2006

Multiple message broadcasting and gossiping in the dynamically orientable graphs

Gayane R. Goltukhchyan
West Virginia University

MULTIPLE MESSAGE BROADCASTING AND GOSSIPING IN THE DYNAMICALLY ORIENTABLE GRAPHS

Gayane R. Goltukhchyan

Dissertation submitted to the
College of Engineering and Mineral Resources
at West Virginia University
in partial fulfillment of the requirements
for the degree of

Doctor of Philosophy
in
Computer and Information Science

Frances L. Van Scoy, Ph.D., Chair
John M. Atkins, Ph.D.
Elaine M. Eschen, Ph.D.
James D. Mooney, Ph.D.
Wafik H. Iskander, Ph.D.

Lane Department of Computer Science and Electrical Engineering

Morgantown, West Virginia
2006

Keywords: Gossiping, Multiple message broadcasting, Dynamically Orientable Graph

ABSTRACT

MULTIPLE MESSAGE BROADCASTING AND GOSSIPING IN THE DYNAMICALLY ORIENTABLE GRAPHS

Gayane Goltukhchyan

This research investigates the problems of gossiping and multiple message broadcasting in dynamically orientable graphs of different network topologies. These are new problems never attempted before. Dynamically orientable graphs and six different network topologies are considered: paths, cycles, stars, binary trees, complete trees and two-dimensional grids. Information dissemination in graphs that are dynamically orientable requires that number of messages sent in each direction along an edge be balanced and therefore necessitates a different approach in gossiping and multiple message broadcasting.

The obvious upper bound for gossiping and multiple message broadcasting in dynamically orientable graphs is twice the best known time for gossiping and multiple message broadcasting in classical graphs. This is obtained by inserting an additional time step t' after each time step t in the classical graph algorithm in which all calls of time step t are repeated with messages moving along the same edges but in the opposite direction to reset the bias of these edges. Finding better bounds for gossiping and multiple message broadcasting in dynamically orientable graphs is the goal of this research.

For each network topology an algorithm is proposed to perform gossiping and multiple message broadcasting. For some network topologies proposed algorithms for dynamically orientable graphs achieved the same upper bound as it is known for classical graphs, for example, gossiping in dynamically orientable grid graphs. In some cases the best time is the twice the best known time for gossiping and multiple message broadcasting in classical graphs, for example, gossiping in dynamically orientable star graphs. In other cases, good time bounds are achieved that are very close to the upper bounds in classical graphs, for example, multiple message broadcasting in dynamically orientable grid graphs. Multiple message broadcasting in dynamically orientable cycle graphs is also a good example of close upper bounds. As number of messages increases bounds become very close to each other.

DEDICATION

I would like to dedicate this dissertation to my mother Iskra and my father Albert, and to my husband Garen and son Artashes. I would also like to thank my brothers Ruben and Levon for their love and support. Without the support and encouragement of my family throughout my entire academic career, I would not be where I am today.

ACKNOWLEDGMENT

I would like to thank my advisor, Dr. Frances L. Van Scoy, for her guidance, support and encouragement. I am grateful for the time she spent advising me, reviewing my dissertation and providing constructive revisions.

I also want to thank Dr. John M. Atkins, Dr. Elaine M. Eschen, Dr. James D. Mooney and Dr. Wafik H. Iskander for serving as members of my committee and for their help and support.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS

The following notation is used throughout the paper:

G	a graph (p.1)
V	a set of vertices (p.1)
E	a set of edges (p.1)
$G\ (V,E)$	graph G with the set of vertices V and set of edges E (p.6)
v_i	vertex i (p.40)
n	number of the vertices in the graph (p.6)
m	number of messages (in case of multiple-message broadcasting) (p.40)
DOG_G	dynamically orientable graph (p.2)
$H1$	half-duplex processor-bound model (p.5)
$F1$	full-duplex processor-bound model (p.5)
k	half-duplex DMA-bound model (p.5)
Fk	full-duplex DMA-bound model (p.5)
H^*	half-duplex link-bound model (p.5)
F^*	full-duplex link-bound model (p.5)
$D\ (G)$	diameter of the graph G (p.6)
$d\ (v)$	degree of the vertex v (p.6)
P_n	path graph (p.6)
C_n	cycle graph (p.6)
B_n	binary tree (p.7)
T_k^m	complete tree (p.8)
S_n	star graph (p.8)

$G_{m,n}$	grid graph (p.9)
$g_M(G)$	the gossiping time of the graph G under model M, where (p.9) $M \in \{F1, F*, Fk, H1, H*, Hk\}$
$b_M(v)$	the broadcast time of a vertex v under model M (p.9)
$b_M(G)$	the broadcast time of a graph G under model M (p.10)
$b\_m_M(v)$	the broadcast time of *m* messages from a vertex *v* under model *M* (p.10)
$b\_m_M(G)$	the broadcast time of *m* messages of a graph *G* under model *M* (p.10)
h	the height of a tree graph (p.27)

Chapter 1

1 Introduction

This dissertation investigates the problems of information dissemination in dynamically orientable graphs (DOG). DOG and other related terms will be defined in this section as well as gossiping and multiple messages broadcasting, the two network communication problems investigated. These problems have not been investigated with regards to dynamically orientable graphs of different network topologies. Reviewing existing gossiping, broadcasting and multiple message broadcasting algorithms for different topologies, new algorithms for dynamically orientable graphs of different network topologies will be proposed.

1.1 Basic Definitions

With the latest developments in parallel and distributed computing, more and more systems utilize parallel architecture. Network communication is a central task in the field of parallel and distributed computing. Modern powerful high-performance computers have thousands of processors interconnected with each other. The processors are connected according to a network topology. The network topology is usually depicted as a *graph* whose vertices (nodes) correspond to processors and whose edges (links) correspond to communication links. A *graph* G is defined as a pair (V, E), where V is a set of vertices, and E is a set of edges between the vertices, $E = \{(u,v) \mid u,v \in V\}$. The edges of the graph can be undirected (u,v), directed from u to v $<u,v>$ or directed from v to u $<v,u>$. A graph is called directed graph if its edges are directed and an undirected graph if no direction is specified for its edges. Throughout this dissertation the terms *processor*, *vertex* and *node* will be used interchangeably as well as terms *link* and *edge*.

Dynamically orientable graphs (DOG) were introduced by Klostermeyer [K1996]. In a DOG, an edge (u,v) can be in one of three states: *neutral* (undirected), directed <u,v> or directed <v,u>. All edges are neutral in the beginning. Once an edge (u,v) is used in the direction from u to v, it becomes directed edge <v,u> and the only way it can be used again is from v to u. After the edge <v,u> is used, it returns to the neutral state and can be used again in either direction. All graphs in this dissertation are dynamically orientable graphs.

Klostermeyer in [K1996] studied a *k-server* problem in DOGs, which he called the *k-train* problem. The *k-server* problem is defined as follows: there are k mobile servers and a sequence of requests that need a server to complete some job. Requests are performed in the FIFO basis. The goal is to minimize total distance traveled by servers satisfying a sequence of requests. The *k-server* problem can be *offline* when a sequence of requests is known in advance and *online* when requests are made one at a time. In [K1996] reference is made to the known fact that weight of the trains create enormous amount of friction as they travel and hence gradually shift the track in the direction of the travel. It is therefore desirable to have an equal number of trains traveling in the opposite directions. Klostermeyer assumed that servers are trains and DOG is a railroad system and studied the offline k-train problem. He showed that the problem is NP-complete for $k \geq 1$ and gave approximation algorithms for the general problem and optimal algorithms for the 2-train problem in trees and 1-train problem in cycles (rings) and discussed the 1-train problem in complete graphs.

Information constantly flows between processors. This information flow is greatly defined by the specific program that runs on these processors. Despite the individuality of such communications, several general types of communications can be identified. They include broadcasting, multiple message broadcasting, gossiping, scattering and gathering.

Communication lines are arranged according to some scheme that allows message passing from one part of the network to another via multiple links. Effective message passing among processors is the key to performance of parallel computers since it constitutes the major part of the computational problem. Broadcasting and gossiping are two common types of network communication.

Given a graph G with V vertices and E edges, *broadcasting* is the information dissemination problem where one node $u \in V$ has a piece of information (message) and needs to pass this information to all other nodes, i.e. after time t every node learns this piece of information. Broadcasting is also called *one-to-all* communication.

Multiple message broadcasting is a modification of broadcasting problem in which the source node has several messages that it needs to broadcast to all other nodes and, for example, due to the size of the messages, it cannot send these messages as one package.

Gossiping is the information dissemination problem where initially each vertex possesses a piece of information and needs to pass it to all other nodes, i.e. after time t every node learns other nodes' messages. Gossiping is also called *all-to-all* communication.

Given a graph G with V vertices and E edges, *scattering* is a problem where one vertex $u \in V$ has to pass different messages to other nodes, i.e. after time t every node learns different piece of information that came from the source. Scattering is also called *personalized one-to-all* communication.

Gathering is the opposite of scattering, when one node collects messages from all other nodes.

These types of general communication occur in many parallel tasks and parallel algorithms, for example, global processor synchronization, image processing, manipulating

distributed databases, matrix multiplication, solving of linear systems, computing of Discrete Fourier Transform, parallel sorting.

When solving an information dissemination problem the goal is to identify a parallel algorithm that leads to efficient problem solution. Each parallel algorithm is the sequence of communication steps, called *communication rounds* [HK1996]. During each communication round, a processor may be engaged in message passing with one of its adjacent processors. One can strive to minimize the time, or in other words communication rounds, of the algorithm or to minimize the number of communications (calls) between any two processors during all communication rounds [HHL1988].

Several communication rules can be defined when dealing with network communications.

Communication links between any two processors can operate in two different ways:

1) In a single round, a message can travel in one direction only, i.e. a processor can be either a sender or a receiver. This resembles telegraph or e-mail communication. In this case, communication is said to be *half-duplex*.
2) In a single round, messages can travel in both directions, i.e. a processor is simultaneously a sender and a receiver. This resembles telephone communication. In this case, communication is said to be *full-duplex*.

During a communication round, the processor may use one link, several links or all links. The communication is:

1) *Processor-bound*, when a processor can only use one of its links. It is also called *whispering* or *1-port*.

2) *Link-bound*, when a processor can use all of its links at the same time. It is also called *shouting* or *n-port*.
3) *DMA-bound*, when processor can use only *k* of its links at the same time. It is also called *k-port*.

Below are some abbreviations that are used to describe these models.

a) H1 – half-duplex processor-bound model.
b) F1 – full-duplex processor-bound model.
c) H*k* – half-duplex DMA-bound model.
d) F*k* – full-duplex DMA-bound model.
e) H* – half-duplex link-bound model.
f) F* – full-duplex link-bound model.

Communications models can be of two kinds: linear and constant models [FL1994]. The linear model is the model where the communication time T depends on the length L of the message: $T = \beta + L\,\tau$, where β is the cost of a start-up and τ is the propagation time of a message of unit length. The constant model is the model where we assume that the communication time T is equal to one time unit ($T = 1$).

1.2 Network topology

In this dissertation gossiping and multiple message passing in DOGs of different network topologies are studied. Below are some notations and definitions of the graphs examined in this dissertation.

G (V,E) denotes the graph *G* with the set of nodes *V* and set of edges *E*. The distance between two nodes in a graph is the length of a shortest path between these two nodes. The number of nodes in the graph is denoted by *n*. The diameter denoted by *D(G)* (or simply D if we know

exactly what graph it refers to) is the maximum distance between any two vertices in *G*. The degree of a vertex *v* denoted by *d(v)* is the number of links it has.

- Path P_n.

In the path of length n (Figure 1.1), nodes are numbered from 1 to *n* and have edges that connect them with two adjacent nodes, except node 1 and *n*.

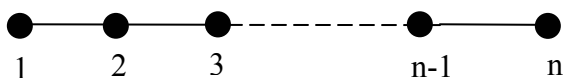


Figure 1.2.1 Path P_n.

The diameter D(P_n) = n -1, d(1) = d(n) = 1, and each vertex *v* has degree d(*v*) = 2 for $1 < v < n$.

- Cycle C_n.

In the cycle graph C_n (Figure 1.2), nodes are numbered from 1 to *n* and have edges that connect them with two adjacent nodes and connect node 1 to node *n*.

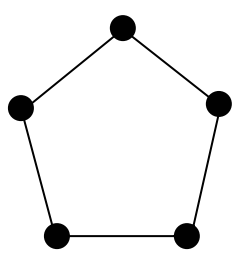


Figure 1.2.2 Cycle C_5.

The diameter D (C_n) = $\left\lfloor \frac{n}{2} \right\rfloor$, and each vertex *v* has degree d (*v*) = 2.

- Binary tree B_n.

A tree graph is a graph whose set of edges connect a set of nodes and contains no cycles. A consequence of this definition is that the number of edges is n-1. A rooted tree is a tree with special node called root. In a rooted tree, each of the nodes that is one edge further away from a given node is called a child. Each edge links a parent node to one of its children. Root node has no parent. Every other node has exactly one parent. Nodes at the bottom of a tree are called leaves. The binary tree is a tree where each parent has at most two children (Figure 1.3). The number of nodes in B_n is n and number of edges is (n-1).

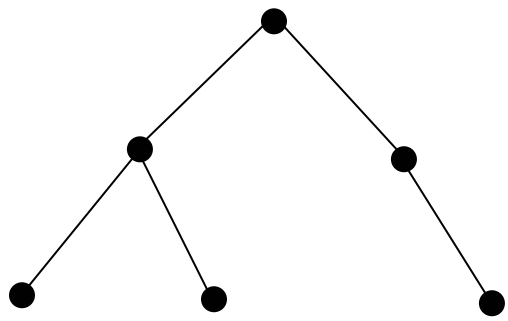


Figure 1.2.3 Binary graph B_6.

The diameter D (B_n) depends on the configuration of the tree. It reaches its maximum value when the tree is a path with diameter of (n-1). Each vertex v in a binary tree has maximum degree d (v) of 3. The root node has maximum degree d (*root*) of 2.

- Complete tree T_k^m

The complete k-ary tree T_k^m of height m is a graph whose nodes are all k-ary strings of length at most m and whose edges connect each string α of length $i\,(0 \le i \le m)$ with the string $\alpha\, a, a \in \{0, \ldots k-1\}$ of length i +1 (Figure 1.4). The root node is an empty string and a node α is at level $i, i \ge 0$ if α is a string of length i.

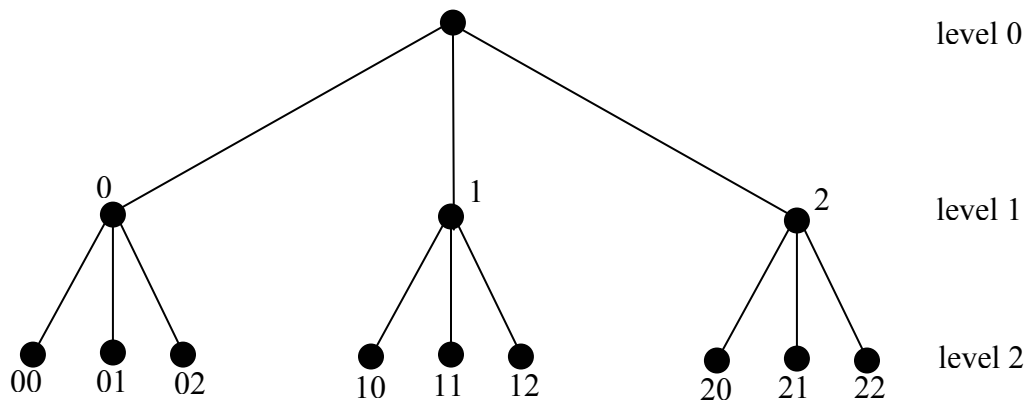


Figure 1.2.4 Complete tree T_3^2.

T_k^m has $\frac{k^{m+1}-1}{k-1}$ nodes. The diameter D (T_k^m) = 2m. The maximum degree of a vertex is k + 1. The minimum degree of the root d (*root*) is k.

- Star graph S_n.

The star graph S_n is a tree with n nodes where (n-1) nodes are connected to one node, the center (Figure 1.5).

The diameter D (S_n) = 2. All vertices except the center have degree of 1, and the center node has degree of (n-1).

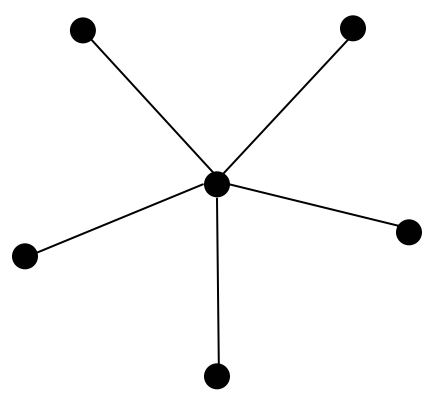


Figure 1.2.5 Star graph S_6.

- Grid $G_{m,n}$.

The 2-dimensional *m x n* grid is a graph with *m* x *n* vertices that are connected in a crisscross manner (Figure 1.6). $G_{m,n}$ is the product of path graphs on *m* and *n* vertices. The number of nodes is *mn*.

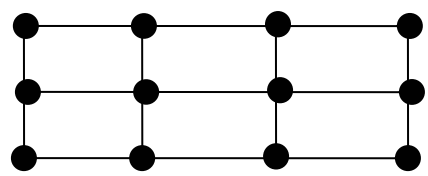


Figure 1.2.6 Grid graph $G_{3,4}$.

The diameter D($G_{m,n}$) = (m-1) + (n-1), and the maximum degree of a vertex, d(*v*), is 4.

1.3 Model assumptions

Commonly accepted notation will be used for gossiping and broadcasting time. Given graph *G(V,E)* the gossiping time, $g_M(G)$, is the number of communication rounds in the algorithm necessary to complete gossiping in the graph *G* under model *M*, where $M \in \{F1, F*, Fk, H1, H*, Hk\}$. The broadcast time of a vertex *v* under model *M,* $b_M(v)$, is the number of communication rounds in the algorithm necessary to complete broadcasting of a message from vertex *v* to all other vertices. The broadcast time of a graph *G* under model *M,* $b_M(G)$, is the maximum of the broadcast time of each vertex in G, i.e. $b_M(G) = \max\{b_M(v) \mid v \in V\}$. The broadcast time of *m* messages from a vertex *v* under model *M,* $b\_m_M(v)$, is the number of communication rounds in the algorithm necessary to complete broadcasting of *m* messages from vertex *v* to all other vertices. The broadcast time of *m* messages of a graph *G* under model *M,* $b\_m_M(G)$, is the maximum broadcast time of *m* messages of any vertex in G, i.e. $b\_m_M(G) = \max\{b\_m_M(v) \mid v \in V\}$.

Below are the conditions under which models are considered:

a) All models are processor-bound, i.e. a processor can only use one of its links at a time.

b) All models are operating in half-duplex mode. Full-duplex mode is meaningless with regards to DOG since edges in DOG have orientation (edges can became directed).

c) Messages are sent in store-and-forward or packet –switched manner. When a processor has several messages it can assemble them into one packet and send it to other processors. In case of multiple message broadcasting it is assumed that either the length of a message or the length of all messages combined is too big to be sent in one packet and it needs to be broken down into several pieces (packets).

d) The model considered is constant model. Communication time, i.e. time to send message from one processor to another along a single edge, is constant. We assume that the length of the message or the size of the packet is relatively small and time necessary to transmit it is one unit of time.

e) Once an algorithm for gossiping or multiple message broadcasting finishes, all edges in a DOG should be in a neutral state, i.e. the system should be in neutral state and ready for another round of the algorithm.

1.4 Goal

This dissertation investigates the problems of gossiping and multiple message broadcasting in DOGs of different network topologies. As was already mentioned these are the new problems never attempted before. Considering graphs that are dynamically orientable puts a restriction on message movements. When attempting to obtain time bounds for network communications it is expected that those bounds will often be worse than the time bound for gossiping and multiple message broadcasting in the usual models. The obvious upper bound for gossiping and multiple

message broadcasting in DOG is twice the best known time for gossiping and multiple message broadcasting in classical graphs. It can be obtained by executing each step of gossiping or multiple- message broadcasting algorithms for classical graphs at odd times and then resetting the edges involved in the odd times at the even times. Thus the goal of the dissertation is to obtain better bounds for gossiping and multiple message broadcasting in DOGs.

Chapter 2

2 Literature Review

The topic of gossiping and multiple message broadcasting in DOG has not been investigated yet. Gossiping, broadcasting and multiple message broadcasting in different kinds of network topologies have been studied, and a short summary of the main results is given below.

2.1 Gossiping

Hedetniemi and Hedetniemi in [HHL1988] conducted a survey of gossiping and broadcasting in different types of communication networks. For grid graphs, letting T(G) be the minimum time necessary to complete gossiping in a graph G, M(G) be the minimum number of calls necessary to complete gossiping in a graph G and N(G) be the minimum number of calls necessary to complete gossiping in minimum time in a graph G, first they have results for any path P_n of length n, $T(P_n) = \begin{cases} n-1 & \textit{for n even} \\ n & \textit{for n odd} \end{cases}$, $N(P_n) = M(P_n) = 2n - 3$. Then for any grid $G_{m,n}$ they have the following results: $T(G_{m,n})$ = the diameter of $G_{m,n}$ (except $G_{3,3}$), $N(G_{m,n}) = 2mn - 4$ if m and n are even, and $M(G_{m,n}) = 2mn - 4$. If $t_L(T)$ is the minimum time required to gossip in a tree, then $2\lceil \log n \rceil - 1 \le t_L(T) \le 2n-3$. If $t_r(n)$ is the minimum time necessary to complete gossiping under the DMA-bound (k-port) half-duplex model, then $t_r(n) = \lceil \log_{r+1} n \rceil$. Other gossiping problems were studied where certain additions were taken into account, for example, no one hears their own message (NOHO), and no transmission of message can be duplicated (NODUP). Krumme, Cybenko and Venkataraman in [KCV1992] studied gossiping under H1 model providing optimal results for several networks. They argued that finding optimal algorithms for

gossiping under the H1 or F1 model is NP-Complete. Krumme et al claimed that gossiping under the H1 model in any graph with n vertices requires at least $\left|\log_{\rho} n\right| \approx 1.44 \lg N$, where ρ is the golden ratio $\frac{1}{2}(1+\sqrt{5})$ which was also discovered by Even and Monien [EM1989]. For *d*-dimensional grids $G_{n1,n2.....nd}$ ($d \geq 2, n_i \geq 9$ for all i) it is possible to finish gossiping in the number of rounds equal to the diameter of the grid which is optimal. In order to be able to solve gossiping in optimal time the grid's size in each dimension should be greater than or equal to four. For 2-dimensional grids the optimal algorithm is known for 7x7 and larger grids. The problem is open for grids between 4x4 and 6x7. For 3 dimensional grids the smallest optimal algorithm known is for 9x9x5 grid. For cycles, the lower bound is $\left\lfloor \frac{n}{2} \right\rfloor + \left\lceil \sqrt{2n} \right\rceil - 1$ and the upper bound is $\left\lfloor \frac{n}{2} \right\rfloor + \left\lceil \sqrt{2n} \right\rceil + 2$. Hromkovic et al [HJM1993] investigated the problem of gossiping in cycles under the H1 model and claimed that for even $n > 3$ the time to gossip is $\frac{n}{2} + \left\lceil \sqrt{2n} \right\rceil - 1$ and for odd $n > 3$ the lower bound is $\left\lceil \frac{n}{2} \right\rceil + \left\lceil \sqrt{2n} - \frac{1}{2} \right\rceil - 1$ and the upper bound is $\left\lceil \frac{n}{2} \right\rceil + \left\lceil 2\sqrt{\left\lceil \frac{n}{2} \right\rceil} \right\rceil - 1$. Fraigniaud and Lazard in [FL1994] provided a survey of communications methods in usual networks. They use the following notations: the gossip time of a graph G under model $M \in \{F1, H1, Fk, Hk, F^*, H^*\}$, $g_M(G)$, is the maximum time necessary to complete gossiping in the graph; the broadcast time of a graph G under model M, $b_M(G)$, is the maximum time necessary to complete broadcasting from any vertex in the graph. In case of constant model the trivial bounds are $\lceil \log_2 N \rceil \leq b_{H1}(G) \leq g_{H1}(G) \leq 2b_{H1}(G)$ and $\lceil \log_2 N \rceil \leq b_{F1}(G) \leq g_{F1}(G) \leq 2b_{F1}(G)$.

As we can see here, the time for gossiping cannot be larger than twice the time for broadcasting.

For cycles C_n, $g_{H1}(C_n) = n/2 + \left|\sqrt{2n}\right| - 1$, if n is even, n > 3, and

$\lceil n/2 \rceil + \left\lceil \sqrt{2n} - \frac{1}{2} \right\rceil - 1 \le g_{H1}(C_n) \le \lceil n/2 \rceil + \left\lceil 2\sqrt{(n+1)/2} \right\rceil - 1$, if n is odd. For d-grid $G_{n1,n2,....nd}$ = P_{n1} x P_{n2} x x P_{nd} assuming that $d \ge 2$ and $n_i \ge 9$ for all i, the gossip problem for the P_{n1} x P_{n2} x x P_{nd} grid under H1 model of communication is solvable in a number of steps equal to the diameter of the grid [KCV1992]. Hromkovic et al [HK1996] conducted a survey of established results for gossiping under F1 and H1 models stating previously mentioned bounds.

Graph	Num. of vertices	Max. degree	Diameter D	Lower bound	Upper bound
P_n	n even n odd	2	n - 1	n n + 1	n n + 1
C_n	n even n odd	2	$\left\lfloor \frac{n}{2} \right\rfloor$	$\frac{n}{2} + \left\lceil \sqrt{2n} \right\rceil - 1$ $\left\lceil \frac{n}{2} \right\rceil + \left\lceil \sqrt{2n} - \frac{1}{2} \right\rceil - 1$	$\frac{n}{2} + \left\lceil \sqrt{2n} \right\rceil - 1$ $\left\lceil \frac{n}{2} \right\rceil + \left\lceil 2\sqrt{\left\lceil \frac{n}{2} \right\rceil} \right\rceil - 1$
B_n	n	3	varies		
T_k^m	$\frac{k^{m+1}-1}{k-1}$	k + 1	2m	Dk	Dk
S_n	n	n-1	2		
$G_{m,n}$	mxn	4	m + n - 2	D	D for 7x7 & larger

Table 2.1 Gossiping under H1 model.

2.2 Broadcasting

Hedetniemi and Hedetniemi in their survey [HHL1988] mentioned the study of broadcasting in grid graphs $G_{p,n}$ with the following results: if the originator of the message is a corner vertex then broadcasting time from a vertex is $p + n - 2$; if the originator of the message is a side vertex then broadcasting time from a vertex is maximum distance from that vertex to a corner vertex; if the originator of the message is an interior vertex at *(i,j)* then broadcasting time from a vertex is

maximum distance from that vertex to a corner vertex plus 1 if $j = \frac{n+2}{2}$ or plus 2 if $j = i = \frac{n+1}{2} = \frac{p+1}{2}$. Fraigniaud and Lazard in [FL1994] conducted a survey of existing at that time communication algorithms. For broadcasting under F1 model they compiled the following results: for cycles they have $b_{F1}(C_n) = \left\lceil \frac{n}{2} \right\rceil = \begin{cases} D & if\ n\ even \\ D+1 & otherwise \end{cases}$; for d-grid graphs $b_{F1}(G_{n1,n2.....nd}) = D$; Hromkovic et al [HK1996] conducted a survey of established results for broadcasting stating previously mentioned bounds.

Graph	Num. of vertices	Max. degree	Diameter D	Lower bound	Upper bound
P_n	n	2	n - 1	$\left\lceil \frac{n}{2} \right\rceil$	n - 1
C_n	n	2	$\left\lfloor \frac{n}{2} \right\rfloor$	$\left\lceil \frac{n}{2} \right\rceil$	$\left\lceil \frac{n}{2} \right\rceil$
B_n	n	3	varies		
T_k^m	$\frac{k^{m+1}-1}{k-1}$	k + 1	2m		
S_n	n	n-1	2	n - 1	n - 1
$G_{m,n}$	mxn	4	m + n - 2	m + n - 2	m + n - 2

Table 2.2. Broadcasting under F1 model.

2.3 Multiple message broadcasting

Farley in [F1980] discussed general lower and upper bounds for broadcasting time in case of one and *m* messages. For one message broadcasting, he gave lower bound of $\lceil \log_2 n \rceil\ for\ n \geq 1$ and upper bound of $(n-1)\ for\ n \geq 1$. For multiple message broadcasting the lower bound is $2(m-1)+D$ where D is the diameter of a graph and the upper bound is

$d_{\max}(m-1)+(n-1)$, where $d_{\max}$ is a maximum degree of a vertex in a graph. Van Scoy and Brooks in [VB1994] proposed two algorithms for broadcasting *m* messages in a $n \times n$ grid that require correspondingly $\max(n+\frac{5}{2}m-1, 2n+2m-4)$ and $2n+\frac{5}{2}m-4$ rounds. They also presented the algorithm for broadcasting *m* messages in an $n \times n \times n$ grid that requires $3n+\frac{10}{3}m+2$ rounds and conjectured similar algorithm exists for a *d*-dimensional grid that requires $dn+\frac{(d^2+1)}{dm}+C$ rounds (C is a constant, n is a number of vertices in each direction). In [VB1994] the assumption was that the corner node of the grid had initial messages. Wojciechowska and Van Scoy in [WV1996] presented an algorithm for broadcasting m messages in a $n \times n$ grid that requires $2n+2m+2$ rounds with corner node as a source and an algorithm that requires $dn+3m+C$ rounds to broadcast m messages from the corner of *d*-dimensional grid. Bar-Noy and Kipnis in [BK1997] studied multiple message broadcasting in postal model which has several assumptions: system is fully connected, i.e. they dealt with complete graph K_n, sending or receiving of a packet (one message) takes one unit of time and message delivery has some communication latency λ. They defined generalized Fibonacci function $F_\lambda(t)$ and its index function $f_\lambda(n)$ as follows: $F_\lambda(t)=\begin{cases}1 & if\ 0 \le t < \lambda, \\ F_\lambda(t-1)+F_\lambda(t-\lambda) & if\ t \ge \lambda\end{cases}$ and $f_\lambda(n)=\min\{t : F_\lambda(t) \ge n\}$. Two algorithms were proposed. The first algorithm has running time of $3m+f_\lambda(n)+O(\lambda)$. The second algorithm has running time of $m+2f_\lambda(n)+O(\lambda)$. Roditty and Shoham in [RS1997] proposed an algorithm for broadcasting m messages in a *d*-dimensional grid $(d \ge 3)$ with the same assumption that a corner vertex of the grid has initial messages. Their time bound is

$d(n+m-2)$ rounds. Diks, Lingas and Pelc in [DLP1999] described an optimal algorithm for broadcasting multiple messages in trees. They considered simultaneous send/receive mode (when a node can send a message to one node and receive a message from another) and half-duplex whispering mode. They concluded that in both modes broadcasting m messages among n processors in a tree can be done in time $O(nm)$. Liestman et al in [LSS2000] proposed an algorithm for broadcasting multiple messages in k-dimensional hypercubes under telephone model. The time to broadcast m messages in H_k for $m \le 2^{d-1}$ (here d is a degree and d = k) is $2m+d-2$, for $m \ge 2^d+1$ is $2m+d-2-\left\lfloor \frac{m-1}{2^{d-1}} \right\rfloor$, and for $2^{d-1}+1 \le m \le 2^d$ is between $2m+d-3$ and $2m+d-2$. The main result of [LSS2000] is disproval of Farley's lower bound in [F1980].

Graph	Num. of vertices	Max. degree d	Diameter D	Lower bound $2(m-1)+D$	Upper bound d(m-1) + (n-1)
P_n	n	2	n - 1	2(m-1) + n -1	2(m-1) + (n-1)
C_n	n	2	$\left\lfloor \frac{n}{2} \right\rfloor$	2(m-1) + $\left\lfloor \frac{n}{2} \right\rfloor$	2(m-1) + (n-1)
B_n	n	3	varies	2(m-1) + D	3(m-1) + (n-1)
T_k^p	n=$\frac{k^{p+1}-1}{k-1}$	k + 1	2p	2(m-1) + 2p	O(nm)
S_n	n	n-1	2	2m	m(n-1)
H_k	2^k	k	k	2(m-1) + k *	for $m \le 2^{k-1}$: $2m+k-2$ for $m \ge 2^k+1$: $2m+k-2-\left\lfloor \frac{m-1}{2^{k-1}} \right\rfloor$ for $2^{k-1}+1 \le m \le 2^k$: btw $2m+k-3$ & $2m+k-2$
$G_{n,n}$	nxn	4	2n - 2	2m+2n-4	2n + 2m + 2

* - proven incorrect for hypercubes

Table 2.3 Multiple message broadcasting.

Chapter 3

3 Gossiping

Given a graph G with V vertices and E edges, *gossiping* is the information dissemination problem where initially each vertex possesses a piece of information and needs to pass it to all other nodes, i.e. after time t every node learns other nodes' messages. Gossiping is also called *all-to-all* communication. When considering gossiping problems, the usual assumption is that each vertex sends cumulative information about messages that it has previously received from other vertices plus its own message.

General lower bounds for any graph G (V, E) with n nodes are proved in [HK1996].

$$g_{H1}(G) \geq g_{F1}(G) \geq \begin{cases} \lceil \log_2 n \rceil & \textit{for even } n \\ \lceil \log_2 n \rceil + 1 & \textit{for odd } n \end{cases}$$

In a gossiping algorithm, if a vertex knows all messages at time, it is called an *expert* at that time and all subsequent times.

In the following subsections gossiping algorithms are presented for dynamically orientable graphs of different network topologies. Networks topologies considered are paths, cycles, binary trees, complete trees, stars and 2-dimensional grids.

Models considered here are half-duplex processor-bound models (H1 models). In a network with n vertices, initially each vertex v_i knows message m_i and after gossiping is finished, each vertex v_i knows all messages $m_1 \ldots m_n$. Each vertex receives one or more messages from one or more vertices, combines them into one cumulative message and passes them along to other vertices.

3.1 Path P_n

In the path of length n (Figure 3.1.1), nodes are numbered from 1 to n and have edges that connect them with two adjacent nodes, except node 1 and n. The diameter $D(P_n) = n - 1$, $d(1) = d(n) = 1$, and each vertex v has degree $d(v) = 2$ for $1 < v < n$.

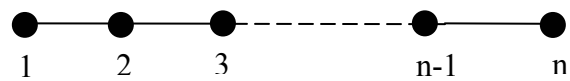


Figure 3.1.1 Path P_n.

The known time for gossiping in the path graph P_n under the H1 model is proven in [HK1996].

$$g_{H1}(P_n) = \begin{cases} n & \text{if } n \text{ is even} \\ n+1 & \text{if } n \text{ is odd} \end{cases}$$

The algorithms given below achieves the same gossiping time for dynamically orientable path DOG_P_n. Gossiping time in DOG_P_n is n when n is even and $(n + 1)$ when n is odd.

Algorithm **H1_GOSSIP_ DOG_P_n** (n is even)

For i = 1 to $(\frac{n}{2} - 1)$ *do*

v_i *passes message* m_i *to* v_{i+1} *at time t = i*

v_{n-i+1} *passes message* m_{n-i+1} *to* v_{n-i} *at time t = i*

For i = $\frac{n}{2}$ v_i *passes its messages* $m_1 \ldots m_{n/2}$ *to* v_{i+1} *at time t = i*

For i = $\frac{n}{2} + 1$ v_i *passes its messages* $m_{n/2+1} \ldots m_n$ *to* v_{i-1} *at time t = i*

For i = $\frac{n}{2} + 2$ *to n do*

v_{n-i+2} *passes messages* $m_1 \ldots m_n$ *to* v_{n-i+1} *at time t = i*

$$v_{i-1} \text{ passes messages } m_1 \ldots m_n \text{ to } v_i \text{ at time } t = i$$

Theorem 3.1.1. Algorithm **H1_GOSSIP_ DOG_P$_n$** (n is even) requires n time units to complete gossiping in DOG paths of even size.

Proof: All messages are passed to the two center vertices $v_{n/2}$ and $v_{n/2+1}$. At time $\frac{n}{2}-1$, $v_{n/2}$ and $v_{n/2+1}$ cumulatively know all messages in the network. Vertex $v_{n/2}$ knows messages $m_1 \ldots m_{n/2}$ and vertex $v_{n/2+1}$ knows messages $m_{n/2+1} \ldots m_n$. It takes two time units for them to exchange messages, after that $v_{n/2}$ and $v_{n/2+1}$ each know all messages. Expert vertices $v_{n/2}$ and $v_{n/2+1}$ send cumulative information about all messages back to the end vertices in time $\frac{n}{2}-1$. The total time to complete gossiping in the path DOG_P$_n$ when n is even is n. □

Figure 3.1.2 illustrates algorithm **H1_GOSSIP_ DOG_P$_n$** (n is even) for P_8.

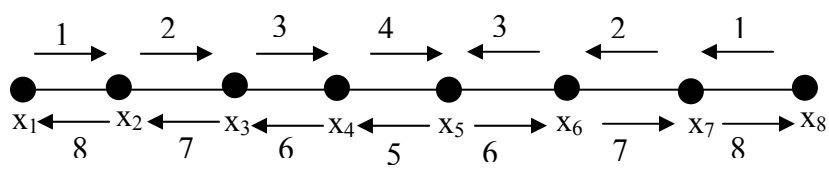


Figure 3.1.2 Gossiping in P_8.

For odd n there are some slight changes in the algorithm.

Algorithm **H1_GOSSIP_ DOG_P$_n$** (n is odd)

For $i = 1$ *to* $(\frac{n-3}{2})$ *do*

v_i *passes message* m_i *to* v_{i+1} *at time* $t = i$

v_{n-i+1} *passes message* m_{n-i+1} *to* v_{n-i} *at time* $t = i$

For $i = \frac{n-1}{2}$ v_i *passes its messages* $m_1 \ldots m_i$ *to* v_{i+1} *at time* $t = i$

For $i = \frac{n+1}{2}$ v_{i+1} *passes its messages* $m_{i+1} \ldots m_n$ *to* v_i *at time* $t = i$

For $i = \frac{n+3}{2}$ v_{i-1} *passes its messages* $m_1 \ldots m_n$ *to* v_{i-2} *at time* $t = i$

For $i = \frac{n+5}{2}$ v_{i-2} *passes its messages* $m_1 \ldots m_n$ *to* v_{i-1} *at time* $t = i$

For $i = \frac{n+7}{2}$ *to* $(n + 1)$ *do*

v_{n-i+3} *passes messages* $m_1 \ldots m_n$ *to* v_{n-i+2} *at time* $t = i$

v_{i-2} *passes messages* $m_1 \ldots m_n$ *to* v_{i-1} *at time* $t = i$

Theorem 3.1.2. Algorithm **H1_GOSSIP_ DOG_P$_n$** (n is odd) requires $(n+1)$ time units to complete gossiping in DOG paths of odd size.

Proof: Messages are passed from the end vertices to the vertices $v_{(n-1)/2}$ and $v_{(n+3)/2}$. At time $\frac{n-3}{2}$, $v_{(n-1)/2}$ knows messages $m_1 \ldots m_{(n-1)/2}$ and $v_{(n+3)/2}$ knows messages $m_{(n+3)/2} \ldots m_n$. At time $\frac{n-1}{2}$ vertex $v_{(n-1)/2}$ passes messages $m_1 \ldots m_{(n-1)/2}$ to vertex $v_{(n+1)/2}$ and vertex $v_{(n+3)/2}$ passes messages $m_{(n+3)/2} \ldots m_n$ to vertex $v_{(n+1)/2}$. Now vertex $v_{(n+1)/2}$ is an expert vertex. Expert vertex $v_{(n+1)/2}$ sends cumulative information about all messages to $v_{(n-1)/2}$ at time $\frac{n+3}{2}$ and to $v_{(n+3)/2}$ at time $\frac{n+5}{2}$. Vertices $v_{(n-1)/2}$ and $v_{(n+3)/2}$ send information about all messages back to the end vertices in time $\frac{n-3}{2}$. The total time to complete gossiping in the path DOG_P$_n$ when n is odd is $(n+1)$. Figure 3.1.3 illustrates algorithm **H1_GOSSIP_ DOG_P$_n$** (n is odd) for P_7 .

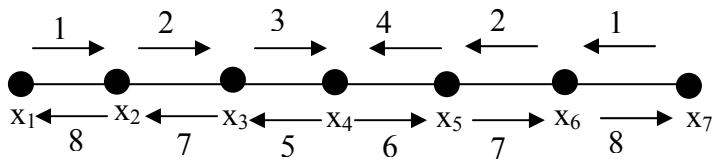


Figure 3.1.3 Gossiping in P_7.

3.2 Cycle C_n

In the cycle graph C_n (Figure 3.2.1), nodes are numbered from 1 to n and have edges that connect them with two adjacent nodes and connect node 1 to node n.

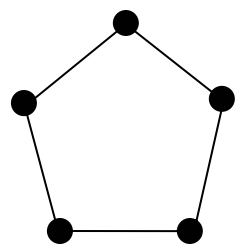


Figure 3.2.1 Cycle C_5.

The diameter $D(C_n) = \left\lfloor \frac{n}{2} \right\rfloor$, and each vertex v has degree $d(v) = 2$.

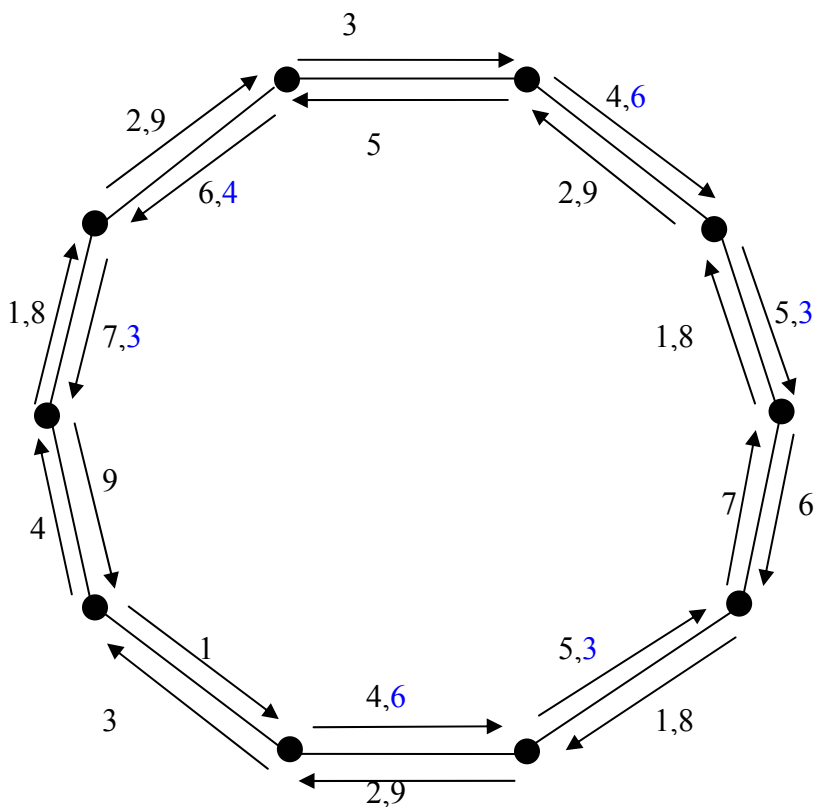


Figure 3.2.2 Gossiping in DOG C_{10}.

The known bounds for gossiping in the classical cycles are $g_{H1}(C_n) = n/2 + \left\lfloor \sqrt{2n} \right\rfloor - 1$, if n is even, n > 3, and $\lceil n/2 \rceil + \left\lceil \sqrt{2n} - \frac{1}{2} \right\rceil - 1 \le g_{H1}(C_n) \le \lceil n/2 \rceil + \left\lceil 2\sqrt{(n+1)/2} \right\rceil - 1$, if n is odd [KCV1992]. The algorithm described in [KCV1992] works well for DOG cycle graphs. One just needs to add some extra empty message passing between nodes in order to leave the system in the stable (neutral state) state. For n > 8, this can be done in such a way that the upper bound does not change. Figure 3.2.2 shows gossiping in DOG C_{10} graph. Time indicated in blue is the time when empty message is passed between vertices in order to orient the edge.

3.3 Star graph S_n

The star graph S_n is a tree with n nodes where (n-1) nodes are connected to one node, the center. The diameter D(S_n) = 2. All vertices except the center have degree of 1, and the center node has degree of (n-1). The star graph S_n is a special case of a complete tree T_{n-1}^{1} (Figure 3.6.1). The gossiping time in S_n is 2n-2.

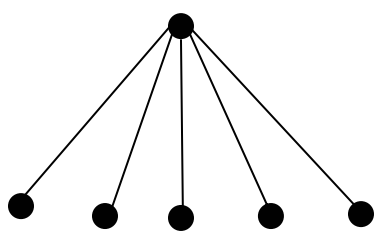


Figure 3.3.1 Star graph S_6.

Algorithm **H1_GOSSIP_ DOG_** S_n

(*Gathering)*

Vertices at level 1 pass messages to their parent (the center) at level 0 in time (n-1).

At time n parent vertex knows messages $m_1 \ldots m_n$

(Broadcasting)

The parent (center node) passes all messages to its children in time (n-1).

Theorem 3.3. Algorithm **H1_GOSSIP_ DOG_** S_n requires (*2n-2*) time units to complete gossiping in DOG stars.

Proof: It is obvious that the parent vertex (the center) is a bottleneck in this case. All messages are gathered at parent vertex in time (n-1). At this time, the parent vertex becomes an expert vertex. It then passes cumulative information about all messages in the system to its children in time (n-1). Total time to gossip in DOG_ S_n is 2n-2. □

Figure 3.6.2 illustrates example of algorithm **H1_GOSSIP_ DOG_** S_n for S_6. Blue lines indicate the gathering part of the algorithm and red lines indicate the broadcasting part.

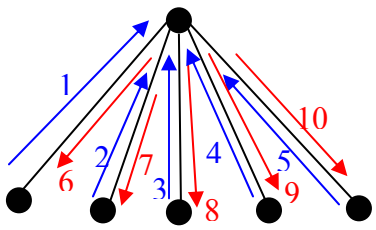


Figure 3.3.2 Gossiping in DOG_ S_6.

3.4 Complete tree T_k^m

A tree graph is a graph whose set of edges connect a set of nodes and contains no cycles. A rooted tree is a tree with special node called root. In a rooted tree, each of the nodes that is one edge further away from a given node is called a child. Each edge links a parent node to one of its children. Root node has no parent. Every other node has exactly one parent. Nodes at the bottom of a tree are called leaves. The complete k-ary tree T_k^m of height m is a tree whose nodes are all k-ary strings of length at most m and whose edges connect each string α of length i $(0 \le i \le m)$

with the string $\alpha a, a \in \{0,\ldots k-1\}$ of length i +1 (Figure 3.4.1). The root node is an empty string and a node α is at level $i, i \geq 0$ if α is a string of length i. T_k^m has $\frac{k^{m+1}-1}{k-1}$ nodes. The diameter D(T_k^m) = $2m$. Maximum degree of a vertex is $k + 1$. Minimum degree of the root d(*root*) is k.

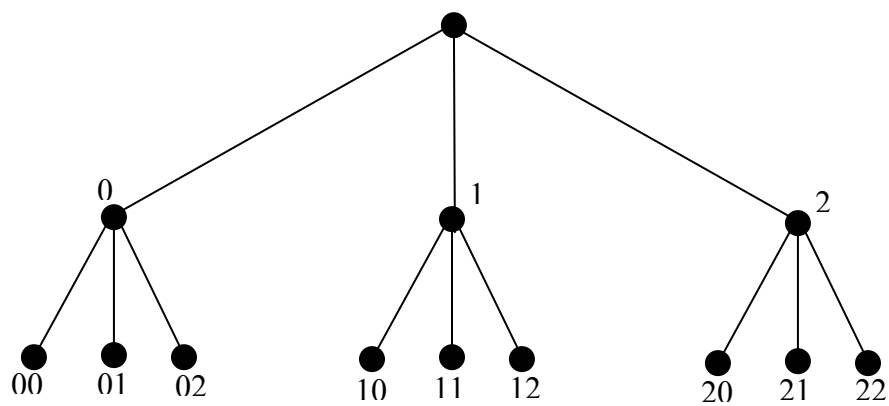


Figure 3.4.1 Complete tree T_3^2.

The known gossiping time for complete trees is $2km$ [HK1996]. The algorithm proposed here requires $2km = Dk$ time units to complete gossiping in DOG_T_k^m which is the same as for classical complete trees.

Algorithm **H1_GOSSIP_ DOG_** T_k^m

(*Gathering)*
For i = m to 1 do
vertices at level i pass messages to their parents at level i-1 in time k.
(Broadcasting)
For i = 0 to m-1 do
vertices at level i pass messages to their children at level i + 1 in time k.

Theorem 3.4. Algorithm **H1_GOSSIP_ DOG_** T_k^m requires *2km* time units to complete gossiping in DOG complete trees.

Proof: After the gathering phase of the algorithm, all edges in the graph are oriented from the root to the leaves. Since gathering is done from the leaves to the root and by levels, no vertex except the leaves passes the messages it knows before it actually receives all messages of its children. The gathering phase requires *mk* time units of time. Thus the root node learns all messages in time *mk* and can start broadcasting at time (*mk* + 1). Broadcasting is done from the root to the leaves and all edges are used in order to transport cumulative message to all vertices thus neutralizing all edges of the graph and leaving the system in the stable state. The broadcasting phase also requires *mk* time units. Again, since broadcasting is done from the root to the leaves by levels, no vertex attempts to broadcast cumulative information before it actually receives it from its parent vertex. The total time to gossip in DOG_T_k^m is $2mk$. □

Figure 3.4.2 illustrates example of algorithm **H1_GOSSIP_ DOG_** T_k^m for T_3^2. Blue lines indicate the gathering phase of the algorithm and red lines indicate the broadcasting phase.

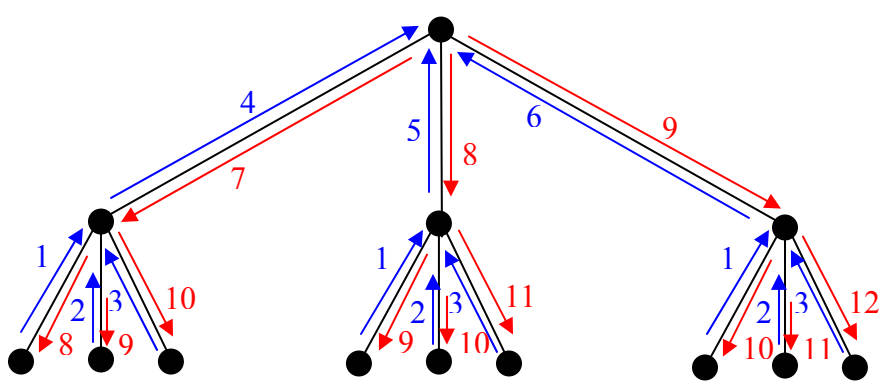


Figure 3.4.2 Gossiping in DOG_T_3^2.

3.5 Binary tree B_n.

The binary tree is a tree in which each parent has at most two children (Figure 3.4.1). Each child is designated as right or left child. The number of nodes in B_n is n and the number of edges is (n-1).

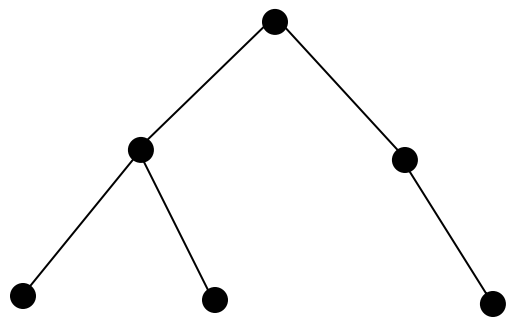


Figure 3.5.1 Binary tree B_6.

The diameter D(B_n) depends on the configuration of the tree. For example, it reaches its maximum value in case when the tree is a path with diameter of (n-1). Each vertex v in a binary tree has maximum degree d(v) of 3. The root node has maximum degree d(*root*) of 2. The height h of the B_n is the maximum distance from the root to the children or the number of levels in the tree. It is known that broadcasting and gossiping time in a binary tree depend on the configuration of the tree. The upper bound of 4h for gossiping in a binary tree is reached when the binary tree is complete, i.e. all vertices have exactly two children with exception of leave nodes. Broadcasting in a complete binary tree T_2^h takes 2h time units and gossiping requires twice the broadcasting time, i.e. 4h. The upper bounds for broadcasting and gossiping in a binary tree with the height h are $b_{H1}(T_2^h) \le 2h$ and $g_{H1}(T_2^h) \le 4h$ correspondingly.

A complete binary tree is a special case of a complete tree T_k^m where k is 2 and m is the height h. The algorithm from Section 3.4 can be applied to complete binary trees resulting in gossip time of 4m, i.e. 4h.

To gossip in an arbitrary binary tree it is best to identify a vertex from which broadcasting can be done in minimum time, gather all messages to that vertex and then broadcast cumulative messages to all vertices. The broadcast center of a tree is a set of vertices from which broadcasting can be done in minimum time (Figure 3.4.2). There exists algorithms that find broadcast center of any tree with *n* vertices in *O(n)* time [SCH1981].

Once the broadcast center of a binary tree is identified, any vertex from the broadcast center can be considered as a root of the tree and the algorithm similar to the algorithm of Section 3.4 can be applied to do gossiping.

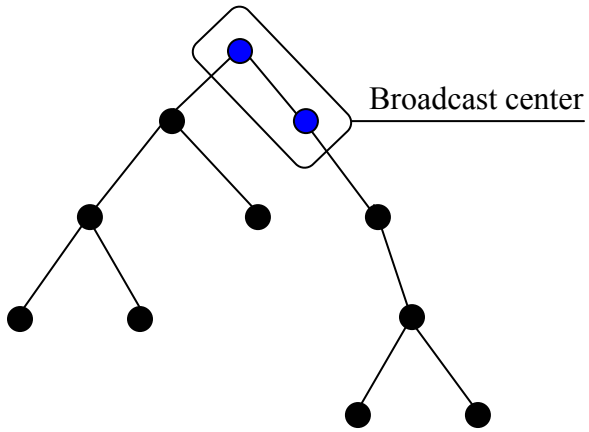


Figure 3.5.2 Broadcast center of B_{11}.

Algorithm **H1_GOSSIP_ DOG_** B_n

Select a vertex from the broadcast center as a root

(*Gathering)*

vertices at level i pass messages to their parents at level i-1 until root knows all messages.

(Broadcasting)

Root propagates cumulative messages down to its children - vertices at level i pass messages to their children at level i + 1.

Theorem 3.5. Algorithm **H1_GOSSIP_ DOG_** B_n requires $4h$ time units to complete gossiping in DOG complete binary trees.

Proof is similar to the one in section 3.4. Gossiping time depends on the configuration of a binary tree. The upper bound on gossiping in a binary tree with the height h is $g_{H1}(B_n) \le 4h$. The same upper bound applies to DOG_B_n. Height h in this case is the maximum distance from the selected root to any vertex. □

Figure 3.5.3 shows the example of **H1_GOSSIP_ DOG_** B_n. Blue lines indicate the gathering phase of the algorithm and red lines indicate the broadcasting phase.

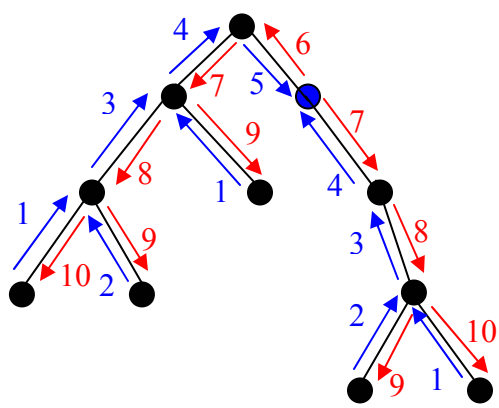


Figure 3.5.3 Gossiping in DOG_B_{11}.

3.6 2-dimensional Grid

The 2-dimensional n x m grid is a graph with n x m vertices that are connected in a crisscross manner (Figure 3.6.1). $G_{n,m}$ is the product of path graphs on m and n vertices. The number of nodes is nm.

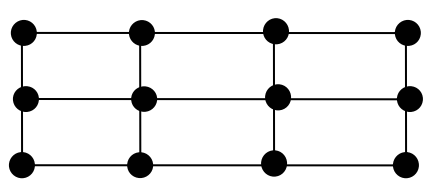


Figure 3.6.1 Grid graph $G_{3,4}$.

The diameter D($G_{n,m}$) = (n-1) + (m-1), and maximum degree of a vertex, d(v), is 4.

For 2-dimensional grids the general optimal algorithm is known for 9×9 and larger grids which equals to the diameter of the graph. Krumme, Cybenko and Venkataraman in [KCV1992] gave a general algorithm for the grids of size 9×9 and larger that accomplishes gossiping in time equal to the diameter of the graph. They developed specific communication patterns for 5x5 and 6x6 grids. Figure 3.6.2 and Figure 3.6.3 depict these communication patterns [KCV1992].

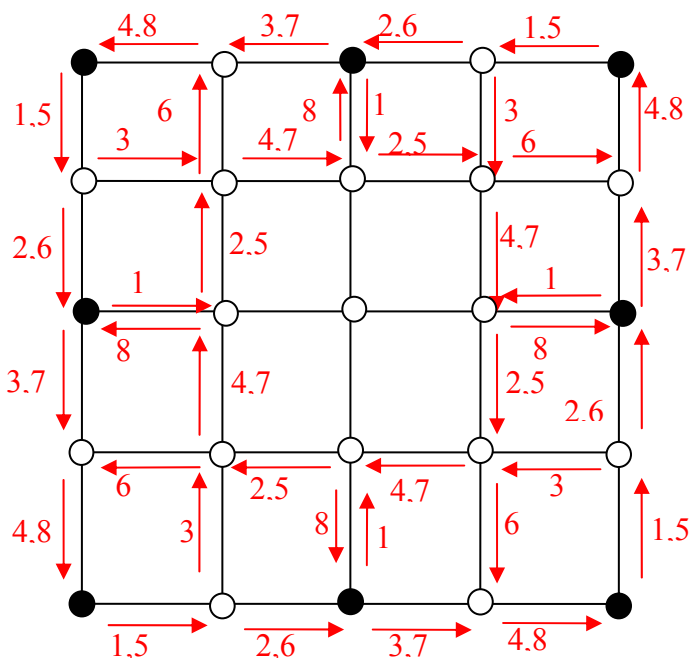


Figure 3.6.2 Communication pattern in $G_{5,5}$ [KCV1992].

Vertices represented by solid dots are called primary points and vertices represented by hollow dots are called secondary points. They showed that messages initially concentrated at primary points are distributed to other primary points in time that equals to the diameter of the graph (in this case $G_{5,5}$ and $G_{6,6}$). They argue that for any grid 9×9 or larger, one can collect messages to the primary points of 5×5 or 6×6 subgrid centered in the larger grid, apply communication patterns from Figure 3.6.2 and 3.6.3 and then sent cumulative message back to other vertices. All of these can be done in optimal time equaled to the diameter of the graph.

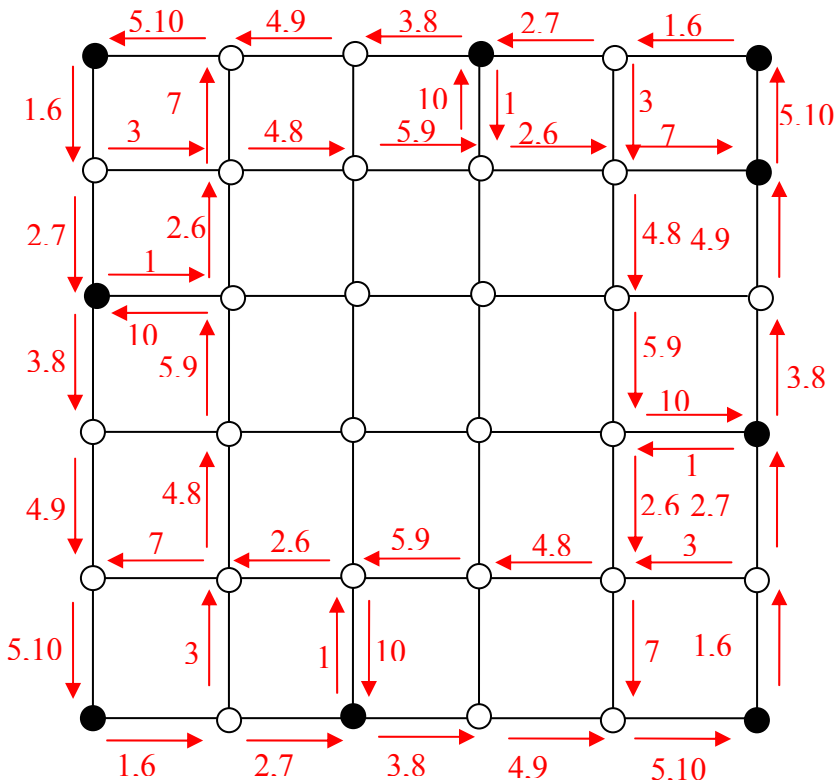


Figure 3.6.3 Communication pattern in $G_{6,6}$ [KCV1992].

The algorithms below are based on the algorithms in [KCV1992] but take into account that grids are dynamically orientable and the system should be in the neutral state when the algorithm is done. Figures 3.6.4 and 3.6.5 show the communication patterns for 5×5 and 6×6 DOG subgrids.

Numbers indicated in green are the times when messages are gathered to the primary points and then cumulative message sent back from primary nodes, times in red are the times when the main communication pattern from [KCV1992] occurs with corresponding delays when necessary and times in blue indicate when empty messages are passed between nodes in order to properly orient them.

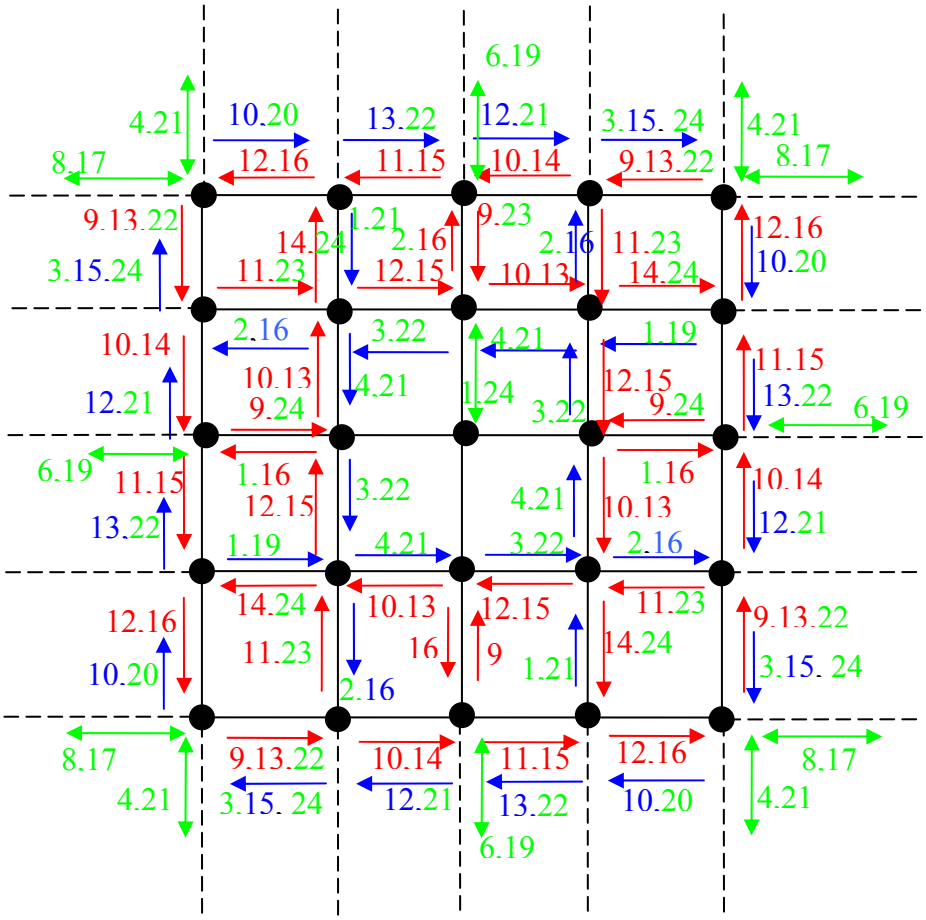


Figure 3.6.4 Communication pattern of $G_{5,5}$ for DOG grids

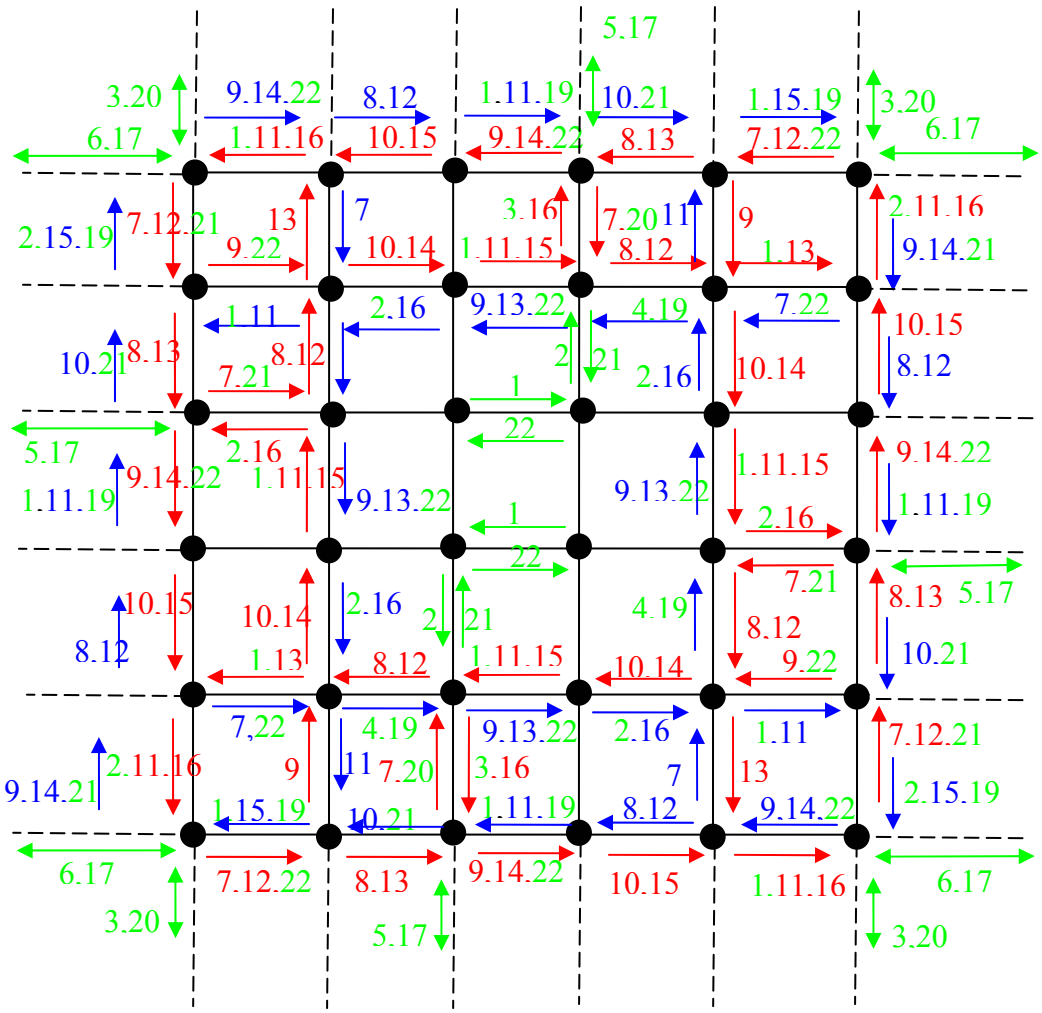


Figure 3.6.5 Communication pattern in DOG $G_{6,6}$ for DOG grids

Algorithm **H1_GOSSIP_ DOG_** G_{nxm}

Identify subgrid $G_{5,5}$ *or* $G_{6,6}$ *that is centered in* $G_{n,m}$.

Collect messages to the corresponding primary points in time $(\frac{n+m}{2}-5)$ *when n and m are odd and in time* $(\frac{n+m}{2}-6)$ *when n and m are even.*

Apply communication pattern shown on figure 3.6.4 or 3.6.5.

Distribute cumulative message back to other vertices.

Theorem 3.5. Algorithm **H1_GOSSIP_ DOG_** G_{nxm} requires $n + m - 2$ time units to complete gossiping in DOG grids of size 12×12 and larger.

Proof: If k is a distance from the corner of the grid $G_{n,m}$ to the corner of the subgrid ($G_{5,5}$ or $G_{6,6}$), it takes k time units to send messages from vertices to primary points [KCV1992]. Figure 3.6.6 shows the distance from corner vertex of grid $G_{n,m}$ to the corner vertex of $G_{5,5}$ grid.

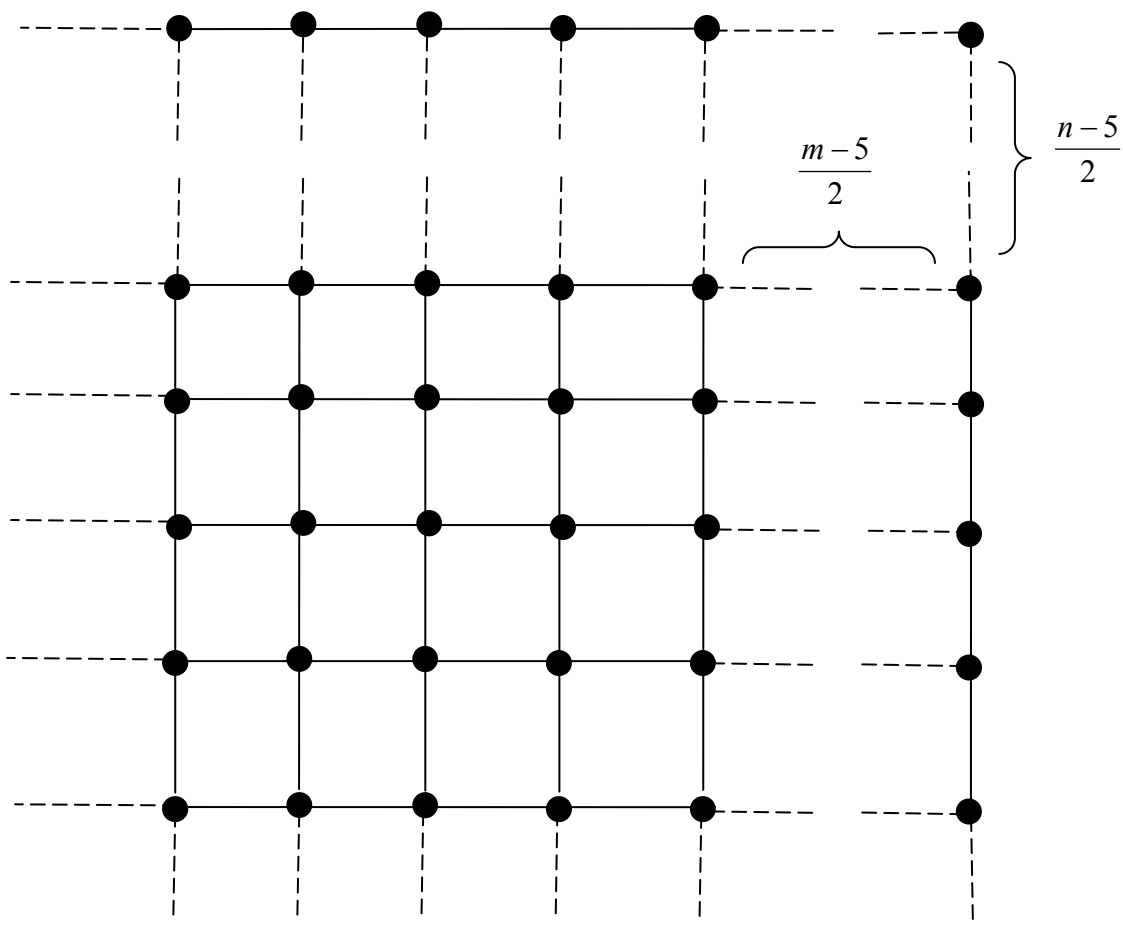


Figure 3.6.6 Distance from corner of $G_{n,m}$ to corner of $G_{5,5}$

The distance from corner vertex of grid $G_{n,m}$ to the corner vertex of $G_{5,5}$ grid is $(\frac{n+m}{2}-5)$ and from corner vertex of grid $G_{n,m}$ to the corner vertex of $G_{6,6}$ grid is $(\frac{n+m}{2}-6)$. The communication patterns shown in Figure 3.6.4 and Figure 3.6.5 require 8 and 10 time units correspondingly. In order to be able to pass messages from secondary vertices inside the subgrids $G_{5,5}$ and $G_{6,6}$ to the primary points and not to interfere with passing messages from outside the

subgrids to the primary points in times $(\frac{n+m}{2}-5)$ and $(\frac{n+m}{2}-6)$ at least 8 and 6 time units required to do that. First, all messages are gathered at primary points, then communication patterns of Figure 3.6.4 and Figure 3.6.5 applied and they ensure that, after gossiping among primary points is finished, all primary points know cumulative information of all messages in the system. At this time all primary points are experts. Primary points distribute the cumulative message back to the other vertices in the same way they received messages in time $(\frac{n+m}{2}-5)$ for odd n and $(\frac{n+m}{2}-6)$ for even n. The algorithm takes optimal time to do gossiping in the DOG_grids when number of vertices is at least 12 in both directions, i.e. total time to gossip in DOG_grids is $n + m - 2$. □

For DOG grids 12×12 and larger algorithm **H1_GOSSIP_ DOG_** G_{nxm} requires D time units, for DOG grids of size 10×10 (Figure 3.6.7) and 11×11(Figure 3.6.8) it needs D + 1 time units, for 9×9 DOG grids (Figure 3.6.6) it needs D + 2 time units. Hence, gossiping in the DOG grid graphs of size 12×12 and larger can be done in optimal time which equals to the diameter of the graph.

3.7 Gossiping Summary

Table 3.1 summarizes existing bounds for classical graphs and achieved bounds for dynamically orientable graphs in case of gossiping. For DOG paths, DOG complete binary trees, DOG complete trees, DOG stars the same gossiping time is achieved as for classical paths, complete binary trees, complete trees and stars. For DOG cycles the same gossiping time is achieved as for classical cycle graphs for $n \geq 8$. For DOG grids $G_{m,n}$ the same gossiping time is achieved as for classical cycle graphs for $m, n \geq 12$.

Graph	Num. of vertices	Max. degree	Diameter D	Lower bound Classical graph	Upper bound Classical graph	Trivial upper bound DOG graph	Upper bound DOG graph
P_n	n even n odd	2	n - 1	n n + 1	n n + 1	2n 2(n+1)	n n + 1
C_n	n even n odd	2	$\left\lfloor \frac{n}{2} \right\rfloor$	$\frac{n}{2}+\left\lceil\sqrt{2n}\right\rceil-1$ $\left\lceil\frac{n}{2}\right\rceil+\left\lceil\sqrt{2n}-\frac{1}{2}\right\rceil-1$	$\frac{n}{2}+\left\lceil\sqrt{2n}\right\rceil-1$ $\left\lceil\frac{n}{2}\right\rceil+\left\lceil 2\sqrt{\left\lceil\frac{n}{2}\right\rceil}\right\rceil-1$	$n+2\left\lceil\sqrt{2n}\right\rceil-2$ $2\left\lceil\frac{n}{2}\right\rceil+2\left\lceil 2\sqrt{\left\lceil\frac{n}{2}\right\rceil}\right\rceil-2$	$\frac{n}{2}+\left\lceil\sqrt{2n}\right\rceil-1$ $\left\lceil\frac{n}{2}\right\rceil+\left\lceil 2\sqrt{\left\lceil\frac{n}{2}\right\rceil}\right\rceil-1$
B_n	n	3	varies		4h (when complete)	8h (when complete)	4h (when complete)
T_k^m	n=$\frac{k^{m+1}-1}{k-1}$	k + 1	2m	Dk	Dk	2Dk	Dk
S_n	n	n-1	2		2(n-1)	4(n-1)	2(n-1)
$G_{m,n}$	mxn	4	m + n - 2	D	D for 9x9 & larger	2D for 9x9 & larger	D for 12x12 & larger

Table 3.1 Gossiping in DOG graphs

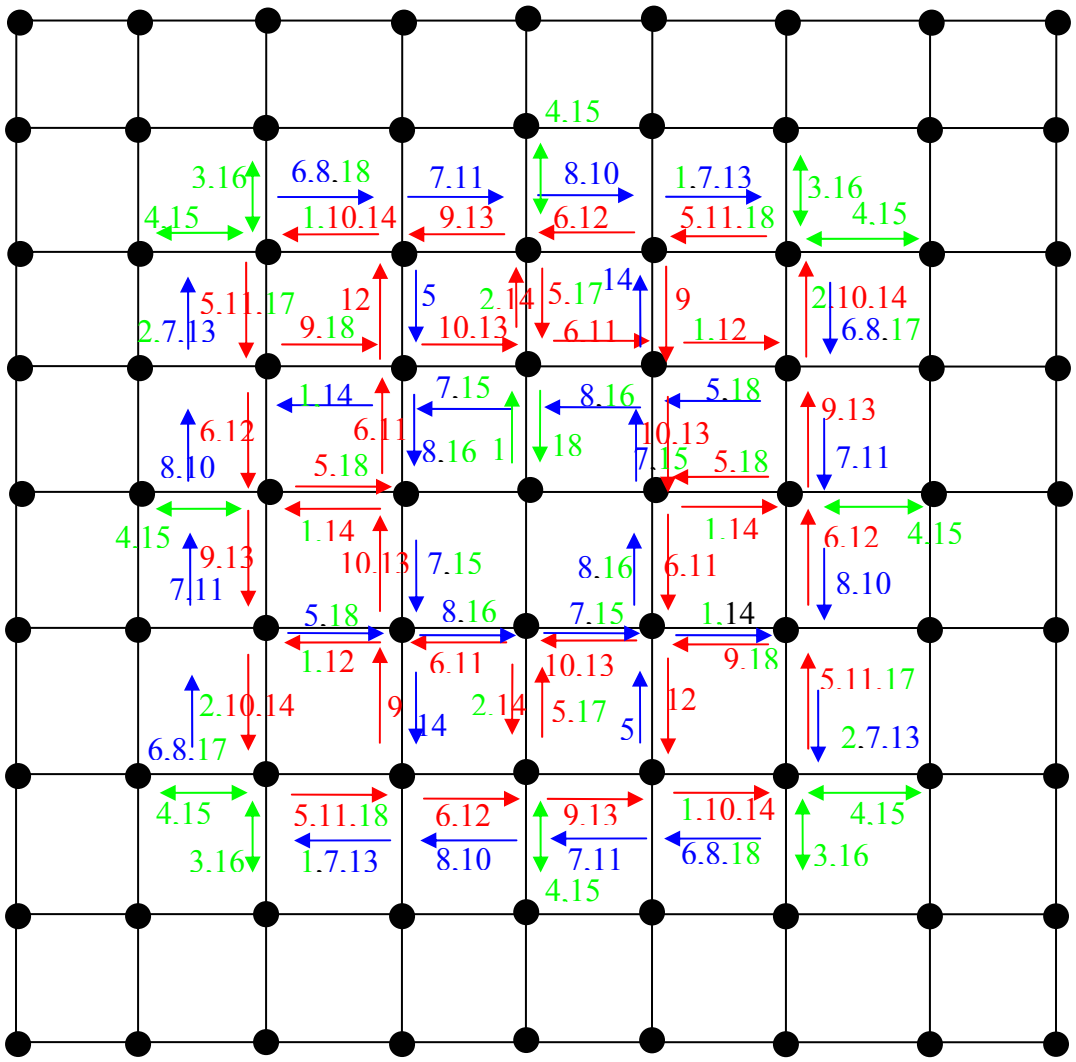


Figure 3.6.7 Communication pattern in DOG $G_{5,5}$ for DOG $G_{9,9}$

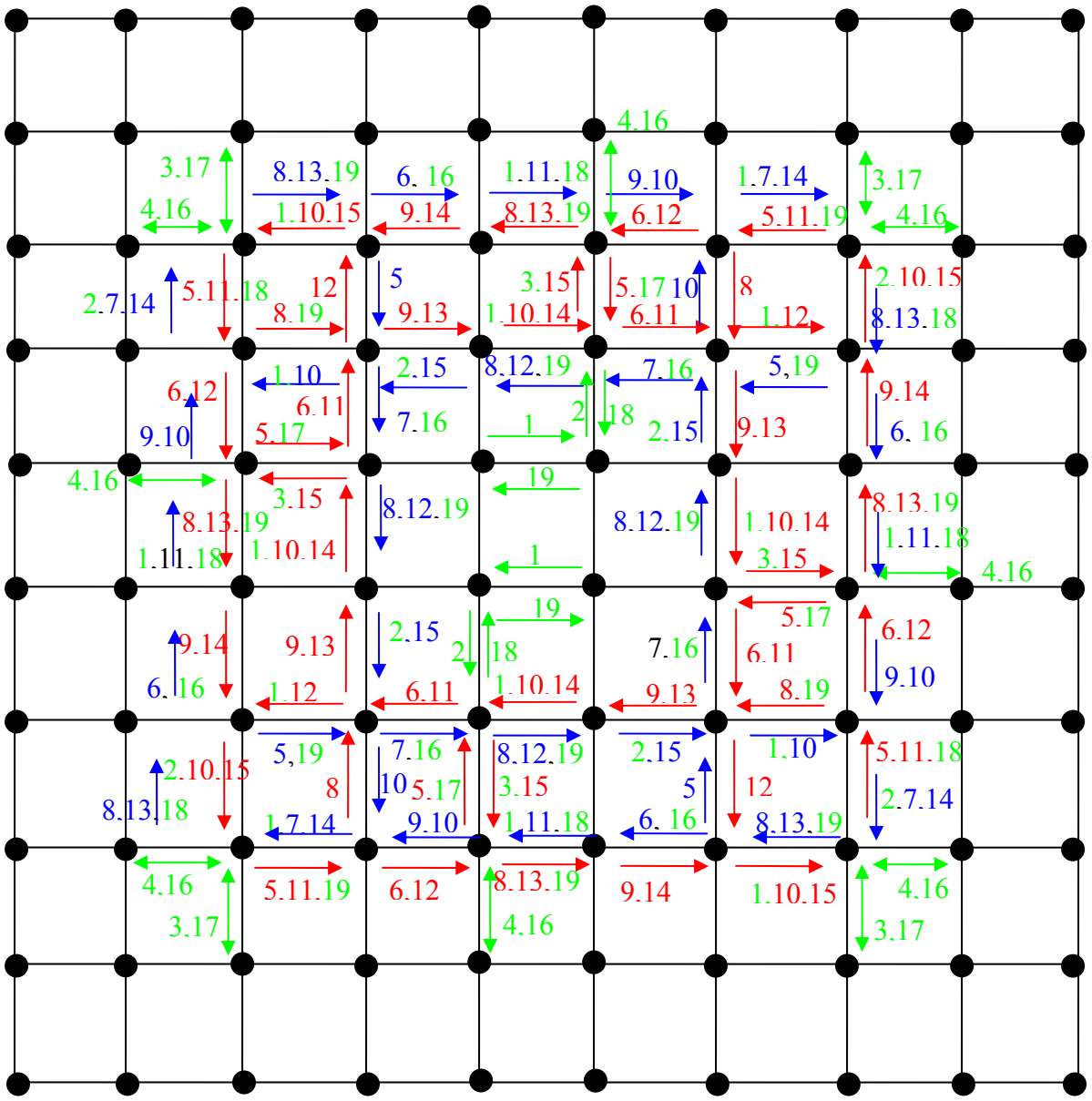


Figure 3.6.8 Communication pattern in DOG $G_{6,6}$ for DOG $G_{10,10}$

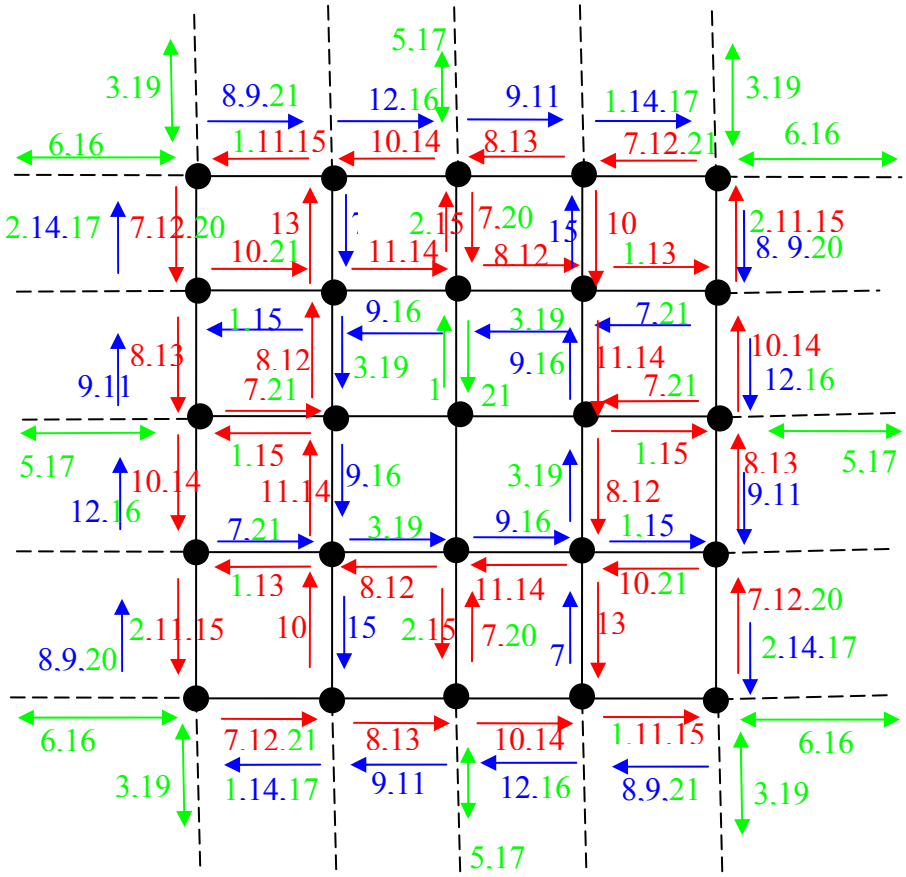


Figure 3.6.9 Communication pattern in DOG $G_{5,5}$ for DOG $G_{11,11}$

Chapter 4

4 Multiple message broadcasting

Given a graph G with V vertices and E edges, *multiple message broadcasting* is the information dissemination problem where one node $u \in V$ has several messages that it needs to broadcast to all other nodes, and due to some reasons, for example, the size of the messages it cannot send these messages as one package.

For the classical graphs the general lower and upper bounds for this problem are defined by Farley in [F1980]. For multiple message broadcasting the lower bound is $2(m-1)+D$ where D is the diameter of a graph and the upper bound is $d_{\max}(m-1)+(n-1)$, where $d_{\max}$ is a maximum degree of a vertex in a graph.

In the following subsections the multiple messages algorithms are presented for different kinds of network topologies that are also satisfy the conditions of dynamically orientable graphs. Networks considered are paths, cycles, binary trees, complete trees, stars and 2-dimensional grids.

Models considered here are half-duplex processor-bound models (H1 models). In a network with n vertices, a vertex s, called a source node, knows messages $m_1 \ldots m_k$ and after multiple message broadcasting is finished, each vertex v_i knows all messages $m_1 \ldots m_k$. Each vertex receives only one message at a time from other vertices and passes known messages along to other vertices also one at a time.

4.1 Path P_n

In the path of length n (Figure 4.1.1), nodes are numbered from 1 to *n* and have edges that connect them with two adjacent nodes, except node 1 and *n*. The diameter $D(P_n) = n - 1$, $d(1) = d(n) = 1$, and each vertex *v* has degree $d(v) = 2$ for $1 < v < n$.

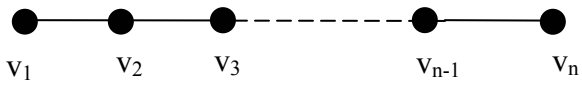


Figure 4.1.1 Path P_n.

According to Farley's theorem the time for multiple message broadcasting in a classical path graph P_n is $2(m-1) + (n-1)$. The algorithm given below broadcasts m messages in dynamically orientable path DOG_P_n.

Algorithm **H1_MMBROAD_ DOG_P_n**

for m_i , i € {1…k}

Source node v_1 sends message m_i to v_2 at time t = 1+ 4(i -1)

and resets its edge with v_2 at time t = 3+ 4(i -1)

Node v_2 sends message m_i to v_3 at time t = 2+ 4(i -1)

and resets its edge with v_3 at time t = 4+ 4(i -1)

For $2 < j < n$, node v_j passes message m_i to v_{j+1} at time t = j + 4(i -1)

and resets its edge with v_{j+1} at time t = j+ 4(i -1)- 2

Figure 4.1.2 illustrates example of algorithm **H1_MMBROAD_ DOG_P_n** for P_7 .

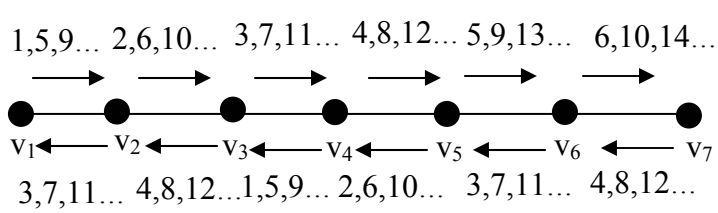


Figure 4.1.2 Multiple-message broadcasting in P_7.

Theorem 4.1. Algorithm **H1_MMBROAD_DOG_P$_n$** requires *D+4(m–1)* time units to complete broadcasting of *m* messages in DOG paths.

Proof: The algorithm considers the end node of a path as a source node. Each consecutive message gets delayed by four time units. First message goes through the path without delays in time equaled to the diameter of the path D. Each consecutive message that is delayed by four time units also goes through the path without delays in time D. The total time to broadcast m messages from the source is $b\_m_{H1}(v) = D + 4(m-1)$ which is 2(m-1) time units more than broadcasting m messages from an end node in a classical path graph, but D time units less than twice the broadcasting in classical path $b_{m,H1}(DOG\_P_n) = D + 4(m-1) < 2b_{m,H1}(P_n) =$

$2D + 4(m-1)$ □

Another possibility is when the source node is not one of the end node but rather one of inner nodes. For classical path graphs, when *n* is odd, time is $(\frac{D}{2}+1) + 2(m-1)$ in best case (source is in the middle of the path) and $D + 2(m-1)$ in the worst case (source is the end node of the path). When *n* is even time is $\left\lceil \frac{D}{2} \right\rceil + 2(m-1)$ in best case (source is in the middle of the path) and $D + 2(m-1)$ in the worst case (source is the end node of the path).

For DOG path graphs, when *n* is odd, multiple message broadcasting time is $b\_m_{H1}(v) = (\frac{D}{2}+1) + 4(m-1)$ in best case (source is in the middle of the path) and $b\_m_{H1}(v) = D + 4(m-1)$ in the worst case (source is the end node of the path). When *n* is even time is $\left\lceil \frac{D}{2} \right\rceil + 4(m-1)$ in best case (source is in the middle of the path) and $D + 4(m-1)$ in the worst case (source is the end node of the path).

As a variation of the problem, we can consider both end nodes of a path as source nodes. Figure 4.1.3 and Figure 4.1.4 illustrate examples of modified algorithm when both end nodes of a path are the source nodes. The main modification of the **H1_MMBROAD_ DOG_P$_n$** algorithm is two sources send different messages. For example, vertex v_1 sends only odd numbered messages and vertex v_n sends only even numbered messages Figure 4.1.5 shows the message path in the algorithm.

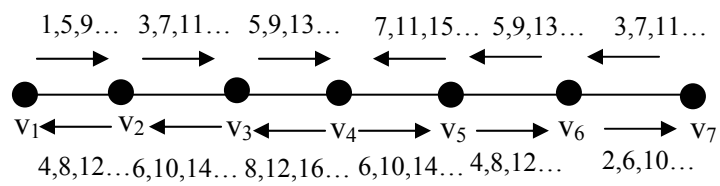


Figure 4.1.3 Multiple-message broadcasting in P_7 (two sources).

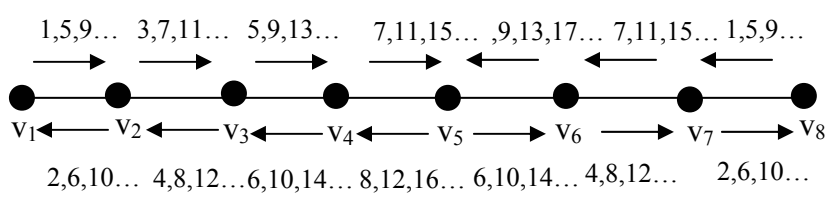


Figure 4.1.4 Multiple-message broadcasting in P_8 (two sources).

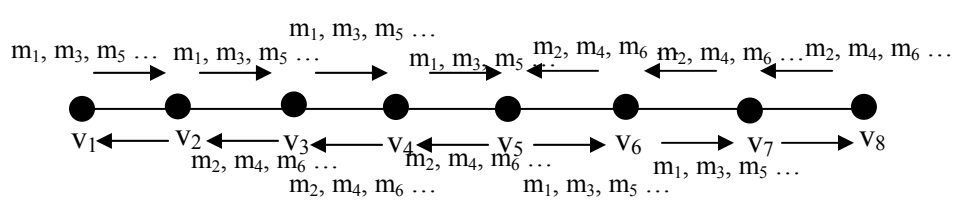


Figure 4.1.5 Message path in P_8 (two sources).

For large m, this approach works better than regular approach of sending all messages from one source node. When both end nodes of a path are the source nodes then broadcasting time is $b_{m,H1}$ (DOG Pn) = $2n + 2m - 8$ for odd n and $b_{m,H1}$ (DOG Pn) = $2n + 2m - 6$ for even n. The same idea can be applied to the multiple message broadcasting in the DOG cycles.

4.2 Cycles C_n

In the cycle graph C_n (Figure 4.2.1), nodes are numbered from 1 to n and have edges that connect them with two adjacent nodes and connect node 1 node n. The diameter $D(C_n) = \left\lfloor \frac{n}{2} \right\rfloor$, and each vertex v has degree $d(v) = 2$.

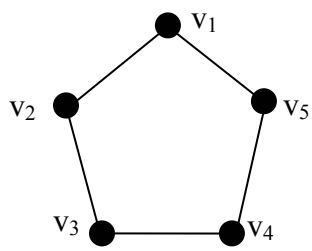


Figure 4.2.1 Cycle C_5.

According to Farley's theorem the lower bound for time for multiple message broadcasting in a classical cycle graph C_n is $2(m-1) + \left\lfloor \frac{n}{2} \right\rfloor$ and the upper bound is $2(m-1) + (n-1)$. The algorithm below broadcasts m messages in dynamically orientable cycles DOG_C_n. The main idea of the algorithm is that the source transmits odd numbered messages to its left neighbors and even numbered messages to its right neighbors. In even cycles, each message except the first one is delayed by four time units at the source and by two time units at each processor. In odd cycles, each message except first one is delayed by six time units at the source and by three time units and by three time units at each processor.

Algorithm **H1_MMBROAD_ DOG_C$_n$** (*n* is even)

For m_i , i € {1…k},

Source node v_1 :

sends odd numbered message m_i to the left (vertex v_2) at time $t = 1 + 2(i -1)$

resets edge with left neighbor at time $t = 4 + 2(i -1)$

sends even numbered message m_i to the right (vertex v_n) at time $t = 2 + 2(i -2)$

resets edge with right neighbor at time $t = 3 + 2(i -2)$

Intermediate node:

Receives message m_i , holds it for 2 time units and sends it to the next processor

Each vertex v_j receives odd numbered message m_i from vertex v_{j-1} at time $t = 2j + 4i – 7$ and (except v_n) send it to vertex v_{j+1} at time $t = 2j + 4i -5$

Each vertex v_j receives even numbered message m_i from vertex v_{j+1} at time $t = 2n - 2j + 4i – 2$ and (except v_2) send it to vertex v_{j-1} at time $t = 2n – 2j + 4i$

Each vertex v_i either reset its edge with empty message or messages moving in opposite directions neutralize the edge. Vertices v_2 … $v_{n/2}$ and $vn_{/2+2}$ … v_{n-1} and edges connecting them need to use empty messages at least once. Vertex $v_{n/2+1}$ does not need empty messages to reset its edges.

Figure 4.2.2 illustrates example of algorithm **H1_MMBROAD_ DOG_C$_n$** for C_8 . Light green lines on the Figure 4.2.2 indicates times when simple resetting of an edge occurs, i.e. no message is passed.

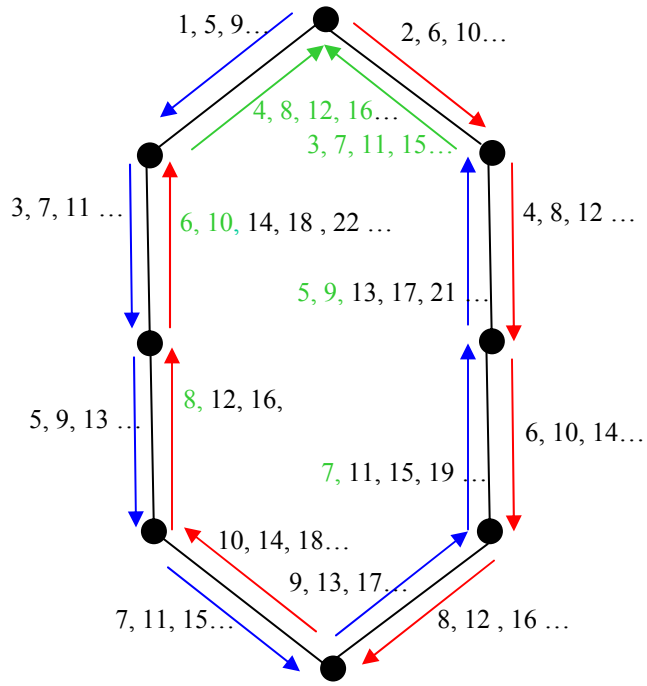


Figure 4.2.2 Multiple-message broadcasting in C_8.

Theorem 4.2.1. Algorithm **H1_MMBROAD_DOG_C$_n$** (n is even) requires $(2n+2m-6)$ time units to complete broadcasting of m messages in even DOG cycles.

Proof: The algorithm ensures that each vertex receives odd and even numbered messages and sends the message only after it receives it. Edges connecting vertices v_1 … $v_{n/2+1}$ send messages in opposite directions with three time units delay between them. Edges connecting vertices $v_{n/2+1}$ … v_1 send messages in opposite directions with one time unit delay between them. All odd numbered messages traveled from v_1 to v_n passing all vertices between them. All even numbered messages traveled from v_1 through v_n to v_2 passing all vertices between them on that path. Once a vertex received a message it holds it for two time units and then passes it to its neighbor. Since a vertex has distinct time slots for odd and even numbered messages, it can use these time slots to reset edges with empty messages in case there is no message passing at that time but it needs resetting. For example, vertex v_3 receives m_1 at time 3, next odd numbered message it will

receive at time 7. Time slot for sending even message is 6, but there is no even numbered message yet, so it can use this time slot to send an empty message thus resetting its edge for message m_3. Figure 4.2.3 shows times slots vertices use for transmitting messages for $k \geq 1$. Since each four time units two new messages are introduced into the network, the coefficient in front of m is 2. The total time to broadcast m messages from the source is $b\_m_{H1}(v) = 2n + 2m - 6$. Farley's lower bound for classical cycle graphs is ($\left\lfloor \frac{n}{2} \right\rfloor$ + 2(m-1)). The trivial lower bound for DOG cycles is (n + 4m – 4). Proposed algorithm reduces the coefficient in front of m by two and performs better then trivial lower bound for $n < 2m + 2$. □

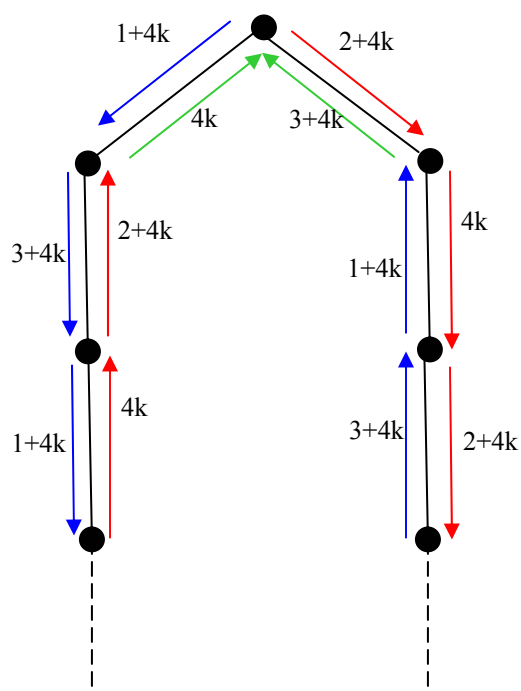


Figure 4.2.3 Time slots for even cycle C_n.

Same approach can be used with odd DOG cycles. The only thing that is different in this case is that each processor holds the message for three time units instead of two time units.

Algorithm **H1_MMBROAD_ DOG_C$_n$** (n is odd)

For m_i , i € {1…k},

Source node:

sends odd numbered message m_i to the left at time t = 1 + 3(i -1)

reset edge with left neighbor at time t = 3 + 3(i -1)

sends even numbered message m_i to the right at time t = 2 + 3(i -2)

reset edge with right neighbor at time t = 6 + 3(i -2)

Intermediate node:

Receive message m_i, hold it for three time units and send it to the next processor

Each vertex v_j receives odd numbered message m_i from vertex v_{j-1} at time t = 3j+ 6i - 11 and (except v_n) send it to vertex v_{j+1} at time t = 3j + 6i - 8

Each vertex v_j receives even numbered message m_i from vertex v_{j+1} at time t = 3n - 3j + 6i – 4 and (except v_2) send it to vertex v_{j-1} at time t = 3n – 3j + 6i - 1

Each vertex v_i either reset it edge with empty message or messages moving in opposite directions neutralize the edge. Vertices v_2 … $v_{(n+1)/2}$ and $v_{(n+1)/2+2}$ … v_{n-1} and edges connecting them need to use empty messages at least once. Vertex $v_{(n+1)/2+1}$ does not need empty messages to reset its edges.

Figure 4.2.4 illustrates example of algorithm **H1_MMBROAD_ DOG_C$_n$** for C_7 . Light green lines on the Figure 4.2.4 indicates times when simple resetting of an edge occurs, i.e. no message is passed.

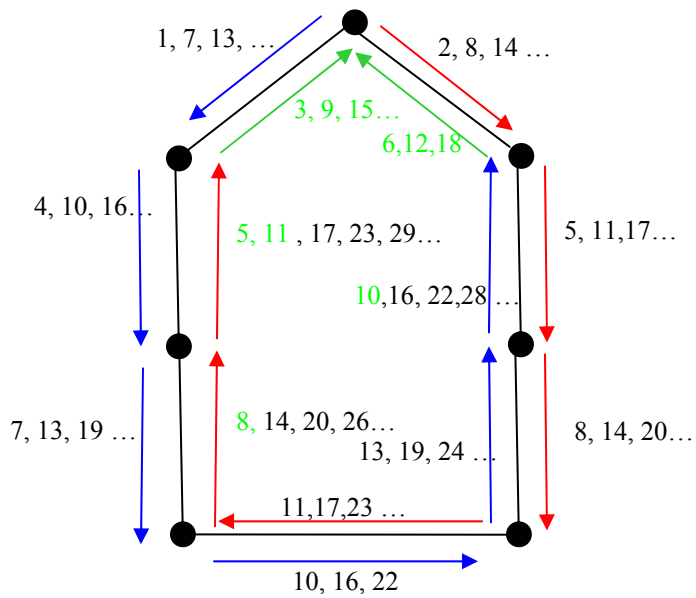


Figure 4.2.4 Multiple-message broadcasting in C_7.

Theorem 4.2.2. Algorithm **H1_MMBROAD_DOG_C_n** (n is odd) requires *(3n+3m–7)* time units to complete broadcasting of *m* messages in odd DOG cycles.

Proof is similar to the even case. Figure 4.2.5 shows times slots vertices use for transmitting messages for $k \geq 1$. Figures 4.2.6 and 4.2.7 show why messages cannot be sent sooner. Vertex v_1 needs at least four time units to send messages to v_2 and to v_n and reset its edges. Next time slots to try to sent message is five and six. Figure 4.2.6 shows why it is not possible to use time slot of five and Figure 4.2.7 shows why it is not possible to use time slot of six. Time slots shown in red indicate collision.

Since each six time units two new messages are introduced into the network, the coefficient in front of *m* is 3. The total time to broadcast *m* messages from the source is $b\_m_{H1}(v) = 3n + 3m - 7$. Farley's lower bound for classical cycle graphs is ($\left\lfloor \frac{n}{2} \right\rfloor$+ 2(m-1)). The trivial lower bound for DOG cycles is (n + 4m – 4). Proposed algorithm reduces the coefficient

in front of m by one and performs better then trivial lower bound for $n < \frac{m}{2} + 1.5$. For odd DOG cycles whenever possible it is better to add one dummy processor for just resending messages, because the time bound for even cycles are better then for odd cycles even when even cycle is more then odd cycle by one vertex. In case if it is not feasible to do, proposed algorithm can be used since it achieves better bound then existing trivial bound for $n < \frac{m}{2} + 1.5$. □

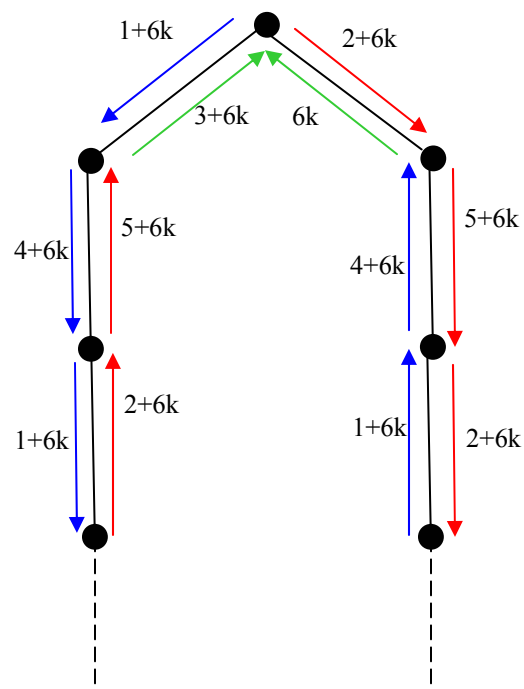


Figure 4.2.5 Time slots for odd cycle C_n with 6-unit delay at the source.

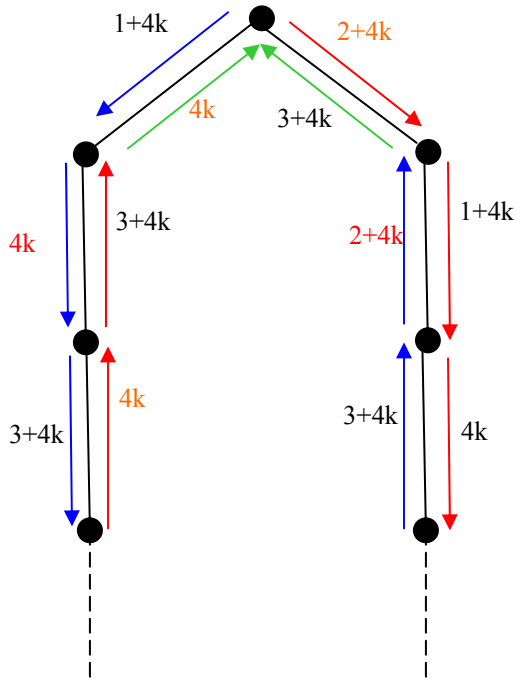


Figure 4.2.6 Time slots for odd cycle C_n with 4-unit delay at the source.

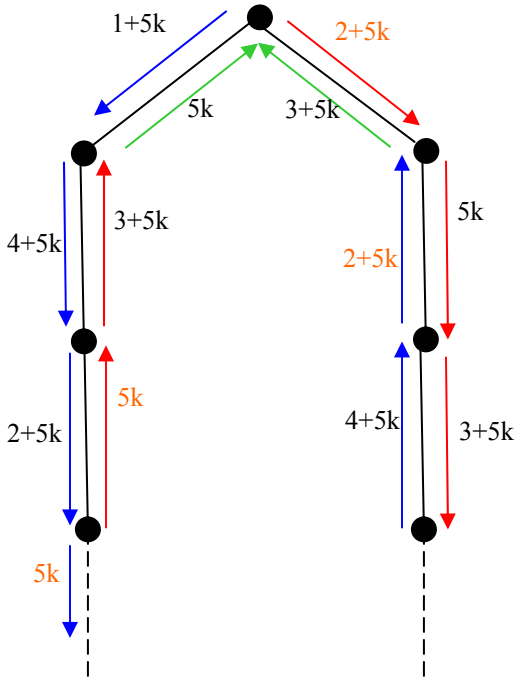


Figure 4.2.7 Time slots for odd cycle C_n with 5-unit delay at the source.

4.3 Star graph S_n

The star graph S_n is a tree with n nodes where (n-1) nodes are connected to one node. The diameter D(S_n) = 2. All vertices except one have degree of 1 and the center node has degree of (n-1) (Figure 4.3.1).

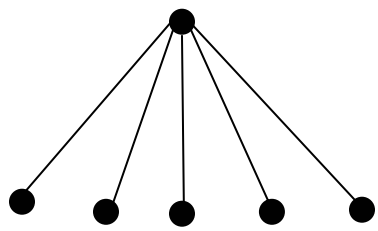


Figure 4.3.1 Star graph S_6.

Time to broadcast multiple messages in the classical star graph S_n is $m(n-1)$ which is equal to Farley's theorem upper bound. Due to the fact, that the center node of a star graph has degree of (n-1), it is a bottleneck in the broadcasting process. The best possible time that can be achieved in the DOG star graphs is twice the multiple message broadcasting time in classical star graphs.

Algorithm **H1_ MMBROAD_ DOG_** S_n

For m_i , $i \in \{1...k\}$,

Center node v_0 *as a Source:*

Center node sends message m_i *to its child* v_j $(1 \leq j \leq (n-1))$ *at time* $t = j + 2(n-1)(i-1)$. *Center node needs two time units on each communication with its child (one time unit to send message and one time unit to reset the edge used to pass that message).*

New message is sent every other 2(n-1) time unit.

Child node v_j *as a Source:*

Child node v_j *sends first message* m_1 *to the center node* v_0 *at time* $t = 1$ *and new message* m_i *every 2(n-1)(i-1) time unit. The center node in turn distributes this message to the rest of the nodes sending a message every two time units (one time*

unit to send message and one time unit to reset the edge used to pass that message).

Theorem 4.3. Algorithm **H1_MMBROAD_DOG**_S_n requires $2m(n-1)$ time units to complete broadcasting of m messages in DOG stars.

Proof: Since center vertex is connected to all other vertices and the only vertex that can connect other vertices to each other, all communications are done through center vertex. If a center node is a source node, it can only transfer messages one by one to each one of its children. It takes (n-1) time units to pass one message. Source vertex also should reset the edges it used to pass this message. It takes another (n-1) time units. Therefore, to finish broadcasting of one message vertex needs 2(n-1) time units. Each additional message also requires (n-1) time units to be broadcasted and (n-1) time units to bring edges into neutral state. Total time to broadcast m messages in DOG_ S_n is $2m(n-1)$. □

Figure 4.3.2 illustrates example of algorithm **H1_ MMBROAD_ DOG_** S_n for S_6. Blue lines indicate message sending and red lines indicate the resetting of an edge.

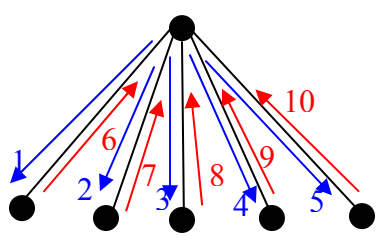


Figure 4.3.2 Multiple message brodcasting in DOG_ S_6.

4.4 Complete tree T_k^p

A tree graph is a graph whose set of edges connect a set of nodes and contains no cycles. A rooted tree is a tree with special node called root. In a rooted tree, each of the nodes that is one

edge further away from a given node is called a child. Each edge links a parent node to one of its children. Root node has no parent. Every other node has exactly one parent. Nodes at the bottom of a tree are called leaves. The complete *k*-ary tree T_k^p of height *p* is a tree whose nodes are all *k*-ary strings of length at most *p* and whose edges connect each string α of length i $(0 \le i \le p)$ with the string $\alpha a, a \in \{0,\ldots k-1\}$ of length *i* +1 (Figure 4.4.1). The root node is an empty string and a node α is at level $i, i \ge 0$ if α is a string of length i. T_k^p has $\frac{k^{p+1}-1}{k-1}$ nodes. The diameter D(T_k^p) = 2*p*. Maximum degree of a vertex is *k* + 1. Minimum degree of the root d(*root*) is *k*.

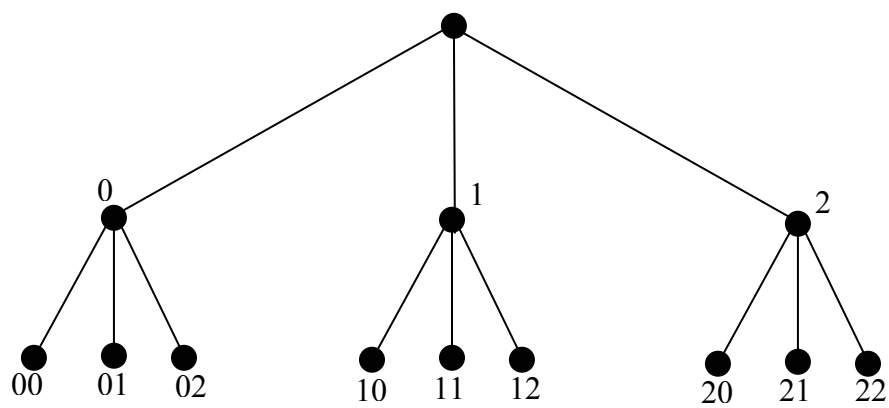


Figure 4.4.1 Complete tree T_3^2.

The known multiple message broadcasting scheme for arbitrary classical tree has an upper bound of *O(nm)* [DLP1999]. This bound holds for complete trees as well. The algorithm proposed here requires *(2k+p)* + 2(*k*+1)(*m*-1) for p = 2 and *pk* + 2(*k*+1)(*m*-1) for p ≥ 3 time units to complete broadcasting of *m* messages in DOG_T_k^p.

Algorithm **H1_ MMBROAD_ DOG_** T_k^p

Root node as a Source.

Root node starts broadcasting by selecting one child after each other, for example, from left to right. It takes k time units. It also resets those edges in k time units.

Root node S sends message m_i to vertex v_j,($j \in \{0,\ldots k-1\}$), at times $t = j + 1 + 2(k+1)(m-1)$.

Root node S resets its edge with vertex v_j,($j \in \{0,\ldots k-1\}$), at times $t = j + k + 2 + 2(k+1)(m-1)$.

If a child vertex v_j receives message m_i at time t, it transfers the message down to its children at times $t+1,\ldots, t + k$. The edge resets at time $t_i + (k+1)$ or $t_i - (k+1)$ when $t_i + (k+1) \geq st_{i+1}$, where t_i is the time when the edge transferred the message m_i and st_{i+1} is the starting time of message i+1 from root vertex.

In case of arbitrary rooted trees, the following addition is made to Algorithm **H1_ MMBROAD_ DOG_** T_k^p **.**

Starting from leaves for each vertex v_i calculate the maximum distance from that vertex to one of each children. Root node starts broadcasting by selecting first the child with the highest maximum distance value, then next highest etc. Each vertex repeats the same procedure.

Theorem 4.4. Algorithm **H1_MMBROAD_DOG_**T_k^p requires $(2k+p) + 2(k+1)(m-1)$ for p = 2 and $pk + 2(k+1)(m-1)$ for $p \geq 3$ time units to complete broadcasting of m messages in DOG_T_k^p .

Proof: Broadcasting is done from the root to the leaves and all edges are used in order to transport messages to all vertices. Broadcasting first message requires pk time units and each subsequent message requires additional $2(k+1)$ time units. Total time to broadcast m messages in DOG_T_k^p is $(2k+p) + 2(k+1)(m-1)$ for p = 2 and $pk + 2(k+1)(m-1)$ for $p \geq 3$. □

Figure 4.4.2 illustrates example of algorithm **H1_ MMBROAD_ DOG_** T_k^p for T_3^3. Red lines indicate broadcasting messages and blue lines indicate edge resetting.

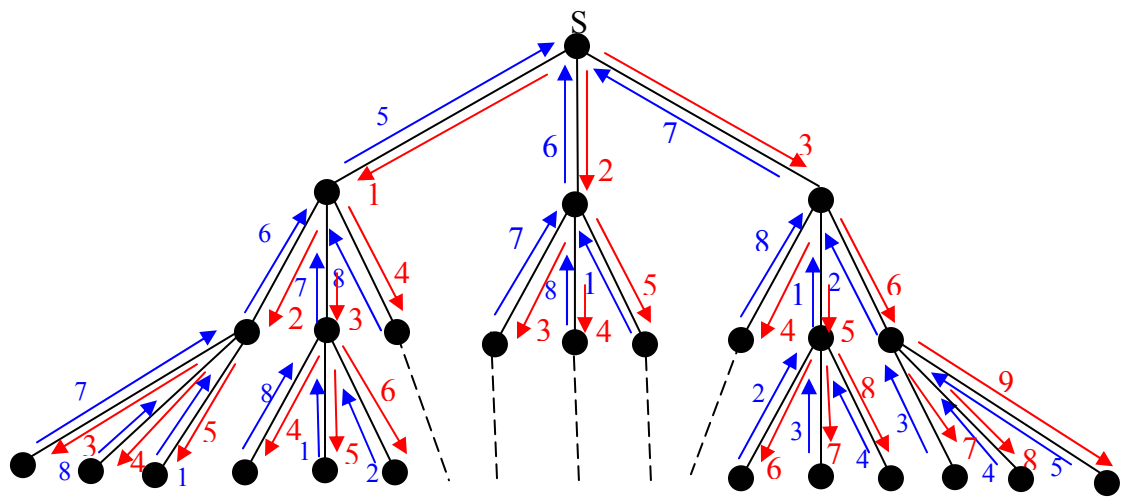


Figure 4.4.2 Multiple message broadcasting in DOG_T_3^3.

4.5 Binary tree B_n

The binary tree is a tree in which each parent has at most two children (Figure 4.5.1). The number of nodes in B_n is n and the number of edges is (n-1).

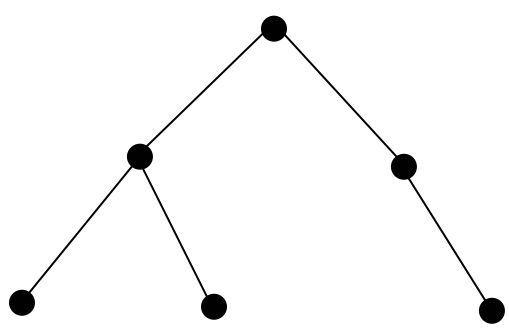


Figure 4.5.1. Binary tree B_6.

The diameter D(B_n) depends on the configuration of the tree. It reaches its maximum value of (n-1), for example, in case when the tree is a path. Each vertex v in a binary tree has maximum

degree d(v) of 3. The root node has maximum degree d(*root*) of 2. The height h of the B_n is the maximum distance from the root to the children or the number of levels in the tree. It is known that broadcasting and gossiping time in a binary tree depend on the configuration of the tree. Broadcasting in a complete binary tree T_2^h takes 2h time units and gossiping requires twice the broadcasting time, i.e. 4h. The upper bounds for broadcasting in a binary tree with the height h are $b_{H1}(B_n) \leq 2h$.

A complete binary tree is a special case of complete tree T_k^p where k is 2 and p is the height h. The algorithm from Section 4.4 can be applied to complete binary trees resulting in a broadcast time of *(4+h)* + 6(m-1) for h = 2 and 2h + 6(m-1) for h ≥ 3.

The upper bound for multiple message broadcasting time in the classical binary tree graph B_n is O(nm). Our upper bound for broadcasting time is 2h + *6(m-1)* that is reached when binary tree is complete.

4.6 2-dimensional Grid

The 2-dimensional $m \times n$ grid is a graph with m x n vertices that are connected in a crisscross manner (Figure 4.8). Number of nodes is mn.

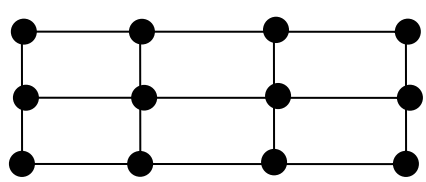


Figure 4.6.1 Grid graph $G_{3,4}$.

Wojciechowska and Van Scoy in [WV1996] presented an algorithm for broadcasting m messages in an $n \times n$ grid that required $2n + 2M + 2$ time units with corner node as a source and

an algorithm that required $dn + 3M + C$ time units to broadcast *m* messages from the corner of d-dimensional grid. *M* is the number of messages that need to be broadcasted. The obvious upper bound for multiple message broadcasting in a DOG $n \times n$ grid is $4n + 4M + 4$ time units. The goal of this dissertation is to find an algorithm that performs better than the obvious upper bound of $4n + 4M + C$. The designed algorithm can perform multiple message broadcasting in a grid in $2n + 2.5M + C$.

4.6.1 Multiple message broadcasting algorithm for DOG grid

In this section, general principle of the proposed algorithm will be explained. In later chapters, detailed algorithms will be given for $n \times n$ and $m \times n$ DOG grids.

Van Scoy in [VB1994] described the algorithm for broadcasting a small number of messages in a square grid. The algorithm divided messages into two sets of odd and even numbered messages. Odd numbered messages were sent along first row and down the columns as well as down the first column. Even numbered messages were sent down first column and along the rows as well as along the first row. The algorithm worked only for small number of messages. Later, Wojciechowska and Van Scoy in [WV1996] presented an algorithm for broadcasting m messages in an $n \times n$ grid that required $2n + 2M + 2$. In their approach the messages were again divided into two sets of odd and even numbered messages. However, odd numbered messages were sent along first row and down the even columns and the last column. Even numbered messages followed the transposed pattern down the first column. In addition to the regular paths messages also traveled in a “curly” pattern. Odd numbered messages were sent left once they reached odd rows and then up, even numbered messages were sent up once they reached odd columns and then left. This algorithm does not work for DOG grids because of the need to maintain correct orientation of edges.

The new algorithm was designed based on the previous approach that takes into account orientation of the edges.

Approach

All messages M divided into four sets of messages as follows:

Set I : Messages numbered (1 + 4k), $k \geq 0$ and $(1+4k) \leq M$

Set II : Messages numbered (2 + 4k), $k \geq 0$ and $(1+4k) \leq M$

Set III : Messages numbered (3 + 4k), $k \geq 0$ and $(1+4k) \leq M$

Set IV : Messages numbered (4 + 4k), $k \geq 0$ and $(1+4k) \leq M$

Algorithm **H1_ MMBROAD_ DOG_ $G_{m,n}$**

Corner vertex $v_{1,1}$ sends Set I messages along first row and down the last column at time $t(mod10) \equiv 1$. Vertices in the first row that have even j index send Set I messages down the corresponding j^{th} column.

If vertex $v_{i,j}$ (i, j are even) receives Set I message from vertex $v_{i-1,j}$, the vertex sends it only to vertex $v_{i+1,,j}$ with the exceptions of the vertices in the last row. In that case the vertex sends the message to its left neighbor, vertex $v_{i,j-1.}$ If vertex $v_{i,j}$ (i is odd) receives Set I message from vertex $v_{i-1,j}$, the vertex sends it to vertex $v_{i+1,j}$ and to vertex $v_{i,j-1}$. Vertices in the last row only send the message to their left neighbor. If vertex $v_{i,j}$ (i , j are odd) receives Set I message from vertex $v_{i,j+1}$, the vertex sends it to vertex $v_{i-1,j}$.

Corner vertex $v_{1,1}$ sends Set III messages along first row and down the last column at time $t(mod10) \equiv 7$. Vertices in the first row that have odd j index send Set III messages down the corresponding j^{th} column.

If vertex $v_{i,j}$ (i, j are odd) receives Set III message from vertex $v_{i-1,j}$, the vertex sends it only to vertex $v_{i+1,,j}$ with the exceptions of the vertices in the last row. In that case the vertex sends the message to its left neighbor, vertex $v_{i,j-1.}$ If vertex $v_{i,j}$ (i is even) receives Set III message from vertex $v_{i-1,j}$, the vertex sends it to vertex $v_{i+1,j}$ and to vertex $v_{i,j-1}$. If vertex $v_{i,j}$ (i , j are even) receives Set III message from vertex $v_{i,j+1}$, the vertex sends it to vertex $v_{i-1,j}$ with the exceptions of the vertices in the second row. Vertices in the second row do not need to send messages up to the first row since all vertices in the first row know all Set III messages. Vertex $v_{i,2}$ (i is even) also sends Set III message to the left neighbor, vertex $v_{i,1.}$ If vertex $v_{i,1}$ (i is even) receives Set III message from vertex $v_{i,2}$ it sends it to vertex $v_{i-1,1}$ with the exceptions of the vertex $v_{2,1}$.

Corner vertex $v_{1,1}$ sends Set II messages down the first column and along the last row at time t(mod10) ≡3 . Vertices in the first column that have even i index send Set II messages along the corresponding i^{th} row.

If vertex $v_{i,j}$ (i, j are even) receives Set II message from vertex $v_{i,j-1}$, the vertex sends it only to vertex $v_{i,,j+1}$ with the exceptions of the vertices in the last column. In that case the vertex also sends the message to its above neighbor, vertex $v_{i-1,j.}$ If vertex $v_{i,j}$ (j is odd) receives Set II message from vertex $v_{i,j-1}$, the vertex sends it to vertex $v_{i,j+1}$ and to vertex $v_{i-1,j}$. If vertex $v_{i,j}$ (i , j are odd) receives Set II message from vertex $v_{i+1,j}$, the vertex sends it to vertex $v_{i,j-1.}$

Corner vertex $v_{1,1}$ sends Set IV messages down the first column and along the last row at time t(mod10) ≡ 9 . Vertices in the first column that have odd i index send Set IV messages along the corresponding i^{th} column.

If vertex $v_{i,j}$ (i, j are odd) receives Set IV message from vertex $v_{i,j-1}$, the vertex sends it only to vertex $v_{i,,j+1}$ with the exceptions of the vertices in the last column. In that case the vertex also sends the message to its above neighbor, vertex $v_{i-1,j}$. If vertex $v_{i,j}$ (j is even) receives Set IV message from vertex $v_{i,j-1}$, the vertex sends it to vertex $v_{i,j+1}$ and to vertex $v_{i-1,j}$. If vertex $v_{i,j}$ (i , j are even) receives Set IV message from vertex $v_{i+1,j}$, the vertex sends it to vertex $v_{i,j-1}$ with the exceptions of the vertices in the second column and the last row. Vertices in the second column do not need to send messages back to the first column since all vertices in the first column know all Set IV messages. Vertices in the last row only need to send messages along the row. Vertex $v_{2,j}$ (j is even) in the second row also sends Set IV message up to the first row vertex $v_{1,j}$ (j is even). If vertex $v_{1,j}$ (j is even) receives Set IV message from vertex $v_{2,j}$ it sends it to vertex $v_{1,j-1}$ with the exceptions of the vertex $v_{1,2}$.

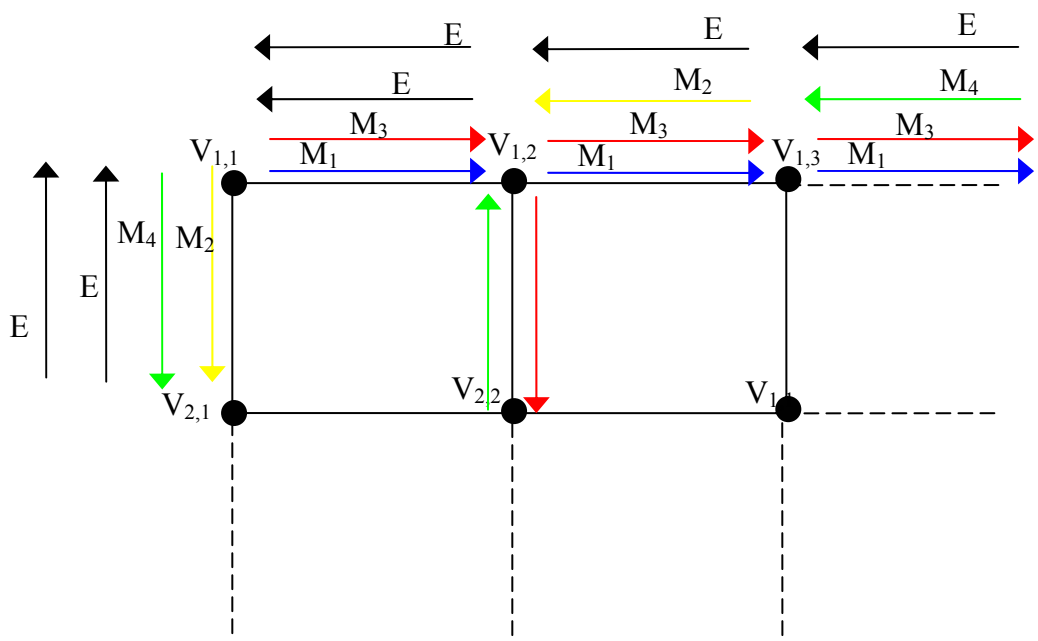
Theorem 4.5. Algorithm **H1_MMBROAD_DOG_ $G_{m,n}$** requires $m + n + 2.5M + C$ time units to complete broadcasting of M messages in DOG grids.

Proof: The algorithm works in such a way that messages traveling between vertices take into account orientation of the edges. All inside vertices are busy eight time units, two units with each neighbor. During these two time units, messages travel in the opposite direction. Outside vertices (vertices in the first row and column and the last row and column) are busy ten time units except four corner vertices. For outside vertices, we need to introduce empty message E that travels between vertices in order to reset them. Due to the fact that four new messages are introduced into the graph every ten time units and for some vertices it takes longer for a message to reach them occasional empty messages will be sent to reset the edge. A vertex receives all messages

from its neighbors. In the following subsections detailed communication for each vertex is given showing that each and every vertex receives all *M* messages.

The algorithm cannot send more messages in ten time units since each vertex has to watch its edges' orientations. Figure 4.6.1.1 shows corner of a grid. Once vertex $v_{1,2}$ receives a message from $v_{1,1}$ at time 1, it must send that message to vertices $v_{1,3}$ and $v_{2,2}$ and reset its edges. Hence, next best time it can receive a message is time 7. Second message it receives need to be sent only to $v_{1,2}$, so another three time units goes on sending second message to $v_{1,2}$, resetting its edges with $v_{1,1}$ and $v_{1,2}$. Third message can be received at time 11 etc.

Figures 4.6.1.2, 4.6.1.3 and 4.6.1.4 demonstrate the way the algorithm works. Blue lines indicate the path of the Set I messages. Yellow lines represent the path of Set II messages. Red lines and green lines represent the paths of Set III and Set IV messages correspondingly.



Figures 4.6.1.1 Communications of vertex $v_{1,2}$

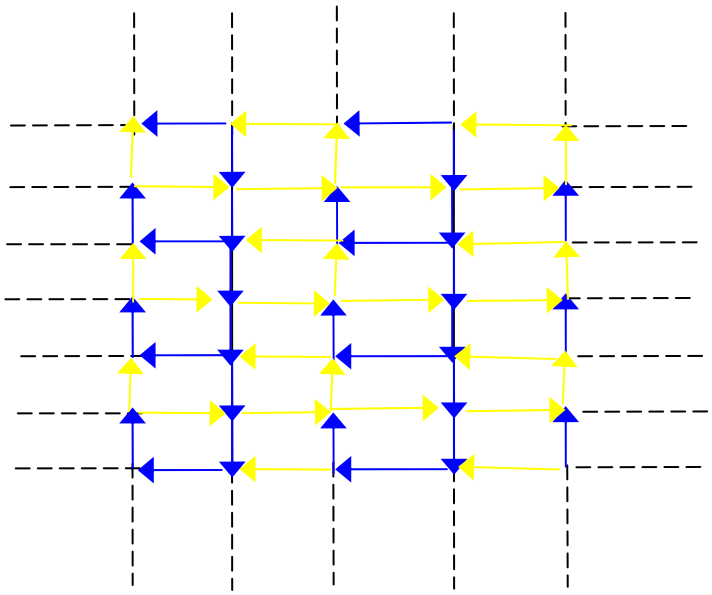


Figure 4.6.1.2 Paths of Set I and Set II messages

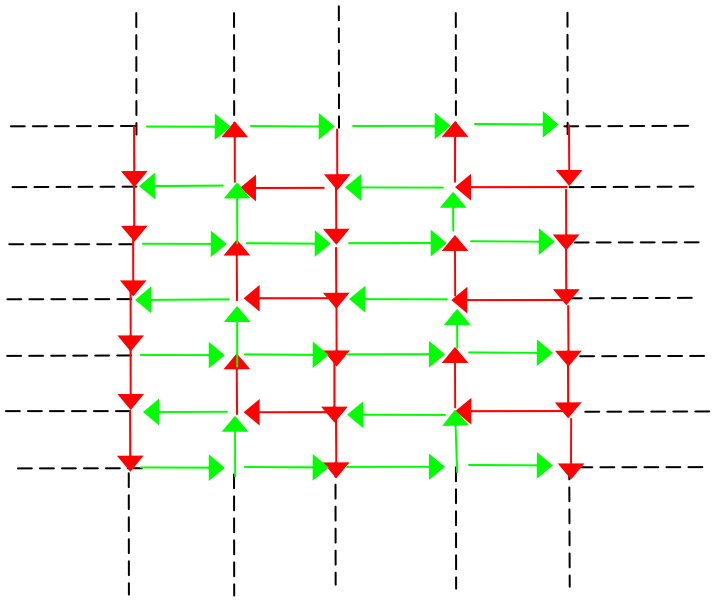


Figure 4.6.1.3 Paths of Set III and Set IV messages

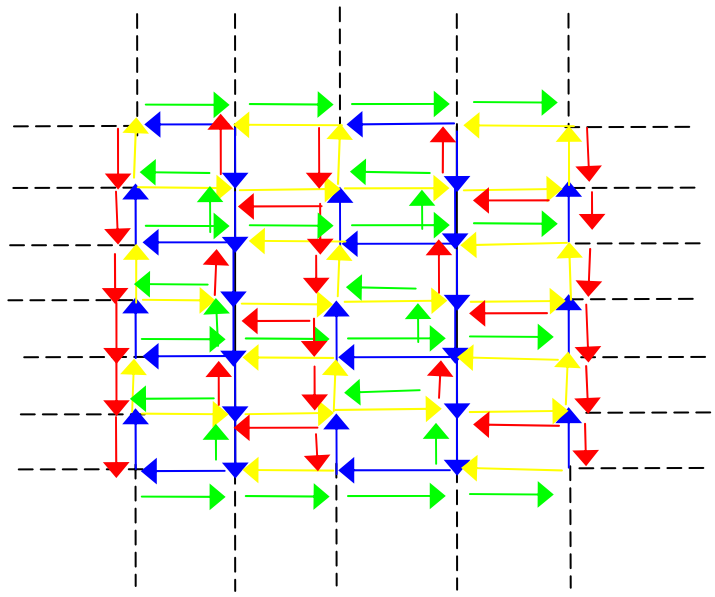


Figure 4.6.1.4 Combined paths of the messages in the grid

According to the algorithm vertex $v_{1,1}$ sends four new messages every ten time units. Thus, total time to broadcast M messages is $m + n + 2.5M + C$ for $G_{m,n}$, or $2n + 2.5M + C$ for $G_{n,n}$ grid. This result is better then the trivial $4n + 4M + C$ bound. □

4.6.2 Multiple message broadcasting in an even by even DOG grid.

Let G_{nxn} be an n x n DOG grid graph. Let n be an even integer, $n > 1$. The following is the detailed algorithm for broadcasting multiple messages from the corner vertex $v_{1,1}$ of the DOG grid.

- **Vertex $v_{1,1}$** (Source node)

- sends odd numbered message (1+4k), k ≥ 0 to vertex $v_{1,2}$ at time (1 + 10k)

- sends odd numbered message (3+4k), k ≥ 0 to vertex $v_{1,2}$ at time (7 + 10k)

- sends even numbered message (2+4k), k ≥ 0 to vertex $v_{2,1}$ at time (3 + 10k)

- sends even numbered message (4+4k), $k \geq 0$ to vertex $v_{2,1}$ at time (9 + 10k)

- receives empty message E_k, $k \geq 0$ from vertex $v_{1,2}$ at time (5 + 10k) and (10 + 10k) and from vertex $v_{2,1}$ at time (2 + 10k) and at time (6 + 10k)

- **Vertex $v_{1,2}$**

- receives odd numbered message (1+4k), $k \geq 0$ from vertex $v_{1,1}$ at time (1 + 10k)

- receives odd numbered message (3+4k), $k \geq 0$ from vertex $v_{1,1}$ at time (7 + 10k)

- sends odd numbered message (1+4k), $k \geq 0$ to vertex $v_{1,3}$ at time (2 + 10k) and to vertex $v_{2,2}$ at time (3 + 10k)

- sends odd numbered message (3+4k), $k \geq 0$ to vertex $v_{1,3}$ at time (8 + 10k)

- receives even numbered message (2+4k), $k \geq 0$ from vertex $v_{1,3}$ at time (16 + 10k)

- receives even numbered message (4+4k), $k \geq 0$ from vertex $v_{2,2}$ at time (19 + 10k)

- receives empty message E_k, $k \geq 0$ from vertex $v_{1,3}$ at time (4 + 10k) and at time 6, from vertex $v_{2,2}$ at time 9

- sends empty message E_k, $k \geq 0$ to vertex $v_{1,1}$ at time (5 + 10k) and at time (10 + 10k), to vertex $v_{2,2}$ at time (3 + 10(k+1)) and to vertex $v_{1,3}$ at time (8 + 10(k+1))

- **Vertex $v_{1,j}$** $(3 \leq j \leq n-1)$

  j is odd

- receives odd numbered message (1+4k), $k \geq 0$ from vertex $v_{1,j-1}$ at time (j - 1 + 10k)

- receives odd numbered message (3+4k), $k \geq 0$ from vertex $v_{1,j-1}$ at time (j + 5 + 10k)

- sends odd numbered message (1+4k), $k \geq 0$ to vertex $v_{1,j+1}$ at time (j + 10k)

- sends odd numbered message (3+4k), $k \geq 0$ to vertex $v_{1,j+1}$ at time (j + 6 + 10k) and to vertex $v_{2,j}$ at time (j + 7 + 10k)

- receives even numbered message (2+4k), $k \geq 0$ from vertex $v_{2,j}$ at time (j + 12 + 10k)

- receives even numbered message (4+4k), $k \geq 0$ from vertex $v_{1,j+1}$ at time (j + 24 + 10k)

- sends even numbered message (2+4k), $k \geq 0$ to vertex $v_{1,j-1}$ at time (j + 13 + 10k)

- receives empty message E_k, $k \geq 0$ from vertex $v_{1,j+1}$ at time (j + 8 + 10k) and at time (j + 4) and (j + 14) and from vertex $v_{1,j-1}$ at time (j + 5 + 10(k+1))

- sends empty message E_k, $k \geq 0$ to vertex $v_{1,j-1}$ at time (j + 1 + 10k) and at time (j + 3) and to vertex $v_{1,j+1}$ at time (j + 10(k+1)) and (j + 10(k+2))

j is even

- receives odd numbered message (1+4k), $k \geq 0$ from vertex $v_{1,j-1}$ at time (j - 1 + 10k)

- receives odd numbered message (3+4k), $k \geq 0$ from vertex $v_{1,j-1}$ at time (j + 5 + 10k)

- sends odd numbered message (1+4k), $k \geq 0$ to vertex $v_{1,j+1}$ at time (j + 10k) and to vertex $v_{2,j}$ at time (j + 1 + 10k)

- sends odd numbered message (3+4k), $k \geq 0$ to vertex $v_{1,j+1}$ at time (j + 6 + 10k)

- receives even numbered message (2+4k), $k \geq 0$ from vertex $v_{1,j+1}$ at time (j + 14 + 10k)

- receives even numbered message (4+4k), $k \geq 0$ from vertex $v_{2,j}$ at time (j + 18 + 10k)

- sends even numbered message (4+4k), $k \geq 0$ to vertex $v_{1,j-1}$ at time (j + 23 + 10k)

- receives empty message E_k, $k \geq 0$ from vertex $v_{1,j+1}$ at time (j + 2 + 10k) and at time (j + 4), from vertex $v_{1,j-1}$ at time (j - 1 + 10(k+1)) and (j - 1 + 10(k+2)) and from vertex $v_{2,j}$ at time (j + 8)

- sends empty message E_k, $k \geq 0$ to vertex $v_{1,j-1}$ at time (j + 7 + 10k), (j + 3) and at time (j + 13), to vertex $v_{1,j+1}$ at time (j + 6 + 10(k+1)) and to vertex $v_{2,j}$ at time (j + 1 + 10(k+1))

- **Vertex $v_{1,n}$**

- receives odd numbered message (1+4k), $k \geq 0$ from vertex $v_{1,n-1}$ at time (n - 1 + 10k)

- receives odd numbered message (3+4k), $k \geq 0$ from vertex $v_{1,n-1}$ at time (n + 5 + 10k)

- sends odd numbered message (1+4k), $k \geq 0$ to vertex $v_{2,n}$ at time (n + 10k)

- sends odd numbered message (3+4k), $k \geq 0$ to vertex $v_{2,n}$ at time (n + 6 + 10k)

- receives even numbered message (2+4k), $k \geq 0$ from vertex $v_{2,n}$ at time (n + 4 + 10k)

- receives even numbered message (4+4k), $k \geq 0$ from vertex $v_{2,n}$ at time (n + 22 + 10k)

- sends even numbered message (4+4k), $k \geq 0$ to vertex $v_{1,n-1}$ at time (n + 23 + 10k)

- receives empty message E_k, $k \geq 0$ from vertex $v_{2,n}$ at time (n + 2) and at time (n + 12) and vertex $v_{1,n-1}$ at time (n - 1 + 10(k+1)) and (n - 1 + 10(k+2))

- sends empty message E_k, $k \geq 0$ to vertex $v_{1,j-1}$ at time (n + 7 + 10k), at time (n + 3) and at time (n + 13) and to vertex $v_{2,n}$ at time (n + 10(k+1)) and (n + 10(k+2))

- **Vertex $v_{2,1}$**

- receives odd numbered message (1+4k), $k \geq 0$ from vertex $v_{3,1}$ at time (7 + 10k)

- receives odd numbered message (3+4k), $k \geq 0$ from vertex $v_{2,2}$ at time (21 + 10k)

- receives even numbered message (2+4k), $k \geq 0$ from vertex $v_{1,1}$ at time (3 + 10k)

- receives even numbered message (4+4k), $k \geq 0$ from vertex $v_{1,1}$ at time (9 + 10k)

- sends even numbered message (2+4k), $k \geq 0$ to vertex $v_{3,1}$ at time (4 + 10k) and to vertex $v_{2,2}$ at time (5 + 10k)

- sends even numbered message (4+4k), $k \geq 0$ to vertex $v_{3,1}$ at time (10 + 10k)

- receives empty message E_k, $k \geq 0$ from vertex $v_{2,2}$ at time 11 and from vertex $v_{3,1}$ at time (8 + 10k).

- sends empty message E_k, $k \geq 0$ to vertex $v_{1,1}$ at time (6 + 10k) and at time (2 + 10k) and to vertex $v_{2,2}$ at time (5 + 10(k+1))

- **Vertex $v_{3,1}$**

- receives odd numbered message (1+4k), $k \geq 0$ from vertex $v_{3,2}$ at time (6 + 10k)

- receives odd numbered message (3+4k), $k \geq 0$ from vertex $v_{4,1}$ at time (29 + 10k)

- sends odd numbered message (1+4k), $k \geq 0$ to vertex $v_{2,1}$ at time (7 + 10k)

- receives even numbered message (2+4k), $k \geq 0$ from vertex $v_{2,1}$ at time (4 + 10k)

- receives even numbered message (4+4k), $k \geq 0$ from vertex $v_{2,1}$ at time (10 + 10k)

- sends even numbered message (2+4k), $k \geq 0$ to vertex $v_{4,1}$ at time (5 + 10k)

- sends even numbered message (4+4k), $k \geq 0$ to vertex $v_{4,1}$ at time (11 + 10k) and to vertex $v_{3,2}$ at time (12 + 10k)

- receives empty message E_k, $k \geq 0$ from vertex $v_{4,1}$ at time 9, 19 and at time (3 + 10k).

- sends empty message E_k, $k \geq 0$ to vertex $v_{2,1}$ at time (8 + 10k) and to vertex $v_{4,1}$ at time (11 + 10(k+1)) and (11 + 10(k+2))

- **Vertex $v_{i,1}$** ($4 \leq i \leq n-1$)

  i is even

- receives odd numbered message (1+4k), $k \geq 0$ from vertex $v_{i+1,1}$ at time (i + 6 + 10k)

- receives odd numbered message (3+4k), $k \geq 0$ from vertex $v_{i,2}$ at time (i + 20 + 10k)

- sends odd numbered message (3+4k), $k \geq 0$ to vertex $v_{i-1,1}$ at time (i + 25 + 10k)

- receives even numbered message (2+4k), $k \geq 0$ from vertex $v_{i-1,1}$ at time (i + 1 + 10k)

- receives even numbered message (4+4k), $k \geq 0$ from vertex $v_{i-1,1}$ at time (i + 7 + 10k)

- sends even numbered message (2+4k), $k \geq 0$ to vertex $v_{i+1,1}$ at time (i + 2 + 10k) and to vertex $v_{i,2}$ at time (i + 3 + 10k).

- sends even numbered message (4+4k), $k \geq 0$ to vertex $v_{i+1,1}$ at time (i + 8 + 10k)

- receives empty message E_k, $k \geq 0$ from vertex $v_{i,2}$ at time (i + 10), from $v_{i+1,1}$ time (i + 4 + 10k) and from vertex $v_{i-1,1}$ at time (i + 7 + 10(k+1)) and (i + 7 + 10(k+2))

- sends empty message E_k, $k \geq 0$ to vertex $v_{i-1,1}$ at time (i + 5), (i + 15) and at time (i - 1 + 10k) and to vertex $v_{i,2}$ at time (i + 3 + 10(k+1))

i is odd

- receives odd numbered message (1+4k), $k \geq 0$ from vertex $v_{i,2}$ at time (i + 4 + 10k)

- receives odd numbered message (3+4k), $k \geq 0$ from vertex $v_{i+1,1}$ at time (i + 26 + 10k)

- sends odd numbered message (1+4k), $k \geq 0$ to vertex $v_{i-1,1}$ at time (i + 5 + 10k)

- receives even numbered message (2+4k), $k \geq 0$ from vertex $v_{i-1,1}$ at time (i + 1 + 10k)

- receives even numbered message (4+4k), $k \geq 0$ from vertex $v_{i-1,1}$ at time (i + 7 + 10k)

- sends even numbered message (2+4k), $k \geq 0$ to vertex $v_{i+1,1}$ at time (i + 2 + 10k)

- sends even numbered message (4+4k), $k \geq 0$ to vertex $v_{i+1,1}$ at time (i + 8 + 10k) and to vertex $v_{i,2}$ at time (i + 9 + 10k)

- receives empty message E_k, $k \geq 0$ from vertex $v_{i+1,1}$ at time (i + 6), (i + 16) and at time (i + 10k).

- sends empty message E_k, $k \geq 0$ to vertex $v_{i-1,1}$ at time (i + 3 + 10k) and to vertex $v_{i+1,1}$ at time (i + 8 + 10(k+1)) and (i + 8 + 10(k+2))

- **Vertex $v_{n,1}$**

- receives odd numbered message (1+4k), $k \geq 0$ from vertex $v_{n,2}$ at time (n + 10 + 10k)

- receives odd numbered message (3+4k), $k \geq 0$ from vertex $v_{n,2}$ at time (n + 16 + 10k)

- sends odd numbered message (3+4k), $k \geq 0$ to vertex $v_{n-1,1}$ at time (n + 25 + 10k)

- receives even numbered message (2+4k), $k \geq 0$ from vertex $v_{n-1,1}$ at time (n + 1 + 10k)

- receives even numbered message (4+4k), $k \geq 0$ from vertex $v_{n-1,1}$ at time (n + 7 + 10k)

- sends even numbered message (2+4k), $k \geq 0$ to vertex $v_{n,2}$ at time (n + 2 + 10k)

- sends even numbered message (4+4k), $k \geq 0$ to vertex $v_{n,2}$ at time (n + 8 + 10k)

- receives empty message E, from vertex $v_{n,2}$ at time (n + 6) and from vertex $v_{n-1,1}$ at time (n + 7 + 10(k+1)) and (n + 7 + 10(k+2))

- sends empty message E_k, $k \geq 0$ to vertex $v_{n-1,1}$ at time (n + 5), (n + 15) and at time (n - 1 + 10k) and to vertex $v_{n,2}$ at time (n + 2 + 10(k+1))

- **Vertex $v_{n,2}$**

- receives odd numbered message (1+4k), $k \geq 0$ from vertex $v_{n-1,2}$ at time (n + 1 + 10k)

- receives odd numbered message (3+4k), $k \geq 0$ from vertex $v_{n,3}$ at time (n + 15 + 10k)

- sends odd numbered message (1+4k), $k \geq 0$ to vertex $v_{n,1}$ at time (n + 10 + 10k)

- sends odd numbered message (3+4k), $k \geq 0$ to vertex $v_{n,1}$ at time (n + 16 + 10k) and to vertex $v_{n-1,2}$ at time (n + 24 + 10k)

- receives even numbered message (2+4k), $k \geq 0$ from vertex $v_{n,1}$ at time (n + 2 + 10k)

- receives even numbered message (4+4k), $k \geq 0$ from vertex $v_{n,1}$ at time (n + 8 + 10k)

- sends even numbered message (2+4k), $k \geq 0$ to vertex $v_{n,3}$ at time (n + 3 + 10k)

- sends even numbered message (4+4k), $k \geq 0$ to vertex $v_{n,3}$ at time (n + 9 + 10k)

- receives empty message E_k, $k \geq 0$ from vertex $v_{n,3}$ at time (n + 5) and at time (n + 7 + 10k), from vertex $v_{n,1}$ at time (n + 2 + 10(k+1)) and from vertex $v_{n-1,2}$ at time (n + 1 + 10(k+1)) and (n + 1 + 10(k+2))

- sends empty message E to vertex $v_{n-1,2}$ at time (n + 4), (n + 14) and to vertex $v_{n,1}$ at time (n + 6) and to vertex $v_{n,3}$ at time (n + 3 + 10(k+1))

- **Vertex $v_{n,j}$ $(3 \leq j \leq n-3)$**

  j is even

- receives odd numbered message (1+4k), $k \geq 0$ from vertex $v_{n-1,j}$ at time (n + j - 1 + 10k)

- receives odd numbered message (3+4k), $k \geq 0$ from vertex $v_{n,j+1}$ at time (n + j + 13 + 10k)

- sends odd numbered message (1+4k), $k \geq 0$ to vertex $v_{n,j-1}$ at time (n + j + 2 + 10k)

- sends odd numbered message (3+4k), $k \geq 0$ to vertex $v_{n-1,j}$ at time (n + j + 14 + 10k).

- receives even numbered message (2+4k), $k \geq 0$ from vertex $v_{n,j-1}$ at time (n + j + 10k)

- receives even numbered message (4+4k), $k \geq 0$ from vertex $v_{n,j-1}$ at time (n + j + 6 + 10k)

- sends even numbered message (2+4k), $k \geq 0$ to vertex $v_{n,j+1}$ at time (n + j + 1 + 10k)

- sends even numbered message (4+4k), $k \geq 0$ to vertex $v_{n,j+1}$ at time (n + j + 7 + 10k)

- receives empty message E_k, $k \geq 0$ from vertex $v_{n,j+1}$ at time (n + j + 3) and at time (n + j + 5 + 10k) and from vertex $v_{n-1,j}$ at time (n + j – 1 + 10(k+1))

- sends empty message E_k, $k \geq 0$ to vertex $v_{n-1,j}$ at time (n + j + 4) and to vertex $v_{n,j-1}$ at time (n + j - 2 + 10k) and to vertex $v_{n,j+1}$ at time (n + j + 1 + 10(k+1))

  j is odd

- receives odd numbered message (1+4k), $k \geq 0$ from vertex $v_{n,j+1}$ at time (n + j + 3 + 10k)

- receives odd numbered message (3+4k), $k \geq 0$ from vertex $v_{n-1,j}$ at time (n + j + 5 + 10k)

- sends odd numbered message (3+4k), $k \geq 0$ to vertex $v_{n,j-1}$ at time (n + j + 12 + 10k)

- receives even numbered message (2+4k), $k \geq 0$ from vertex $v_{n,j-1}$ at time (n + j + 10k)

- receives even numbered message (4+4k), $k \geq 0$ from vertex $v_{n,j-1}$ at time (n + j + 6 + 10k)

- sends even numbered message (2+4k), $k \geq 0$ to vertex $v_{n,j+1}$ at time (n + j + 1 + 10k) and to vertex $v_{n-1,j}$ at time (n + j + 8 + 10k)

- sends even numbered message (4+4k), $k \geq 0$ to vertex $v_{n,j+1}$ at time (n + j + 7 + 10k)

- receives empty message E_k, $k \geq 0$ from vertex $v_{n,j+1}$ at time (n + j - 1 + 10k) and from vertex $v_{n,j-1}$ at time (n + j + 10(k+1))

- sends empty message E_k, $k \geq 0$ to vertex $v_{n,j-1}$ at time (n + j + 2) and at time (n + j + 4 + 10k)

- **Vertex $v_{n,n-2}$**

- receives odd numbered message (1+4k), $k \geq 0$ from vertex $v_{n-1,n-2}$ at time (2n - 3 + 10k)

- receives odd numbered message (3+4k), $k \geq 0$ from vertex $v_{n,n-1}$ at time (2n + 11 + 10k)

- sends odd numbered message (1+4k), $k \geq 0$ to vertex $v_{n,n-3}$ at time (2n + 10k)

- sends odd numbered message (3+4k), $k \geq 0$ to vertex $v_{n-1,n-2}$ at time (2n + 13 + 10k)

- receives even numbered message (2+4k), $k \geq 0$ from vertex $v_{n,n-3}$ at time (2n - 2 + 10k)

- receives even numbered message (4+4k), $k \geq 0$ from vertex $v_{n,n-3}$ at time (2n + 4 + 10k)

- sends even numbered message (2+4k), $k \geq 0$ to vertex $v_{n,n-1}$ at time (2n - 1 + 10k)

- sends even numbered message (4+4k), $k \geq 0$ to vertex $v_{n,n-1}$ at time (2n + 5 + 10k)

- receives empty message E_k, $k \geq 0$ from vertex $v_{n,n-1}$ at time (2n + 1) and at time (2n + 2 + 10k) and from vertex $v_{n-1,n-2}$ at time (2n – 3 + 10(k+1))

- sends empty message E_k, $k \geq 0$ to vertex $v_{n-1,n-2}$ at time (2n + 3), to vertex $v_{n,n-3}$ at time (2n - 4 + 10k) and to vertex $v_{n,n-1}$ at time (2n – 1 + 10(k+1))

- **Vertex $v_{n,n-1}$**

- receives odd numbered message (1+4k), $k \geq 0$ from vertex $v_{n,n}$ at time (2n + 3 + 10k)

- receives odd numbered message (3+4k), $k \geq 0$ from vertex $v_{n-1,n-1}$ at time (2n + 4 + 10k)

- sends odd numbered message (3+4k), $k \geq 0$ to vertex $v_{n,n-2}$ at time (2n + 11 + 10k)

- receives even numbered message (2+4k), $k \geq 0$ from vertex $v_{n,n-2}$ at time (2n - 1 + 10k)

- receives even numbered message (4+4k), $k \geq 0$ from vertex $v_{n,n-2}$ at time (2n + 5 + 10k)

- sends even numbered message (2+4k), $k \geq 0$ to vertex $v_{n,n}$ at time (2n + 10k) and to vertex $v_{n-1,n-1}$ at time (2n + 8 + 10k)

- sends even numbered message (4+4k), $k \geq 0$ to vertex $v_{n,n}$ at time (2n + 6 + 10k)

- receives empty message E_k, $k \geq 0$ from vertex $v_{n,n}$ at time (2n - 3 + 10k) and from vertex $v_{n,n-2}$ at time (2n - 1 + 10(k+1))

- sends empty message E_k, $k \geq 0$ to vertex $v_{n,n-2}$ at time (2n + 1) and at time (2n + 2 + 10k)

- **Vertex $v_{n,n}$**

- receives odd numbered message (1+4k), $k \geq 0$ from vertex $v_{n-1,n}$ at time (2n -2 + 10k)

- receives odd numbered message (3+4k), $k \geq 0$ from vertex $v_{n-1,n}$ at time (2n + 4 + 10k)

- sends odd numbered message (1+4k), $k \geq 0$ to vertex $v_{n,n-1}$ at time (2n + 3 + 10k)

- receives even numbered message (2+4k), $k \geq 0$ from vertex $v_{n,n-1}$ at time (2n + 10k)

- receives even numbered message (4+4k), $k \geq 0$ from vertex $v_{n,n-1}$ at time (2n + 6 + 10k)

- sends even numbered message (2+4k), $k \geq 0$ to vertex $v_{n-1,n}$ at time (2n + 2 + 10k)

- sends empty message E_k, $k \geq 0$ to vertex $v_{n,n-1}$ at time (2n - 3 + 10k) and to vertex $v_{n-1,n}$ at time (2n - 1 + 10k)

- **Vertex $v_{2,n}$**

- receives odd numbered message (1+4k), $k \geq 0$ from vertex $v_{1,n}$ at time (n + 10k)

- receives odd numbered message (3+4k), $k \geq 0$ from vertex $v_{1,n}$ at time (n + 6 + 10k)

- sends odd numbered message (1+4k), $k \geq 0$ to vertex $v_{3,n}$ at time (n + 1 + 10k)

- sends odd numbered message (3+4k), $k \geq 0$ to vertex $v_{3,n}$ at time (n + 7 + 10k)

- receives even numbered message (2+4k), $k \geq 0$ from vertex $v_{2,n-1}$ at time (n + 3 + 10k)

- receives even numbered message (4+4k), $k \geq 0$ from vertex $v_{3,n}$ at time (n + 15 + 10k)

- sends even numbered message (2+4k), $k \geq 0$ to vertex $v_{1,n}$ at time (n + 4 + 10k)

- sends even numbered message (4+4k), $k \geq 0$ to vertex $v_{2,n-1}$ at time (n + 18 + 10k) and to vertex $v_{1,n}$ at time (n + 22 + 10k)

- receives empty message E_k, $k \geq 0$ from vertex $v_{3,n}$ at time (n + 5) and (n - 1 + 10k), from vertex $v_{2,n-1}$ at time (n + 3 + 10(k+1)) and from vertex $v_{1,n}$ at time (n + 10(k+1)) and (n + 10(k+2))

- sends empty message E_k, $k \geq 0$ to vertex $v_{2,n-1}$ at time (n + 8), to vertex $v_{1,n}$ at time (n + 2) and (n + 12) and to vertex $v_{3,n}$ at time (n + 7 + 10(k+1))

- **Vertex $v_{i,n}$ $(3 \leq i \leq n-2)$**

  i is even

- receives odd numbered message (1+4k), $k \geq 0$ from vertex $v_{i-1,n}$ at time (i + n -2 + 10k)

- receives odd numbered message (3+4k), $k \geq 0$ from vertex $v_{i-1,n}$ at time (i + n + 4 + 10k)

- sends odd numbered message (1+4k), $k \geq 0$ to vertex $v_{i+1,n}$ at time (i + n - 1 + 10k)

- sends odd numbered message (3+4k), $k \geq 0$ to vertex $v_{i+1,n}$ at time (i + n + 5 + 10k)

- receives even numbered message (2+4k), $k \geq 0$ from vertex $v_{i,n-1}$ at time (i + n + 1 + 10k)

- receives even numbered message (4+4k), $k \geq 0$ from vertex $v_{i+1,n}$ at time (i + n + 13 + 10k)

- sends even numbered message (2+4k), $k \geq 0$ to vertex $v_{i-1,n}$ at time (i + n + 2 + 10k)

- sends even numbered message (4+4k), $k \geq 0$ to vertex $v_{i,n-1}$ at time (i + n + 16 + 10k)

- receives empty message E_k, $k \geq 0$ from vertex $v_{i+1,n}$ at time (i + n + 3) and (i + n - 3 + 10k), from vertex $v_{i,n-1}$ at time (i + n + 1 + 10(k+1))

- sends empty message E_k, $k \geq 0$ to vertex $v_{i,n-1}$ at time (i + n + 6), to vertex $v_{i-1,n}$ at time (i + n + 10k) and to vertex $v_{i+1,n}$ at time (i + n + 5 + 10(k+1))

i is odd

- receives odd numbered message (1+4k), $k \geq 0$ from vertex $v_{i-1,n}$ at time (i + n -2 + 10k)

- receives odd numbered message (3+4k), $k \geq 0$ from vertex $v_{i-1,n}$ at time (i + n + 4 + 10k)

- sends odd numbered message (1+4k), $k \geq 0$ to vertex $v_{i+1,n}$ at time (i + n - 1 + 10k) and to vertex $v_{i,n-1}$ at time (i + n + 10k)

- sends odd numbered message (3+4k), $k \geq 0$ to vertex $v_{i+1,n}$ at time (i + n + 5 + 10k)

- receives even numbered message (2+4k), $k \geq 0$ from vertex $v_{i+1,n}$ at time (i + n + 3 + 10k)

- receives even numbered message (4+4k), $k \geq 0$ from vertex $v_{i,n-1}$ at time (i + n + 7 + 10k)

- sends even numbered message (4+4k), $k \geq 0$ to vertex $v_{i-1,n}$ at time (i + n + 12 + 10k)

- receives empty message E_k, $k \geq 0$ from vertex $v_{i+1,n}$ at time (i + n + 1 + 10k) and from vertex $v_{i-1,n}$ at time (i + n + 4 + 10(k+1))

- sends empty message E_k, $k \geq 0$ to vertex $v_{i-1,n}$ at time (i + n - 4 + 10k) and to vertex $v_{i-1,n}$ at time (i + n + 2)

- **Vertex $v_{n-1,n}$**

- receives odd numbered message (1+4k), $k \geq 0$ from vertex $v_{n-2,n}$ at time (2n - 3 + 10k)

- receives odd numbered message (3+4k), $k \geq 0$ from vertex $v_{n-2,n}$ at time (2n + 3 + 10k)

- sends odd numbered message (1+4k), $k \geq 0$ to vertex $v_{n,n}$ at time (2n - 2 + 10k) and to vertex $v_{n-1,n-1}$ at time (2n + 10k)

- sends odd numbered message (3+4k), $k \geq 0$ to vertex $v_{n,n}$ at time (2n + 4 + 10k)

- receives even numbered message (2+4k), $k \geq 0$ from vertex $v_{n,n}$ at time (2n + 2 + 10k)

- receives even numbered message (4+4k), $k \geq 0$ from vertex $v_{n-1,n-1}$ at time (2n + 6 + 10k)

- sends even numbered message (4+4k), $k \geq 0$ to vertex $v_{n-2,n}$ at time (2n + 11 + 10k)

- receives empty message E_k, $k \geq 0$ from vertex $v_{n,n}$ at time (2n - 1 + 10k) and from vertex $v_{n-2,n}$ at time (2n + 3 + 10(k+1))

- sends empty message E_k, $k \geq 0$ to vertex $v_{n-2,n}$ at time (2n + 1) and (2n - 5 + 10k)

- **Vertex $v_{2,2}$**

- receives odd numbered message (1+4k), $k \geq 0$ from vertex $v_{1,2}$ at time (3 + 10k)

- receives odd numbered message (3+4k), $k \geq 0$ from vertex $v_{2,3}$ at time (12 + 10k)

- sends odd numbered message (1+4k), $k \geq 0$ to vertex $v_{3,2}$ at time (4 + 10k)

- sends odd numbered message (3+4k), $k \geq 0$ to vertex $v_{2,1}$ at time (21 + 10k)

- receives even numbered message (2+4k), $k \geq 0$ from vertex $v_{2,1}$ at time (5 + 10k)

- receives even numbered message (4+4k), $k \geq 0$ from vertex $v_{3,2}$ at time (18 + 10k)

- sends even numbered message (2+4k), $k \geq 0$ to vertex $v_{2,3}$ at time (6 + 10k)

- sends even numbered message (4+4k), $k \geq 0$ to vertex $v_{1,2}$ at time (19 + 10k)

- receives empty message E from vertex $v_{3,2}$ at time 8, from vertex $v_{1,2}$ at time (3 + 10(k+1)) and from vertex $v_{2,1}$ at time (5 + 10(k+1))

- sends empty message E to vertex $v_{1,2}$ at time 9, to vertex $v_{2,1}$ at time 11 and to vertex $v_{3,2}$ at time (4 + 10(k+1))

- **Vertex $v_{2,j}$** $(3 \le j \le n-3)$

j is even

- receives odd numbered message (1+4k), $k \ge 0$ from vertex $v_{1,j}$ at time (j + 1 + 10k)

- receives odd numbered message (3+4k), $k \ge 0$ from vertex $v_{2,j+1}$ at time (j + 10 + 10k)

- sends odd numbered message (1+4k), $k \ge 0$ to vertex $v_{3,j}$ at time (j + 2 + 10k)

- receives even numbered message (2+4k), $k \ge 0$ from vertex $v_{2,j-1}$ at time (j + 3 + 10k)

- receives even numbered message (4+4k), $k \ge 0$ from vertex $v_{3,j}$ at time (j + 16 + 10k)

- sends even numbered message (2+4k), $k \ge 0$ to vertex $v_{2,j+1}$ at time (j + 4 + 10k)

- sends even numbered message (4+4k), $k \ge 0$ to vertex $v_{1,j}$ at time (j + 18 + 10k) and to vertex $v_{2,j-1}$ at time (j + 19 + 10k)

- receives empty message E from vertex $v_{3,j}$ at time (j + 6), from vertex $v_{1,j}$ at time (j + 1 + 10(k+1)) and from vertex $v_{2,j-1}$ at time (j + 3 + 10(k+1))

- sends empty message E to vertex $v_{2,j-1}$ at time (j + 9), to vertex $v_{1,j}$ at time (j + 8) and to vertex $v_{3,j}$ at time (j + 2 + 10(k+1))

j is odd

- receives odd numbered message (1+4k), $k \ge 0$ from vertex $v_{3,j}$ at time (j + 6 + 10k)

- receives odd numbered message (3+4k), $k \ge 0$ from vertex $v_{1,j}$ at time (j + 7 + 10k)

- sends odd numbered message (3+4k), $k \ge 0$ to vertex $v_{3,j}$ at time (j + 8 + 10k) and to vertex $v_{2,j-1}$ at time (j+ 9 + 10k)

- receives even numbered message (2+4k), $k \ge 0$ from vertex $v_{2,j-1}$ at time (j + 3 + 10k)

- receives even numbered message (4+4k), $k \ge 0$ from vertex $v_{2,j+1}$ at time (j + 20 + 10k)

- sends even numbered message (2+4k), $k \ge 0$ to vertex $v_{2,j+1}$ at time (j + 4 + 10k) and to vertex $v_{1,j}$ at time (j + 12 + 10k)

- receives empty message E from vertex $v_{2,j+1}$ at time (j + 10)

- sends empty message E to vertex $v_{2,j+1}$ at time (j + 4 + 10(k+1))

- **Vertex $v_{2,n-2}$**

- receives odd numbered message (1+4k), $k \geq 0$ from vertex $v_{1,n-2}$ at time (n-1 + 10k)

- receives odd numbered message (3+4k), $k \geq 0$ from vertex $v_{2,n-1}$ at time (n + 15 + 10k)

- sends odd numbered message (1+4k), $k \geq 0$ to vertex $v_{3,n-2}$ at time (n + 10k)

- receives even numbered message (2+4k), $k \geq 0$ from vertex $v_{2,n-3}$ at time (n + 1 + 10k)

- receives even numbered message (4+4k), $k \geq 0$ from vertex $v_{3,n-2}$ at time (n + 14 + 10k)

- sends even numbered message (2+4k), $k \geq 0$ to vertex $v_{2,n-1}$ at time (n + 2 + 10k)

- sends even numbered message (4+4k), $k \geq 0$ to vertex $v_{1,n-2}$ at time (n + 16 + 10k) and to vertex $v_{2,n-3}$ at time (n + 17 + 10k)

- receives empty message E from vertex $v_{2,n-1}$ at time (n + 5), from vertex $v_{3,n-2}$ at time (n + 4), from vertex $v_{2,n-3}$ at time (n + 1 + 10(k+1)) and from vertex $v_{1,n-2}$ at time (n - 1 + 10(k+1))

- sends empty message E to vertex $v_{2,n-3}$ at time (n + 7), to vertex $v_{1,n-2}$ at time (n + 6), to vertex $v_{3,n-2}$ at time (n + 10(k+1)) and to vertex $v_{2,n-1}$ at time (n + 2 + 10(k+1))

- **Vertex $v_{2,n-1}$**

- receives odd numbered message (1+4k), $k \geq 0$ from vertex $v_{3,n-1}$ at time (n + 4 + 10k)

- receives odd numbered message (3+4k), $k \geq 0$ from vertex $v_{1,n-1}$ at time (n + 6 + 10k)

- sends odd numbered message (3+4k), $k \geq 0$ to vertex $v_{3,n-1}$ at time (n + 7 + 10k) and to vertex $v_{2,n-2}$ at time (n + 15 + 10k)

- receives even numbered message (2+4k), $k \geq 0$ from vertex $v_{2,n-2}$ at time (n + 2 + 10k)

- receives even numbered message (4+4k), $k \geq 0$ from vertex $v_{2,n}$ at time (n + 18 + 10k)

- sends even numbered message (2+4k), $k \geq 0$ to vertex $v_{2,n}$ at time (n + 3 + 10k) and to vertex $v_{1,n-1}$ at time (n + 11 + 10k)

- receives empty message E from vertex $v_{2,n}$ at time (n + 8) and from vertex $v_{2,n-2}$ at time (n + 2 + 10(k+1))

- sends empty message E to vertex $v_{2,n-2}$ at time (n + 5) and to vertex $v_{2,n}$ at time (n + 3 + 10(k+1))

- **Vertex $v_{3,2}$**

- receives odd numbered message (1+4k), $k \geq 0$ from vertex $v_{2,2}$ at time (4 + 10k)

- receives odd numbered message (3+4k), $k \geq 0$ from vertex $v_{4,2}$ at time (21 + 10k)

- sends odd numbered message (1+4k), $k \geq 0$ to vertex $v_{4,2}$ at time (5 + 10k) and to vertex $v_{3,1}$ at time (6 + 10k)

- receives even numbered message (2+4k), $k \geq 0$ from vertex $v_{3,3}$ at time (17 + 10k)

- receives even numbered message (4+4k), $k \geq 0$ from vertex $v_{3,1}$ at time (12 + 10k)

- sends even numbered message (4+4k), $k \geq 0$ to vertex $v_{3,3}$ at time (13 + 10k) and to vertex $v_{2,2}$ at time (18 + 10k)

- receives empty message E from vertex $v_{4,2}$ at time 11 and from vertex $v_{2,2}$ at time (4 + 10(k+1))

- sends empty message E to vertex $v_{2,2}$ at time 8 and to vertex $v_{4,2}$ at time (5 + 10(k+1))

- **Vertex $v_{i,2}$** $(4 \leq i \leq n-2)$

  i is even

- receives odd numbered message (1+4k), $k \geq 0$ from vertex $v_{i-1,2}$ at time (i + 1 + 10k)

- receives odd numbered message (3+4k), $k \geq 0$ from vertex $v_{i,3}$ at time (i +16 + 10k)

- sends odd numbered message (1+4k), $k \geq 0$ to vertex $v_{i+1,2}$ at time (i + 2 + 10k)

- sends odd numbered message (3+4k), $k \geq 0$ to vertex $v_{i-1,2}$ at time (i + 17 + 10k) and to vertex $v_{i,1}$ at time (i + 20 + 10k)

- receives even numbered message (2+4k), $k \geq 0$ from vertex $v_{i,1}$ at time (i + 3 + 10k)

- receives even numbered message (4+4k), $k \geq 0$ from vertex $v_{i+1,2}$ at time (i + 18 + 10k)

- sends even numbered message (2+4k), $k \geq 0$ to vertex $v_{i,3}$ at time (i + 4 + 10k)

- receives empty message E from vertex $v_{i+1,2}$ at time (i + 8), from vertex $v_{i,3}$ at time (i + 6), from vertex $v_{i-1,2}$ at time (i + 1 + 10(k+1)) and from vertex $v_{i,1}$ at time (i + 3 + 10(k+1))

- sends empty message E to vertex $v_{i-1,2}$ at time (i + 7), to vertex $v_{i,1}$ at time (i + 10), to vertex $v_{i+1,2}$ at time (i + 2 + 10(k+1)) and to vertex $v_{i,3}$ at time (i + 4 + 10(k+1))

i is odd

- receives odd numbered message (1+4k), $k \geq 0$ from vertex $v_{i-1,2}$ at time (i + 1 + 10k)

- receives odd numbered message (3+4k), $k \geq 0$ from vertex $v_{i+1,2}$ at time (i + 18 + 10k)

- sends odd numbered message (1+4k), $k \geq 0$ to vertex $v_{i+1,2}$ at time (i + 2 + 10k) and to vertex $v_{i,1}$ at time (i + 4 + 10k)

- receives even numbered message (2+4k), $k \geq 0$ from vertex $v_{i,3}$ at time (i + 16 + 10k)

- receives even numbered message (4+4k), $k \geq 0$ from vertex $v_{i,1}$ at time (i + 9 + 10k)

- sends even numbered message (4+4k), $k \geq 0$ to vertex $v_{i,3}$ at time (i + 10 + 10k) and to vertex $v_{i-1,2}$ at time (i + 17 + 10k)

- receives empty message E from vertex $v_{i+1,2}$ at time (i + 8) and from vertex $v_{i-1,2}$ at time (i + 1 + 10(k+1))

- sends empty message E to vertex $v_{i-1,2}$ at time (i + 7) and to vertex $v_{i+1,2}$ at time (i + 2 + 10(k+1))

- **Vertex $v_{n-1,2}$**

- receives odd numbered message (1+4k), k ≥ 0 from vertex $v_{n-2,2}$ at time (n + 10k)

- receives odd numbered message (3+4k), k ≥ 0 from vertex $v_{n,2}$ at time (n + 24 + 10k)

- sends odd numbered message (1+4k), k ≥ 0 to vertex $v_{n,2}$ at time (n + 1 + 10k) and to vertex $v_{n-1,1}$ at time (n + 3 + 10k)

- receives even numbered message (2+4k), k ≥ 0 from vertex $v_{n-1,3}$ at time (n + 12 + 10k)

- receives even numbered message (4+4k), k ≥ 0 from vertex $v_{n-1,1}$ at time (n + 8 + 10k)

- sends even numbered message (4+4k), k ≥ 0 to vertex $v_{n-1,3}$ at time (n + 9 + 10k) and to vertex $v_{n-2,2}$ at time (n + 16 + 10k)

- receives empty message E from vertex $v_{n,2}$ at time (n + 4) and (n + 14) and from vertex $v_{n-2,2}$ at time (n + 10(k+1))

- sends empty message E to vertex $v_{n-2,2}$ at time (n + 6) and to vertex $v_{n,2}$ at time (n + 1 + 10(k+1)) and (n + 1 + 10(k+2))

- **Vertex $v_{n-1,3}$**

- receives odd numbered message (1+4k), k ≥ 0 from vertex $v_{n-1,4}$ at time (n + 4 + 10k)

- receives odd numbered message (3+4k), k ≥ 0 from vertex $v_{n-2,3}$ at time (n + 7 + 10k)

- sends odd numbered message (1+4k), k ≥ 0 to vertex $v_{n-2,3}$ at time (n + 15 + 10k)

- sends odd numbered message (3+4k), k ≥ 0 to vertex $v_{n,3}$ at time (n + 8 + 10k)

- receives even numbered message (2+4k), k ≥ 0 from vertex $v_{n,3}$ at time (n + 11 + 10k)

- receives even numbered message (4+4k), $k \geq 0$ from vertex $v_{n-1,2}$ at time (n + 9 + 10k)

- sends even numbered message (2+4k), $k \geq 0$ to vertex $v_{n-1,2}$ at time (n + 12 + 10k)

- sends even numbered message (4+4k), $k \geq 0$ to vertex $v_{n-1,4}$ at time (n + 10 + 10k)

- **Vertex $v_{n-1,j}$** $(4 \leq j \leq n-3)$

j is even

- receives odd numbered message (1+4k), $k \geq 0$ from vertex $v_{n-2,j}$ at time (n + j – 2 + 10k)

- receives odd numbered message (3+4k), $k \geq 0$ from vertex $v_{n,j}$ at time (n + j + 14 + 10k)

- sends odd numbered message (1+4k), $k \geq 0$ to vertex $v_{n,j}$ at time (n + j – 1 + 10k) and to vertex $v_{n-1,j-1}$ at time (n + j + 10k)

- receives even numbered message (2+4k), $k \geq 0$ from vertex $v_{n-1,j+1}$ at time (n + j + 11 + 10k)

- receives even numbered message (4+4k), $k \geq 0$ from vertex $v_{n-1,j-1}$ at time (n + j + 6 + 10k)

- sends even numbered message (4+4k), $k \geq 0$ to vertex $v_{n-1,j+1}$ at time (n + j + 7 + 10k) and to vertex $v_{n-2,j}$ at time (n + j + 12 + 10k)

- receives empty message E from vertex $v_{n,j}$ at time (n + j + 4) and from vertex $v_{n-2,j}$ at time (n + j – 2 + 10(k+1))

- sends empty message E to vertex $v_{n-2,j}$ at time (n + j + 2) and to vertex $v_{n,j}$ at time (n + j – 1 + 10(k+1))

j is odd

- receives odd numbered message (1+4k), $k \geq 0$ from vertex $v_{n-1,j+1}$ at time (n + j + 1 + 10k)

- receives odd numbered message (3+4k), $k \geq 0$ from vertex $v_{n-2,j}$ at time (n + j + 4 + 10k)

- sends odd numbered message (1+4k), $k \geq 0$ to vertex $v_{n-2,j}$ at time (n + j + 12 + 10k)

- sends odd numbered message (3+4k), $k \geq 0$ to vertex $v_{n,j}$ at time (n + j + 5 + 10k)

- receives even numbered message (2+4k), $k \geq 0$ from vertex $v_{n,j}$ at time (n + j + 8 + 10k)

- receives even numbered message (4+4k), $k \geq 0$ from vertex $v_{n-1,j-1}$ at time (n + j + 6 + 10k)

- sends even numbered message (2+4k), $k \geq 0$ to vertex $v_{n-1,j-1}$ at time (n + j + 10 + 10k) - sends even numbered message (4+4k), $k \geq 0$ to vertex $v_{n-1,j+1}$ at time (n + j + 7 + 10k)

- **Vertex $v_{n-1,n-2}$**

- receives odd numbered message (1+4k), $k \geq 0$ from vertex $v_{n-2,n-2}$ at time (2n - 4 + 10k)

- receives odd numbered message (3+4k), $k \geq 0$ from vertex $v_{n,n-2}$ at time (2n + 13 + 10k)

- sends odd numbered message (1+4k), $k \geq 0$ to vertex $v_{n,n-2}$ at time (2n - 3 + 10k) and to vertex $v_{n-1,n-3}$ at time (2n - 2 + 10k)

- receives even numbered message (2+4k), $k \geq 0$ from vertex $v_{n-1,n-1}$ at time (2n + 9 + 10k)

- receives even numbered message (4+4k), $k \geq 0$ from vertex $v_{n-1,n-3}$ at time (2n + 4 + 10k)

- sends even numbered message (4+4k), $k \geq 0$ to vertex $v_{n-1,n-1}$ at time (2n + 5 + 10k) and to vertex $v_{n-2,n-2}$ at time (2n + 12 + 10k)

- receives empty message E from vertex $v_{n,n-2}$ at time (2n + 3) and from vertex $v_{n-2,n-2}$ at time (2n – 4 + 10(k+1))

- sends empty message E to vertex $v_{n-2,n-2}$ at time (2n + 2) and to vertex $v_{n,n-2}$ at time (2n – 3 + 10(k+1))

- **Vertex $v_{n-1,n-1}$**

- receives odd numbered message (1+4k), $k \geq 0$ from vertex $v_{n-1,n}$ at time (2n + 10k)

- receives odd numbered message (3+4k), $k \geq 0$ from vertex $v_{n-2,n-1}$ at time (2n + 3 + 10k)

- sends odd numbered message (1+4k), $k \geq 0$ to vertex $v_{n-2,n-1}$ at time (2n + 7 + 10k)

- sends odd numbered message (3+4k), $k \geq 0$ to vertex $v_{n,n-1}$ at time (2n + 4 + 10k)

- receives even numbered message (2+4k), $k \geq 0$ from vertex $v_{n,n-1}$ at time (2n + 8 + 10k)

- receives even numbered message (4+4k), $k \geq 0$ from vertex $v_{n-1,n-2}$ at time (2n + 5 + 10k)

- sends even numbered message (2+4k), $k \geq 0$ to vertex $v_{n-1,n-2}$ at time (2n + 9 + 10k)

- sends even numbered message (4+4k), $k \geq 0$ to vertex $v_{n-1,n}$ at time (2n + 6 + 10k)

- **Vertex $v_{3,n-2}$**

- receives odd numbered message (1+4k), $k \geq 0$ from vertex $v_{2,n-2}$ at time (n + 10k)

- receives odd numbered message (3+4k), $k \geq 0$ from vertex $v_{4,n-2}$ at time (n + 17 + 10k)

- sends odd numbered message (1+4k), $k \geq 0$ to vertex $v_{4,n-2}$ at time (n + 1 + 10k) and to vertex $v_{3,n-3}$ at time (n + 2 + 10k)

- receives even numbered message (2+4k), $k \geq 0$ from vertex $v_{3,n-1}$ at time (n + 15 + 10k)

- receives even numbered message (4+4k), $k \geq 0$ from vertex $v_{3,n-3}$ at time (n + 8 + 10k)

- sends even numbered message (4+4k), $k \geq 0$ to vertex $v_{3,n-1}$ at time (n + 9 + 10k) and to vertex $v_{2,n-2}$ at time (n + 14 + 10k)

- receives empty message E from vertex $v_{4,n-2}$ at time (n + 7) and from vertex $v_{2,n-2}$ at time (n + 10(k+1))

- sends empty message E to vertex $v_{2,n-2}$ at time (n + 4) and to vertex $v_{4,n-2}$ at time (n + 1 + 10(k+1))

- **Vertex $v_{3,n-1}$**

- receives odd numbered message (1+4k), $k \geq 0$ from vertex $v_{3,n}$ at time (n + 3 + 10k)

- receives odd numbered message (3+4k), $k \geq 0$ from vertex $v_{2,n-1}$ at time (n + 7 + 10k)

- sends odd numbered message (1+4k), $k \geq 0$ to vertex $v_{2,n-1}$ at time (n + 4 + 10k)

- sends odd numbered message (3+4k), $k \geq 0$ to vertex $v_{4,n-1}$ at time (n + 8 + 10k)

- receives even numbered message (2+4k), $k \geq 0$ from vertex $v_{4,n-1}$ at time (n + 12 + 10k)

- receives even numbered message (4+4k), $k \geq 0$ from vertex $v_{3,n-2}$ at time (n + 9 + 10k)

- sends even numbered message (2+4k), $k \geq 0$ to vertex $v_{3,n-2}$ at time (n + 15 + 10k)

- sends even numbered message (4+4k), $k \geq 0$ to vertex $v_{3,n}$ at time (n + 10 + 10k)

- **Vertex $v_{i,n-1}$** $(4 \leq i \leq n-2)$

i is even

- receives odd numbered message (1+4k), $k \geq 0$ from vertex $v_{i+1,n-1}$ at time (i + n + 9 + 10k)

- receives odd numbered message (3+4k), $k \geq 0$ from vertex $v_{i-1,n-1}$ at time (i + n + 4 + 10k)

- sends odd numbered message (3+4k), $k \geq 0$ to vertex $v_{i+1,n-1}$ at time (i + n + 5 + 10k) and to vertex $v_{i,n-2}$ at time (i + n + 12 + 10k)

- receives even numbered message (2+4k), $k \geq 0$ from vertex $v_{i,n-2}$ at time (i + n + 10k)

- receives even numbered message (4+4k), $k \geq 0$ from vertex $v_{i,n}$ at time (i + n + 16 + 10k)

- sends even numbered message (2+4k), $k \geq 0$ to vertex $v_{i,n}$ at time (i + n + 1 + 10k) and to vertex $v_{i-1,n-1}$ at time (i + n + 8 + 10k)

- receives empty message E from vertex $v_{i,n}$ at time (i + n + 6) and from vertex $v_{i,n-2}$ at time (i + n + 10(k+1))

- sends empty message E to vertex $v_{i,n-2}$ at time (i + n + 2) and to vertex $v_{i,n}$ at time (i + n + 1 + 10(k+1))

i is odd

- receives odd numbered message (1+4k), $k \geq 0$ from vertex $v_{i,n}$ at time (i + n + 10k)

- receives odd numbered message (3+4k), $k \geq 0$ from vertex $v_{i-1,n-1}$ at time (i + n + 4 + 10k)

- sends odd numbered message (1+4k), $k \geq 0$ to vertex $v_{i-1,n-1}$ at time (i + n + 8 + 10k)

- sends odd numbered message (3+4k), $k \geq 0$ to vertex $v_{i+1,n-1}$ at time (i + n + 5 + 10k)

- receives even numbered message (2+4k), $k \geq 0$ from vertex $v_{i+1,n-1}$ at time (i + n + 9 + 10k)

- receives even numbered message (4+4k), $k \geq 0$ from vertex $v_{i,n-2}$ at time (i + n + 6 + 10k)

- sends even numbered message (2+4k), $k \geq 0$ to vertex $v_{i,n-2}$ at time (i + n + 12 + 10k)

- sends even numbered message (4+4k), $k \geq 0$ to vertex $v_{i,n}$ at time (i + n + 7 + 10k)

- **Vertex $v_{3,j}$** $(3 \leq j \leq n-3)$

j is even

- receives odd numbered message (1+4k), $k \geq 0$ from vertex $v_{2,j}$ at time (j + 2 + 10k)

- receives odd numbered message (3+4k), $k \geq 0$ from vertex $v_{4,j}$ at time (j + 19 + 10k)

- sends odd numbered message (1+4k), $k \geq 0$ to vertex $v_{4,j}$ at time (j + 3 + 10k) and to vertex $v_{3,j-1}$ at time (j + 4 + 10k)

- receives even numbered message (2+4k), $k \geq 0$ from vertex $v_{3,j+1}$ at time (j + 15 + 10k)

- receives even numbered message (4+4k), $k \geq 0$ from vertex $v_{3,j-1}$ at time (j + 10 + 10k) - sends even numbered message (4+4k), $k \geq 0$ to vertex $v_{3,j+1}$ at time (j + 11 + 10k) and to vertex $v_{2,j}$ at time (j + 16 + 10k)

- receives empty message E from vertex $v_{4,j}$ at time (j + 9) and from vertex $v_{2,j}$ at time (j + 2 + 10(k+1))

- sends empty message E to vertex $v_{2,j}$ at time (j + 6) and to vertex $v_{4,j}$ at time (j + 3 + 10(k+1))

j is odd

- receives odd numbered message (1+4k), $k \geq 0$ from vertex $v_{3,j+1}$ at time (j + 5 + 10k)

- receives odd numbered message (3+4k), $k \geq 0$ from vertex $v_{2,j}$ at time (j + 8 + 10k)

- sends odd numbered message (1+4k), $k \geq 0$ to vertex $v_{2,j}$ at time (j + 6 + 10k)

- sends odd numbered message (3+4k), $k \geq 0$ to vertex $v_{4,j}$ at time (j + 9 + 10k)

- receives even numbered message (2+4k), $k \geq 0$ from vertex $v_{4,j}$ at time (j + 13 + 10k)

- receives even numbered message (4+4k), $k \geq 0$ from vertex $v_{3,j-1}$ at time (j + 10 + 10k)

- sends even numbered message (2+4k), $k \geq 0$ to vertex $v_{3,j-1}$ at time (j + 14 + 10k) - sends even numbered message (4+4k), $k \geq 0$ to vertex $v_{3,j+1}$ at time (j + 11 + 10k)

- **Vertex $v_{i,3}$** ($4 \leq i \leq n-3$)

  i is even

- receives odd numbered message (1+4k), $k \geq 0$ from vertex $v_{i+1,3}$ at time (i + 13 + 10k)

- receives odd numbered message (3+4k), $k \geq 0$ from vertex $v_{i-1,3}$ at time (i + 8 + 10k)

- sends odd numbered message (3+4k), $k \geq 0$ to vertex $v_{i+1,3}$ at time (i + 9 + 10k) and to vertex $v_{i,2}$ at time (i + 16 + 10k)

- receives even numbered message (2+4k), $k \geq 0$ from vertex $v_{i,2}$ at time (i + 4 + 10k)

- receives even numbered message (4+4k), $k \geq 0$ from vertex $v_{i,4}$ at time (i + 21 + 10k)

- sends even numbered message (2+4k), $k \geq 0$ to vertex $v_{i,4}$ at time (i + 5 + 10k) and to vertex $v_{i-1,3}$ at time (i + 12 + 10k)

- receives empty message E from vertex $v_{i,4}$ at time (i + 11) and from vertex $v_{i,2}$ at time (i + 4 + 10(k+1))

- sends empty message E to vertex $v_{i,2}$ at time (i + 6) and to vertex $v_{i,4}$ at time (i + 5 + 10(k+1))

  i is odd

- receives odd numbered message (1+4k), $k \geq 0$ from vertex $v_{i,4}$ at time (i + 5 + 10k)

- receives odd numbered message (3+4k), $k \geq 0$ from vertex $v_{i-1,3}$ at time (i + 8 + 10k)

- sends odd numbered message (1+4k), $k \geq 0$ to vertex $v_{i-1,3}$ at time (i + 12 + 10k)

- sends odd numbered message (3+4k), $k \geq 0$ to vertex $v_{i+1,3}$ at time (i + 9 + 10k)

- receives even numbered message (2+4k), $k \geq 0$ from vertex $v_{i+1,3}$ at time (i + 13 + 10k)

- receives even numbered message (4+4k), $k \geq 0$ from vertex $v_{i,2}$ at time (i + 10 + 10k)

- sends even numbered message (2+4k), $k \geq 0$ to vertex $v_{i,2}$ at time (i + 16 + 10k)

- sends even numbered message (4+4k), $k \geq 0$ to vertex $v_{i,4}$ at time (i + 11 + 10k)

- **Vertex $v_{n-2,3}$**

- receives odd numbered message (1+4k), $k \geq 0$ from vertex $v_{n-1,3}$ at time (n + 15 + 10k)

- receives odd numbered message (3+4k), $k \geq 0$ from vertex $v_{n-3,3}$ at time (n + 6 + 10k)

- sends odd numbered message (3+4k), $k \geq 0$ to vertex $v_{n-1,3}$ at time (n + 7 + 10k) and to vertex $v_{n-2,2}$ at time (n + 14 + 10k)

- receives even numbered message (2+4k), $k \geq 0$ from vertex $v_{n-2,2}$ at time (n + 2 + 10k)

- receives even numbered message (4+4k), $k \geq 0$ from vertex $v_{n-2,4}$ at time (n + 19 + 10k)

- sends even numbered message (2+4k), $k \geq 0$ to vertex $v_{n-2,4}$ at time (n + 3 + 10k) and to vertex $v_{n-3,3}$ at time (n + 10 + 10k)

- receives empty message E from vertex $v_{n-2,4}$ at time (n + 9) and from vertex $v_{n-2,2}$ at time (n + 2 + 10(k+1))

- sends empty message E to vertex $v_{n-2,2}$ at time (n + 4) and to vertex $v_{n-2,4}$ at time (n + 3 + 10(k+1))

- **Vertex $v_{n-2,j}$** $(4 \le j \le n-3)$

j is even

- receives odd numbered message (1+4k), $k \ge 0$ from vertex $v_{n-3,j}$ at time (n + j - 3 + 10k)

- receives odd numbered message (3+4k), $k \ge 0$ from vertex $v_{n-2,j+1}$ at time (n + j + 6 + 10k)

- sends odd numbered message (1+4k), $k \ge 0$ to vertex $v_{n-1,j}$ at time (n + j - 2 + 10k)

- sends odd numbered message (3+4k), $k \ge 0$ to vertex $v_{n-3,j}$ at time (n + j + 13 + 10k)

- receives even numbered message (2+4k), $k \ge 0$ from vertex $v_{n-2,j-1}$ at time (n + j - 1 + 10k)

- receives even numbered message (4+4k), $k \ge 0$ from vertex $v_{n-1,j}$ at time (n + j + 12 + 10k)

- sends even numbered message (2+4k), $k \ge 0$ to vertex $v_{n-2,j+1}$ at time (n + j + 10k)

- sends even numbered message (4+4k), $k \ge 0$ to vertex $v_{n-2,j-1}$ at time (n + j + 15 + 10k)

- receives empty message E from vertex $v_{n-1,j}$ at time (n + j + 2), from vertex $v_{n-3,j}$ at time (n + j - 3 + 10(k+1)) and from vertex $v_{n-2,j-1}$ at time (n + j - 1 + 10(k+1))

- sends empty message E to vertex $v_{n-3,j}$ at time (n + j + 3), to vertex $v_{n-2,j-1}$ at time (n + j + 5) and to vertex $v_{n-1,j}$ at time (n + j - 2 + 10(k+1))

j is odd

- receives odd numbered message (1+4k), $k \ge 0$ from vertex $v_{n-1,j}$ at time (n + j + 12 + 10k)

- receives odd numbered message (3+4k), $k \ge 0$ from vertex $v_{n-3,j}$ at time (n + j + 3 + 10k)

- sends odd numbered message (3+4k), $k \ge 0$ to vertex $v_{n-1,j}$ at time (n + j + 4 + 10k) and to vertex $v_{n-2,j-1}$ at time (n + j + 5 + 10k)

- receives even numbered message (2+4k), $k \ge 0$ from vertex $v_{n-2,j-1}$ at time (n + j - 1 + 10k)

- receives even numbered message (4+4k), $k \ge 0$ from vertex $v_{n-2,j+1}$ at time (n + j + 16 + 10k)

- sends even numbered message (2+4k), $k \ge 0$ to vertex $v_{n-2,j+1}$ at time (n + j + 10k) and to vertex $v_{n-3,j}$ at time (n + j + 7 + 10k)

- receives empty message E from vertex $v_{n-2,j+1}$ at time (n + j + 6)

- sends empty message E to vertex $v_{n-2,j+1}$ at time (n + j + 10(k+1))

- **Vertex $v_{i,n-2}$** $(4 \le i \le n-2)$

  i is even

- receives odd numbered message (1+4k), $k \ge 0$ from vertex $v_{i-1,n-2}$ at time (n + i - 3 + 10k)

- receives odd numbered message (3+4k), $k \ge 0$ from vertex $v_{i,n-1}$ at time (n + i + 12 + 10k)

- sends odd numbered message (1+4k), $k \ge 0$ to vertex $v_{i+1,n-2}$ at time (n + i - 2 + 10k)

- sends odd numbered message (3+4k), $k \ge 0$ to vertex $v_{i-1,n-2}$ at time (n + i + 13 + 10k)

- receives even numbered message (2+4k), $k \ge 0$ from vertex $v_{i,n-3}$ at time (n + i - 1 + 10k)

- receives even numbered message (4+4k), $k \ge 0$ from vertex $v_{i+1,n-2}$ at time (n + i + 14 + 10k)

- sends even numbered message (2+4k), $k \ge 0$ to vertex $v_{i,n-1}$ at time (n + i + 10k)

- sends even numbered message (4+4k), $k \ge 0$ to vertex $v_{i,n-3}$ at time (n + i + 15 + 10k)

- receives empty message E from vertex $v_{i,n-1}$ at time (n + i + 2), from vertex $v_{i+1,n-2}$ at time (n + i + 4), from vertex $v_{i-1,n-2}$ at time (n + i - 3 + 10(k+1)) and from vertex $v_{i,n-3}$ at time (n + i - 1 + 10(k+1))

- sends empty message E to vertex $v_{i-1,n-2}$ at time (n + i + 3), to vertex $v_{i,n-3}$ at time (n + i + 5), to vertex $v_{i,n-1}$ at time (n + i + 10(k+1)) and to vertex $v_{i+1,n-2}$ at time (n + i – 2 + 10(k+1))

  i is odd

- receives odd numbered message (1+4k), $k \ge 0$ from vertex $v_{i-1,n-2}$ at time (n + i - 3 + 10k)

- receives odd numbered message (3+4k), $k \ge 0$ from vertex $v_{i+1,n-2}$ at time (n + i + 14 + 10k)

- sends odd numbered message (1+4k), $k \ge 0$ to vertex $v_{i+1,n-2}$ at time (n + i - 2 + 10k) and to vertex $v_{i,n-3}$ at time (n + i -1 + 10k)

- receives even numbered message (2+4k), $k \geq 0$ from vertex $v_{i,n-1}$ at time (n + i + 12 + 10k)

- receives even numbered message (4+4k), $k \geq 0$ from vertex $v_{i,n-3}$ at time (n + i + 5 + 10k)

- sends even numbered message (4+4k), $k \geq 0$ to vertex $v_{i,n-1}$ at time (n + i + 6 + 10k) and to vertex $v_{i-1,n-2}$ at time (n + i + 13 + 10k)

- receives empty message E from vertex $v_{i+1,n-2}$ at time (n + i + 4) and from vertex $v_{i-1,n-2}$ at time (n + i – 3 + 10(k+1))

- sends empty message E to vertex $v_{i-1,n-2}$ at time (n + i + 3) and to vertex $v_{i+1,n-2}$ at time (n + i – 2 + 10(k+1))

- **Vertex $v_{i,j}$**

$(4 \leq i \leq n-3)$ and $(4 \leq j \leq n-3)$

i odd, j odd:

- receives odd numbered message (1+4k), $k \geq 0$ from vertex $v_{i,j+1}$ at time (i + j + 2 + 10k)

- receives odd numbered message (3+4k), $k \geq 0$ from vertex $v_{i-1,j}$ at time (i + j + 5 + 10k)

- sends odd numbered message (1+4k), $k \geq 0$ to vertex $v_{i-1,j}$ at time (i + j + 9 + 10k)

- sends odd numbered message (3+4k), $k \geq 0$ to vertex $v_{i+1,j}$ at time (i + j + 6 + 10k)

- receives even numbered message (2+4k), $k \geq 0$ from vertex $v_{i+1,j}$ at time (i + j + 10 + 10k)

- receives even numbered message (4+4k), $k \geq 0$ from vertex $v_{i,j-1}$ at time (i + j + 7 + 10k)

- sends even numbered message (2+4k), $k \geq 0$ to vertex $v_{i,j-1}$ at time (i + j + 11 + 10k)

- sends even numbered message (4+4k), $k \geq 0$ to vertex $v_{i,j+1}$ at time (i + j + 8 + 10k)

i odd, j even:

- receives odd numbered message (1+4k), $k \geq 0$ from vertex $v_{i-1,j}$ at time (i + j -1 + 10k)

- receives odd numbered message (3+4k), $k \geq 0$ from vertex $v_{i+1,j}$ at time (i + j + 16 + 10k)

- sends odd numbered message (1+4k), $k \geq 0$ to vertex $v_{i+1,j}$ at time (i + j + 10k) and to vertex $v_{i,j-1}$ at time (i + j + 1 +10k)

- receives even numbered message (2+4k), $k \geq 0$ from vertex $v_{i,j+1}$ at time (i + j + 12 + 10k)

- receives even numbered message (4+4k), $k \geq 0$ from vertex $v_{i,j-1}$ at time (i + j + 7+ 10k)

- sends even numbered message (4+4k), $k \geq 0$ to vertex $v_{i,j+1}$ at time (i + j + 8 + 10k) and to vertex $v_{i-1,j}$ at time (i + j + 15 +10k)

- receives empty message E from vertex $v_{i-1,j}$ at time (i + j - 1 + 10*(k+1)) and from vertex v_{i+1j} at time (i + j + 6)

- sends empty message E to vertex $v_{i+1,j}$ at time (i + j + 5) and to vertex $v_{i-1,j}$ at time (i + j + 10*(k+1))

i even, j odd:

- receives odd numbered message (1+4k), $k \geq 0$ from vertex $v_{i+1,j}$ at time (i + j + 10 + 10k)

- receives odd numbered message (3+4k), $k \geq 0$ from vertex $v_{i-1,j}$ at time (i + j + 5 + 10k)

- sends odd numbered message (3+4k), $k \geq 0$ to vertex $v_{i+1,j}$ at time (i + j + 6 + 10k) and to vertex $v_{i,j-1}$ at time (i + j +7 +10k)

- receives even numbered message (2+4k), $k \geq 0$ from vertex $v_{i,j-1}$ at time (i + j + 1 + 10k)

- receives even numbered message (4+4k), $k \geq 0$ from vertex $v_{i,j+1}$ at time (i + j + 18 + 10k)

- sends even numbered message (2+4k), $k \geq 0$ to vertex $v_{i,j+1}$ at time (i + j + 2 + 10k) and to vertex $v_{i-1,j}$ at time (i + j + 9 +10k)

- receives empty message E from vertex $v_{i,j+1}$ at time (i + j + 8)

- sends empty message E to vertex $v_{i,j+1}$ at time (i + j + 2 + 10*(k+1))

i even, j even:

- receives odd numbered message (1+4k), $k \geq 0$ from vertex $v_{i-1,j}$ at time (i + j -1 + 10k)

- receives odd numbered message (3+4k), k ≥ 0 from vertex $v_{i,j+1}$ at time (i + j + 8 + 10k)

- sends odd numbered message (1+4k), k ≥ 0 to vertex $v_{i+1,j}$ at time (i + j + 10k)

- sends odd numbered message (3+4k), k ≥ 0 to vertex $v_{i-1,j}$ at time (i + j + 15 + 10k)

- receives even numbered message (2+4k), k ≥ 0 from vertex $v_{i,j-1}$ at time (i + j + 1 + 10k)

- receives even numbered message (4+4k), k ≥ 0 from vertex $v_{i+1,j}$ at time (i + j + 16 + 10k)

- sends even numbered message (2+4k), k ≥ 0 to vertex $v_{i,j+1}$ at time (i + j + 2 + 10k)

- sends even numbered message (4+4k), k ≥ 0 to vertex $v_{i,j-1}$ at time (i + j + 17 + 10k)

- receives empty message E from vertex $v_{i-1,j}$ at time (i + j - 1 + 10*(k+1)), from vertex v_{i+1j} at time (i + j + 6) and from vertex $v_{i,j-1}$ at time (i + j + 1 + 10*(k+1))

- sends empty message E to vertex $v_{i-1,j}$ at time (i + j + 5), to vertex $v_{i+1,j}$ at time (i + j + 10*(k+1)) and to vertex $v_{i,j-1}$ at time (i + j + 7).

Total time of described algorithm that performs multiple message broadcasting in an even by even DOG grid G_{mxn} is (m + n + 2.5M + 4).

4.6.3 Multiple message broadcasting in an odd by odd DOG grid.

Let G_{nxn} be an *n* x *n* DOG grid graph. Let *n* be an odd integer, *n* > 1. The algorithm for the odd by odd grid is basically the same as for the even by even case with the exception of several vertices. The following are those vertices that behave differently.

- **Vertex $v_{1,n}$**

- receives odd numbered message (1+4k), k ≥ 0 from vertex $v_{1,n-1}$ at time (n - 1 + 10k)

- receives odd numbered message (3+4k), k ≥ 0 from vertex $v_{1,n-1}$ at time (n + 5 + 10k)

- sends odd numbered message (1+4k), k ≥ 0 to vertex $v_{2,n}$ at time (n + 10k)

- sends odd numbered message (3+4k), $k \geq 0$ to vertex $v_{2,n}$ at time (n + 6 + 10k)

- receives even numbered message (2+4k), $k \geq 0$ from vertex $v_{2,n}$ at time (n + 4 + 10k)

- receives even numbered message (4+4k), $k \geq 0$ from vertex $v_{2,n}$ at time (n + 22 + 10k)

- sends even numbered message (2+4k), $k \geq 0$ to vertex $v_{1,n-1}$ at time (n + 13 + 10k)

- receives empty message E_k, $k \geq 0$ from vertex $v_{2,n}$ at time (n + 2) and at time (n + 12) and from vertex $v_{1,n-1}$ at time (n + 5 + 10(k+1))

- sends empty message E_k, $k \geq 0$ to vertex $v_{1,n-1}$ at time (n + 1 + 10k) and at time (n + 3) and to vertex $v_{2,n}$ at time (n + 10(k+1)) and (n + 10(k+2))

- **Vertex $v_{n,1}$**

- receives odd numbered message (1+4k), $k \geq 0$ from vertex $v_{n,2}$ at time (n + 4 + 10k)

- receives odd numbered message (3+4k), $k \geq 0$ from vertex $v_{n,2}$ at time (n + 20 + 10k)

- sends odd numbered message (1+4k), $k \geq 0$ to vertex $v_{n-1,1}$ at time (n + 5 + 10k)

- receives even numbered message (2+4k), $k \geq 0$ from vertex $v_{n-1,1}$ at time (n + 1 + 10k)

- receives even numbered message (4+4k), $k \geq 0$ from vertex $v_{n-1,1}$ at time (n + 7 + 10k)

- sends even numbered message (2+4k), $k \geq 0$ to vertex $v_{n,2}$ at time (n + 2 + 10k)

- sends even numbered message (4+4k), $k \geq 0$ to vertex $v_{n,2}$ at time (n + 8 + 10k)

- receives empty message E, from vertex $v_{n,2}$ at time (n + 10)

- sends empty message E_k, $k \geq 0$ to vertex $v_{n-1,1}$ at time (n + 3 + 10k) and to vertex $v_{n,2}$ at time (n + 8 + 10(k+1))

- **Vertex $v_{n,2}$**

- receives odd numbered message (1+4k), $k \geq 0$ from vertex $v_{n-1,2}$ at time (n + 1 + 10k)

- receives odd numbered message (3+4k), $k \geq 0$ from vertex $v_{n,3}$ at time (n + 15 + 10k)

- sends odd numbered message (1+4k), $k \geq 0$ to vertex $v_{n,1}$ at time (n + 4 + 10k)

- sends odd numbered message (3+4k), $k \geq 0$ to vertex $v_{n,1}$ at time (n + 20 + 10k)

- receives even numbered message (2+4k), $k \geq 0$ from vertex $v_{n,1}$ at time (n + 2 + 10k)

- receives even numbered message (4+4k), $k \geq 0$ from vertex $v_{n,1}$ at time (n + 8 + 10k)

- sends even numbered message (2+4k), $k \geq 0$ to vertex $v_{n,3}$ at time (n + 3 + 10k)

- sends even numbered message (4+4k), $k \geq 0$ to vertex $v_{n,3}$ at time (n + 9 + 10k) and to vertex $v_{n-1,2}$ at time (n + 16 + 10k)

- receives empty message E, from vertex $v_{n,3}$ at time (n + 5) and (n + 7 + 10k), from vertex $v_{n-1,2}$ at time (n + 1 + 10(k+1)) and from vertex $v_{n,1}$ at time (n + 8 + 10(k+1))

- sends empty message E_k, $k \geq 0$ to vertex $v_{n-1,2}$ at time (n + 6), to vertex $v_{n,1}$ at time (n + 10) and to vertex $v_{n,3}$ at time (n + 3 + 10(k+1))

- **Vertex $v_{n,j}$ $(3 \leq j \leq n-4)$**

  j is even

- receives odd numbered message (1+4k), $k \geq 0$ from vertex $v_{n-1,j}$ at time (n + j - 1 + 10k)

- receives odd numbered message (3+4k), $k \geq 0$ from vertex $v_{n,j+1}$ at time (n + j + 13 + 10k)

- sends odd numbered message (1+4k), $k \geq 0$ to vertex $v_{n,j-1}$ at time (n + j + 2 + 10k)

- receives even numbered message (2+4k), $k \geq 0$ from vertex $v_{n,j-1}$ at time (n + j + 10k)

- receives even numbered message (4+4k), $k \geq 0$ from vertex $v_{n,j-1}$ at time (n + j + 6 + 10k)

- sends even numbered message (2+4k), $k \geq 0$ to vertex $v_{n,j+1}$ at time (n + j + 1 + 10k)

- sends even numbered message (4+4k), $k \geq 0$ to vertex $v_{n,j+1}$ at time (n + j + 7 + 10k) and to vertex $v_{n-1,j}$ at time (n + j + 14 + 10k)

- receives empty message E_k, $k \geq 0$ from vertex $v_{n,j+1}$ at time (n + j + 3) and at time (n + j + 5 + 10k) and from vertex $v_{n-1,j}$ at time (n + j – 1 + 10(k+1))

- sends empty message E_k, $k \geq 0$ to vertex $v_{n-1,j}$ at time (n + j + 4), to vertex $v_{n,j-1}$ at time (n + j + 8 + 10k) and to vertex $v_{n,j+1}$ at time (n + j + 1 + 10(k+1))

j is odd

- receives odd numbered message (1+4k), $k \geq 0$ from vertex $v_{n,j+1}$ at time (n + j + 3 + 10k)

- receives odd numbered message (3+4k), $k \geq 0$ from vertex $v_{n-1,j}$ at time (n + j + 5 + 10k)

- sends odd numbered message (1+4k), $k \geq 0$ to vertex $v_{n-1,j}$ at time (n + j + 8 + 10k)

- sends odd numbered message (3+4k), $k \geq 0$ to vertex $v_{n,j-1}$ at time (n + j + 12 + 10k)

- receives even numbered message (2+4k), $k \geq 0$ from vertex $v_{n,j-1}$ at time (n + j + 10k)

- receives even numbered message (4+4k), $k \geq 0$ from vertex $v_{n,j-1}$ at time (n + j + 6 + 10k)

- sends even numbered message (2+4k), $k \geq 0$ to vertex $v_{n,j+1}$ at time (n + j + 1 + 10k)

- sends even numbered message (4+4k), $k \geq 0$ to vertex $v_{n,j+1}$ at time (n + j + 7 + 10k)

- receives empty message E_k, $k \geq 0$ from vertex $v_{n,j+1}$ at time (n + j + 9 + 10k) and from vertex $v_{n,j-1}$ at time (n + j + 10(k+1))

- sends empty message E_k, $k \geq 0$ to vertex $v_{n,j-1}$ at time (n + j + 2) and at time (n + j + 4 + 10k)

- **Vertex $v_{n,n-3}$**

- receives odd numbered message (1+4k), $k \geq 0$ from vertex $v_{n-1,n-3}$ at time (2n - 4 + 10k)

- receives odd numbered message (3+4k), $k \geq 0$ from vertex $v_{n,n-2}$ at time (2n + 10 + 10k)

- sends odd numbered message (1+4k), $k \geq 0$ to vertex $v_{n,n-4}$ at time (2n - 1 + 10k)

- receives even numbered message (2+4k), $k \geq 0$ from vertex $v_{n,n-4}$ at time (2n - 3 + 10k)

- receives even numbered message (4+4k), $k \geq 0$ from vertex $v_{n,n-4}$ at time (2n + 3 + 10k)

- sends even numbered message (2+4k), $k \geq 0$ to vertex $v_{n,n-2}$ at time (2n - 2 + 10k)

- sends even numbered message (4+4k), $k \geq 0$ to vertex $v_{n,n-2}$ at time (2n + 4 + 10k) and to vertex $v_{n-1,n-3}$ at time (2n + 12 + 10k)

- receives empty message E_k, $k \geq 0$ from vertex $v_{n,n-2}$ at time 2n and at time (2n + 1 + 10k) and from vertex $v_{n-1,n-3}$ at time (2n – 4 + 10(k+1))

- sends empty message E_k, $k \geq 0$ to vertex $v_{n-1,n-3}$ at time (2n + 2), to vertex $v_{n,n-4}$ at time (2n + 5 + 10k) and to vertex $v_{n,n-2}$ at time (2n – 2 + 10(k+1))

- **Vertex $v_{n,n-2}$**

- receives odd numbered message (1+4k), $k \geq 0$ from vertex $v_{n,n-1}$ at time (2n + 2 + 10k)

- receives odd numbered message (3+4k), $k \geq 0$ from vertex $v_{n-1,n-2}$ at time (2n + 3 + 10k)

- sends odd numbered message (1+4k), $k \geq 0$ to vertex $v_{n-1,n-2}$ at time (2n + 6 + 10k)

- sends odd numbered message (3+4k), $k \geq 0$ to vertex $v_{n,n-3}$ at time (2n + 10 + 10k)

- receives even numbered message (2+4k), $k \geq 0$ from vertex $v_{n,n-3}$ at time (2n - 2 + 10k)

- receives even numbered message (4+4k), $k \geq 0$ from vertex $v_{n,n-3}$ at time (2n + 4 + 10k)

- sends even numbered message (2+4k), $k \geq 0$ to vertex $v_{n,n-1}$ at time (2n - 1 + 10k)

- sends even numbered message (4+4k), $k \geq 0$ to vertex $v_{n,n-1}$ at time (2n + 5 + 10k)

- receives empty message E_k, $k \geq 0$ from vertex $v_{n,n-1}$ at time (2n - 3 + 10k) and from vertex $v_{n,n-3}$ at time (2n – 2 + 10(k+1))

- sends empty message E_k, $k \geq 0$ to vertex $v_{n,n-3}$ at time 2n and at time (2n + 1 + 10k)

- **Vertex $v_{n,n-1}$**

- receives odd numbered message (1+4k), $k \geq 0$ from vertex $v_{n-1,n-1}$ at time (2n - 2 + 10k)

- receives odd numbered message (3+4k), $k \geq 0$ from vertex $v_{n,n}$ at time (2n + 11 + 10k)

- sends odd numbered message (1+4k), $k \geq 0$ to vertex $v_{n,n-2}$ at time (2n + 2 + 10k)

- receives even numbered message (2+4k), $k \geq 0$ from vertex $v_{n,n-2}$ at time (2n - 1 + 10k)

- receives even numbered message (4+4k), $k \geq 0$ from vertex $v_{n,n-2}$ at time (2n + 5 + 10k)

- sends even numbered message (2+4k), $k \geq 0$ to vertex $v_{n,n}$ at time (2n + 10k)

- sends even numbered message (4+4k), $k \geq 0$ to vertex $v_{n,n}$ at time (2n + 6 + 10k) and to vertex $v_{n-1,n-1}$ at time (2n + 14 + 10k)

- receives empty message E_k, $k \geq 0$ from vertex $v_{n,n}$ at time (2n + 1) and at time (2n + 3 + 10k) and from vertex $v_{n-1,n-1}$ at time (2n – 2 + 10(k+1))

- sends empty message E_k, $k \geq 0$ to vertex $v_{n-1,n-1}$ at time (2n + 4), to vertex $v_{n,n-2}$ at time (2n - 3 + 10k) and to vertex $v_{n,n}$ at time (2n + 10(k+1))

- **Vertex $v_{n,n}$**

- receives odd numbered message (1+4k), $k \geq 0$ from vertex $v_{n-1,n}$ at time (2n - 2 + 10k)

- receives odd numbered message (3+4k), $k \geq 0$ from vertex $v_{n-1,n}$ at time (2n + 4 + 10k)

- sends odd numbered message (3+4k), $k \geq 0$ to vertex $v_{n,n-1}$ at time (2n + 11 + 10k)

- receives even numbered message (2+4k), $k \geq 0$ from vertex $v_{n,n-1}$ at time (2n + 10k)

- receives even numbered message (4+4k), $k \geq 0$ from vertex $v_{n,n-1}$ at time (2n + 6 + 10k)

- sends even numbered message (4+4k), $k \geq 0$ to vertex $v_{n-1,n}$ at time (2n + 12 + 10k)

- receives empty message E_k, $k \geq 0$ from vertex $v_{n,n-1}$ at time (2n + 10(k+1)) and from vertex $v_{n-1,n}$ at time (2n – 2 + 10(k+1))

- sends empty message E_k, $k \geq 0$ to vertex $v_{n,n-1}$ at time (2n + 1), (2n + 3 + 10k) and to vertex $v_{n-1,n}$ at time (2n + 2) and (2n + 5 + 10k)

- **Vertex $v_{2,n}$**

- receives odd numbered message (1+4k), $k \geq 0$ from vertex $v_{1,n}$ at time (n + 10k)

- receives odd numbered message (3+4k), $k \geq 0$ from vertex $v_{1,n}$ at time (n + 6 + 10k)

- sends odd numbered message (1+4k), $k \geq 0$ to vertex $v_{3,n}$ at time (n + 1 + 10k)

- sends odd numbered message (3+4k), $k \geq 0$ to vertex $v_{3,n}$ at time (n + 7 + 10k) and to vertex $v_{2,n-1}$ at time (n + 8 + 10k)

- receives even numbered message (2+4k), $k \geq 0$ from vertex $v_{2,n-1}$ at time (n + 3 + 10k)

- receives even numbered message (4+4k), $k \geq 0$ from vertex $v_{3,n}$ at time (n + 15 + 10k)

- sends even numbered message (2+4k), $k \geq 0$ to vertex $v_{1,n}$ at time (n + 4 + 10k)

- sends even numbered message (4+4k), $k \geq 0$ to vertex $v_{1,n}$ at time (n + 22 + 10k)

- receives empty message E_k, $k \geq 0$ from vertex $v_{3,n}$ at time (n + 5) and at time (n + 9 + 10k), from vertex $v_{1,n}$ at time (n + 10(k+1)) and (n + 10(k+2))

- sends empty message E_k, $k \geq 0$ to vertex $v_{1,n}$ at time (n + 2) and (n + 12) and to vertex $v_{3,n}$ at time (n + 1 + 10(k+1))

- **Vertex $v_{i,n}$ $(3 \leq i \leq n-2)$**

  i is even

- receives odd numbered message (1+4k), $k \geq 0$ from vertex $v_{i-1,n}$ at time (i + n -2 + 10k)

- receives odd numbered message (3+4k), $k \geq 0$ from vertex $v_{i-1,n}$ at time (i + n + 4 + 10k)

- sends odd numbered message (1+4k), $k \geq 0$ to vertex $v_{i+1,n}$ at time (i + n - 1 + 10k)

- sends odd numbered message (3+4k), $k \geq 0$ to vertex $v_{i+1,n}$ at time (i + n + 5 + 10k) and to vertex $v_{i,n-1}$ at time (i + n + 6 + 10k)

- receives even numbered message (2+4k), $k \geq 0$ from vertex $v_{i,n-1}$ at time (i + n + 1 + 10k)

- receives even numbered message (4+4k), $k \geq 0$ from vertex $v_{i+1,n}$ at time (i + n + 13 + 10k)

- sends even numbered message (2+4k), $k \geq 0$ to vertex $v_{i-1,n}$ at time (i + n + 2 + 10k)

- receives empty message E_k, $k \geq 0$ from vertex $v_{i+1,n}$ at time (i + n + 3) and at time (i + n + 7 + 10k)

- sends empty message E_k, $k \geq 0$ to vertex $v_{i-1,n}$ at time (i + n + 10k) and to vertex $v_{i+1,n}$ at time (i + n – 1 + 10(k+1))

i is odd

- receives odd numbered message (1+4k), $k \geq 0$ from vertex $v_{i-1,n}$ at time (i + n -2 + 10k)

- receives odd numbered message (3+4k), $k \geq 0$ from vertex $v_{i-1,n}$ at time (i + n + 4 + 10k)

- sends odd numbered message (1+4k), $k \geq 0$ to vertex $v_{i+1,n}$ at time (i + n - 1 + 10k)

- sends odd numbered message (3+4k), $k \geq 0$ to vertex $v_{i+1,n}$ at time (i + n + 5 + 10k)

- receives even numbered message (2+4k), $k \geq 0$ from vertex $v_{i+1,n}$ at time (i + n + 3 + 10k)

- receives even numbered message (4+4k), $k \geq 0$ from vertex $v_{i,n-1}$ at time (i + n + 7 + 10k)

- sends even numbered message (2+4k), $k \geq 0$ to vertex $v_{i,n-1}$ at time (i + n + 10 + 10k)

- sends even numbered message (4+4k), $k \geq 0$ to vertex $v_{i-1,n}$ at time (i + n + 12 + 10k)

- receives empty message E_k, $k \geq 0$ from vertex $v_{i+1,n}$ at time (i + n + 1 + 10k) and from vertex $v_{i-1,n}$ at time (i + n - 2 + 10(k+1))

- sends empty message E_k, $k \geq 0$ to vertex $v_{i-1,n}$ at time (i + n + 2) and at time (i + n + 6 + 10k)

- **Vertex $v_{n-1,n}$**

- receives odd numbered message (1+4k), $k \geq 0$ from vertex $v_{n-2,n}$ at time (2n - 3 + 10k)

- receives odd numbered message (3+4k), $k \geq 0$ from vertex $v_{n-2,n}$ at time (2n + 3 + 10k)

- sends odd numbered message (1+4k), $k \geq 0$ to vertex $v_{n,n}$ at time (2n - 2 + 10k)

- sends odd numbered message (3+4k), $k \geq 0$ to vertex $v_{n,n}$ at time (2n + 4 + 10k) and to vertex $v_{n-1,n-1}$ at time (2n + 6 + 10k)

- receives even numbered message (2+4k), $k \geq 0$ from vertex $v_{n-1,n-1}$ at time (2n + 10k)

- receives even numbered message (4+4k), $k \geq 0$ from vertex $v_{n,n}$ at time (2n + 12 + 10k)

- sends even numbered message (2+4k), $k \geq 0$ to vertex $v_{n-2,n}$ at time (2n + 1 + 10k)

- receives empty message E_k, $k \geq 0$ from vertex $v_{n,n}$ at time (2n + 2) and at time (2n + 5 + 10k)

- sends empty message E_k, $k \geq 0$ to vertex $v_{n-2,n}$ at time (2n - 1 + 10k) and to vertex $v_{n,n}$ at time (2n - 2 + 10(k+1))

- **Vertex $v_{2,n-2}$**

- receives odd numbered message (1+4k), $k \geq 0$ from vertex $v_{3,n-2}$ at time (n + 13 + 10k)

- receives odd numbered message (3+4k), $k \geq 0$ from vertex $v_{1,n-2}$ at time (n + 5 + 10k)

- sends odd numbered message (3+4k), $k \geq 0$ to vertex $v_{3,n-2}$ at time (n + 6 + 10k) and to vertex $v_{2,n-3}$ at time (n + 7 + 10k)

- receives even numbered message (2+4k), $k \geq 0$ from vertex $v_{2,n-3}$ at time (n + 1 + 10k)

- receives even numbered message (4+4k), $k \geq 0$ from vertex $v_{2,n-1}$ at time (n + 19 + 10k)

- sends even numbered message (2+4k), $k \geq 0$ to vertex $v_{2,n-1}$ at time (n + 2 + 10k) and to vertex $v_{1,n-2}$ at time (n + 10 + 10k)

- receives empty message E from vertex $v_{2,n-1}$ at time (n + 9)

- sends empty message E_k, $k \geq 0$ to vertex $v_{2,n-1}$ at time (n + 2 + 10(k+1))

- **Vertex $v_{2,n-1}$**

- receives odd numbered message (1+4k), $k \geq 0$ from vertex $v_{1,n-1}$ at time (n + 10k)

- receives odd numbered message (3+4k), $k \geq 0$ from vertex $v_{2,n}$ at time (n + 8 + 10k)

- sends odd numbered message (1+4k), $k \geq 0$ to vertex $v_{3,n-1}$ at time (n + 1 + 10k)

- receives even numbered message (2+4k), $k \geq 0$ from vertex $v_{2,n-2}$ at time (n + 2 + 10k)

- receives even numbered message (4+4k), $k \geq 0$ from vertex $v_{3,n-1}$ at time (n + 15 + 10k)

- sends even numbered message (2+4k), $k \geq 0$ to vertex $v_{2,n}$ at time (n + 3 + 10k)

- sends even numbered message (4+4k), $k \geq 0$ to vertex $v_{2,n-2}$ at time (n + 19 + 10k) and to vertex $v_{1,n-1}$ at time (n + 17 + 10k)

- receives empty message E from vertex $v_{3,n-1}$ at time (n + 5), from vertex $v_{2,n-2}$ at time (n + 2 + 10(k+1)) and from vertex $v_{1,n-1}$ at time (n + 10(k+1))

- sends empty message E to vertex $v_{2,n-2}$ at time (n + 9), to vertex $v_{1,n-1}$ at time (n + 7) and to vertex $v_{3,n-1}$ at time (n + 1 + 10(k+1))

- **Vertex $v_{n-2,2}$**

- receives odd numbered message (1+4k), $k \geq 0$ from vertex $v_{n-3,2}$ at time (n – 1 + 10k)

- receives odd numbered message (3+4k), $k \geq 0$ from vertex $v_{n-1,2}$ at time (n + 24 + 10k)

- sends odd numbered message (1+4k), $k \geq 0$ to vertex $v_{n-1,2}$ at time (n + 10k) and to vertex $v_{n-2,1}$ at time (n + 2 + 10k)

- receives even numbered message (2+4k), $k \geq 0$ from vertex $v_{n-2,3}$ at time (n + 21 + 10k)

- receives even numbered message (4+4k), $k \geq 0$ from vertex $v_{n-2,1}$ at time (n + 7 + 10k)

- sends even numbered message (4+4k), $k \geq 0$ to vertex $v_{n-2,3}$ at time (n + 8 + 10k) and to vertex $v_{n-3,2}$ at time (n + 15 + 10k)

- receives empty message E from vertex $v_{n-1,2}$ at time (n + 4) and (n + 14), from vertex $v_{n-2,3}$ at time (n + 11) and from vertex $v_{n-3,2}$ at time (n – 1 + 10(k+1))

- sends empty message E to vertex $v_{n-3,2}$ at time (n + 5), to vertex $v_{n-2,3}$ at time (n + 8 + 10(k+1)) and to vertex $v_{n-1,2}$ at time (n + 10(k+1)) and (n + 10(k+2))

- **Vertex $v_{n-1,2}$**

- receives odd numbered message (1+4k), k ≥ 0 from vertex $v_{n-2,2}$ at time (n + 10k)

- receives odd numbered message (3+4k), k ≥ 0 from vertex $v_{n-1,3}$ at time (n + 15 + 10k)

- sends odd numbered message (1+4k), k ≥ 0 to vertex $v_{n,2}$ at time (n + 1 + 10k)

- sends odd numbered message (3+4k), k ≥ 0 to vertex $v_{n-1,1}$ at time (n + 19 + 10k) and to vertex $v_{n-2,2}$ at time (n + 24 + 10k)

- receives even numbered message (2+4k), k ≥ 0 from vertex $v_{n-1,1}$ at time (n + 2 + 10k)

- receives even numbered message (4+4k), k ≥ 0 from vertex $v_{n,2}$ at time (n + 16 + 10k)

- sends even numbered message (2+4k), k ≥ 0 to vertex $v_{n-1,3}$ at time (n + 3 + 10k)

- receives empty message E from vertex $v_{n,2}$ at time (n + 6), from vertex $v_{n-1,3}$ at time (n + 5), from vertex $v_{n-1,1}$ at time (n + 2 + 10(k+1)) and from vertex $v_{n-2,2}$ at time (n + 10(k+1)) and (n + 10(k+2))

- sends empty message E to vertex $v_{n-2,2}$ at time (n + 4) and (n + 14), to vertex $v_{n-1,1}$ at time (n + 9), to vertex $v_{n,2}$ at time (n + 1 + 10(k+1)) and to vertex $v_{n-1,3}$ at time (n + 3 + 10(k+1))

- **Vertex $v_{n-1,3}$**

- receives odd numbered message (1+4k), k ≥ 0 from vertex $v_{n,3}$ at time (n + 11 + 10k)

- receives odd numbered message (3+4k), k ≥ 0 from vertex $v_{n-2,3}$ at time (n + 7 + 10k)

- sends odd numbered message (3+4k), k ≥ 0 to vertex $v_{n,3}$ at time (n + 8 + 10k) and to vertex $v_{n-1,2}$ at time (n + 15 + 10k)

- receives even numbered message (2+4k), $k \geq 0$ from vertex $v_{n-1,2}$ at time (n + 3 + 10k)

- receives even numbered message (4+4k), $k \geq 0$ from vertex $v_{n-1,4}$ at time (n + 20 + 10k)

- sends even numbered message (2+4k), $k \geq 0$ to vertex $v_{n-1,4}$ at time (n + 4 + 10k) and to vertex $v_{n-2,3}$ at time (n + 12 + 10k)

- receives empty message E from vertex $v_{n-1,4}$ at time (n + 10) and from vertex $v_{n-1,2}$ at time (n + 3 + 10(k+1))

- sends empty message E to vertex $v_{n-1,2}$ at time (n + 5) and to vertex $v_{n-1,4}$ at time (n + 4 + 10(k+1))

- **Vertex $v_{n-1,j}$** $(4 \leq j \leq n - 4)$

  j is even

- receives odd numbered message (1+4k), $k \geq 0$ from vertex $v_{n-2,j}$ at time (n + j – 2 + 10k)

- receives odd numbered message (3+4k), $k \geq 0$ from vertex $v_{n-1,j+1}$ at time (n + j + 7 + 10k)

- sends odd numbered message (1+4k), $k \geq 0$ to vertex $v_{n,j}$ at time (n + j – 1 + 10k)

- sends odd numbered message (3+4k), $k \geq 0$ to vertex $v_{n-2,j}$ at time (n + j + 12 + 10k)

- receives even numbered message (2+4k), $k \geq 0$ from vertex $v_{n-1,j-1}$ at time (n + j + 10k)

- receives even numbered message (4+4k), $k \geq 0$ from vertex $v_{n,j}$ at time (n + j + 14 + 10k)

- sends even numbered message (2+4k), $k \geq 0$ to vertex $v_{n-1,j+1}$ at time (n + j + 1 + 10k) - sends even numbered message (4+4k), $k \geq 0$ to vertex $v_{n-1,j-1}$ at time (n + j + 16 + 10k)

- receives empty message E from vertex $v_{n,j}$ at time (n + j + 4), from vertex $v_{n-1,j-1}$ at time (n + j + 10(k+1)) and from vertex $v_{n-2,j}$ at time (n + j – 2 + 10(k+1))

- sends empty message E to vertex $v_{n-2,j}$ at time (n + j + 2), to vertex $v_{n-1,j-1}$ at time (n + j + 6), to vertex $v_{n,j}$ at time (n + j – 1 + 10(k+1))

j is odd

- receives odd numbered message (1+4k), $k \geq 0$ from vertex $v_{n,j}$ at time (n + j + 8 + 10k)

- receives odd numbered message (3+4k), $k \geq 0$ from vertex $v_{n-2,j}$ at time (n + j + 4 + 10k)

- sends odd numbered message (3+4k), $k \geq 0$ to vertex $v_{n-1,j-1}$ at time (n + j + 6 + 10k) and to vertex $v_{n,j}$ at time (n + j + 5 + 10k)

- receives even numbered message (2+4k), $k \geq 0$ from vertex $v_{n-1,j-1}$ at time (n + j + 10k)

- receives even numbered message (4+4k), $k \geq 0$ from vertex $v_{n-1,j+1}$ at time (n + j + 17 + 10k)

- sends even numbered message (2+4k), $k \geq 0$ to vertex $v_{n-1,j+1}$ at time (n + j + 1 + 10k) and to vertex $v_{n-2,j}$ at time (n + j + 2 + 10k)

- receives empty message E from vertex $v_{n-1,j+1}$ at time (n + j + 7)

- sends empty message E to vertex $v_{n-1,j+1}$ at time (n + j + 1 + 10(k+1))

- **Vertex $v_{n-1,n-3}$**

- receives odd numbered message (1+4k), $k \geq 0$ from vertex $v_{n-2,n-3}$ at time (2n – 5 + 10k)

- receives odd numbered message (3+4k), $k \geq 0$ from vertex $v_{n-1,n-2}$ at time (2n + 4 + 10k)

- sends odd numbered message (1+4k), $k \geq 0$ to vertex $v_{n,n-3}$ at time (2n - 4 + 10k)

- sends odd numbered message (3+4k), $k \geq 0$ to vertex $v_{n-2,n-3}$ at time (2n + 9 + 10k)

- receives even numbered message (2+4k), $k \geq 0$ from vertex $v_{n-1,n-4}$ at time (2n - 3 + 10k)

- receives even numbered message (4+4k), $k \geq 0$ from vertex $v_{n,n-3}$ at time (2n + 12 + 10k)

- sends even numbered message (2+4k), $k \geq 0$ to vertex $v_{n-1,n-2}$ at time (2n - 2 + 10k)

- sends even numbered message (4+4k), $k \geq 0$ to vertex $v_{n-1,n-4}$ at time (2n + 13 + 10k)

- receives empty message E from vertex $v_{n,n-3}$ at time (2n + 2), from vertex $v_{n-1,n-4}$ at time (2n – 3 + 10(k+1)) and from vertex $v_{n-2,n-3}$ at time (2n – 5 + 10(k+1))

- sends empty message E to vertex $v_{n-2,n-3}$ at time (2n - 1), to vertex $v_{n-1,n-4}$ at time (2n + 3) and to vertex $v_{n,n-3}$ at time (2n – 4 + 10(k+1))

- **Vertex $v_{n-1,n-2}$**

- receives odd numbered message (1+4k), k ≥ 0 from vertex $v_{n,n-2}$ at time (2n + 6 + 10k)

- receives odd numbered message (3+4k), k ≥ 0 from vertex $v_{n-2,n-2}$ at time (2n + 2 + 10k)

- sends odd numbered message (3+4k), k ≥ 0 to vertex $v_{n,n-2}$ at time (2n + 3 + 10k) and to vertex $v_{n-1,n-3}$ at time (2n + 4 + 10k)

- receives even numbered message (2+4k), k ≥ 0 from vertex $v_{n-1,n-3}$ at time (2n - 2 + 10k)

- receives even numbered message (4+4k), k ≥ 0 from vertex $v_{n-1,n-1}$ at time (2n + 21 + 10k)

- sends even numbered message (2+4k), k ≥ 0 to vertex $v_{n-1,n-1}$ at time (2n - 1 + 10k) and to vertex $v_{n-2,n-2}$ at time (2n + 10k)

- receives empty message E from vertex $v_{n-1,n-1}$ at time (2n + 1) and at time (2n + 11)

- sends empty message E to vertex $v_{n-1,n-1}$ at time (2n – 1 + 10(k+1)) and (2n – 1 + 10(k+2))

- **Vertex $v_{n-1,n-1}$**

- receives odd numbered message (1+4k), k ≥ 0 from vertex $v_{n-2,n-1}$ at time (2n – 3 + 10k)

- receives odd numbered message (3+4k), k ≥ 0 from vertex $v_{n-1,n}$ at time (2n + 6 + 10k)

- sends odd numbered message (1+4k), k ≥ 0 to vertex $v_{n,n-1}$ at time (2n – 2 + 10k)

- sends odd numbered message (3+4k), k ≥ 0 to vertex $v_{n-2,n-1}$ at time (2n + 13 + 10k)

- receives even numbered message (2+4k), k ≥ 0 from vertex $v_{n-1,n-2}$ at time (2n – 1 + 10k)

- receives even numbered message (4+4k), k ≥ 0 from vertex $v_{n,n-1}$ at time (2n + 14 + 10k)

- sends even numbered message (2+4k), k ≥ 0 to vertex $v_{n-1,n}$ at time (2n + 10k)

- sends even numbered message (4+4k), $k \geq 0$ to vertex $v_{n-1,n-2}$ at time (2n + 21 + 10k)

- receives empty message E from vertex $v_{n,n-1}$ at time (2n + 4), from vertex $v_{n-1,n-2}$ at time (2n – 1 + 10(k+1)) and (2n – 1 + 10(k+2)) and from vertex $v_{n-2,n-1}$ at time (2n – 3 + 10(k+1))

- sends empty message E to vertex $v_{n-1,n-2}$ at time (2n + 1) and at time (2n + 11), to vertex $v_{n-2,n-1}$ at time (2n + 3) and to vertex $v_{n,n-1}$ at time (2n – 2 + 10(k+1))

- **Vertex $v_{3,n-1}$**

- receives odd numbered message (1+4k), $k \geq 0$ from vertex $v_{2,n-1}$ at time (n + 1 + 10k)

- receives odd numbered message (3+4k), $k \geq 0$ from vertex $v_{4,n-1}$ at time (n + 18 + 10k)

- sends odd numbered message (1+4k), $k \geq 0$ to vertex $v_{4,n-1}$ at time (n + 2 + 10k) and to vertex $v_{3,n-2}$ at time (n + 4 + 10k)

- receives even numbered message (2+4k), $k \geq 0$ from vertex $v_{3,n}$ at time (n + 13 + 10k)

- receives even numbered message (4+4k), $k \geq 0$ from vertex $v_{3,n-2}$ at time (n + 9 + 10k)

- sends even numbered message (4+4k), $k \geq 0$ to vertex $v_{2,n-1}$ at time (n + 15 + 10k) and to vertex $v_{3,n}$ at time (n + 10 + 10k)

- receives empty message E from vertex $v_{4,n-1}$ at time (n + 8) and from vertex $v_{2,n-1}$ at time (n + 1 + 10(k+1))

- sends empty message E to vertex $v_{2,n-1}$ at time (n + 5) and to vertex $v_{4,n-1}$ at time (n + 2 + 10(k+1))

- **Vertex $v_{i,n-1}$** $(4 \leq i \leq n-2)$

  i is even

- receives odd numbered message (1+4k), $k \geq 0$ from vertex $v_{i-1,n-1}$ at time (i + n – 2 + 10k)

- receives odd numbered message (3+4k), $k \geq 0$ from vertex $v_{i,n}$ at time (i + n + 6 + 10k)

- sends odd numbered message (1+4k), $k \geq 0$ to vertex $v_{i+1,n-1}$ at time (i + n – 1 + 10k)

- sends odd numbered message (3+4k), $k \geq 0$ to vertex $v_{i-1,n-1}$ at time (i + n + 14 + 10k)

- receives even numbered message (2+4k), $k \geq 0$ from vertex $v_{i,n-2}$ at time (i + n + 10k)

- receives even numbered message (4+4k), $k \geq 0$ from vertex $v_{i+1,n-1}$ at time (i + n + 15 + 10k)

- sends even numbered message (2+4k), $k \geq 0$ to vertex $v_{i,n}$ at time (i + n + 1 + 10k)

- sends even numbered message (4+4k), $k \geq 0$ to vertex $v_{i,n-2}$ at time (i + n + 22 + 10k)

- receives empty message E from vertex $v_{i+1,n-1}$ at time (i + n + 5), from vertex $v_{i-1,n-1}$ at time (i + n – 2 + 10(k+1)) and from vertex $v_{i,n-2}$ at time (i + n + 10(k+1)) and (i + n + 10(k+2))

- sends empty message E to vertex $v_{i-1,n-1}$ at time (i + n + 4), to vertex $v_{i,n-2}$ at time (i + n + 2) and at time (i + n + 12) and to vertex $v_{i+1,n-1}$ at time (i + n – 1 + 10(k+1))

i is odd

- receives odd numbered message (1+4k), $k \geq 0$ from vertex $v_{i-1,n-1}$ at time (i + n – 2 + 10k)

- receives odd numbered message (3+4k), $k \geq 0$ from vertex $v_{i+1,n-1}$ at time (i + n + 15 + 10k)

- sends odd numbered message (1+4k), $k \geq 0$ to vertex $v_{i+1,n-1}$ at time (i + n – 1 + 10k) and to vertex $v_{i,n-2}$ at time (i + n + 1 + 10k)

- receives even numbered message (2+4k), $k \geq 0$ from vertex $v_{i,n}$ at time (i + n + 10 + 10k)

- receives even numbered message (4+4k), $k \geq 0$ from vertex $v_{i,n-2}$ at time (i + n + 6 + 10k)

- sends even numbered message (4+4k), $k \geq 0$ to vertex $v_{i-1,n-1}$ at time (i + n + 14 + 10k) and to vertex $v_{i,n}$ at time (i + n + 7 + 10k)

- receives empty message E from vertex $v_{i+1,n-1}$ at time (i + n + 5) and from vertex $v_{i-1,n-1}$ at time (i + n – 2 + 10(k+1))

- sends empty message E to vertex $v_{i-1,n-1}$ at time (i + n + 4) and to vertex $v_{i+1,n-1}$ at time (i + n – 1 + 10(k+1))

- **Vertex $v_{n-2,3}$**

- receives odd numbered message (1+4k), k ≥ 0 from vertex $v_{n-2,4}$ at time (n + 3 + 10k)

- receives odd numbered message (3+4k), k ≥ 0 from vertex $v_{n-3,3}$ at time (n + 6 + 10k)

- sends odd numbered message (1+4k), k ≥ 0 to vertex $v_{n-3,3}$ at time (n + 10 + 10k)

- sends odd numbered message (3+4k), k ≥ 0 to vertex $v_{n-1,3}$ at time (n + 7 + 10k)

- receives even numbered message (2+4k), k ≥ 0 from vertex $v_{n-1,3}$ at time (n + 12 + 10k)

- receives even numbered message (4+4k), k ≥ 0 from vertex $v_{n-2,2}$ at time (n + 8 + 10k)

- sends even numbered message (2+4k), k ≥ 0 to vertex $v_{n-2,2}$ at time (n + 21 + 10k)

- sends even numbered message (4+4k), k ≥ 0 to vertex $v_{n-2,4}$ at time (n + 9 + 10k)

- receives empty message E from vertex $v_{n-2,2}$ at time (n + 8 + 10(k+1))

- sends empty message E to vertex $v_{n-2,2}$ at time (n + 11)

- **Vertex $v_{n-2,j}$** (4 ≤ j ≤ n-3)

  j is even

- receives odd numbered message (1+4k), k ≥ 0 from vertex $v_{n-3,j}$ at time (n + j - 3 + 10k)

- receives odd numbered message (3+4k), k ≥ 0 from vertex $v_{n-1,j}$ at time (n + j + 12 + 10k)

- sends odd numbered message (1+4k), k ≥ 0 to vertex $v_{n-1,j}$ at time (n + j - 2 + 10k) and to vertex $v_{n-2,j-1}$ at time (n + j – 1 + 10k)

- receives even numbered message (2+4k), k ≥ 0 from vertex $v_{n-2,j+1}$ at time (n + j + 10 + 10k)

- receives even numbered message (4+4k), k ≥ 0 from vertex $v_{n-2,j-1}$ at time (n + j + 5 + 10k)

- sends even numbered message (4+4k), $k \geq 0$ to vertex $v_{n-2,j+1}$ at time (n + j + 6 + 10k) and to vertex $v_{n-3,j}$ at time (n + j + 13 + 10k)

- receives empty message E from vertex $v_{n-1,j}$ at time (n + j + 2) and from vertex $v_{n-3,j}$ at time (n + j – 3 + 10(k+1))

- sends empty message E to vertex $v_{n-3,j}$ at time (n + j + 3) and to vertex $v_{n-1,j}$ at time (n + j – 2 + 10(k+1))

j is odd

- receives odd numbered message (1+4k), $k \geq 0$ from vertex $v_{n-2,j+1}$ at time (n + j + 10k)

- receives odd numbered message (3+4k), $k \geq 0$ from vertex $v_{n-3,j}$ at time (n + j + 3 + 10k)

- sends odd numbered message (1+4k), $k \geq 0$ to vertex $v_{n-3,j}$ at time (n + j + 7 + 10k)

- sends odd numbered message (3+4k), $k \geq 0$ to vertex $v_{n-1,j}$ at time (n + j + 4 + 10k)

- receives even numbered message (2+4k), $k \geq 0$ from vertex $v_{n-1,j}$ at time (n + j + 2 + 10k)

- receives even numbered message (4+4k), $k \geq 0$ from vertex $v_{n-2,j-1}$ at time (n + j + 5 + 10k)

- sends even numbered message (2+4k), $k \geq 0$ to vertex $v_{n-2,j-1}$ at time (n + j + 9 + 10k)

- sends even numbered message (4+4k), $k \geq 0$ to vertex $v_{n-2,j+1}$ at time (n + j + 6 + 10k)

- **Vertex $v_{3,n-2}$**

- receives odd numbered message (1+4k), $k \geq 0$ from vertex $v_{3,n-1}$ at time (n + 4 + 10k)

- receives odd numbered message (3+4k), $k \geq 0$ from vertex $v_{2,n-2}$ at time (n + 6 + 10k)

- sends odd numbered message (1+4k), $k \geq 0$ to vertex $v_{2,n-2}$ at time (n + 13 + 10k)

- sends odd numbered message (3+4k), $k \geq 0$ to vertex $v_{4,n-2}$ at time (n + 7 + 10k)

- receives even numbered message (2+4k), $k \geq 0$ from vertex $v_{4,n-2}$ at time (n + 11 + 10k)

- receives even numbered message (4+4k), $k \geq 0$ from vertex $v_{3,n-3}$ at time (n + 8 + 10k)

- sends even numbered message (2+4k), $k \geq 0$ to vertex $v_{3,n-3}$ at time (n + 12 + 10k)

- sends even numbered message (4+4k), $k \geq 0$ to vertex $v_{3,n-1}$ at time (n + 9 + 10k)

- **Vertex $v_{i,n-2}$** $(4 \leq i \leq n-3)$

i is even

- receives odd numbered message (1+4k), $k \geq 0$ from vertex $v_{i+1,n-2}$ at time (n + i + 8 + 10k)

- receives odd numbered message (3+4k), $k \geq 0$ from vertex $v_{i-1,n-2}$ at time (n + i + 3 + 10k)

- sends odd numbered message (3+4k), $k \geq 0$ to vertex $v_{i+1,n-2}$ at time (n + i + 4 + 10k) and to vertex $v_{i,n-3}$ at time (n + i + 5 + 10k)

- receives even numbered message (2+4k), $k \geq 0$ from vertex $v_{i,n-3}$ at time (n + i - 1 + 10k)

- receives even numbered message (4+4k), $k \geq 0$ from vertex $v_{i,n-1}$ at time (n + i + 22 + 10k)

- sends even numbered message (2+4k), $k \geq 0$ to vertex $v_{i,n-1}$ at time (n + i + 10k) and to vertex $v_{i-1,n-2}$ at time (n + i + 7 + 10k)

- receives empty message E from vertex $v_{i,n-1}$ at time (n + i + 2) and (n + i + 12)

- sends empty message E to vertex $v_{i,n-1}$ at time (n + i + 10(k+1)) and (n + i + 10(k+2))

i is odd

- receives odd numbered message (1+4k), $k \geq 0$ from vertex $v_{i,n-1}$ at time (n + i + 1 + 10k)

- receives odd numbered message (3+4k), $k \geq 0$ from vertex $v_{i-1,n-2}$ at time (n + i + 3 + 10k)

- sends odd numbered message (1+4k), $k \geq 0$ to vertex $v_{i-1,n-2}$ at time (n + i + 7 + 10k)

- sends odd numbered message (3+4k), $k \geq 0$ to vertex $v_{i+1,n-2}$ at time (n + i + 4 + 10k)

- receives even numbered message (2+4k), $k \geq 0$ from vertex $v_{i+1,n-2}$ at time (n + i + 8 + 10k)

- receives even numbered message (4+4k), $k \geq 0$ from vertex $v_{i,n-3}$ at time (n + i + 5 + 10k)

- sends even numbered message (2+4k), $k \geq 0$ to vertex $v_{i,n-3}$ at time (n + i + 9 + 10k)

- sends even numbered message (4+4k), $k \geq 0$ to vertex $v_{i,n-1}$ at time (n + i + 6 + 10k)

- **Vertex $v_{n-2,n-2}$**

- receives odd numbered message (1+4k), $k \geq 0$ from vertex $v_{n-2,n-1}$ at time (2n – 1 + 10k)

- receives odd numbered message (3+4k), $k \geq 0$ from vertex $v_{n-3,n-2}$ at time (2n + 1 + 10k)

- sends odd numbered message (1+4k), $k \geq 0$ to vertex $v_{n-3,n-2}$ at time (2n + 5 + 10k)

- sends odd numbered message (3+4k), $k \geq 0$ to vertex $v_{n-1,n-2}$ at time (2n + 2 + 10k)

- receives even numbered message (2+4k), $k \geq 0$ from vertex $v_{n-1,n-2}$ at time (2n + 10k)

- receives even numbered message (4+4k), $k \geq 0$ from vertex $v_{n-2,n-3}$ at time (2n + 3 + 10k)

- sends even numbered message (2+4k), $k \geq 0$ to vertex $v_{n-2,n-3}$ at time (2n + 7 + 10k)

- sends even numbered message (4+4k), $k \geq 0$ to vertex $v_{n-2,n-1}$ at time (2n + 4 + 10k)

Figure 4.4.3.1 shows the path of the four messages in the 9x9 DOG grid. Edge labels correspond to message number and time it is transmitted.

Total time of described algorithm that performs multiple message broadcasting in an odd by odd DOG grid G_{mxn} is $(m + n + 2.5M + 11)$.

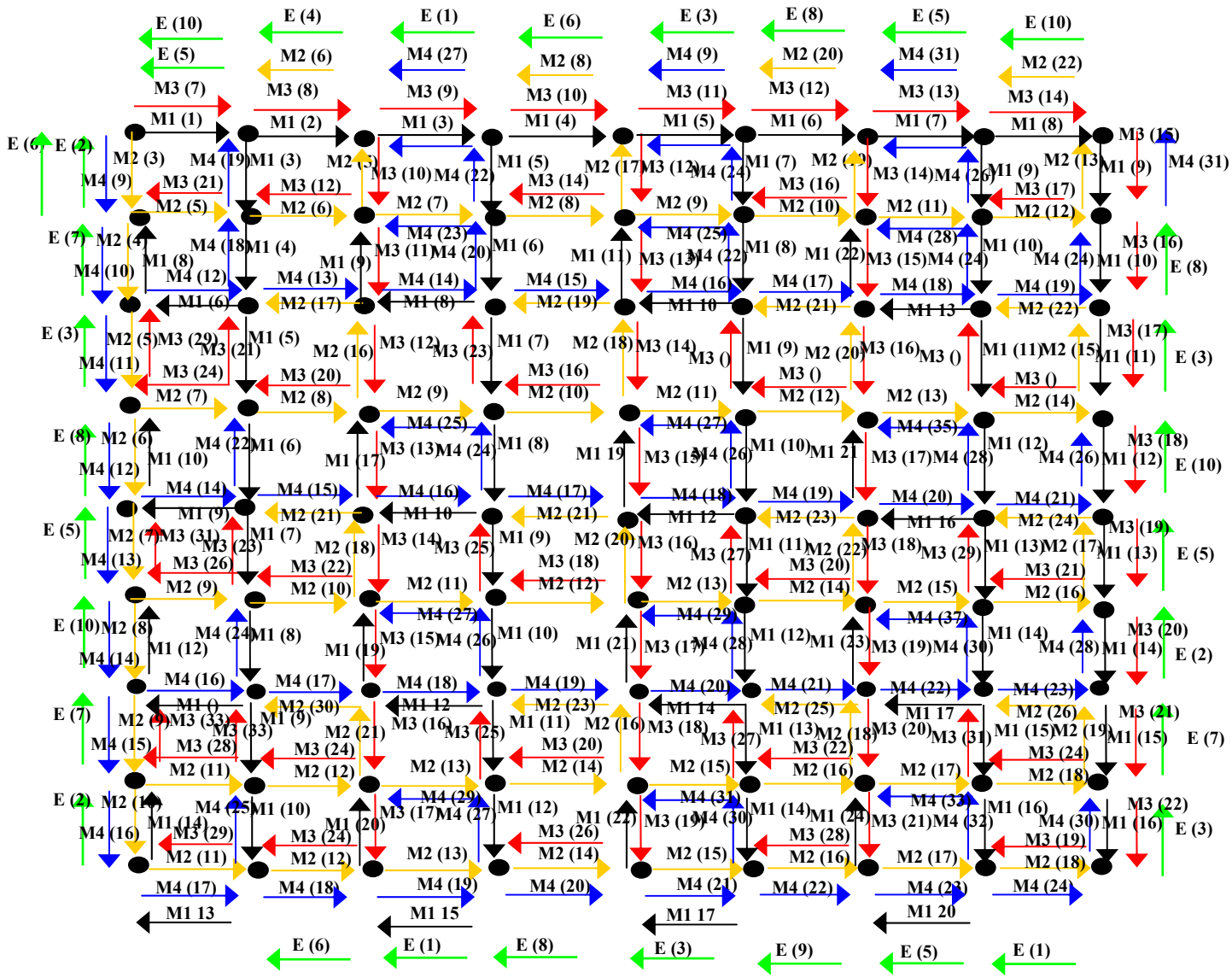


Figure 4.6.3.1 Broadcasting four messages in 9x9 DOG grid

4.6.4 Multiple message broadcasting in an arbitrary size DOG grid.

Algorithm for *n x n* grid works exactly the same for the case *m x n* ($m \neq n$) where *m* and *n* are even or *m* and *n* are odd. In the above described rules one just need to replace *n* with *m* and *2n* with ($m + n$) accordingly.

Algorithms for even by even and odd by odd grids can be modified to do multiple message broadcasting in an arbitrary size DOG grid *m x n* where either *m* is odd, *n* is even or vice versa. Since the most vertices in any grid follow the same algorithm only vertices in the last three columns and three rows should be handled differently. For example, if grid has odd number of rows and even number of columns, then the last three rows' vertices follow the algorithm for odd by odd grid and the last three columns' vertices follow the algorithm for even by even grid. However, the lower right corner of the grid should be handled differently.

Let us look at 17x18 DOG grid. Figure 4.4.3.1 shows the lower right corner of the grid.

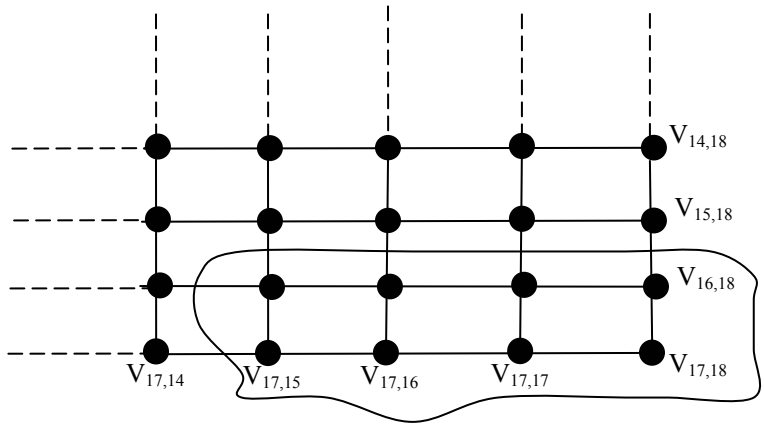


Figure 4.6.4.1 DOG grid G_{17x18}

For *m x n* grid where *m* is odd and *n* is even the following vertices need slightly different rules by which they send and receive messages.

- **Vertex $v_{m,n-3}$**

- receives odd numbered message (1+4k), $k \geq 0$ from vertex $v_{m,n-2}$ at time (m + n + 10k)

- receives odd numbered message (3+4k), $k \geq 0$ from vertex $v_{m-1,n-3}$ at time (m + n + 2 + 10k)

- sends odd numbered message (1+4k), $k \geq 0$ to vertex $v_{m-1,n-3}$ at time (m + n + 5 + 10k)

- sends odd numbered message (3+4k), $k \geq 0$ to vertex $v_{m,n-4}$ at time (n + m + 9 + 10k)

- receives even numbered message (2+4k), $k \geq 0$ from vertex $v_{m,n-4}$ at time (m + n – 3 + 10k)

- receives even numbered message (4+4k), $k \geq 0$ from vertex $v_{m,n-4}$ at time (n + m + 3 + 10k)

- sends even numbered message (2+4k), $k \geq 0$ to vertex $v_{m,n-2}$ at time (m + n – 2 + 10k)

- sends even numbered message (4+4k), $k \geq 0$ to vertex $v_{m,n-2}$ at time (n + m + 4 + 10k)

- receives empty message E from vertex $v_{m,n-2}$ at time (n + m + 6 + 10k) and from vertex $v_{m,n-4}$ at time (m + n – 3 + 10(k+1))

- sends empty message E to vertex $v_{m,n-4}$ at time (n + m – 1) and at time (n + m + 1 + 10k)

- **Vertex $v_{m,n-2}$**

- receives odd numbered message (1+4k), $k \geq 0$ from vertex $v_{m-1,n-2}$ at time (m + n – 3 + 10k)

- receives odd numbered message (3+4k), $k \geq 0$ from vertex $v_{m,n-1}$ at time (m + n + 11 + 10k)

- sends odd numbered message (1+4k), $k \geq 0$ to vertex $v_{m,n-3}$ at time (m + n + 10k)

- receives even numbered message (2+4k), $k \geq 0$ from vertex $v_{m,n-3}$ at time (m + n – 2 + 10k)

- receives even numbered message (4+4k), $k \geq 0$ from vertex $v_{m,n-3}$ at time (n + m + 4 + 10k)

- sends even numbered message (2+4k), $k \geq 0$ to vertex $v_{m,n-1}$ at time (m + n – 1 + 10k)

- sends even numbered message (4+4k), $k \geq 0$ to vertex $v_{m,n-1}$ at time (n + m + 5 + 10k) and to vertex $v_{m-1,n-2}$ at time (n + m + 13 + 10k)

- receives empty message E from vertex $v_{m,n-1}$ at time (n + m + 1) and at time (n + m + 2 + 10k) and from vertex $v_{m-1,n-2}$ at time (m + n – 3 + 10(k+1))

- sends empty message E to vertex $v_{m,n-3}$ at time (n + m + 6 +10k), to vertex $v_{m-1,n-2}$ at time (n + m + 3) and to vertex $v_{m,n-1}$ at time (m + n – 1 + 10(k+1))

- **Vertex $v_{m,n-1}$**

- receives odd numbered message (1+4k), k ≥ 0 from vertex $v_{m,n}$ at time (m + n + 3 + 10k)

- receives odd numbered message (3+4k), k ≥ 0 from vertex $v_{m-1,n-1}$ at time (m + n + 4 + 10k)

- sends odd numbered message (1+4k), k ≥ 0 to vertex $v_{m-1,n-1}$ at time (m + n + 8 + 10k)

- sends odd numbered message (3+4k), k ≥ 0 to vertex $v_{m,n-2}$ at time (n + m + 11 + 10k)

- receives even numbered message (2+4k), k ≥ 0 from vertex $v_{m,n-2}$ at time (m + n – 1 + 10k)

- receives even numbered message (4+4k), k ≥ 0 from vertex $v_{m,n-2}$ at time (n + m + 5 + 10k)

- sends even numbered message (2+4k), k ≥ 0 to vertex $v_{m,n}$ at time (m + n + 10k)

- sends even numbered message (4+4k), k ≥ 0 to vertex $v_{m,n}$ at time (n + m + 6 + 10k)

- receives empty message E from vertex $v_{m,n}$ at time (n + m + 7 + 10k) and from vertex $v_{m,n-2}$ at time (m + n – 1 + 10(k+1))

- sends empty message E to vertex $v_{m,n-2}$ at time (n + m + 1) and at time (n + m + 2 + 10k)

- **Vertex $v_{m,n}$**

- receives odd numbered message (1+4k), k ≥ 0 from vertex $v_{m-1,n}$ at time (m + n – 2 + 10k)

- receives odd numbered message (3+4k), k ≥ 0 from vertex $v_{m-1,n}$ at time (m + n + 4 + 10k)

- sends odd numbered message (1+4k), k ≥ 0 to vertex $v_{m,n-1}$ at time (m + n + 3 + 10k)

- receives even numbered message (2+4k), k ≥ 0 from vertex $v_{m,n-1}$ at time (m + n + 10k)

- receives even numbered message (4+4k), $k \geq 0$ from vertex $v_{m,n-1}$ at time (n + m + 6 + 10k)

- sends even numbered message (4+4k), $k \geq 0$ to vertex $v_{m-1,n}$ at time (n + m + 12 + 10k)

- receives empty message E from vertex $v_{m-1,n}$ at time (m + n – 2 + 10(k+1))

- sends empty message E to vertex $v_{m-1,n}$ at time (n + m + 5 +10k) and at time (n + m + 2) and to vertex $v_{m,n-1}$ at time (n + m + 7+10k)

- **Vertex $v_{m-1,n-3}$**

- receives odd numbered message (1+4k), $k \geq 0$ from vertex $v_{m,n-3}$ at time (m + n + 5 + 10k)

- receives odd numbered message (3+4k), $k \geq 0$ from vertex $v_{m-2,n-3}$ at time (m + n + 1 + 10k)

- sends odd numbered message (3+4k), $k \geq 0$ to vertex $v_{m,n-3}$ at time (n + m + 2+ 10k) and to vertex $v_{m-1,n-4}$ at time (n + m + 3+ 10k)

- receives even numbered message (2+4k), $k \geq 0$ from vertex $v_{m-1,n-4}$ at time (m + n – 3 + 10k)

- receives even numbered message (4+4k), $k \geq 0$ from vertex $v_{m-1,n-2}$ at time (n + m + 14 + 10k)

- sends even numbered message (2+4k), $k \geq 0$ to vertex $v_{m-1,n-2}$ at time (m + n – 2 + 10k) and to vertex $v_{m-2,n-3}$ at time (n + m - 1 + 10k)

- receives empty message E from vertex $v_{m-1,n-2}$ at time (m + n + 4)

- sends empty message E to vertex $v_{m-1,n-2}$ at time (m+ n – 2 +10(k+1))

- **Vertex $v_{m-1,n-2}$**

- receives odd numbered message (1+4k), $k \geq 0$ from vertex $v_{m-2,n-2}$ at time (m + n – 4 + 10k)

- receives odd numbered message (3+4k), $k \geq 0$ from vertex $v_{m-1,n-1}$ at time (m + n + 5 + 10k)

- sends odd numbered message (1+4k), $k \geq 0$ to vertex $v_{m,n-2}$ at time (m + n – 3 + 10k)

- sends odd numbered message (3+4k), $k \geq 0$ to vertex $v_{m-2,n-2}$ at time (m + n + 12 + 10k)

- receives even numbered message (2+4k), $k \geq 0$ from vertex $v_{m-1,n-3}$ at time (m + n – 2 + 10k)

- receives even numbered message (4+4k), $k \geq 0$ from vertex $v_{m,n-2}$ at time (n + m + 13 + 10k)

- sends even numbered message (2+4k), $k \geq 0$ to vertex $v_{m-1,n-1}$ at time (m + n – 1 + 10k)

- sends even numbered message (4+4k), $k \geq 0$ to vertex $v_{m-1,n-3}$ at time (n + m + 14 + 10k)

- receives empty message E from vertex $v_{m,n-2}$ at time (n + m + 3), from vertex $v_{m-1,n-3}$ at time (m+ n – 2 +10(k+1)) and from vertex $v_{m-2,n-2}$ at time (m+ n – 4 +10(k+1))

- sends empty message E to vertex $v_{m-1,n-3}$ at time (n + m + 4), to vertex $v_{m-2,n-2}$ at time (n + m + 2) and to vertex $v_{m,n-2}$ at time (m+ n – 3 +10(k+1))

- **Vertex $v_{m-1,n-1}$**

- receives odd numbered message (1+4k), $k \geq 0$ from vertex $v_{m,n-1}$ at time (m + n + 8 + 10k)

- receives odd numbered message (3+4k), $k \geq 0$ from vertex $v_{m-2,n-1}$ at time (m + n + 3 + 10k)

- sends odd numbered message (3+4k), $k \geq 0$ to vertex $v_{m,n-1}$ at time (n + m + 4+ 10k) and to vertex $v_{m-1,n-2}$ at time (n + m + 5+ 10k)

- receives even numbered message (2+4k), $k \geq 0$ from vertex $v_{m-1,n-2}$ at time (m + n – 1 + 10k)

- receives even numbered message (4+4k), $k \geq 0$ from vertex $v_{m-1,n}$ at time (n + m + 16 + 10k)

- sends even numbered message (2+4k), $k \geq 0$ to vertex $v_{m-1,n}$ at time (m + n + 10k) and to vertex $v_{m-2,n-1}$ at time (n + m + 7 + 10k)

- receives empty message E from vertex $v_{m-1,n}$ at time (n + m + 6)

- sends empty message E to vertex $v_{m-1,n}$ at time (m + n +10(k+1))

- **Vertex $v_{m-1,n}$**

- receives odd numbered message (1+4k), $k \geq 0$ from vertex $v_{m-2,n}$ at time (m + n – 3 + 10k)

- receives odd numbered message (3+4k), k ≥ 0 from vertex $v_{m-2,n}$ at time (m + n + 3 + 10k)

- sends odd numbered message (1+4k), k ≥ 0 to vertex $v_{m,n}$ at time (m + n – 2 + 10k)

- sends odd numbered message (3+4k), k ≥ 0 to vertex $v_{m,n}$ at time (m + n + 4 + 10k)

- receives even numbered message (2+4k), k ≥ 0 from vertex $v_{m-1,n-1}$ at time (m + n + 10k)

- receives even numbered message (4+4k), k ≥ 0 from vertex $v_{m,n}$ at time (n + m + 12 + 10k)

- sends even numbered message (2+4k), k ≥ 0 to vertex $v_{m-2,n}$ at time (n + m + 1 + 10k)

- sends even numbered message (4+4k), k ≥ 0 to vertex $v_{m-1,n-1}$ at time (n + m + 16 + 10k)

- receives empty message E from vertex $v_{m,n}$ at time (n + m + 2) and (n + m + 5 + 10k) and from vertex $v_{m-1,n-1}$ at time (m + n +10(k+1))

- sends empty message E to vertex $v_{m-1,n-1}$ at time (n + m + 6), to vertex $v_{m-2,n}$ at time (n + m – 1 +10k) and to vertex $v_{m,n}$ at time (m + n – 2 +10(k+1))

Total time of described algorithm that performs multiple message broadcasting in an odd by even DOG grid G_{mxn} is $(m + n + 2.5M + 6)$. C++ program included in Appendix was used to run model for large n, m and M in case of odd number of rows and even number of columns.

4.6.5 Multiple message broadcasting in DOG grid with arbitrary number of messages

The algorithm discussed above works for cases when number of messages is a multiple of four. In case when number of messages is equal to $(1 + 4k)$, $(2 + 4k)$ or $(3 + 4k)$, where k is some integer, a slight modification of the algorithm is needed. The modification adds some constant number to the total time of multiple message broadcasting namely 4 in case of $M = (1 + 4k)$, 8 in case of $M = (2 + 4k)$ and 16 in case of $M = (3 + 4k)$.

Let us look at the case when number of messages needed to be broadcasted M equal to $(1 + 4k)$. Before starting the main algorithm described in the above sections several steps need to be executed that will take care of the first message in the different way, then the rest $M - 1$ messages can be broadcasted according to the main algorithm. Those extra steps require 4 time units. Figure 4.4.5.1 demonstrates those steps for G_{7x7} grid.

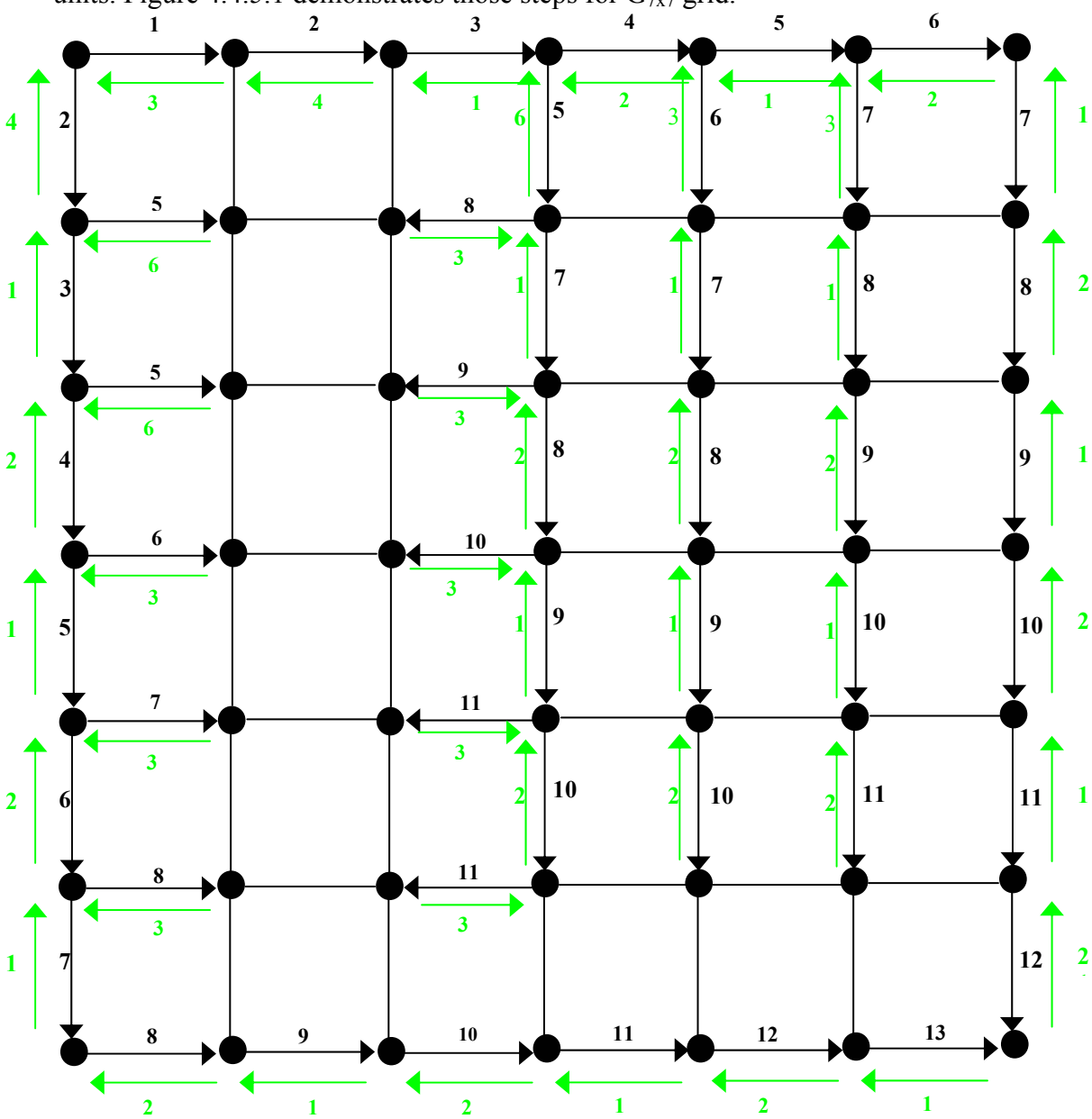


Figure 4.6.5.1 Path of the first message of M = (1+4k) messages in G_{7x7} grid.

Thus total time for multiple message broadcasting in the even by even grid G_{mxn}, when number of messages M is equal to $(1 + 4k)$ where k is integer and $k \geq 0$, is $(m + n + 2.5(M\text{-}1) + 8)$. Total time for multiple message broadcasting in the odd by odd grid G_{mxn}, when number of messages M is equal to $(1 + 4k)$ where k is integer and $k \geq 0$, is $(m + n + 2.5(M\text{-}1) + 15)$. Total time for multiple message broadcasting in the odd by even grid G_{mxn}, when number of messages M is equal to $(1 + 4k)$ where k is integer and $k \geq 0$, is $(m + n + 2.5(M\text{-}1) +10)$.

Same approach works for the cases when number of messages needed to be broadcasted M equal to $(2 + 4k)$ and $(3 + 4k)$. Each additional message requires four more time units of time. Main algorithm will start 8 and 12 time units late correspondingly. Total time for multiple message broadcasting in the even by even grid G_{mxn} is $(m + n + 2.5(M\text{-}2) + 12)$ when number of messages M is equal to $(2 + 4k)$ where k is integer and $k \geq 0$ and $(m + n + 2.5(M\text{-}3) + 16)$ when number of messages M is equal to $(3 + 4k)$ where k is integer and $k \geq 0$. Total time for multiple message broadcasting in the odd by odd grid G_{mxn} is $(m + n + 2.5(M\text{-}2) + 19)$ when number of messages M is equal to $(2 + 4k)$ where k is integer and $k \geq 0$ and $(m + n + 2.5(M\text{-}3) + 23)$ when number of messages M is equal to $(3 + 4k)$ where k is integer and $k \geq 0$. Total time for multiple message broadcasting in the odd by even grid G_{mxn} is $(m + n + 2.5(M\text{-}2) + 14)$ when number of messages M is equal to $(2 + 4k)$ where k is integer and $k \geq 0$ and $(m + n + 2.5(M\text{-}3) + 18)$ when number of messages M is equal to $(3 + 4k)$ where k is integer and $k \geq 0$.

4.6.6 Multiple message broadcasting in DOG grid with arbitrary source node

Discussed algorithm was designed for the case when the source node is the corner node. To consider an arbitrary node as a source the algorithm requires certain changes that increase the broadcasting time. In the case when source node is the corner node, the broadcasting time for the

n x n grid is $2n + 2.5M + C$ where M is the number of messages. However, when the source node is an arbitrary node in the graph to achieve coefficient of 2.5 is not possible. Currently, the source node is sending four new messages every ten time units. The source is busy eight of this ten time units sending and resetting edges which leave us with only two idle time units that we can use in order to have 2.5 coefficient in front of M. Figure 4.4.6.1 shows a grid with arbitrary source node. The lower right of the grid is highlighted to indicate the part where existing algorithm works. We call it main grid. Vertices in the first row and the first column (except corner nodes) of main grid are busy all ten time units so we cannot use them to broadcast messages outside main grid. The algorithm needs to be redesigned in order to solve this new problem. This task can be investigated in the future research work.

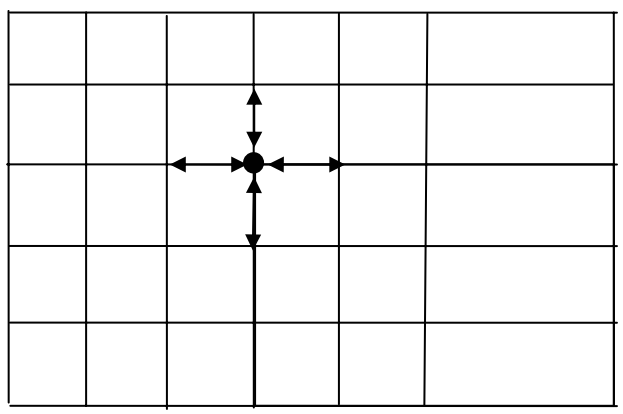


Figure 4.6.6.1 Grid with arbitrary source node

4.7 Multiple message broadcasting summary

Table 4.1 summarizes the time bounds for classical graphs and achieved bounds for dynamically orientable graphs in case of multiple message broadcasting. Lower and upper bound for classical graphs are based on Farley's bounds [F1980]. In case when better bounds exist Farley's bound replaced by best known bounds. For DOG grids, DOG complete trees and DOG

cycles the achieved times are very close to the upper bounds for classical grids and cycles. For DOG stars the best one can do is the twice the time required for classical graphs. For DOG paths and DOG binary trees the coefficient in front of m is twice the coefficient in front of m for the classical paths and binary trees.

Graph	Num. of vertices	Max. Degree d	Diameter D	Lower bound $2(m-1)+D$	Upper bound Classical graph d(m-1) + (n-1)	Trivial upper bound DOG graph	Upper bound DOG graph
P_n	n	2	n - 1	2(m-1) + n -1	2(m-1) + n -1	2(n-1) +4(m-1)	(n-1)+ 4(m-1)
C_n	n is even n is odd	2	$\left\lfloor \frac{n}{2} \right\rfloor$	2(m-1) + $\left\lfloor \frac{n}{2} \right\rfloor$	n + 2m -3	2n + 4m - 6	2n +2m – 6 3n + 3m - 7
B_n (height h)	n	3	varies	2(m-1) + D	3(m-1) + n -1	2(n-1) + 6(m-1)	*(4+h)* + 6(*m*-1) for h = 2; 2h + 6(*m*-1) for h ≥ 3
T_k^p	n = $\frac{k^{p+1}-1}{k-1}$	k + 1	2p	2(m-1) + 2p	O(nm)	O(nm)	*(2k+p)* + 2(*k*+1)(*m*-1) for p = 2; *pk* + 2(*k*+1)(*m*-1) for p ≥ 3
S_n	n	n-1	2	2m	m(n -1)	2m(n-1)	2m(n-1)
$G_{n,n}$	nxn	4	2n - 2	2m+2n-4	2n + 2m + 2	4n + 4m + 4	2n + 2.5m + C

Table 4.1 Multiple message broadcasting in DOG graphs

Chapter 5

5 Conclusion and Future Research

5.1 Conclusion

This dissertation investigates the problems of gossiping and multiple message broadcasting in the DOG of different network topologies. These problems have not been investigated with regards to dynamically orientable graphs of different network topologies. Algorithms for gossiping and multiple message broadcasting in dynamically orientable graphs of different network topologies are proposed. Considering graphs that are dynamically orientable puts a restriction on message movements. The obvious upper bound for gossiping and multiple message broadcasting in DOG is twice the best known time for gossiping and multiple message broadcasting in classical graphs. It can be obtained by executing each step of gossiping or multiple- message broadcasting algorithms for classical graphs at odd times and then resetting the edges involved in the odd times at the even times. The goal of the dissertation is to obtain better bounds for gossiping and multiple message broadcasting in DOG. In some network topologies proposed algorithms for DOG achieved the same upper bound as it is known for classical graphs, for example, gossiping in DOG grids. In some cases the best time is the twice the best known time for gossiping and multiple message broadcasting in classical graphs, for example, gossiping in DOG star. In other cases, good time bounds are achieved that are very close to the upper bounds in classical graphs, for example, multiple message broadcasting in DOG grid.

In summary, Chapter 3 discusses problems of gossiping in the dynamically orientable graphs and Chapter 4 investigates multiple message broadcasting for the same type of graphs.

Particularly, for DOG path, time to gossip is $g_{H1}(P_n) = \begin{cases} n & if\ n\ is\ even \\ n+1 & if\ n\ is\ odd \end{cases}$, which is the same as for classical graphs and time to broadcast m messages from an end node is $b\_m_{H1}(v) = D + 4(m-1)$ which is 2(m-1) time units more than broadcasting m messages from an end node in a classical path graph, but D time units less than twice the broadcasting in classical path. For DOG cycles, time to gossip is the same as for classical graphs, which is $g_{H1}(C_n) = n/2 + \left|\sqrt{2n}\right| - 1$, if n is even, n > 3, and $\lceil n/2 \rceil + \left\lceil \sqrt{2n} - \frac{1}{2} \right\rceil - 1 \le g_{H1}(C_n) \le \lceil n/2 \rceil + \left\lceil 2\sqrt{(n+1)/2} \right\rceil - 1$, if n is odd. According to Farley's theorem the lower bound for time for multiple message broadcasting in a classical cycle graph C_n is 2(m-1) + $\left\lfloor \frac{n}{2} \right\rfloor$ and the upper bound is 2(m-1) + (n-1). For DOG cycles the total time to broadcast m messages from the source is $b\_m_{H1}(v) = 2n + 2m - 6$, which is less than twice the broadcasting time in even classical cycle for $n < 2m + 2$ when n is even and is $b\_m_{H1}(v) = 3n + 3m - 7$, which is less than twice the broadcasting time in odd classical cycle for $n < \frac{m}{2} + 1.5$ when n is odd. In case of star graphs, the gossiping time in classical and DOG graphs is 2n-2 and time to broadcast multiple messages in the DOG star graphs is 2m(n-1) which is twice the multiple message broadcasting time in classical star graphs. For DOG complete tree the gossiping algorithm proposed here requires $2km = Dk$ time units and known gossiping time for classical complete trees is $2km$. The known multiple message broadcasting scheme for arbitrary classical tree has an upper bound of $O(nm)$. This bound holds for complete trees as well. The algorithm proposed here requires $pk + 2(k+1)(m-1)$ time units to complete broadcasting of m messages in DOG_T_k^m. For DOG binary tree gossiping time depends on the configuration

of a binary tree. The upper bound for gossiping in a binary tree with the height h is $g_{H1}(B_n) \leq 4h$. Upper bound for multiple message broadcasting time is $2h + 6(m-1)$ that is reached when binary tree is complete. For DOG grids 12×12 and larger with diameter D proposed gossiping algorithm requires D time units, for DOG grids of size 10×10 and 11×11 it requites D + 1 time units, for 9×9 DOG grids it requires D + 2 time units. Hence, gossiping in the DOG grid graphs of size 12×12 and larger can be done in optimal time, which equals to the diameter of the graph. Wojciechowska and Van Scoy presented an algorithm for broadcasting m messages in an $n \times n$ grid that required $2n + 2m + 2$ time units with corner node as a source. Proposed algorithm performs multiple message broadcasting in a grid in $2n + 2.5m + C$.

Table 5.1 summarizing existing bounds for classical graphs and achieved bounds for dynamically orientable graphs in case of gossiping. Table 5.2 summarizing time bounds for classical graphs and achieved bounds for dynamically orientable graphs in case of multiple message broadcasting. Lower and upper bound for classical graphs are based on Farley's bounds [F1980]. In case when better bounds exist Farley's bound replaced by best known bounds. Last column in both tables shows the bound achieved by algorithms presented in this dissertation.

5.2 Future Research

Future work may include problems of gossiping and multiple message broadcasting in dynamically orientable graphs of other network topologies, such as hypercubes, de Bruijn graphs, shuffle-exchange graphs and butterfly graphs. Particularly, good algorithm is designed for gossiping in DOG de Bruijn graphs. The algorithm improves the upper bound for n < 32,000. Other interesting topologies include shuffle-exchange and butterfly graphs. For dynamically

orientable grid graphs, development of algorithm for multiple message broadcasting from arbitrary source is next topic to be researched.

Graph	Num. of vertices	Max. degree	Diameter D	Lower bound Classical graph	Upper bound Classical graph	Trivial upper bound DOG graph	Upper bound DOG graph
P_n	n even n odd	2	n - 1	n n + 1	n n + 1	2n 2(n+1)	n n + 1
C_n	n even n odd	2	$\left\lfloor \frac{n}{2} \right\rfloor$	$\frac{n}{2}+\left\lceil\sqrt{2n}\right\rceil-1$ $\left\lceil\frac{n}{2}\right\rceil+\left\lceil\sqrt{2n}-\frac{1}{2}\right\rceil-1$	$\frac{n}{2}+\left\lceil\sqrt{2n}\right\rceil-1$ $\left\lceil\frac{n}{2}\right\rceil+\left\lceil 2\sqrt{\left\lceil\frac{n}{2}\right\rceil}\right\rceil-1$	$n+2\left\lceil\sqrt{2n}\right\rceil-2$ $2\left\lceil\frac{n}{2}\right\rceil+2\left\lceil 2\sqrt{\left\lceil\frac{n}{2}\right\rceil}\right\rceil-2$	$\frac{n}{2}+\left\lceil\sqrt{2n}\right\rceil-1$ $\left\lceil\frac{n}{2}\right\rceil+\left\lceil 2\sqrt{\left\lceil\frac{n}{2}\right\rceil}\right\rceil-1$
B_n	n	3	varies				4h (when complete)
T_k^m	n=$\frac{k^{m+1}-1}{k-1}$	k + 1	2m	Dk	Dk	2Dk	Dk
S_n	n	n-1	2		2(n-1)	4(n-1)	2(n-1)
$G_{m,n}$	mxn	4	m + n - 2	D	D for 9x9 & larger	2D for 9x9 & larger	D for 12x12 & larger

Table 5.1 Gossiping in DOG graphs

Graph	Num. of vertices	Max. Degree d	Diameter D	Lower bound $2(m-1)+D$	Upper bound Classical graph d(m-1) + (n-1)	Trivial upper bound DOG graph	Upper bound DOG graph
P_n	n	2	n - 1	2(m-1) + n -1	2(m-1) + n -1	2(n-1) +4(m-1)	(n-1)+ 4(m-1)
C_n	n is even n is odd	2	$\left\lfloor \frac{n}{2} \right\rfloor$	2(m-1) + $\left\lfloor \frac{n}{2} \right\rfloor$	n + 2m -3	2n + 4m - 6	2n +2m – 6 3n + 3m - 7
B_n	n (height h)	3	varies	2(m-1) + D	3(m-1) + n -1	2(n-1) + 6(m-1)	*(4+h)* + 6(*m*-1) for h = 2; 2h + 6(*m*-1) for h ≥ 3
T_k^p	n = $\frac{k^{p+1}-1}{k-1}$	k + 1	2p	2(m-1) + 2p	O(nm)	O(nm)	*(2k+p)* + 2(*k*+1)(*m*-1) for p = 2; *pk* + 2(*k*+1)(*m*-1) for p ≥ 3
S_n	n	n-1	2	2m	m(n -1)	2m(n-1)	2m(n-1)
$G_{n,n}$	nxn	4	2n - 2	2m+2n-4	2n + 2m + 2	4n + 4m + 4	2n + 2.5m + C

Table 5.2 Multiple message broadcasting in DOG graphs

Bibliography

[BH1999] A. Bar-Noy, C. Ho (1999). Broadcasting multiple messages in the Multiport Model. *IEEE Transactions on Parallel and Distributed Systems 10(5), 500-508*.

[BK1997] A. Bar-Noy, S. Kipnis (1997). Multiple message broadcasting in the postal model. *Networks 29(1), 1-10*.

[BKS2000] A. Bar-Noy, S. Kipnis, B. Schieber (2000). Optimal multiple message broadcasting in telephone-like communication systems. *Discrete Applied Mathematics 100, 1–15.*

[BP1988] J. Bermond, C. Peyrat (1988). Broadcasting in de Bruijn networks. *Proceeding of 19th South-eastern Conference on Combinatorics, Graph theory and Computing, Congressus Numerantium 66, 283-292.*

[CT1980] E. Cockayne, A. Thomason (1980). Optimal multi-message broadcasting in complete graphs. *Utilitas Mathematica 18, 181-199*.

[DLP1999] K. Diks, A. Lingas, A. Pelc (1999). An optimal algorithm for broadcasting multiple messages in trees. *Journal of Parallel and Distributed Computing 59, 465 -474.*

[ES1979] R. Entringer, P. Slater (1979). Gossips and Telegraphs. *Journal of the Franklin Institute 307, 353-359*.

[EM1989] S. Even, B. Monien (1989). On the number of rounds necessary to disseminate information. *Proceedings of the first annual ACM symposium on Parallel algorithms and architectures, 318-327*.

[F1980] A. Farley (1980). Broadcast time in communication networks. *SIAM Journal on Applied Mathematics 39 (2), 385 – 390*.

[F2004] A. Farley (2004). Minimal path broadcast networks. *Networks 43(2), 61-70.*

[FL1994] P. Fraigniaud, E. Lazard (1994). Methods and problems of communication in usual networks. *Discrete Applied Mathematics 53, 79-133*.

[HHL1988] S. Hedetniemi, S. Hedetniemi, A. Liestman (1988). A survey of gossiping and broadcasting in communication networks. *Networks 18, 319-349*.

[HJM1993] J. Hromkovic, C. Jeschke, B. Monien (1993). Optimal algorithms for dissemination of information in some interconnection networks. *Algorithmica 10-1, 24-40*.

[HJM1994] J. Hromkovic, C. Jeschke, B. Monien (1994). Note on optimal gossiping in some weak-connected graphs. *Theoretical Computer Science 127, 395-402*.

[HK1996] J. Hromkovic, R. Klasing et all (1996). Dissemination of information in interconnection networks (broadcasting and gossiping). *Combinatorial Network Theory, Kluwer Academic Publishers, 152-212*.

[KMPS1994] R. Klasing, B. Monien, R. Peine, E. Stohr (1994). Broadcasting in Butterfly and deBruijn Networks. *Discrete Applied Mathematics 53, 183-197*.

[K1996] W. Klostermeyer (1996). Path problems in dynamically orientable graphs. *Australasian Journal of Combinatorics 14, 21-30*.

[K1992] D. Krumme (1992). Fast gossiping for the hypercube. *SIAM Journal on Computing 21-2, 365-380*.

[KCV1992] D. Krumme, G. Cybenko, K. Venkataraman (1992). Gossiping in minimal time. *SIAM Journal on Computing 21-1, 111-139*.

[LZ2003] F. Lau, S. Zhang (2003). Optimal gossiping in paths and cycles. *Journal of Discrete Algorithms 1, 461-475*.

[LSS2000] A. Liestman, T. Shermen, M. Suderman (2000). Broadcasting multiple messages in hypercubes. *Proceedings of International Symposium on Parallel Architectures, Algorithms and Networks, 274-281*.

[P1996] S. Perennes (1996). Broadcasting and gossiping on de Bruijn, shuffle-exchange and similar networks. *Discrete Applied Mathematics 83, 247-262*.

[RS1997] Y.Roditty, B. Shoham (1997). On broadcasting multiple messages in a *d*-dimensional grid. *Discrete Applied Mathematics 75, 277-284*.

[SCH1981] P. Slater, E. Cockayne, S.Hedetniemi (1981). Information dissemination in trees. *SIAM Journal on Computing 10, 692-701*.

[VB1994] F. VanScoy, J. Brooks (1994). Broadcasting multiple messages in a grid. *Discrete Applied Mathematics 53, 321–336.*

[WV1996] I. Wojciechowska, F. Van Scoy (1996). A nearly optimal algorithm for broadcasting multiple messages in a two-dimensional grid. *Congressum Numerantium 118, 81 –96.*

Appendix

```
//
// Multiple messages broadcasting in an arbitrary size DOG grid
//

#include <stdio.h>
#include <stdlib.h>
#include <iostream>
#include <string.h>
#include <sstream>
#include <fstream>
#include <list>
#include <vector>

using namespace std;

std::stringstream ss;
std::string str1, str2, str3, str4, str_index;
std::ofstream fout("output_MMB_arbitrary.txt");

int M; // total  number of messages
int k;  // multiplier in message count

int row_m; // number of rows
int column_n; // nuber of columns


string itos(int i)      // convert int to string
      {
            stringstream s;
            s << i;
            return s.str();
      }


// Message Class
class Message {

public:

      // constructor
    Message() {

      mess_number = 0;
      time_received = 0;
      time_sent = 0;
      }


      // constructor
Message(int num, int rec_time, string r_from, int sent_time, string s_to)
{
      mess_number = num;
      time_received = rec_time;
      time_sent = sent_time;
    received_from = r_from;
      sent_to = s_to;
}

   // destructor
      ~Message() {}
```

```
      int get_message_number() { return mess_number;}
      int get_time_received() { return time_received;}
      int get_time_sent() { return time_sent;}
      string get_received_from() { return received_from;}
      string get_sent_to() { return sent_to;}

private:
      int mess_number;
      int time_received;
      int time_sent;
      string received_from;
      string sent_to;
};

struct Node {
      Message *Item;
      Node  *Next;
};

// message_list Class
class message_list {

public:

      // constructor
      message_list();

      // destructor
      ~message_list();

      Node *Nodes;
      void Add(Message *NewItem);
      void Report_Received_Mes();
      void Report_Sent_Mes();
      void Find_Max_received_t();
      void Find_Max_sent_t();
      int Max_received_time();
      int Max_sent_time();

};

message_list::message_list ()                 // constructor
{
   Nodes = NULL;
}

message_list::~message_list()                 // destructor
{
   while (Nodes != NULL) {
      Node *N = Nodes;
      delete(N->Item);
      Nodes = N->Next;
      delete N;
   };
}

void message_list::Add(Message *NewItem)
{
   Node *N;
   N = new Node;
   N->Item = NewItem;
   N->Next = Nodes;
   Nodes = N;
```

```
}

void message_list::Report_Received_Mes()
{
 Node *Current =Nodes;
 cout << " \nReceived Messages statistics: ";
 fout << " \nReceived Messages statistics: ";

 while (Current != NULL)

 {
         fout << " \nMessage number: "<< Current->Item->get_message_number();
         fout << " \nReceived from " << Current->Item->get_received_from() <<
                  ", Time_received: "<< Current->Item->get_time_received();
         fout << " \n";
         Current = Current -> Next;
 };
}

void message_list::Report_Sent_Mes()
{
 Node *Current =Nodes;
 cout << " \nSent Messages statistics: ";
 fout << " \nSent Messages statistics: ";

 while (Current != NULL)

 {
         fout << " \nMessage number: "<< Current->Item->get_message_number();
         fout << " \nSent to " << Current->Item->get_sent_to() <<
                  ", Time_sent: "<< Current->Item->get_time_sent();
         fout << " \n";
         Current = Current -> Next;

 };
}


void message_list::Find_Max_received_t()
{
      Node *Current =Nodes;

      int Max_received = 0;
      int max_rec_message_num = -1;

      while (Current != NULL)
            {
             if (Current->Item->get_time_received() > Max_received)
                {
                Max_received = Current->Item->get_time_received();
                  max_rec_message_num = Current->Item->get_message_number();
                }

                Current =Current->Next;
             }

fout<< " Maximum time for received message:\n";
fout << " \nMessage number: "<< max_rec_message_num;
fout << " Time_received: "<< Max_received;

} // end of void message_list::Find_Max_received_t()
```

```
void message_list::Find_Max_sent_t()
{
      Node *Current =Nodes;

      int Max_send = 0;
      int max_sent_message_num = -1;


      while (Current != NULL)
            {
              if (Current->Item->get_time_sent() > Max_send)
                {
                Max_send = Current->Item->get_time_sent();
                  max_sent_message_num = Current->Item->get_message_number();
                }

              Current =Current->Next;
             }

fout<< " \nMaximum time for sent message:\n";
fout << " \nMessage number: "<< max_sent_message_num;
fout << " Time_received: "<< Max_send;
} // end of void message_list::Find_Max_sent_t()


int message_list::Max_received_time()
{
      Node *Current =Nodes;

      int Max_received = 0;

      while (Current != NULL)
            {
             if (Current->Item->get_time_received() > Max_received)
                {
                Max_received = Current->Item->get_time_received();
                  }

                Current =Current->Next;
             }

return Max_received;

}



int message_list::Max_sent_time()
{
      Node *Current =Nodes;

      int Max_send = 0;

      while (Current != NULL)
            {
              if (Current->Item->get_time_sent() > Max_send)
                {
                Max_send = Current->Item->get_time_sent();
                  }

              Current =Current->Next;
             }

return Max_send;
```

```
}


class Vertex {


public:

// constructor
    Vertex() {}


// constructor
      Vertex(int i, int j)
{
      index_i = i;
      index_j = j;
}

// destructor
~Vertex() {}


int get_index_i() { return index_i; }
int get_index_j() { return index_j; }
message_list Received_Messages;
message_list Sent_Messages;

private:

 int index_i;
 int index_j;

};



struct Vertex_Node {
   Vertex *Item;
   Vertex_Node  *Next;
};


class Vertex_list {
   Vertex_Node *Nodes;

public:

// constructor
   Vertex_list();

// destructor
   ~Vertex_list();

   void Add(Vertex *NewItem);

   void Report_Vertex();
   void Report_Max_time();
   int Max_received_t();
   int Max_sent_t();

   Vertex* Find_Vertex(int search_i, int search_j);
```

```
};


Vertex_list::Vertex_list ()                 // constructor
{
   Nodes = NULL;
}

Vertex_list::~Vertex_list()                 // destructor
{
   while (Nodes != NULL) {
      Vertex_Node *N = Nodes;
      delete(N->Item);
      Nodes = N->Next;
      delete N;
   };
}

void Vertex_list::Add(Vertex *NewItem)
{
   Vertex_Node *N;
   N = new Vertex_Node;
   N->Item = NewItem;
   N->Next = Nodes;
   Nodes = N;
}


void Vertex_list::Report_Vertex()
{
 Vertex_Node *Current =Nodes;

 while (Current != NULL)

 {
         fout << " \nVertex v( "<< Current->Item->get_index_i() << "," <<
                  Current->Item->get_index_j() << "): \n";
         Current->Item->Received_Messages.Report_Received_Mes();
         fout << " \n";
         Current->Item->Sent_Messages.Report_Sent_Mes();
         fout << " \n";
         Current = Current -> Next;

 };
}


void Vertex_list::Report_Max_time()
{

Vertex_Node *Current =Nodes;

 while (Current != NULL)

 {
      fout << " \nVertex v( "<< Current->Item->get_index_i() << "," <<
                  Current->Item->get_index_j() << "): \n";
      Current->Item->Received_Messages.Find_Max_received_t();
      fout << " \n";
      Current->Item->Sent_Messages.Find_Max_sent_t();
      fout << " \n";
      Current = Current -> Next;
```

```
 };
}


int Vertex_list::Max_received_t()
{

int max_time = 0;

Vertex_Node *Current =Nodes;

 while (Current != NULL)

 {
       if ( Current->Item->Received_Messages.Max_received_time() > max_time)
       max_time = Current->Item->Received_Messages.Max_received_time();

         Current = Current -> Next;

 };
return max_time;
}


int Vertex_list::Max_sent_t()
{

int max_time = 0;

Vertex_Node *Current =Nodes;

 while (Current != NULL)

 {
       if ( Current->Item->Sent_Messages.Max_sent_time() > max_time)
       max_time = Current->Item->Sent_Messages.Max_sent_time();

         Current = Current -> Next;

 };
return max_time;
}



Vertex* Vertex_list::Find_Vertex(int search_i, int search_j)
{

Vertex_Node *Current =Nodes;

      while (Current != NULL)
            {
            if ((Current->Item->get_index_i() == search_i) && (Current->Item->
            get_index_j() == search_j))
                {
                return (Current->Item);
                }
             Current =Current->Next;
             }
      return NULL;
}

/////////////////////////////////////////////////////////////////
```

```
void calculate_vertex_i_j ( Vertex *ver, int num_i, int num_j)

{
int i = num_i;
int j = num_j;
Vertex *v = ver;
Message *m;
int t;

                  str1.append("v(");
                  str2.append("v(");
                  str3.append("v(");
                  str4.append("v(");

                  str_index = itos(i-1);
                  str1.append(str_index);
                  str_index = itos(i+1);
                  str2.append(str_index);
                  str1.append(",");
                  str2.append(",");

                  str_index = itos(j);
                  str1.append(str_index);
                  str2.append(str_index);
                  str1.append(")");
                  str2.append(")");

                  str_index = itos(i);
                  str3.append(str_index);
                  str4.append(str_index);
                  str3.append(",");
                  str4.append(",");

                  str_index = itos(j-1);
                  str3.append(str_index);
                  str_index = itos(j+1);
                  str4.append(str_index);
                  str3.append(")");
                  str4.append(")");


if (i%2 != 0 && j%2 != 0)
{

// Messages (1 + 4k)

      //Receives
      for ( t = 0; t <= k ; t++ ) {
            m = new Message((1+4*t),(i + j + 2 + 10*t),str4, 0, " ");
            (*v).Received_Messages.Add(m);
            }

      //Sends
      for (  t = 0; t <= k ; t++ ) {
            m = new Message((1+4*t),0," ",(i + j + 9 + 10*t), str1);
            (*v).Sent_Messages.Add(m);
            }

// Messages (2 + 4k)

      //Receives
      for (  t = 0; t <= k ; t++ ) {
            m = new Message((2+4*t),(i + j + 10 + 10*t),str2, 0, " ");
```

```
            (*v).Received_Messages.Add(m);
            }

      //Sends
      for (  t = 0; t <= k ; t++ ) {
            m = new Message((2+4*t),0," ",(i + j + 11 + 10*t), str3);
            (*v).Sent_Messages.Add(m);
            }

// Messages (3 + 4k)

      //Receives
      for (  t = 0; t <= k ; t++ ) {
            m = new Message((3 + 4*t),(i + j + 5 + 10*t),str1,0, " ");
            (*v).Received_Messages.Add(m);
            }

      //Sends
      for (  t = 0; t <= k ; t++ ) {
            m = new Message((3+4*t),0," ",(i + j + 6 + 10*t), str2);
            (*v).Sent_Messages.Add(m);
            }

// Messages (4 + 4k)

      //Receives
      for (  t = 0; t <= k ; t++ ) {
            m = new Message((4 + 4*t),(i + j + 7 + 10*t), str3,0, " " );
            (*v).Received_Messages.Add(m);
            }

      //Sends
      for (  t = 0; t <= k ; t++ ) {
            m = new Message((4 + 4*t),0," ",(i + j + 8 + 10*t), str4);
            (*v).Sent_Messages.Add(m);
            }


} // end of IF (i%2 != 0 && j%2 != 0)


if (i%2 != 0 && j%2 == 0)

      {
// Messages (1 + 4k)

      //Receives
      for ( t = 0; t <= k ; t++ ) {
            m = new Message((1+4*t),(i + j - 1 + 10*t),str1, 0, " ");
            (*v).Received_Messages.Add(m);
            }

      //Sends
      for (  t = 0; t <= k ; t++ ) {
            m = new Message((1+4*t),0," ",(i + j + 10*t), str2);
            (*v).Sent_Messages.Add(m);
            }

      for (  t = 0; t <= k ; t++ ) {
            m = new Message((1+4*t),0," ",(i + j + 1 + 10*t), str3);
            (*v).Sent_Messages.Add(m);
            }

// Messages (2 + 4k)
```

```
      //Receives
      for (  t = 0; t <= k ; t++ ) {
            m = new Message((2+4*t),(i + j + 12 + 10*t),str4, 0, " ");
            (*v).Received_Messages.Add(m);
            }

// Messages (3 + 4k)

      //Receives
      for (  t = 0; t <= k ; t++ ) {
            m = new Message((3 + 4*t),(i + j + 16 + 10*t),str2,0, " ");
            (*v).Received_Messages.Add(m);
            }

// Messages (4 + 4k)

      //Receives
      for (  t = 0; t <= k ; t++ ) {
            m = new Message((4 + 4*t),(i + j + 7 + 10*t), str3,0, " " );
            (*v).Received_Messages.Add(m);
            }

      //Sends
      for (  t = 0; t <= k ; t++ ) {
            m = new Message((4 + 4*t),0," ",(i + j + 8 + 10*t), str4);
            (*v).Sent_Messages.Add(m);
            }

      for (  t = 0; t <= k ; t++ ) {
            m = new Message((4 + 4*t),0," ",(i + j + 15 + 10*t), str1);
            (*v).Sent_Messages.Add(m);
            }

// Messages 0

      //Receives
            m = new Message(0,(i + j - 1 + 10*(k+1)), str1,0, " " );
            (*v).Received_Messages.Add(m);

            m = new Message(0,(i + j + 6), str2,0, " " );
            (*v).Received_Messages.Add(m);

      //Sends
      m = new Message(0,0," ",(i + j + 5), str1);
      (*v).Sent_Messages.Add(m);

      m = new Message(0,0," ",(i + j + 10*(k+1)), str2);
      (*v).Sent_Messages.Add(m);

} // end of IF (i%2 != 0 && j%2 == 0)


if (i%2 == 0 && j%2 != 0)
{
// Messages (1 + 4k)

      //Receives
      for ( t = 0; t <= k ; t++ ) {
            m = new Message((1+4*t),(i + j + 10 + 10*t),str2, 0, " ");
            (*v).Received_Messages.Add(m);
            }

// Messages (2 + 4k)
```

```
      //Receives
      for (  t = 0; t <= k ; t++ ) {
            m = new Message((2+4*t),(i + j + 1 + 10*t),str3, 0, " ");
            (*v).Received_Messages.Add(m);
            }

      //Sends
      for (  t = 0; t <= k ; t++ ) {
            m = new Message((2+4*t),0," ",(i + j + 2 + 10*t), str4);
            (*v).Sent_Messages.Add(m);
            }

      for (  t = 0; t <= k ; t++ ) {
            m = new Message((2+4*t),0," ",(i + j + 9 + 10*t), str1);
            (*v).Sent_Messages.Add(m);
            }

// Messages (3 + 4k)

      //Receives
      for (  t = 0; t <= k ; t++ ) {
            m = new Message((3 + 4*t),(i + j + 5 + 10*t),str1,0, " ");
            (*v).Received_Messages.Add(m);
            }

      //Sends
      for (  t = 0; t <= k ; t++ ) {
            m = new Message((3+4*t),0," ",(i + j + 6 + 10*t), str2);
            (*v).Sent_Messages.Add(m);
            }

      for (  t = 0; t <= k ; t++ ) {
            m = new Message((3+4*t),0," ",(i + j + 7 + 10*t), str3);
            (*v).Sent_Messages.Add(m);
            }

// Messages (4 + 4k)

      //Receives
      for (  t = 0; t <= k ; t++ ) {
            m = new Message((4 + 4*t),(i + j + 18 + 10*t), str4,0, " " );
            (*v).Received_Messages.Add(m);
            }

// Messages 0

      //Receives
            m = new Message(0,(i + j + 8), str4,0, " " );
            (*v).Received_Messages.Add(m);

      //Sends

            m = new Message(0,0," ",(i + j + 2 + 10*(k+1)), str4);
            (*v).Sent_Messages.Add(m);

      } // end of IF (i%2 == 0 && j%2 != 0)


if (i%2 == 0 && j%2 == 0)

{

// Messages (1 + 4k)
```

```
      //Receives
      for ( t = 0; t <= k ; t++ ) {
            m = new Message((1+4*t),(i + j - 1 + 10*t),str1, 0, " ");
            (*v).Received_Messages.Add(m);
            }

      //Sends
      for (  t = 0; t <= k ; t++ ) {
            m = new Message((1+4*t),0," ",(i + j + 10*t), str2);
            (*v).Sent_Messages.Add(m);
            }

// Messages (2 + 4k)

      //Receives
      for (  t = 0; t <= k ; t++ ) {
            m = new Message((2+4*t),(i + j + 1 + 10*t),str3, 0, " ");
            (*v).Received_Messages.Add(m);
            }

      //Sends
      for (  t = 0; t <= k ; t++ ) {
            m = new Message((2+4*t),0," ",(i + j + 2 + 10*t), str4);
            (*v).Sent_Messages.Add(m);
            }

// Messages (3 + 4k)

      //Receives
      for (  t = 0; t <= k ; t++ ) {
            m = new Message((3 + 4*t),(i + j + 8 + 10*t),str4,0, " ");
            (*v).Received_Messages.Add(m);
            }

      //Sends
      for (  t = 0; t <= k ; t++ ) {
            m = new Message((3+4*t),0," ",(i + j + 15 + 10*t), str1);
            (*v).Sent_Messages.Add(m);
            }

// Messages (4 + 4k)

      //Receives
      for (  t = 0; t <= k ; t++ ) {
            m = new Message((4 + 4*t),(i + j + 16 + 10*t), str2,0, " " );
            (*v).Received_Messages.Add(m);
            }

      //Sends
      for (  t = 0; t <= k ; t++ ) {
            m = new Message((4 + 4*t),0," ",(i + j + 17 + 10*t), str3);
            (*v).Sent_Messages.Add(m);
            }


// Messages 0

      //Receives

            m = new Message(0,(i + j - 1 + 10*(k+1)), str1,0, " " );
            (*v).Received_Messages.Add(m);

            m = new Message(0,(i + j + 6), str2,0, " " );
```

```
            (*v).Received_Messages.Add(m);

            m = new Message(0,(i + j + 1 + 10*(k+1)), str3,0, " " );
            (*v).Received_Messages.Add(m);

      //Sends
            m = new Message(0,0," ",(i + j + 5), str1);
            (*v).Sent_Messages.Add(m);

            m = new Message(0,0," ",(i + j + 10*(k+1)), str2);
            (*v).Sent_Messages.Add(m);

            m = new Message(0,0," ",(i + j + 7), str3);
            (*v).Sent_Messages.Add(m);

} // end of IF (i%2 == 0 && j%2 == 0)

      str1 = "";
      str2 = "";
      str3 = "";
      str4 = "";
} // end of calculate_vertex_i_j() function


////////////////////////////////////////////////////////////////////////////

void calculate_vertex_row_minus_2_2(Vertex *ver) {

// Vertex v(row_m-2,2)

Vertex *v = ver;
Message *m;

                  str1.append("v(");
                  str_index = itos(row_m-3);
                  str1.append(str_index);
                  str1.append(",");
                  str_index = itos(2);
                  str1.append(str_index);
                  str1.append(")");


                  str2.append("v(");
                  str_index = itos(row_m-1);
                  str2.append(str_index);
                  str2.append(",");
                  str_index = itos(2);
                  str2.append(str_index);
                  str2.append(")");


                  str3.append("v(");
                  str_index = itos(row_m-2);
                  str3.append(str_index);
                  str3.append(",");
                  str_index = itos(1);
                  str3.append(str_index);
                  str3.append(")");

                  str4.append("v(");
                  str_index = itos(row_m-2);
                  str4.append(str_index);
                  str4.append(",");
                  str_index = itos(3);
```

```
                  str4.append(str_index);
                  str4.append(")");

// Messages (1 + 4k)

      //Receives
      for ( int j = 0; j <= k ; j++ ) {
            m = new Message((1+4*j),(row_m - 1 + 10*j),str1, 0, " ");
            (*v).Received_Messages.Add(m);
            }

      //Sends
      for (  j = 0; j <= k ; j++ ) {
            m = new Message((1 + 4*j),0," ",(row_m + 10*j), str2);
            (*v).Sent_Messages.Add(m);
            }

      for (  j = 0; j <= k ; j++ ) {
            m = new Message((1 + 4*j),0," ",(row_m + 2 + 10*j), str3);
            (*v).Sent_Messages.Add(m);
            }

// Messages (2 + 4k)

      //Receives
      for (  j = 0; j <= k ; j++ ) {
            m = new Message((2+4*j),(row_m + 21 + 10*j),str4, 0, " ");
            (*v).Received_Messages.Add(m);
            }

// Messages (3 + 4k)

      //Receives
      for (  j = 0; j <= k ; j++ ) {
            m = new Message((3 + 4*j),(row_m + 24 + 10*j),str2,0, " ");
            (*v).Received_Messages.Add(m);
            }

// Messages (4 + 4k)

      //Receives
      for (  j = 0; j <= k ; j++ ) {
            m = new Message((4 + 4*j),(row_m + 7 + 10*j), str3,0, " " );
            (*v).Received_Messages.Add(m);
            }

      //Sends
      for (  j = 0; j <= k ; j++ ) {
            m = new Message((4 + 4*j),0," ",(row_m + 8 + 10*j), str4);
            (*v).Sent_Messages.Add(m);
            }

      for (  j = 0; j <= k ; j++ ) {
            m = new Message((4 + 4*j),0," ",(row_m + 15 + 10*j), str1);
            (*v).Sent_Messages.Add(m);
            }

// Messages (0)

      //Receives
            m = new Message(0,(row_m + 4), str2, 0, " ");
            (*v).Received_Messages.Add(m);

            m = new Message(0,(row_m + 14), str2, 0, " ");
```

```
            (*v).Received_Messages.Add(m);
            m = new Message(0,(row_m + 11), str4, 0, " ");
            (*v).Received_Messages.Add(m);

            m = new Message(0,(row_m - 1 + 10*(k+1)), str1, 0, " ");
            (*v).Received_Messages.Add(m);

      //Sends
            m = new Message(0,0," ",(row_m + 5), str1);
            (*v).Sent_Messages.Add(m);

            m = new Message(0,0, " ", (row_m + 8 + 10*(k+1)), str4);
            (*v).Sent_Messages.Add(m);

            m = new Message(0,0, " ", (row_m + 10*(k+1)), str2);
            (*v).Sent_Messages.Add(m);

            m = new Message(0,0, " ", (row_m + 10*(k+2)), str2);
            (*v).Sent_Messages.Add(m);

      str1 = "";
      str2 = "";
      str3 = "";
      str4 = "";
      }

////////////////////////////////////////////////////////////////////////////////

void calculate_vertex_row_column_minus_3 ( Vertex *ver)
{
// Vertex v(row_m,column_n - 3)

Vertex *v = ver;
Message *m;


                  str1.append("v(");
                  str_index = itos(row_m);
                  str1.append(str_index);
                  str1.append(",");
                  str_index = itos(column_n - 4);
                  str1.append(str_index);
                  str1.append(")");


                  str2.append("v(");
                  str_index = itos(row_m);
                  str2.append(str_index);
                  str2.append(",");
                  str_index = itos(column_n - 2);
                  str2.append(str_index);
                  str2.append(")");


                  str3.append("v(");
                  str_index = itos(row_m-1);
                  str3.append(str_index);
                  str3.append(",");
                  str_index = itos(column_n - 3);
                  str3.append(str_index);
                  str3.append(")");


// Messages (1 + 4k)
```

```
      //Receives
      for ( int i = 0; i <= k ; i++ ) {
      m = new Message((1+4*i),(row_m + column_n + 10*i),str2, 0, " ");
      (*v).Received_Messages.Add(m);
      }

      //Sends
      for (  i = 0; i <= k ; i++ ) {
      m = new Message((1+4*i),0," ",(row_m + column_n + 5 + 10*i), str3);
      (*v).Sent_Messages.Add(m);
      }

// Messages (2 + 4k)

      //Receives
      for (  i = 0; i <= k ; i++ ) {
      m = new Message((2+4*i),(row_m + column_n - 3 + 10*i),str1, 0, " ");
      (*v).Received_Messages.Add(m);
      }

      //Sends
      for (  i = 0; i <= k ; i++ ) {
      m = new Message((2+4*i),0, " ", (row_m + column_n - 2 + 10*i),str2);
      (*v).Sent_Messages.Add(m);
      }

// Messages (3 + 4k)

      //Receives
      for (  i = 0; i <= k ; i++ ) {
      m = new Message((3 + 4*i),(row_m + column_n + 2 + 10*i),str3,0, " ");
      (*v).Received_Messages.Add(m);
      }

      //Sends
      for (  i = 0; i <= k ; i++ ) {
      m = new Message((3+4*i),0, " ", (row_m + column_n + 9 + 10*i),str1);
      (*v).Sent_Messages.Add(m);
      }

// Messages (4 + 4k)

      //Receives
      for (  i = 0; i <= k ; i++ ) {
      m = new Message((4 + 4*i),(row_m + column_n + 3 + 10*i), str1,0, " " );
      (*v).Received_Messages.Add(m);
      }

      //Sends
      for (  i = 0; i <= k ; i++ ) {
      m = new Message((4 + 4*i),0," ",(row_m + column_n + 4 + 10*i), str2);
      (*v).Sent_Messages.Add(m);
      }

// Messages (0)

      //Receives
      for (  i = 0; i <= k ; i++ ) {
      m = new Message(0,(row_m + column_n + 6 + 10*i),str2, 0, " ");
      (*v).Received_Messages.Add(m);
      }

      m = new Message(0,(row_m + column_n - 3 + 10*(k+1)), str1, 0, " ");
```

```
      (*v).Received_Messages.Add(m);

      //Sends
      for (  i = 0; i <= k ; i++ ) {
            m = new Message(0,0," ",(row_m + column_n + 1 + 10*i), str1);
            (*v).Sent_Messages.Add(m);
            }

            m = new Message(0,0," ",(row_m + column_n - 1),str1);
            (*v).Sent_Messages.Add(m);

      str1 = "";
      str2 = "";
      str3 = "";

}


////////////////////////////////////////////////////////////////////////////////

void calculate_vertex_row_column_minus_2 (Vertex *ver) {

// Vertex v(row_m,column_n-2)

Vertex *v = ver;
Message *m;


                  str1.append("v(");
                  str_index = itos(row_m);
                  str1.append(str_index);
                  str1.append(",");
                  str_index = itos(column_n-3);
                  str1.append(str_index);
                  str1.append(")");


                  str2.append("v(");
                  str_index = itos(row_m);
                  str2.append(str_index);
                  str2.append(",");
                  str_index = itos(column_n-1);
                  str2.append(str_index);
                  str2.append(")");


                  str3.append("v(");
                  str_index = itos(row_m-1);
                  str3.append(str_index);
                  str3.append(",");
                  str_index = itos(column_n-2);
                  str3.append(str_index);
                  str3.append(")");


// Messages (1 + 4k)

      //Receives
      for ( int i = 0; i <= k ; i++ ) {
      m = new Message((1+4*i),(row_m + column_n - 3 + 10*i),str3, 0, " ");
      (*v).Received_Messages.Add(m);
      }

      //Sends
```

```
      for (  i = 0; i <= k ; i++ ) {
      m = new Message((1+4*i),0," ",(row_m + column_n + 10*i), str1);
      (*v).Sent_Messages.Add(m);
      }

// Messages (2 + 4k)

      //Receives
      for (  i = 0; i <= k ; i++ ) {
      m = new Message((2+4*i),(row_m + column_n - 2 + 10*i),str1, 0, " ");
      (*v).Received_Messages.Add(m);
      }

      //Sends
      for (  i = 0; i <= k ; i++ ) {
      m = new Message((2+4*i),0, " ", (row_m + column_n - 1 + 10*i),str2);
      (*v).Sent_Messages.Add(m);
      }

// Messages (3 + 4k)

      //Receives
      for (  i = 0; i <= k ; i++ ) {
      m = new Message((3 + 4*i),(row_m + column_n + 11 + 10*i),str2,0, " ");
      (*v).Received_Messages.Add(m);
      }

// Messages (4 + 4k)

      //Receives
      for (  i = 0; i <= k ; i++ ) {
      m = new Message((4 + 4*i),(row_m + column_n + 4 + 10*i), str1,0, " " );
      (*v).Received_Messages.Add(m);
      }

      //Sends
      for (  i = 0; i <= k ; i++ ) {
      m = new Message((4 + 4*i),0," ",(row_m + column_n + 5 + 10*i), str2);
      (*v).Sent_Messages.Add(m);
      }

      for (  i = 0; i <= k ; i++ ) {
      m = new Message((4 + 4*i),0," ",(row_m + column_n + 13 + 10*i), str3);
      (*v).Sent_Messages.Add(m);
      }

// Messages (0)

      //Receives
      for (  i = 0; i <= k ; i++ ) {
      m = new Message(0,(row_m + column_n + 2 + 10*i),str2, 0, " ");
      (*v).Received_Messages.Add(m);
      }

      m = new Message(0,(row_m + column_n + 1), str2, 0, " " );
      (*v).Received_Messages.Add(m);

      m = new Message(0,(row_m + column_n - 3 + 10*(k+1)), str3, 0, " ");
      (*v).Received_Messages.Add(m);

      //Sends
      for (  i = 0; i <= k ; i++ ) {
            m = new Message(0,0," ",(row_m + column_n  + 6 + 10*i), str1);
            (*v).Sent_Messages.Add(m);
```

```
            }

      m = new Message(0,0," ",(row_m + column_n + 3),str3);
      (*v).Sent_Messages.Add(m);

      m = new Message(0,0, " ", (row_m + column_n - 1 + 10*(k+1)), str2);
      (*v).Sent_Messages.Add(m);

      str1 = "";
      str2 = "";
      str3 = "";

}


////////////////////////////////////////////////////////////////////////////////

void calculate_vertex_row_column_minus_1 (Vertex *ver) {

// Vertex v(row_m,column_n-1)

Vertex *v = ver;
Message *m;

                  str1.append("v(");
                  str_index = itos(row_m);
                  str1.append(str_index);
                  str1.append(",");
                  str_index = itos(column_n-2);
                  str1.append(str_index);
                  str1.append(")");


                  str2.append("v(");
                  str_index = itos(row_m);
                  str2.append(str_index);
                  str2.append(",");
                  str_index = itos(column_n);
                  str2.append(str_index);
                  str2.append(")");


                  str3.append("v(");
                  str_index = itos(row_m-1);
                  str3.append(str_index);
                  str3.append(",");
                  str_index = itos(column_n-1);
                  str3.append(str_index);
                  str3.append(")");

// Messages (1 + 4k)

      //Receives
      for ( int i = 0; i <= k ; i++ ) {
      m = new Message((1+4*i),(row_m + column_n + 3 + 10*i),str2, 0, " ");
      (*v).Received_Messages.Add(m);
      }

      //Sends
      for (  i = 0; i <= k ; i++ ) {
      m = new Message((1+4*i),0," ",(row_m + column_n + 8 + 10*i), str3);
      (*v).Sent_Messages.Add(m);
      }
```

```
// Messages (2 + 4k)

      //Receives
      for (  i = 0; i <= k ; i++ ) {
      m = new Message((2+4*i),(row_m + column_n - 1 + 10*i),str1, 0, " ");
      (*v).Received_Messages.Add(m);
      }

      //Sends
      for (  i = 0; i <= k ; i++ ) {

      m = new Message((2+4*i),0," ",(row_m + column_n + 10*i), str2);
      (*v).Sent_Messages.Add(m);
      }

// Messages (3 + 4k)

      //Receives
      for (  i = 0; i <= k ; i++ ) {
      m = new Message((3 + 4*i),(row_m + column_n + 4 + 10*i),str3,0, " ");
      (*v).Received_Messages.Add(m);
      }

      //Sends
      for (  i = 0; i <= k ; i++ ) {
      m = new Message((3+4*i),0," ",(row_m + column_n + 11 + 10*i), str3);
      (*v).Sent_Messages.Add(m);
      }

// Messages (4 + 4k)

      //Receives
      for (  i = 0; i <= k ; i++ ) {
      m = new Message((4 + 4*i),(row_m + column_n + 5 + 10*i), str1,0, " " );
      (*v).Received_Messages.Add(m);
      }

      //Sends
      for (  i = 0; i <= k ; i++ ) {
      m = new Message((4+4*i),0, " ", (row_m + column_n + 6 + 10*i),str2);
      (*v).Sent_Messages.Add(m);
      }

// Messages (0)

      //Receives
      for (  i = 0; i <= k ; i++ ) {
            m = new Message(0,(row_m + column_n + 7 + 10*i),str2, 0, " ");
            (*v).Received_Messages.Add(m);
            }
      m = new Message(0,(row_m + column_n - 1 + 10*(k+1) ), str1, 0, " );
      (*v).Received_Messages.Add(m);

      //Sends
      for (  i = 0; i <= k ; i++ ) {
            m = new Message(0,0," ",(row_m + column_n + 2 + 10*i), str1);
            (*v).Sent_Messages.Add(m);
            }

            m = new Message(0,0," ",(row_m + column_n + 1),str1);
            (*v).Sent_Messages.Add(m);


      str1 = "";
```

```
      str2 = "";
      str3 = "";

}

////////////////////////////////////////////////////////////////////////////////

void calculate_vertex_row_column (Vertex *ver) {

// Vertex v(row_m,column_n)

Vertex *v = ver;
Message *m;


                  str1.append("v(");
                  str_index = itos(row_m);
                  str1.append(str_index);
                  str1.append(",");
                  str_index = itos(column_n-1);
                  str1.append(str_index);
                  str1.append(")");


                  str3.append("v(");
                  str_index = itos(row_m-1);
                  str3.append(str_index);
                  str3.append(",");
                  str_index = itos(column_n);
                  str3.append(str_index);
                  str3.append(")");


// Messages (1 + 4k)

      //Receives
      for ( int i = 0; i <= k ; i++ ) {
      m = new Message((1+4*i),(row_m + column_n - 2 + 10*i),str3, 0, " ");
      (*v).Received_Messages.Add(m);
      }

      //Sends
      for (  i = 0; i <= k ; i++ ) {
      m = new Message((1+4*i),0, " ", (row_m + column_n + 3 + 10*i),str1);
      (*v).Sent_Messages.Add(m);
      }

// Messages (2 + 4k)

      //Receives
      for (  i = 0; i <= k ; i++ ) {
            m = new Message((2+4*i),(row_m + column_n + 10*i),str1, 0, " ");
            (*v).Received_Messages.Add(m);
            }

// Messages (3 + 4k)

      //Receives
      for (  i = 0; i <= k ; i++ ) {
      m = new Message((3 + 4*i),(row_m + column_n + 4 + 10*i),str3,0, " ");
      (*v).Received_Messages.Add(m);
      }

// Messages (4 + 4k)
```

```
      //Receives
      for (  i = 0; i <= k ; i++ ) {
      m = new Message((4 + 4*i),(row_m + column_n + 6 + 10*i), str1,0, " " );
      (*v).Received_Messages.Add(m);
      }

      //Sends
      for (  i = 0; i <= k ; i++ ) {
      m = new Message((4+4*i),0, " ", (row_m + column_n + 12 + 10*i),str3);
      (*v).Sent_Messages.Add(m);
      }

// Messages (0)

      //Receives
      m = new Message(0,(row_m + column_n - 2 + 10*(k+1)), str3, 0, " ");
      (*v).Received_Messages.Add(m);

      //Sends
      for (  i = 0; i <= k ; i++ ) {
      m = new Message(0,0," ",(row_m + column_n  + 5 + 10*i), str3);
      (*v).Sent_Messages.Add(m);
      }

      m = new Message(0,0," ",(row_m + column_n  + 2), str3);
      (*v).Sent_Messages.Add(m);

      for (  i = 0; i <= k ; i++ ) {
            m = new Message(0,0," ",(row_m + column_n + 7 + 10*i),str1);
            (*v).Sent_Messages.Add(m);
            }

      str1 = "";
      str3 = "";

}

////////////////////////////////////////////////////////////////////////////////

void calculate_vertex_row_minus_1_column(Vertex *ver) {

// Vertex v(row_m-1,column_n)

Vertex *v=ver;
Message *m;


                  str1.append("v(");
                  str_index = itos(row_m-2);
                  str1.append(str_index);
                  str1.append(",");
                  str_index = itos(column_n);
                  str1.append(str_index);
                  str1.append(")");


                  str2.append("v(");
                  str_index = itos(row_m);
                  str2.append(str_index);
                  str2.append(",");
                  str_index = itos(column_n);
                  str2.append(str_index);
```

```
                    str2.append(")");


                    str3.append("v(");
                    str_index = itos(row_m-1);
                    str3.append(str_index);
                    str3.append(",");
                    str_index = itos(column_n-1);
                    str3.append(str_index);
                    str3.append(")");


// Messages (1 + 4k)

      //Receives
      for ( int j = 0; j <= k ; j++ ) {
      m = new Message((1+4*j),(row_m + column_n - 3 + 10*j),str1, 0, " ");
      (*v).Received_Messages.Add(m);
      }

      //Sends
      for (  j = 0; j <= k ; j++ ) {
      m = new Message((1 + 4*j),0," ",(row_m + column_n - 2 + 10*j), str2);
      (*v).Sent_Messages.Add(m);
      }

// Messages (2 + 4k)

      //Receives
      for (  j = 0; j <= k ; j++ ) {
      m = new Message((2+4*j),(row_m + column_n + 10*j),str3, 0, " ");
      (*v).Received_Messages.Add(m);
      }

      //Sends
      for (  j = 0; j <= k ; j++ ) {
      m = new Message((2 + 4*j),0," ",(row_m + column_n + 1 + 10*j), str1);
      (*v).Sent_Messages.Add(m);
      }

// Messages (3 + 4k)

      //Receives
      for (  j = 0; j <= k ; j++ ) {
      m = new Message((3 + 4*j),(row_m + column_n + 3 + 10*j),str1,0, " ");
      (*v).Received_Messages.Add(m);
      }

      //Sends
      for (  j = 0; j <= k ; j++ ) {
      m = new Message((3 + 4*j),0," ",(row_m + column_n + 4 + 10*j), str2);
      (*v).Sent_Messages.Add(m);
      }

// Messages (4 + 4k)

      //Receives
      for (  j = 0; j <= k ; j++ ) {
      m = new Message((4 + 4*j),(row_m + column_n + 12 + 10*j), str2,0, " " );
      (*v).Received_Messages.Add(m);
      }

      //Sends
      for (  j = 0; j <= k ; j++ ) {
```

```
      m = new Message((4 + 4*j),0," ",(row_m + column_n + 16 + 10*j), str3);
      (*v).Sent_Messages.Add(m);
      }

// Messages (0)

      //Receives
      for (  j = 0; j <= k ; j++ ) {
            m = new Message(0,(row_m + column_n + 5 + 10*j), str2, 0, " ");
            (*v).Received_Messages.Add(m);
            }

            m = new Message(0,(row_m + column_n + 2), str2, 0, " ");
            (*v).Received_Messages.Add(m);

            m = new Message(0,(row_m + column_n + 10*(k+1)), str3, 0, " ");
            (*v).Received_Messages.Add(m);


      //Sends
      for (  j = 0; j <= k ; j++ ) {
      m = new Message(0,0," ",(row_m + column_n - 1 + 10*j), str1);
      (*v).Sent_Messages.Add(m);
      }

      m = new Message(0,0, " ", (row_m + column_n - 2 + 10*(k+1)), str2);
      (*v).Sent_Messages.Add(m);

      m = new Message(0,0, " ", (row_m + column_n + 6), str3);
      (*v).Sent_Messages.Add(m);

      str1 = "";
      str2 = "";
      str3 = "";

}

////////////////////////////////////////////////////////////////////////////////

void calculate_vertex_row_minus_1_column_minus_1(Vertex *ver){

//Vertex v(row_m-1,column_n - 1)

Vertex *v = ver;
Message *m;


                  str1.append("v(");
                  str2.append("v(");
                  str3.append("v(");
                  str4.append("v(");


                  str_index = itos(row_m - 2);
                  str1.append(str_index);
                  str_index = itos(row_m);
                  str2.append(str_index);
                  str1.append(",");
                  str2.append(",");

                  str_index = itos(column_n - 1);
                  str1.append(str_index);
                  str2.append(str_index);
                  str1.append(")");
```

```
                    str2.append(")");

                    str_index = itos(row_m - 1);
                    str3.append(str_index);
                    str4.append(str_index);
                    str3.append(",");
                    str4.append(",");

                    str_index = itos(column_n - 2);
                    str3.append(str_index);

                    str_index = itos(column_n);
                    str4.append(str_index);
                    str3.append(")");
                    str4.append(")");


// Messages (1 + 4k)

      //Receives
      for ( int i = 0; i <= k ; i++ ) {
      m = new Message((1+4*i),(row_m + column_n + 8 + 10*i),str2, 0, " ");
      (*v).Received_Messages.Add(m);
      }

// Messages (2 + 4k)

      //Receives
      for (  i = 0; i <= k ; i++ ) {
      m = new Message((2+4*i),(row_m + column_n - 1 + 10*i),str3, 0, " ");
      (*v).Received_Messages.Add(m);
      }

      //Sends
      for (  i = 0; i <= k ; i++ ) {
      m = new Message((2+4*i),0," ",(row_m + column_n + 10*i), str4);
      (*v).Sent_Messages.Add(m);
      }

      for (  i = 0; i <= k ; i++ ) {
      m = new Message((2+4*i),0," ",(row_m + column_n + 7 + 10*i), str1);
      (*v).Sent_Messages.Add(m);
      }

// Messages (3 + 4k)

      //Receives
      for (  i = 0; i <= k ; i++ ) {
      m = new Message((3 + 4*i),(row_m + column_n + 3 + 10*i),str1,0, " ");
      (*v).Received_Messages.Add(m);
      }

      //Sends
      for (  i = 0; i <= k ; i++ ) {
      m = new Message((3 + 4*i),0," ",(row_m + column_n + 4 + 10*i), str2);
      (*v).Sent_Messages.Add(m);
      }

      for (  i = 0; i <= k ; i++ ) {
      m = new Message((3 + 4*i),0," ",(row_m + column_n + 5 + 10*i), str3);
      (*v).Sent_Messages.Add(m);
      }

// Messages (4 + 4k)
```

```
      //Receives
      for (  i = 0; i <= k ; i++ ) {
      m = new Message((4 + 4*i),(row_m + column_n + 16 + 10*i), str4,0, " " );
      (*v).Received_Messages.Add(m);
      }


// Messages (0)

      //Receives
      m = new Message(0,(row_m + column_n + 6), str4, 0, " " );
      (*v).Received_Messages.Add(m);

      //Sends
      m = new Message(0,0, " ", (row_m + column_n + 10*(k+1)), str4);
      (*v).Sent_Messages.Add(m);

      str1 = "";
      str2 = "";
      str3 = "";
      str4 = "";



}


////////////////////////////////////////////////////////////////////////////////

void calculate_vertex_row_minus_1_column_minus_2(Vertex *ver){

// Vertex v(row_m-1,column_n - 2)
Vertex *v = ver;
Message *m;

                  str1.append("v(");
                  str2.append("v(");
                  str3.append("v(");
                  str4.append("v(");


                  str_index = itos(row_m - 2);
                  str1.append(str_index);
                  str_index = itos(row_m);
                  str2.append(str_index);
                  str1.append(",");
                  str2.append(",");

                  str_index = itos(column_n - 2);
                  str1.append(str_index);
                  str2.append(str_index);
                  str1.append(")");
                  str2.append(")");

                  str_index = itos(row_m - 1);
                  str3.append(str_index);
                  str4.append(str_index);
                  str3.append(",");
                  str4.append(",");

                  str_index = itos(column_n - 3);
                  str3.append(str_index);
                  str_index = itos(column_n - 1);
```

```
                    str4.append(str_index);
                    str3.append(")");
                    str4.append(")");

// Messages (1 + 4k)

      //Receives
      for ( int i = 0; i <= k ; i++ ) {
      m = new Message((1+4*i),(row_m + column_n - 4 + 10*i),str1, 0, " ");
      (*v).Received_Messages.Add(m);
      }

      //Sends
      for (  i = 0; i <= k ; i++ ) {
      m = new Message((1+4*i),0," ",(row_m + column_n - 3 + 10*i), str2);
      (*v).Sent_Messages.Add(m);
      }

// Messages (2 + 4k)

      //Receives
      for (  i = 0; i <= k ; i++ ) {
      m = new Message((2+4*i),(row_m + column_n - 2 + 10*i),str3, 0, " ");
      (*v).Received_Messages.Add(m);
      }

      //Sends
      for (  i = 0; i <= k ; i++ ) {
      m = new Message((2+4*i),0," ",(row_m + column_n - 1 + 10*i), str4);
      (*v).Sent_Messages.Add(m);
      }

// Messages (3 + 4k)

      //Receives
      for (  i = 0; i <= k ; i++ ) {
      m = new Message((3 + 4*i),(row_m + column_n + 5 + 10*i),str4,0, " ");
      (*v).Received_Messages.Add(m);
      }

      //Sends
      for (  i = 0; i <= k ; i++ ) {
      m = new Message((3+4*i),0," ",(row_m + column_n + 12 + 10*i), str1);
      (*v).Sent_Messages.Add(m);
      }

// Messages (4 + 4k)

      //Receives
      for (  i = 0; i <= k ; i++ ) {
      m = new Message((4 + 4*i),(row_m + column_n + 13 + 10*i), str2,0, " " );
      (*v).Received_Messages.Add(m);
      }

      //Sends
      for (  i = 0; i <= k ; i++ ) {
      m = new Message((4+4*i),0," ",(row_m + column_n + 14 + 10*i), str3);
      (*v).Sent_Messages.Add(m);
      }

// Messages (0)

      //Receives
      m = new Message(0,(row_m + column_n + 3), str2, 0, " " );
```

```
        (*v).Received_Messages.Add(m);

        m = new Message(0,(row_m + column_n - 2 + 10*(k+1)), str3, 0, " " );
        (*v).Received_Messages.Add(m);

        m = new Message(0,(row_m + column_n - 4 + 10*(k+1)), str1, 0, " " );
        (*v).Received_Messages.Add(m);

        //Sends
        m = new Message(0,0, " ", (row_m + column_n - 3 + 10*(k+1)), str2);
        (*v).Sent_Messages.Add(m);

        m = new Message(0,0, " ", (row_m + column_n + 4), str3);
        (*v).Sent_Messages.Add(m);

        m = new Message(0,0, " ", (row_m + column_n + 2), str1);
        (*v).Sent_Messages.Add(m);

        str1 = "";
        str2 = "";
        str3 = "";
        str4 = "";

}


////////////////////////////////////////////////////////////////////////////////

void calculate_vertex_row_minus_1_column_minus_3(Vertex *ver) {

// Vertex v(row_m-1,column_n - 3)

Vertex *v = ver;
Message *m;
                    str1.append("v(");
                    str2.append("v(");
                    str3.append("v(");
                    str4.append("v(");

                    str_index = itos(row_m - 2);
                    str1.append(str_index);

                    str_index = itos(row_m);
                    str2.append(str_index);
                    str1.append(",");
                    str2.append(",");

                    str_index = itos(column_n - 3);
                    str1.append(str_index);
                    str2.append(str_index);
                    str1.append(")");
                    str2.append(")");

                    str_index = itos(row_m - 1);
                    str3.append(str_index);
                    str4.append(str_index);
                    str3.append(",");
                    str4.append(",");

                    str_index = itos(column_n - 4);
                    str3.append(str_index);
                    str_index = itos(column_n - 2);
                    str4.append(str_index);
```

```
                    str3.append(")");
                    str4.append(")");


// Messages (1 + 4k)

      //Receives
      for ( int i = 0; i <= k ; i++ ) {
      m = new Message((1+4*i),(row_m + column_n + 5 + 10*i),str2, 0, " ");
      (*v).Received_Messages.Add(m);
      }

// Messages (2 + 4k)

      //Receives
      for (  i = 0; i <= k ; i++ ) {
      m = new Message((2+4*i),(row_m + column_n - 3 + 10*i),str3, 0, " ");
      (*v).Received_Messages.Add(m);
      }

      //Sends
      for (  i = 0; i <= k ; i++ ) {
      m = new Message((2+4*i),0," ",(row_m + column_n - 2 + 10*i), str4);
      (*v).Sent_Messages.Add(m);
      }

      for (  i = 0; i <= k ; i++ ) {
      m = new Message((2+4*i),0," ",(row_m + column_n - 1 + 10*i), str1);
      (*v).Sent_Messages.Add(m);
      }

// Messages (3 + 4k)

      //Receives
      for (  i = 0; i <= k ; i++ ) {
      m = new Message((3 + 4*i),(row_m + column_n + 1 + 10*i),str1,0, " ");
      (*v).Received_Messages.Add(m);
      }

      //Sends
      for (  i = 0; i <= k ; i++ ) {
      m = new Message((3+4*i),0," ",(row_m + column_n + 2 + 10*i), str2);
      (*v).Sent_Messages.Add(m);
      }

      for (  i = 0; i <= k ; i++ ) {
      m = new Message((3+4*i),0," ",(row_m + column_n + 3 + 10*i), str3);
      (*v).Sent_Messages.Add(m);
      }

// Messages (4 + 4k)

      //Receives
      for (  i = 0; i <= k ; i++ ) {
      m = new Message((4 + 4*i),(row_m + column_n + 14 + 10*i), str4,0, " " );
      (*v).Received_Messages.Add(m);
      }

// Messages (0)

      //Receives
            m = new Message(0,(row_m + column_n + 4), str4, 0, " " );
            (*v).Received_Messages.Add(m);
```

```
      //Sends
      m = new Message(0,0, " ", (row_m + column_n - 2 + 10*(k+1)), str4);
      (*v).Sent_Messages.Add(m);

      str1 = "";
      str2 = "";
      str3 = "";
      str4 = "";
}

////////////////////////////////////////////////////////////////////////////////

//////---------------------------------- MAIN --------------------------------


void main ( int argc, char **argv )

{

Vertex_list vertices;
Message *m;

cout<< " Please enter number of rows : " ;
cin>> row_m;

cout<< " \nPlease enter number of rows : " ;
cin>>column_n;

int mod = 1;
while ( mod != 0 ) {
      cout<< " \nPlease enter number of messages multiple of 4 : " ;
      cin>>M;
      mod = (M%4);
} // end of while ( mod != 0 ) loop

k = (M-4)/4;


// Creating a list of vertices
for ( int i = 1; i <= row_m; i++ )
      for (int j = 1; j <= column_n; j++ ) {
            Vertex *v = new Vertex(i,j);
            vertices.Add(v);
            }


////////////////////////////////////////////////////////////////////////////////

// Vertex v(1,1)

Vertex *v = vertices.Find_Vertex(1,1);

// Messages (1 + 4k)

      //Sends
      for (  i = 0; i <= k ; i++ ) {
            m = new Message((1+4*i),0," ",(1+ 10*i), "v(1,2)");
            (*v).Sent_Messages.Add(m);
            }

// Messages (2 + 4k)

      //Sends
      for ( i = 0; i <= k ; i++ ) {
```

```
            m = new Message((2 + 4*i),0," ",( 3 + 10*i),"v(2,1)");
            (*v).Sent_Messages.Add(m);
            }

// Messages (3 + 4k)

      //Sends
      for ( i = 0; i <= k ; i++ ) {
            m = new Message((3 + 4*i),0," " ,( 7 + 10*i),"v(1,2)");
            (*v).Sent_Messages.Add(m);
            }



// Messages (4 + 4k)

      //Sends
      for ( i = 0; i <= k ; i++ ) {
            m = new Message((4 + 4*i),0, " ", ( 9 + 10*i),"v(2,1)");
            (*v).Sent_Messages.Add(m);
            }

// Messages 0

      //Receives
      for ( i = 0; i <= k ; i++ ) {
            m = new Message(0,( 5 + 10*i),"v(1,2)",0, " ");
            (*v).Received_Messages.Add(m);
            }

      for ( i = 0; i <= k ; i++ ) {
            m = new Message(0,( 10 + 10*i),"v(1,2)",0, " ");
            (*v).Received_Messages.Add(m);
            }

      for ( i = 0; i <= k ; i++ ) {
            m = new Message(0,( 2 + 10*i),"v(2,1)",0, " ");
            (*v).Received_Messages.Add(m);
            }

      for ( i = 0; i <= k ; i++ ) {
            m = new Message(0,( 6 + 10*i),"v(2,1)",0, " ");
            (*v).Received_Messages.Add(m);
            }

////////////////////////////////////////////////////////////////////////////////

// Vertex v(1,2)

v = vertices.Find_Vertex(1,2);

// Messages (1 + 4k)

      //Receives
      for ( i = 0; i <= k ; i++ ) {
            m = new Message((1+4*i),(1+ 10*i),"v(1,1)", 0, " ");
            (*v).Received_Messages.Add(m);
            }

      //Sends
      for (  i = 0; i <= k ; i++ ) {
            m = new Message((1+4*i),0," ",(2 + 10*i), "v(1,3)");
            (*v).Sent_Messages.Add(m);
```

```
            }

      for (  i = 0; i <= k ; i++ ) {
            m = new Message((1+4*i),0," ", (3 + 10*i), "v(2,2)");
            (*v).Sent_Messages.Add(m);
            }

// Messages (2 + 4k)

      //Receives
      for (  i = 0; i <= k ; i++ ) {
            m = new Message((2+4*i),(16 + 10*i),"v(1,3)", 0, " ");
            (*v).Received_Messages.Add(m);
            }

// Messages (3 + 4k)

      //Receives
      for (  i = 0; i <= k ; i++ ) {
            m = new Message((3 + 4*i),(7 + 10*i),"v(1,1)",0, " ");
            (*v).Received_Messages.Add(m);
            }

      //Sends
      for (  i = 0; i <= k ; i++ ) {
            m = new Message((3 + 4*i),0," ",(8 + 10*i), "v(1,3)");
            (*v).Sent_Messages.Add(m);
            }

// Messages (4 + 4k)

      //Receives
      for (  i = 0; i <= k ; i++ ) {
            m = new Message((4 + 4*i),(19 + 10*i), "v(2,2)",0, " " );
            (*v).Received_Messages.Add(m);
            }

// Messages (0)

      //Receives
      for (  i = 0; i <= k ; i++ ) {
            m = new Message(0,(4 + 10*i),"v(1,3)", 0, " ");
            (*v).Received_Messages.Add(m);
            }

            m = new Message(0,6, "v(1,3)", 0, " " );
            (*v).Received_Messages.Add(m);

            m = new Message(0,9, "v(2,2)", 0, " ");
            (*v).Received_Messages.Add(m);

      //Sends
      for (  i = 0; i <= k ; i++ ) {
            m = new Message(0,0," ",(5 + 10*i), "v(1,1)");
            (*v).Sent_Messages.Add(m);
            }

      for (  i = 0; i <= k ; i++ ) {
            m = new Message(0,0," ",(10 + 10*i),"v(1,1)");
            (*v).Sent_Messages.Add(m);
            }

            m = new Message(0,0," ",(3 + 10*(k+1)),"v(2,2)");
            (*v).Sent_Messages.Add(m);
```

```
            m = new Message(0,0," ",(8 + 10*(k+1)),"v(1,3)");
            (*v).Sent_Messages.Add(m);

////////////////////////////////////////////////////////////////////////////////

// Vertex v(1,j) (3 <= j <= (column_n - 1)

for ( int j = 3; j <= (column_n-1); j++)

{
v = vertices.Find_Vertex(1,j);

if ( j%2 != 0 )  // j is ODD
{
            str1.append("v(1,");
            str_index = itos(j-1);
            str1.append(str_index);
            str1.append(")");

            str2.append("v(1,");
            str_index = itos(j+1);
            str2.append(str_index);
            str2.append(")");

            str3.append("v(2,");
            str_index = itos(j);
            str3.append(str_index);
            str3.append(")");

// Messages (1 + 4k)

      //Receives
      for ( i = 0; i <= k ; i++ ) {
            m = new Message((1+4*i),(j - 1 + 10*i),str1, 0, " ");
            (*v).Received_Messages.Add(m);
            }

      //Sends
      for (  i = 0; i <= k ; i++ ) {
            m = new Message((1+4*i),0," ",(j + 10*i), str2);
            (*v).Sent_Messages.Add(m);
            }

// Messages (2 + 4k)

      //Receives
      for (  i = 0; i <= k ; i++ ) {
            m = new Message((2+4*i),(j + 12 + 10*i),str3, 0, " ");
            (*v).Received_Messages.Add(m);
            }

      //Sends
      for (  i = 0; i <= k ; i++ ) {
            m = new Message((2+4*i),0, " ", (j + 13 + 10*i),str1);
            (*v).Sent_Messages.Add(m);
            }

// Messages (3 + 4k)

      //Receives
      for (  i = 0; i <= k ; i++ ) {
            m = new Message((3 + 4*i),(j + 5 + 10*i),str1,0, " ");
```

```
            (*v).Received_Messages.Add(m);
            }

      //Sends
      for (  i = 0; i <= k ; i++ ) {
            m = new Message((3 + 4*i),0," ",(j + 6 + 10*i), str2);
            (*v).Sent_Messages.Add(m);
            }

      for (  i = 0; i <= k ; i++ ) {
            m = new Message((3 + 4*i),0," ",(j + 7 + 10*i), str3);
            (*v).Sent_Messages.Add(m);
            }

// Messages (4 + 4k)

      //Receives
      for (  i = 0; i <= k ; i++ ) {
            m = new Message((4 + 4*i),(j + 24 + 10*i), str2,0, " " );
            (*v).Received_Messages.Add(m);
            }

// Messages (0)

      //Receives
      for (  i = 0; i <= k ; i++ ) {
            m = new Message(0,(j + 8 + 10*i),str2, 0, " ");
            (*v).Received_Messages.Add(m);
            }

            m = new Message(0,(j + 4), str2, 0, " " );
            (*v).Received_Messages.Add(m);

            m = new Message(0,(j + 14), str2, 0, " ");
            (*v).Received_Messages.Add(m);

            m = new Message(0,(j + 5 + 10*(k+1)), str1, 0, " ");
            (*v).Received_Messages.Add(m);

      //Sends
      for (  i = 0; i <= k ; i++ ) {
            m = new Message(0,0," ",(j + 1 + 10*i), str1);
            (*v).Sent_Messages.Add(m);
            }

            m = new Message(0,0," ",(j + 3),str1);
            (*v).Sent_Messages.Add(m);

            m = new Message(0,0, " ", (j + 10*(k+1)), str2);
            (*v).Sent_Messages.Add(m);

            m = new Message(0,0, " ", (j + 10*(k+2)), str2);
            (*v).Sent_Messages.Add(m);

      str1 = "";
      str2 = "";
      str3 = "";


      }
      else {  // j is EVEN


            str1.append("v(1,");
```

```
            str_index = itos(j-1);
            str1.append(str_index);
            str1.append(")");

            str2.append("v(1,");
            str_index = itos(j+1);
            str2.append(str_index);
            str2.append(")");

            str3.append("v(2,");
            str_index = itos(j);
            str3.append(str_index);
            str3.append(")");

// Messages (1 + 4k)

      //Receives
      for ( i = 0; i <= k ; i++ ) {
            m = new Message((1+4*i),(j - 1 + 10*i),str1, 0, " ");
            (*v).Received_Messages.Add(m);
            }

      //Sends
      for (  i = 0; i <= k ; i++ ) {
            m = new Message((1+4*i),0," ",(j + 10*i), str2);
            (*v).Sent_Messages.Add(m);
            }

      for (  i = 0; i <= k ; i++ ) {
            m = new Message((1+4*i),0," ",(j + 1 + 10*i), str3);
            (*v).Sent_Messages.Add(m);
            }

// Messages (2 + 4k)

      //Receives
      for (  i = 0; i <= k ; i++ ) {
            m = new Message((2+4*i),(j + 14 + 10*i),str2, 0, " ");
            (*v).Received_Messages.Add(m);
            }

// Messages (3 + 4k)

      //Receives
      for (  i = 0; i <= k ; i++ ) {
            m = new Message((3 + 4*i),(j + 5 + 10*i),str1,0, " ");
            (*v).Received_Messages.Add(m);
            }

      //Sends
      for (  i = 0; i <= k ; i++ ) {
            m = new Message((3 + 4*i),0," ",(j + 6 + 10*i), str2);
            (*v).Sent_Messages.Add(m);
            }

// Messages (4 + 4k)

      //Receives
      for (  i = 0; i <= k ; i++ ) {
            m = new Message((4 + 4*i),(j + 18 + 10*i), str3,0, " " );
            (*v).Received_Messages.Add(m);
            }

      //Sends
```

```
      for (  i = 0; i <= k ; i++ ) {
            m = new Message((4+4*i),0, " ", (j + 23 + 10*i),str1);
            (*v).Sent_Messages.Add(m);
            }

// Messages (0)

      //Receives
      for (  i = 0; i <= k ; i++ ) {
            m = new Message(0,(j + 2 + 10*i),str2, 0, " ");
            (*v).Received_Messages.Add(m);
            }

            m = new Message(0,(j + 4), str2, 0, " " );
            (*v).Received_Messages.Add(m);

            m = new Message(0,(j - 1 + 10*(k+1) ), str1, 0, " ");
            (*v).Received_Messages.Add(m);

            m = new Message(0,(j - 1 + 10*(k+2)), str1, 0, " ");
            (*v).Received_Messages.Add(m);

            m = new Message(0,(j + 8), str3, 0, " ");
            (*v).Received_Messages.Add(m);

      //Sends
      for (  i = 0; i <= k ; i++ ) {
            m = new Message(0,0," ",(j + 7 + 10*i), str1);
            (*v).Sent_Messages.Add(m);
            }

            m = new Message(0,0," ",(j + 3),str1);
            (*v).Sent_Messages.Add(m);

            m = new Message(0,0," ",(j + 13),str1);
            (*v).Sent_Messages.Add(m);

            m = new Message(0,0, " ", (j + 6 + 10*(k+1)), str2);
            (*v).Sent_Messages.Add(m);

            m = new Message(0,0, " ", (j + 1 + 10*(k+1)), str3);
            (*v).Sent_Messages.Add(m);

      str1 = "";
      str2 = "";
      str3 = "";

} //end of ELSE


} //end of j FOR loop


////////////////////////////////////////////////////////////////////////////////

// Vertex v(1,column_n)

 v = vertices.Find_Vertex(1,column_n);

                  str1.append("v(1,");
                  str_index = itos(column_n - 1);
                  str1.append(str_index);
                  str1.append(")");
```

```
                  str3.append("v(2,");
                  str_index = itos(column_n);
                  str3.append(str_index);
                  str3.append(")");

// Messages (1 + 4k)

      //Receives
      for (  i = 0; i <= k ; i++ ) {
            m = new Message((1+4*i),( column_n - 1 + 10*i),str1,0, " ");
            (*v).Received_Messages.Add(m);
            }

      //Sends
      for (  i = 0; i <= k ; i++ ) {
            m = new Message((1+4*i),0," ",(column_n + 10*i), str3);
            (*v).Sent_Messages.Add(m);
            }

// Messages (2 + 4k)

      //Receives
      for (  i = 0; i <= k ; i++ ) {
            m = new Message((2+4*i),( column_n + 4 + 10*i),str3,0, " ");
            (*v).Received_Messages.Add(m);
            }

// Messages (3 + 4k)

      //Receives
      for (  i = 0; i <= k ; i++ ) {
            m = new Message((3+4*i),( column_n + 5 + 10*i),str1,0, " ");
            (*v).Received_Messages.Add(m);
            }

      //Sends
      for ( i = 0; i <= k ; i++ ) {
            m = new Message((3 + 4*i),0," " ,( column_n + 6 + 10*i),str3);
            (*v).Sent_Messages.Add(m);
            }

// Messages (4 + 4k)

      //Receives
      for (  i = 0; i <= k ; i++ ) {
            m = new Message((4+4*i),( column_n + 22 + 10*i),str3,0, " ");
            (*v).Received_Messages.Add(m);
            }

      //Sends
      for ( i = 0; i <= k ; i++ ) {
            m = new Message((4 + 4*i),0, " ", ( column_n + 23 + 10*i),str1);
            (*v).Sent_Messages.Add(m);
            }

// Messages 0

      //Receives
            m = new Message(0,( column_n + 2),str3,0, " ");
            (*v).Received_Messages.Add(m);

            m = new Message(0,(column_n + 12),str3,0, " ");
            (*v).Received_Messages.Add(m);
```

```
            m = new Message(0,( column_n - 1 + 10*(k+1)),str1,0, " ");
            (*v).Received_Messages.Add(m);

            m = new Message(0,( column_n - 1 + 10*(k+2)),str1,0, " ");
            (*v).Received_Messages.Add(m);

      //Sends
      for ( i = 0; i <= k ; i++ ) {
            m = new Message(0,0, " ",( column_n + 7 + 10*i),str1);
            (*v).Sent_Messages.Add(m);
            }

            m = new Message(0,0, " ",( column_n + 3),str1);
            (*v).Sent_Messages.Add(m);

            m = new Message(0,0, " ",( column_n + 13),str1);
            (*v).Sent_Messages.Add(m);

            m = new Message(0,0, " ",( column_n + 10*(k+1)),str3);
            (*v).Sent_Messages.Add(m);

            m = new Message(0,0, " ",( column_n + 10*(k+2)),str3);
            (*v).Sent_Messages.Add(m);

str1 = "";
str2 = "";
str3 = "";

////////////////////////////////////////////////////////////////////////////////

// Vertex v(2,1)

 v = vertices.Find_Vertex(2,1);

// Messages (1 + 4k)

      //Receives
      for (  i = 0; i <= k ; i++ ) {
            m = new Message((1+4*i),(7 + 10*i),"v(3,1)",0, " ");
            (*v).Received_Messages.Add(m);
            }

// Messages (2 + 4k)

      //Receives
      for (  i = 0; i <= k ; i++ ) {
            m = new Message((2+4*i),(3 + 10*i),"v(1,1)",0, " ");
            (*v).Received_Messages.Add(m);
            }

      //Sends
      for (  i = 0; i <= k ; i++ ) {
            m = new Message((2+4*i),0," ",(4 + 10*i), "v(3,1)");
            (*v).Sent_Messages.Add(m);
            }

      for (  i = 0; i <= k ; i++ ) {
            m = new Message((2+4*i),0," ",(5 + 10*i), "v(2,2)");
            (*v).Sent_Messages.Add(m);
            }

// Messages (3 + 4k)

      //Receives
```

```
      for (  i = 0; i <= k ; i++ ) {
            m = new Message((3+4*i),(21 + 10*i),"v(2,2)",0, " ");
            (*v).Received_Messages.Add(m);
            }

// Messages (4 + 4k)

      //Receives
      for (  i = 0; i <= k ; i++ ) {
            m = new Message((4+4*i),(9 + 10*i),"v(1,1)",0, " ");
            (*v).Received_Messages.Add(m);
            }

      //Sends
      for ( i = 0; i <= k ; i++ ) {
            m = new Message((4 + 4*i),0, " ", (10 + 10*i),"v(3,1)");
            (*v).Sent_Messages.Add(m);
            }

// Messages 0

      //Receives
            m = new Message(0,11,"v(2,2)",0, " ");
            (*v).Received_Messages.Add(m);

      for ( i = 0; i <= k ; i++ ) {
            m = new Message(0,(8 + 10*i),"v(3,1)",0, " ");
            (*v).Received_Messages.Add(m);
            }

      //Sends
      for ( i = 0; i <= k ; i++ ) {
            m = new Message(0,0, " ",(2 + 10*i),"v(1,1)");
            (*v).Sent_Messages.Add(m);
            }

      for ( i = 0; i <= k ; i++ ) {
            m = new Message(0,0, " ",(6 + 10*i),"v(1,1)");
            (*v).Sent_Messages.Add(m);
            }

            m = new Message(0,0, " ",( 5 + 10*(k+1)),"v(2,2)");
            (*v).Sent_Messages.Add(m);


////////////////////////////////////////////////////////////////////////////////

// Vertex v(3,1)

 v = vertices.Find_Vertex(3,1);

// Messages (1 + 4k)

      //Receives
      for (  i = 0; i <= k ; i++ ) {
            m = new Message((1+4*i),(6 + 10*i),"v(3,2)",0, " ");
            (*v).Received_Messages.Add(m);
            }

      //Sends
      for (  i = 0; i <= k ; i++ ) {
            m = new Message((1+4*i),0," ",(7 + 10*i), "v(2,1)");
            (*v).Sent_Messages.Add(m);
            }
```

```
// Messages (2 + 4k)

      //Receives
      for (  i = 0; i <= k ; i++ ) {
            m = new Message((2+4*i),(4 + 10*i),"v(2,1)",0, " ");
            (*v).Received_Messages.Add(m);
            }

      //Sends
      for (  i = 0; i <= k ; i++ ) {
            m = new Message((2+4*i),0," ",(5 + 10*i), "v(4,1)");
            (*v).Sent_Messages.Add(m);
            }

// Messages (3 + 4k)

      //Receives
      for (  i = 0; i <= k ; i++ ) {
            m = new Message((3+4*i),(29 + 10*i),"v(4,1)",0, " ");
            (*v).Received_Messages.Add(m);
            }

// Messages (4 + 4k)

      //Receives
      for (  i = 0; i <= k ; i++ ) {
            m = new Message((4+4*i),(10 + 10*i),"v(2,1)",0, " ");
            (*v).Received_Messages.Add(m);
            }

      //Sends
      for ( i = 0; i <= k ; i++ ) {
            m = new Message((4 + 4*i),0, " ", (11 + 10*i),"v(4,1)");
            *v).Sent_Messages.Add(m);
            }

      for (  i = 0; i <= k ; i++ ) {
            m = new Message((4+4*i),0," ",(12 + 10*i), "v(3,2)");
            (*v).Sent_Messages.Add(m);
            }

// Messages 0

      //Receives
            m = new Message(0,9,"v(4,1)",0, " ");
            (*v).Received_Messages.Add(m);

            m = new Message(0,19,"v(4,1)",0, " ");
            (*v).Received_Messages.Add(m);

      for ( i = 0; i <= k ; i++ ) {
            m = new Message(0,(3 + 10*i),"v(4,1)",0, " ");
            (*v).Received_Messages.Add(m);
            }

      //Sends
      for ( i = 0; i <= k ; i++ ) {
            m = new Message(0,0, " ",(8 + 10*i),"v(2,1)");
            (*v).Sent_Messages.Add(m);
            }

            m = new Message(0,0, " ",( 11 + 10*(k+1)),"v(4,1)");
            (*v).Sent_Messages.Add(m);
```

```
            m = new Message(0,0, " ",( 11 + 10*(k+2)),"v(4,1)");
            (*v).Sent_Messages.Add(m);

////////////////////////////////////////////////////////////////////////////////

// Vertex v(i,1) (4 <= i <= (row_m - 1)

for ( i = 4; i <= (row_m - 1); i++)

{
v = vertices.Find_Vertex(i, 1);

if ( i%2 == 0 )  // i is EVEN
{
      str1.append("v(");
      str_index = itos(i-1);
      str1.append(str_index);
      str1.append(",1)");

      str2.append("v(");
      str_index = itos(i+1);
      str2.append(str_index);
      str2.append(",1)");
      str3.append("v(");
      str_index = itos(i);
      str3.append(str_index);
      str3.append(",2)");


// Messages (1 + 4k)

      //Receives
      for ( j = 0; j <= k ; j++ ) {
            m = new Message((1+4*j),(i + 6 + 10*j),str2, 0, " ");
            (*v).Received_Messages.Add(m);
            }

// Messages (2 + 4k)

      //Receives
      for (  j = 0; j <= k ; j++ ) {
            m = new Message((2+4*j),(i + 1 + 10*j),str1, 0, " ");
            (*v).Received_Messages.Add(m);
            }

      //Sends

      for (  j = 0; j <= k ; j++ ) {
            m = new Message((2+4*j),0, " ", (i + 2 + 10*j),str2);
            (*v).Sent_Messages.Add(m);
            }

      for (  j = 0; j <= k ; j++ ) {
            m = new Message((2+4*j),0, " ", (i + 3 + 10*j),str3);
            (*v).Sent_Messages.Add(m);
            }

// Messages (3 + 4k)

      //Receives
      for (  j = 0; j <= k ; j++ ) {
            m = new Message((3 + 4*j),(i + 20 + 10*j),str3,0, " ");
            (*v).Received_Messages.Add(m);
```

```
            }

      //Sends
      for (  j = 0; j <= k ; j++ ) {
            m = new Message((3 + 4*j),0," ",(i + 25 + 10*j), str1);
            (*v).Sent_Messages.Add(m);
            }

// Messages (4 + 4k)

      //Receives
      for (  j = 0; j <= k ; j++ ) {
            m = new Message((4 + 4*j),(i + 7 + 10*j), str1,0, " " );
            (*v).Received_Messages.Add(m);
            }

      //Sends
      for (  j = 0; j <= k ; j++ ) {
            m = new Message((4 + 4*j),0," ",(i + 8 + 10*j), str2);
            (*v).Sent_Messages.Add(m);
            }

// Messages (0)

      //Receives
      for (  j = 0; j <= k ; j++ ) {
            m = new Message(0,(i + 4 + 10*j), str2, 0, " " );
            (*v).Received_Messages.Add(m);
            }

            m = new Message(0,(i + 10), str3, 0, " " );
            (*v).Received_Messages.Add(m);

            m = new Message(0,(i + 7 + 10*(k+1)), str1, 0, " ");
            (*v).Received_Messages.Add(m);

            m = new Message(0,(i + 7 + 10*(k+2)), str1, 0, " ");
            (*v).Received_Messages.Add(m);

      //Sends
      for (  j = 0; j <= k ; j++ ) {
            m = new Message(0,0," ",(i - 1 + 10*j), str1);
            (*v).Sent_Messages.Add(m);
            }

            m = new Message(0,0," ",(i + 5),str1);
            (*v).Sent_Messages.Add(m);

            m = new Message(0,0," ",(i + 15),str1);
            (*v).Sent_Messages.Add(m);

            m = new Message(0,0, " ", (i + 3 + 10*(k+1)), str3);
            (*v).Sent_Messages.Add(m);


      str1 = "";
      str2 = "";
      str3 = "";


}
else {  // i is ODD

            str1.append("v(");
```

```
            str_index = itos(i-1);
            str1.append(str_index);
            str1.append(",1)");

            str2.append("v(");
            str_index = itos(i+1);
            str2.append(str_index);
            str2.append(",1)");

            str3.append("v(");
            str_index = itos(i);
            str3.append(str_index);
            str3.append(",2)");

// Messages (1 + 4k)

      //Receives
      for ( j = 0; j <= k ; j++ ) {
            m = new Message((1+4*j),(i + 4 + 10*j),str3, 0, " ");
            (*v).Received_Messages.Add(m);
            }

      //Sends
      for (  j = 0; j <= k ; j++ ) {
            m = new Message((1 + 4*j),0," ",(i + 5 + 10*j), str1);
            (*v).Sent_Messages.Add(m);
            }

// Messages (2 + 4k)

      //Receives
      for (  j = 0; j <= k ; j++ ) {
            m = new Message((2+4*j),(i + 1 + 10*j),str1, 0, " ");
            (*v).Received_Messages.Add(m);
            }

      //Sends
      for (  j = 0; j <= k ; j++ ) {
            m = new Message((2+4*j),0, " ", (i + 2 + 10*j),str2);
            (*v).Sent_Messages.Add(m);
            }

// Messages (3 + 4k)

      //Receives
      for (  j = 0; j <= k ; j++ ) {
            m = new Message((3 + 4*j),(i + 26 + 10*j),str2,0, " ");
            (*v).Received_Messages.Add(m);
            }

// Messages (4 + 4k)

      //Receives
      for (  j = 0; j <= k ; j++ ) {
            m = new Message((4 + 4*j),(i + 7 + 10*j), str1,0, " " );
            (*v).Received_Messages.Add(m);
            }
      //Sends
      for (  j = 0; j <= k ; j++ ) {
            m = new Message((4 + 4*j),0," ",(i + 8 + 10*j), str2);
            (*v).Sent_Messages.Add(m);
            }
      for (  j = 0; j <= k ; j++ ) {
            m = new Message((4+4*j),0, " ", (i + 9 + 10*j),str3);
```

```
            (*v).Sent_Messages.Add(m);
            }

// Messages (0)

      //Receives
            m = new Message(0,(i + 6), str2, 0, " " );
            (*v).Received_Messages.Add(m);

            m = new Message(0,(i + 16), str2, 0, " " );
            (*v).Received_Messages.Add(m);

      for (  j = 0; j <= k ; j++ ) {
            m = new Message(0,(i + 10*j), str2, 0, " ");
            (*v).Received_Messages.Add(m);
            }

      //Sends
      for (  j = 0; j <= k ; j++ ) {
            m = new Message(0,0," ",(i + 3 + 10*j), str1);
            (*v).Sent_Messages.Add(m);
            }

            m = new Message(0,0, " ", (i + 8 + 10*(k+1)), str2);
            (*v).Sent_Messages.Add(m);

            m = new Message(0,0, " ", (i + 8 + 10*(k+2)), str2);
            (*v).Sent_Messages.Add(m);


str1 = "";
str2 = "";
str3 = "";

} //end of ELSE


} //end of j FOR loop


////////////////////////////////////////////////////////////////////////////////

// Vertex v(row_m,1)

v = vertices.Find_Vertex(row_m,1);

                  str1.append("v(");
                  str_index = itos(row_m - 1);
                  str1.append(str_index);
                  str1.append(",1)");

                  str3.append("v(");
                  str_index = itos(row_m);
                  str3.append(str_index);
                  str3.append(",2)");

// Messages (1 + 4k)

      //Receives
      for (  i = 0; i <= k ; i++ ) {
            m = new Message((1+4*i),( row_m + 4 + 10*i),str3,0, " ");
            (*v).Received_Messages.Add(m);
            }
```

```
      //Sends
      for (  i = 0; i <= k ; i++ ) {
            m = new Message((1+4*i),0," ",(row_m + 5 + 10*i), str1);
            (*v).Sent_Messages.Add(m);
            }

// Messages (2 + 4k)

      //Receives
      for (  i = 0; i <= k ; i++ ) {
            m = new Message((2+4*i),( row_m  + 1 + 10*i),str1,0, " ");
            (*v).Received_Messages.Add(m);
            }

      //Sends
      for (  i = 0; i <= k ; i++ ) {
            m = new Message((2+4*i),0," ",(row_m + 2 + 10*i), str3);
            (*v).Sent_Messages.Add(m);
            }

// Messages (3 + 4k)

      //Receives
      for (  i = 0; i <= k ; i++ ) {
            m = new Message((3+4*i),( row_m + 20 + 10*i),str3,0, " ");
            (*v).Received_Messages.Add(m);
            }

// Messages (4 + 4k)

      //Receives
      for (  i = 0; i <= k ; i++ ) {
            m = new Message((4+4*i),( row_m + 7 + 10*i),str1,0, " ");
            (*v).Received_Messages.Add(m);
            }

      //Sends
      for ( i = 0; i <= k ; i++ ) {
            m = new Message((4 + 4*i),0, " ", ( row_m + 8 + 10*i),str3);
          (*v).Sent_Messages.Add(m);
            }

// Messages 0

      //Receives
            m = new Message(0,( row_m + 10),str3,0, " ");
            (*v).Received_Messages.Add(m);

      //Sends
      for ( i = 0; i <= k ; i++ ) {
            m = new Message(0,0, " ",( row_m + 3 + 10*i),str1);
            (*v).Sent_Messages.Add(m);
            }

            m = new Message(0,0, " ",( row_m + 8 + 10*(k+1)),str3);
            (*v).Sent_Messages.Add(m);

str1 = "";
str2 = "";
str3 = "";

////////////////////////////////////////////////////////////////////////////////
```

```
// Vertex v(row_m,2)

v = vertices.Find_Vertex(row_m,2);

                  str1.append("v(");
                  str_index = itos(row_m);
                  str1.append(str_index);
                  str1.append(",1)");

                  str2.append("v(");
                  str_index = itos(row_m);
                  str2.append(str_index);
                  str2.append(",3)");

                  str3.append("v(");
                  str_index = itos(row_m - 1);
                  str3.append(str_index);
                  str3.append(",2)");


// Messages (1 + 4k)

      //Receives
      for ( i = 0; i <= k ; i++ ) {
            m = new Message((1+4*i),( row_m + 1 + 10*i),str3, 0, " ");
            (*v).Received_Messages.Add(m);
            }

      //Sends
      for (  i = 0; i <= k ; i++ ) {
            m = new Message((1+4*i),0," ",(row_m + 4 + 10*i), str1);
            (*v).Sent_Messages.Add(m);
            }

// Messages (2 + 4k)

      //Receives
      for (  i = 0; i <= k ; i++ ) {
            m = new Message((2+4*i),(row_m + 2 + 10*i),str1, 0, " ");
            (*v).Received_Messages.Add(m);
            }

      //Sends
      for (  i = 0; i <= k ; i++ ) {
            m = new Message((2+4*i),0," ",(row_m + 3 + 10*i), str2);
            (*v).Sent_Messages.Add(m);
            }

// Messages (3 + 4k)

      //Receives
      for (  i = 0; i <= k ; i++ ) {
            m = new Message((3 + 4*i),(row_m + 15 + 10*i),str2,0, " ");
            (*v).Received_Messages.Add(m);
            }

      //Sends
      for (  i = 0; i <= k ; i++ ) {
            m = new Message((3 + 4*i),0," ",(row_m + 20 + 10*i), str1);
            (*v).Sent_Messages.Add(m);
            }

// Messages (4 + 4k)
```

```
      //Receives
      for (  i = 0; i <= k ; i++ ) {
            m = new Message((4 + 4*i),(row_m + 8 + 10*i), str1,0, " " );
            (*v).Received_Messages.Add(m);
            }
      //Sends
      for (  i = 0; i <= k ; i++ ) {
            m = new Message((4+4*i),0," ", (row_m + 9 + 10*i), str2);
            (*v).Sent_Messages.Add(m);
            }

      for (  i = 0; i <= k ; i++ ) {
            m = new Message((4+4*i),0," ", (row_m + 16 + 10*i), str3);
            (*v).Sent_Messages.Add(m);
            }

// Messages (0)

      //Receives
      for (  i = 0; i <= k ; i++ ) {
            m = new Message(0,(row_m + 7 + 10*i),str2, 0, " ");
            (*v).Received_Messages.Add(m);
            }

            m = new Message(0,(row_m + 5), str2, 0, " " );
            (*v).Received_Messages.Add(m);

            m = new Message(0,(row_m + 8 + 10*(k+1)), str1, 0, " ");
            (*v).Received_Messages.Add(m);

            m = new Message(0,(row_m + 1 + 10*(k+1)), str3, 0, " ");
            (*v).Received_Messages.Add(m);

      //Sends
            m = new Message(0,0," ",(row_m + 6), str3);
            (*v).Sent_Messages.Add(m);

            m = new Message(0,0," ",(row_m + 10),str1);
            (*v).Sent_Messages.Add(m);

            m = new Message(0,0," ",(row_m + 3 + 10*(k+1)),str2);
            (*v).Sent_Messages.Add(m);

str1 = "";
str2 = "";
str3 = "";

////////////////////////////////////////////////////////////////////////////////

// Vertex v(row_m,j) (3 <= j <= (column_n - 4)
for ( j = 3; j <= (column_n-4); j++)

{
v = vertices.Find_Vertex(row_m,j);

                  str1.append("v(");
                  str_index = itos(row_m);
                  str1.append(str_index);
                  str1.append(",");
                  str_index = itos(j-1);
                  str1.append(str_index);
                  str1.append(")");

                  str2.append("v(");
```

```
                    str_index = itos(row_m);
                    str2.append(str_index);
                    str2.append(",");
                    str_index = itos(j+1);
                    str2.append(str_index);
                    str2.append(")");

                    str3.append("v(");
                    str_index = itos(row_m-1);
                    str3.append(str_index);
                    str3.append(",");
                    str_index = itos(j);
                    str3.append(str_index);
                    str3.append(")");


if ( j%2 == 0 )  // j is EVEN
{

// Messages (1 + 4k)

      //Receives
      for ( i = 0; i <= k ; i++ ) {
            m = new Message((1+4*i),(row_m + j - 1 + 10*i),str3, 0, " ");
            (*v).Received_Messages.Add(m);
            }


      //Sends
      for (  i = 0; i <= k ; i++ ) {
            m = new Message((1+4*i),0," ",(row_m + j + 2 + 10*i), str1);
            (*v).Sent_Messages.Add(m);
            }

// Messages (2 + 4k)

      //Receives
      for (  i = 0; i <= k ; i++ ) {
            m = new Message((2+4*i),(row_m + j + 10*i),str1, 0, " ");
            (*v).Received_Messages.Add(m);
            }

      //Sends
      for (  i = 0; i <= k ; i++ ) {
            m = new Message((2+4*i),0, " ", (row_m + j + 1 + 10*i),str2);
            (*v).Sent_Messages.Add(m);
            }

// Messages (3 + 4k)

      //Receives
      for (  i = 0; i <= k ; i++ ) {
            m = new Message((3 + 4*i),(row_m + j + 13 + 10*i),str2,0, " ");
            (*v).Received_Messages.Add(m);
            }

// Messages (4 + 4k)

      //Receives
      for (  i = 0; i <= k ; i++ ) {
            m = new Message((4 + 4*i),(row_m + j + 6 + 10*i), str1,0, " " );
            (*v).Received_Messages.Add(m);
            }
```

```
      //Sends
      for (  i = 0; i <= k ; i++ ) {
            m = new Message((4 + 4*i),0," ",(row_m + j + 7 + 10*i), str2);
            (*v).Sent_Messages.Add(m);
            }

      for (  i = 0; i <= k ; i++ ) {
            m = new Message((4 + 4*i),0," ",(row_m + j + 14 + 10*i), str3);
            (*v).Sent_Messages.Add(m);
            }

// Messages (0)

      //Receives
      for (  i = 0; i <= k ; i++ ) {
            m = new Message(0,(row_m + j + 5 + 10*i),str2, 0, " ");
            (*v).Received_Messages.Add(m);
            }

      m = new Message(0,(row_m + j + 3), str2, 0, " " );
      (*v).Received_Messages.Add(m);

      m = new Message(0,(row_m + j - 1 + 10*(k+1)), str3, 0, " ");
      (*v).Received_Messages.Add(m);

      //Sends
      for (  i = 0; i <= k ; i++ ) {
            m = new Message(0,0," ",(row_m + j + 8 + 10*i), str1);
            (*v).Sent_Messages.Add(m);
            }
            m = new Message(0,0," ",(row_m + j + 4),str3);
            (*v).Sent_Messages.Add(m);

            m = new Message(0,0, " ", (row_m + j + 1 + 10*(k+1)), str2);
            (*v).Sent_Messages.Add(m);

      }
      else {  // j is ODD


// Messages (1 + 4k)

      //Receives
      for ( i = 0; i <= k ; i++ ) {

            m = new Message((1+4*i),(row_m + j + 3 + 10*i),str2, 0, " ");
            (*v).Received_Messages.Add(m);
            }

      //Sends
      for (  i = 0; i <= k ; i++ ) {
            m = new Message((1+4*i),0," ",(row_m + j + 8 + 10*i), str3);
            (*v).Sent_Messages.Add(m);
            }

// Messages (2 + 4k)

      //Receives
      for (  i = 0; i <= k ; i++ ) {
            m = new Message((2+4*i),(row_m + j + 10*i),str1, 0, " ");
            (*v).Received_Messages.Add(m);
            }
```

```
      //Sends
      for (  i = 0; i <= k ; i++ ) {
            m = new Message((2+4*i),0," ",(row_m + j + 1 + 10*i), str2);
            (*v).Sent_Messages.Add(m);
            }

// Messages (3 + 4k)

      //Receives
      for (  i = 0; i <= k ; i++ ) {
            m = new Message((3 + 4*i),(row_m + j + 5 + 10*i),str3,0, " ");
            (*v).Received_Messages.Add(m);
            }

      //Sends
      for (  i = 0; i <= k ; i++ ) {
            m = new Message((3 + 4*i),0," ",(row_m + j + 12 + 10*i), str1);
            (*v).Sent_Messages.Add(m);
            }

// Messages (4 + 4k)

      //Receives
      for (  i = 0; i <= k ; i++ ) {
            m = new Message((4 + 4*i),(row_m + j + 6 + 10*i), str1,0, " " );
            (*v).Received_Messages.Add(m);
            }
      //Sends
      for (  i = 0; i <= k ; i++ ) {
            m = new Message((4+4*i),0, " ", (row_m + j + 7 + 10*i),str2);
            (*v).Sent_Messages.Add(m);
            }

// Messages (0)

      //Receives
      for (  i = 0; i <= k ; i++ ) {
            m = new Message(0,(row_m + j + 9 + 10*i),str2, 0, " ");
            (*v).Received_Messages.Add(m);
            }

            m = new Message(0,(row_m + j + 10*(k+1) ), str1, 0, " ");
            (*v).Received_Messages.Add(m);

      //Sends
      for (  i = 0; i <= k ; i++ ) {
            m = new Message(0,0," ",(row_m + j + 4 + 10*i), str1);
            (*v).Sent_Messages.Add(m);
            }

            m = new Message(0,0," ",(row_m + j + 2),str1);
            (*v).Sent_Messages.Add(m);


} //end of ELSE

      str1 = "";
      str2 = "";
      str3 = "";
} //end of j FOR loop

////////////////////////////////////////////////////////////////////////////////
```

```
// Vertex v(row_m,column_n-3)  NEW

v = vertices.Find_Vertex(row_m,column_n - 3);
calculate_vertex_row_column_minus_3 (v);


////////////////////////////////////////////////////////////////////////////

// Vertex v(row_m,column_n-2)

v = vertices.Find_Vertex(row_m,column_n-2);
calculate_vertex_row_column_minus_2 (v);


////////////////////////////////////////////////////////////////////////////

// Vertex v(row_m,column_n-1)
v = vertices.Find_Vertex(row_m,column_n-1);
calculate_vertex_row_column_minus_1 (v);


////////////////////////////////////////////////////////////////////////////

// Vertex v(row_m,column_n)
v = vertices.Find_Vertex(row_m,column_n);
calculate_vertex_row_column (v);


////////////////////////////////////////////////////////////////////////////

// Vertex v(2,column_n)

 v = vertices.Find_Vertex(2,column_n);
 (*v).get_index_j() << "\n";

                  str1.append("v(1,");
                  str_index = itos(column_n);
                  str1.append(str_index);
                  str1.append(")");

                  str2.append("v(3,");
                  str_index = itos(column_n);
                  str2.append(str_index);
                  str2.append(")");

                  str3.append("v(2,");
                  str_index = itos(column_n - 1);
                  str3.append(str_index);
                  str3.append(")");

// Messages (1 + 4k)

      //Receives
      for (  i = 0; i <= k ; i++ ) {
            m = new Message((1+4*i),( column_n + 10*i),str1,0, " ");
            (*v).Received_Messages.Add(m);
            }

      //Sends
      for (  i = 0; i <= k ; i++ ) {
            m = new Message((1+4*i),0," ",(column_n + 1 + 10*i), str2);
            (*v).Sent_Messages.Add(m);
            }
```

```
// Messages (2 + 4k)

      //Receives
      for (  i = 0; i <= k ; i++ ) {
            m = new Message((2+4*i),( column_n + 3 + 10*i),str3,0, " ");
            (*v).Received_Messages.Add(m);
            }

      //Sends
      for (  i = 0; i <= k ; i++ ) {
            m = new Message((2+4*i),0," ",(column_n + 4 + 10*i), str1);
            (*v).Sent_Messages.Add(m);
            }

// Messages (3 + 4k)

      //Receives
      for (  i = 0; i <= k ; i++ ) {
            m = new Message((3+4*i),( column_n + 6 + 10*i),str1,0, " ");
            (*v).Received_Messages.Add(m);
            }

      //Sends
      for ( i = 0; i <= k ; i++ ) {
            m = new Message((3 + 4*i),0," " ,( column_n + 7 + 10*i),str2);
            (*v).Sent_Messages.Add(m);
            }

// Messages (4 + 4k)

      //Receives
      for (  i = 0; i <= k ; i++ ) {
            m = new Message((4+4*i),( column_n + 15 + 10*i),str2,0, " ");
            (*v).Received_Messages.Add(m);
            }

      //Sends
      for ( i = 0; i <= k ; i++ ) {
            m = new Message((4 + 4*i),0, " ", ( column_n + 22 + 10*i),str1);
            (*v).Sent_Messages.Add(m);
            }

      for ( i = 0; i <= k ; i++ ) {
            m = new Message((4 + 4*i),0, " ", ( column_n + 18 + 10*i),str3);
            (*v).Sent_Messages.Add(m);
            }

// Messages 0

      //Receives
      for ( i = 0; i <= k ; i++ ) {
            m = new Message(0,( column_n - 1 + 10*i),str2,0, " ");
            (*v).Received_Messages.Add(m);
            }

            m = new Message(0,(column_n + 5),str2,0, " ");
            (*v).Received_Messages.Add(m);

            m = new Message(0,( column_n + 10*(k+1)),str1,0, " ");
            (*v).Received_Messages.Add(m);

            m = new Message(0,( column_n + 10*(k+2)),str1,0, " ");
            (*v).Received_Messages.Add(m);
```

```
            m = new Message(0,( column_n + 3 + 10*(k+1)),str3,0, " ");
            (*v).Received_Messages.Add(m);

      //Sends
            m = new Message(0,0, " ",( column_n + 2),str1);
            (*v).Sent_Messages.Add(m);

            m = new Message(0,0, " ",( column_n + 12),str1);
            (*v).Sent_Messages.Add(m);

            m = new Message(0,0, " ",( column_n + 8),str3);
            (*v).Sent_Messages.Add(m);

            m = new Message(0,0, " ",( column_n + 7 + 10*(k+1)),str2);
            (*v).Sent_Messages.Add(m);

str1 = "";
str2 = "";
str3 = "";

////////////////////////////////////////////////////////////////////////////////

// Vertex v(i,column_n) (3 <= i <= (row_m - 2)

for ( i = 3; i <= (row_m - 2); i++)
{
v = vertices.Find_Vertex(i, column_n);

                  str1.append("v(");
                  str_index = itos(i-1);
                  str1.append(str_index);
                  str1.append(",");
                  str_index = itos(column_n);
                  str1.append(str_index);
                  str1.append(")");

                  str2.append("v(");
                  str_index = itos(i+1);
                  str2.append(str_index);
                  str2.append(",");
                  str_index = itos(column_n);
                  str2.append(str_index);
                  str2.append(")");

                  str3.append("v(");
                  str_index = itos(i);
                  str3.append(str_index);
                  str3.append(",");
                  str_index = itos(column_n-1);
                  str3.append(str_index);
                  str3.append(")");


if ( i%2 == 0 )  // i is EVEN
{

// Messages (1 + 4k)

      //Receives
      for ( j = 0; j <= k ; j++ ) {
            m = new Message((1+4*j),(i + column_n - 2 + 10*j),str1, 0, " ");
            (*v).Received_Messages.Add(m);
            }
```

```
      //Sends
      for (  j = 0; j <= k ; j++ ) {
            m = new Message((1+4*j),0, " ", (i + column_n - 1 + 10*j),str2);
            (*v).Sent_Messages.Add(m);
            }

// Messages (2 + 4k)

      //Receives
      for (  j = 0; j <= k ; j++ ) {
            m = new Message((2+4*j),(i + column_n + 1 + 10*j),str3, 0, " ");
            (*v).Received_Messages.Add(m);
            }

      //Sends
      for (  j = 0; j <= k ; j++ ) {
            m = new Message((2+4*j),0, " ", (i + column_n + 2 + 10*j),str1);
            (*v).Sent_Messages.Add(m);
            }

// Messages (3 + 4k)

      //Receives
      for (  j = 0; j <= k ; j++ ) {
            m = new Message((3 + 4*j),(i + column_n + 4 + 10*j),str1,0, " ");
            (*v).Received_Messages.Add(m);
            }

      //Sends
      for (  j = 0; j <= k ; j++ ) {
            m = new Message((3 + 4*j),0," ",(i + column_n + 5 + 10*j), str2);
            (*v).Sent_Messages.Add(m);
            }

// Messages (4 + 4k)

      //Receives
      for (  j = 0; j <= k ; j++ ) {
            m = new Message((4 + 4*j),(i + column_n + 13 + 10*j), str2,0, " "
            (*v).Received_Messages.Add(m);
            }
      //Sends
      for (  j = 0; j <= k ; j++ ) {
            m = new Message((4 + 4*j),0," ",(i + column_n + 16 + 10*j), str3);
            (*v).Sent_Messages.Add(m);
            }

// Messages (0)

      //Receives
      for (  j = 0; j <= k ; j++ ) {
            m = new Message(0,(i + column_n - 3 + 10*j), str2, 0, " " );
            (*v).Received_Messages.Add(m);
            }

            m = new Message(0,(i + column_n + 3), str2, 0, " " );
            (*v).Received_Messages.Add(m);

            m = new Message(0,(i + column_n + 1 + 10*(k+1)), str3, 0, " ");
            (*v).Received_Messages.Add(m);

      //Sends
      for (  j = 0; j <= k ; j++ ) {
            m = new Message(0,0," ",(i + column_n + 10*j), str1);
```

```
            (*v).Sent_Messages.Add(m);
            }

            m = new Message(0,0," ",(i + column_n + 6),str3);
            (*v).Sent_Messages.Add(m);

            m = new Message(0,0, " ", (i + column_n + 5 + 10*(k+1)), str2);
            (*v).Sent_Messages.Add(m);

      }
      else {  // i is ODD

// Messages (1 + 4k)

      //Receives
      for ( j = 0; j <= k ; j++ ) {
            m = new Message((1+4*j),(i + column_n - 2 + 10*j),str1, 0, " ");
            (*v).Received_Messages.Add(m);
            }

      //Sends
      for (  j = 0; j <= k ; j++ ) {
            m = new Message((1 + 4*j),0," ",(i + column_n - 1 + 10*j), str2);
            (*v).Sent_Messages.Add(m);
            }

      for (  j = 0; j <= k ; j++ ) {
            m = new Message((1 + 4*j),0," ",(i + column_n + 10*j), str3);
            (*v).Sent_Messages.Add(m);
            }

// Messages (2 + 4k)

      //Receives
      for (  j = 0; j <= k ; j++ ) {
            m = new Message((2+4*j),(i + column_n + 3 + 10*j),str2, 0, " ");
            (*v).Received_Messages.Add(m);
            }

// Messages (3 + 4k)

      //Receives
      for (  j = 0; j <= k ; j++ ) {
            m = new Message((3 + 4*j),(i + column_n + 4 + 10*j),str1,0, " ");
            (*v).Received_Messages.Add(m);
            }

      //Sends
      for (  j = 0; j <= k ; j++ ) {
            m = new Message((3 + 4*j),0," ",(i + column_n + 5 + 10*j), str2);
            (*v).Sent_Messages.Add(m);
            }

// Messages (4 + 4k)

      //Receives
      for (  j = 0; j <= k ; j++ ) {
            m = new Message((4 + 4*j),(i + column_n + 7 + 10*j), str3,0, " "
            (*v).Received_Messages.Add(m);
            }

       //Sends
      for (  j = 0; j <= k ; j++ ) {
            m = new Message((4 + 4*j),0," ",(i + column_n + 12 + 10*j), str1);
```

```
            (*v).Sent_Messages.Add(m);
            }

// Messages (0)

      //Receives
      for (  j = 0; j <= k ; j++ ) {
            m = new Message(0,(i + column_n + 1 + 10*j), str2, 0, " ");
            (*v).Received_Messages.Add(m);
            }

            m = new Message(0,(i + column_n + 4 + 10*(k+1)), str1, 0, " ");
            (*v).Received_Messages.Add(m);

      //Sends
      for (  j = 0; j <= k ; j++ ) {
            m = new Message(0,0," ",(i + column_n - 4 + 10*j), str1);
            (*v).Sent_Messages.Add(m);
            }

            m = new Message(0,0, " ", (i + column_n + 2), str1);
            (*v).Sent_Messages.Add(m);

} //end of ELSE

      str1 = "";
      str2 = "";
      str3 = "";

} //end of i FOR loop





////////////////////////////////////////////////////////////////////////////////

// Vertex v(row_m-1,column_n)

v = vertices.Find_Vertex(row_m-1, column_n);
calculate_vertex_row_minus_1_column(v);

////////////////////////////////////////////////////////////////////////////////

// Vertex v(2,2)

v = vertices.Find_Vertex(2,2);

                  str1.append("v(1,2)");
                  str2.append("v(3,2)");
                  str3.append("v(2,1)");
                  str4.append("v(2,3)");

// Messages (1 + 4k)

      //Receives
      for ( j = 0; j <= k ; j++ ) {
            m = new Message((1+4*j),(3 + 10*j),str1, 0, " ");
            (*v).Received_Messages.Add(m);
            }

      //Sends
```

```
      for (  j = 0; j <= k ; j++ ) {
            m = new Message((1 + 4*j),0," ",(4 + 10*j), str2);
            (*v).Sent_Messages.Add(m);
            }

// Messages (2 + 4k)

      //Receives
      for (  j = 0; j <= k ; j++ ) {
            m = new Message((2+4*j),(5 + 10*j),str3, 0, " ");
            (*v).Received_Messages.Add(m);
            }

      //Sends
      for (  j = 0; j <= k ; j++ ) {
            m = new Message((2 + 4*j),0," ",(6 + 10*j), str4);
            (*v).Sent_Messages.Add(m);
            }

// Messages (3 + 4k)

      //Receives
      for (  j = 0; j <= k ; j++ ) {
            m = new Message((3 + 4*j),(12 + 10*j),str4,0, " ");
            (*v).Received_Messages.Add(m);
            }

      //Sends
      for (  j = 0; j <= k ; j++ ) {
            m = new Message((3 + 4*j),0," ",(21 + 10*j), str3);
            (*v).Sent_Messages.Add(m);
            }

// Messages (4 + 4k)

      //Receives
      for (  j = 0; j <= k ; j++ ) {
            m = new Message((4 + 4*j),(18 + 10*j), str2,0, " " );
            (*v).Received_Messages.Add(m);
            }

      //Sends
      for (  j = 0; j <= k ; j++ ) {
            m = new Message((4 + 4*j),0," ",(19 + 10*j), str1);
            (*v).Sent_Messages.Add(m);
            }

// Messages (0)

      //Receives
      m = new Message(0,8, str2, 0, " ");
      (*v).Received_Messages.Add(m);

      m = new Message(0,(3 + 10*(k+1)), str1, 0, " ");
      (*v).Received_Messages.Add(m);

      m = new Message(0,(5 + 10*(k+1)), str3, 0, " ");
      (*v).Received_Messages.Add(m);

      //Sends
      m = new Message(0,0," ",9, str1);
      (*v).Sent_Messages.Add(m);

      m = new Message(0,0," ",11, str3);
```

```
      (*v).Sent_Messages.Add(m);

      m = new Message(0,0, " ", (4 + 10*(k+1)), str2);
      (*v).Sent_Messages.Add(m);


      str1 = "";
      str2 = "";
      str3 = "";
      str4 = "";

////////////////////////////////////////////////////////////////////////////////

// Vertex v(2,j) (3 <= j <= (column_n - 3)

for ( j = 3; j <= (column_n-3); j++)

{
v = vertices.Find_Vertex(2,j);

                  str1.append("v(1,");
                  str_index = itos(j);
                  str1.append(str_index);
                  str1.append(")");

                  str2.append("v(3,");
                  str_index = itos(j);
                  str2.append(str_index);
                  str2.append(")");

                  str3.append("v(2,");
                  str_index = itos(j-1);
                  str3.append(str_index);
                  str3.append(")");

                  str4.append("v(2,");
                  str_index = itos(j+1);
                  str4.append(str_index);
                  str4.append(")");


if ( j%2 == 0 )  // j is EVEN
{

// Messages (1 + 4k)

      //Receives
      for ( i = 0; i <= k ; i++ ) {
            m = new Message((1+4*i),(j + 1 + 10*i),str1, 0, " ");
            (*v).Received_Messages.Add(m);
            }

      //Sends
      for (  i = 0; i <= k ; i++ ) {
            m = new Message((1+4*i),0," ",(j + 2 + 10*i), str2);
            (*v).Sent_Messages.Add(m);
            }



// Messages (2 + 4k)

      //Receives
```

```
      for (  i = 0; i <= k ; i++ ) {
            m = new Message((2+4*i),(j + 3 + 10*i),str3, 0, " ");
            (*v).Received_Messages.Add(m);
            }

      //Sends
      for (  i = 0; i <= k ; i++ ) {
            m = new Message((2+4*i),0, " ", (j + 4 + 10*i),str4);
            (*v).Sent_Messages.Add(m);
            }

// Messages (3 + 4k)

      //Receives
      for (  i = 0; i <= k ; i++ ) {
            m = new Message((3 + 4*i),(j + 10 + 10*i),str4,0, " ");
            (*v).Received_Messages.Add(m);
            }

// Messages (4 + 4k)

      //Receives
      for (  i = 0; i <= k ; i++ ) {
            m = new Message((4 + 4*i),(j + 16 + 10*i), str2,0, " " );
            (*v).Received_Messages.Add(m);
            }

      //Sends
      for (  i = 0; i <= k ; i++ ) {
            m = new Message((4 + 4*i),0," ",(j + 18 + 10*i), str1);
            (*v).Sent_Messages.Add(m);
            }

      for (  i = 0; i <= k ; i++ ) {
            m = new Message((4 + 4*i),0," ",(j + 19 + 10*i), str3);
            (*v).Sent_Messages.Add(m);
            }

// Messages (0)

      //Receives
      m = new Message(0,(j + 6), str2, 0, " " );
      (*v).Received_Messages.Add(m);

      m = new Message(0,(j + 1 + 10*(k+1)), str1, 0, " ");
      (*v).Received_Messages.Add(m);

      m = new Message(0,(j + 3 + 10*(k+1)), str3, 0, " ");
      (*v).Received_Messages.Add(m);

      //Sends
      m = new Message(0,0," ",(j + 9),str3);
      (*v).Sent_Messages.Add(m);

      m = new Message(0,0, " ", (j + 8), str1);
      (*v).Sent_Messages.Add(m);

      m = new Message(0,0, " ", (j + 2 + 10*(k+1)), str2);
      (*v).Sent_Messages.Add(m);
}
else {  // j is ODD


// Messages (1 + 4k)
```

```
      //Receives
      for ( i = 0; i <= k ; i++ ) {
            m = new Message((1+4*i),(j + 6 + 10*i),str2, 0, " ");
            (*v).Received_Messages.Add(m);
            }

// Messages (2 + 4k)

      //Receives
      for (  i = 0; i <= k ; i++ ) {
            m = new Message((2+4*i),(j + 3 + 10*i),str3, 0, " ");
            (*v).Received_Messages.Add(m);
            }

      //Sends
      for (  i = 0; i <= k ; i++ ) {
            m = new Message((2+4*i),0," ",(j + 4 + 10*i), str4);
            (*v).Sent_Messages.Add(m);
            }

      for (  i = 0; i <= k ; i++ ) {
            m = new Message((2+4*i),0," ",(j + 12 + 10*i), str1);
            (*v).Sent_Messages.Add(m);
            }

// Messages (3 + 4k)

      //Receives
      for (  i = 0; i <= k ; i++ ) {
            m = new Message((3 + 4*i),(j + 7 + 10*i),str1,0, " ");
            (*v).Received_Messages.Add(m);
            }

      //Sends
      for (  i = 0; i <= k ; i++ ) {
            m = new Message((3 + 4*i),0," ",(j + 8 + 10*i), str2);
            (*v).Sent_Messages.Add(m);
            }

      for (  i = 0; i <= k ; i++ ) {
            m = new Message((3 + 4*i),0," ",(j + 9 + 10*i), str3);
            (*v).Sent_Messages.Add(m);
            }

// Messages (4 + 4k)

      //Receives
      for (  i = 0; i <= k ; i++ ) {
            m = new Message((4 + 4*i),(j + 20 + 10*i), str4,0, " " );
            (*v).Received_Messages.Add(m);
            }

// Messages (0)

      //Receives
      m = new Message(0,(j + 10), str4, 0, " ");
      (*v).Received_Messages.Add(m);

      //Sends
      m = new Message(0,0," ",(j + 4 + 10*(k+1)),str4);
      (*v).Sent_Messages.Add(m);

} //end of ELSE
```

```
      str1 = "";
      str2 = "";
      str3 = "";
      str4 = "";

} //end of j FOR loop

////////////////////////////////////////////////////////////////////////////////

Vertex v(2,column_n - 2)

v = vertices.Find_Vertex(2,column_n - 2);

                  str1.append("v(1,");
                  str_index = itos(column_n - 2);
                  str1.append(str_index);
                  str1.append(")");

                  str2.append("v(3,");
                  str_index = itos(column_n - 2);
                  str2.append(str_index);
                  str2.append(")");

                  str3.append("v(2,");
                  str_index = itos(column_n - 3);
                  str3.append(str_index);
                  str3.append(")");

                  str4.append("v(2,");
                  str_index = itos(column_n - 1);
                  str4.append(str_index);
                  str4.append(")");


// Messages (1 + 4k)

      //Receives
      for ( i = 0; i <= k ; i++ ) {
            m = new Message((1+4*i),(column_n - 1 + 10*i),str1, 0, " ");
            (*v).Received_Messages.Add(m);
            }

      //Sends
      for (  i = 0; i <= k ; i++ ) {
            m = new Message((1+4*i),0," ",(column_n + 10*i), str2);
            (*v).Sent_Messages.Add(m);
            }

// Messages (2 + 4k)

      //Receives
      for (  i = 0; i <= k ; i++ ) {
            m = new Message((2+4*i),(column_n + 1 + 10*i),str3, 0, " ");
            (*v).Received_Messages.Add(m);
            }

      //Sends
      for (  i = 0; i <= k ; i++ ) {
            m = new Message((2+4*i),0, " ", (column_n + 2 + 10*i),str4);
            (*v).Sent_Messages.Add(m);
            }
```

```
// Messages (3 + 4k)

      //Receives
      for (  i = 0; i <= k ; i++ ) {
            m = new Message((3 + 4*i),(column_n + 15 + 10*i),str4,0, " ");
            (*v).Received_Messages.Add(m);
            }

// Messages (4 + 4k)

      //Receives
      for (  i = 0; i <= k ; i++ ) {
            m = new Message((4 + 4*i),(column_n + 14 + 10*i), str2,0, " " );
            (*v).Received_Messages.Add(m);
            }

      //Sends
      for (  i = 0; i <= k ; i++ ) {
            m = new Message((4 + 4*i),0," ",(column_n + 16 + 10*i), str1);
            (*v).Sent_Messages.Add(m);
            }

      for (  i = 0; i <= k ; i++ ) {
            m = new Message((4 + 4*i),0," ",(column_n + 17 + 10*i), str3);
            (*v).Sent_Messages.Add(m);
            }

// Messages (0)

      //Receives
      m = new Message(0,(column_n + 5), str4, 0, " " );
      (*v).Received_Messages.Add(m);

      m = new Message(0,(column_n + 4), str2, 0, " ");
      (*v).Received_Messages.Add(m);

      m = new Message(0,(column_n + 1 + 10*(k+1)), str3, 0, " ");
      (*v).Received_Messages.Add(m);

      m = new Message(0,(column_n - 1 + 10*(k+1)), str1, 0, " ");
      (*v).Received_Messages.Add(m);

      //Sends
      m = new Message(0,0," ",(column_n + 7),str3);
      (*v).Sent_Messages.Add(m);

      m = new Message(0,0, " ", (column_n + 6), str1);
      (*v).Sent_Messages.Add(m);

      m = new Message(0,0, " ", (column_n + 10*(k+1)), str2);
      (*v).Sent_Messages.Add(m);

      m = new Message(0,0, " ", (column_n + 2 + 10*(k+1)), str4);
      (*v).Sent_Messages.Add(m);

      str1 = "";
      str2 = "";
      str3 = "";
      str4 = "";

////////////////////////////////////////////////////////////////////////////////

// Vertex v(2,column_n - 1)
```

```
v = vertices.Find_Vertex(2,column_n - 1);

                  str1.append("v(1,");
                  str_index = itos(column_n - 1);
                  str1.append(str_index);
                  str1.append(")");

                  str2.append("v(3,");
                  str_index = itos(column_n - 1);
                  str2.append(str_index);
                  str2.append(")");

                  str3.append("v(2,");
                  str_index = itos(column_n - 2);
                  str3.append(str_index);
                  str3.append(")");

                  str4.append("v(2,");
                  str_index = itos(column_n);
                  str4.append(str_index);
                  str4.append(")");

// Messages (1 + 4k)

      //Receives
      for ( i = 0; i <= k ; i++ ) {
            m = new Message((1+4*i),(column_n + 4 + 10*i),str2, 0, " ");
            (*v).Received_Messages.Add(m);
            }

// Messages (2 + 4k)

      //Receives
      for (  i = 0; i <= k ; i++ ) {
            m = new Message((2+4*i),(column_n + 2 + 10*i),str3, 0, " ");
            (*v).Received_Messages.Add(m);
            }

      //Sends
      for (  i = 0; i <= k ; i++ ) {
            m = new Message((2+4*i),0," ",(column_n + 3 + 10*i), str4);
            (*v).Sent_Messages.Add(m);
            }

      for (  i = 0; i <= k ; i++ ) {
            m = new Message((2+4*i),0," ",(column_n + 11 + 10*i), str1);
            (*v).Sent_Messages.Add(m);
            }

// Messages (3 + 4k)

      //Receives
      for (  i = 0; i <= k ; i++ ) {
            m = new Message((3 + 4*i),(column_n + 6 + 10*i),str1,0, " ");
            (*v).Received_Messages.Add(m);
            }

      //Sends
      for (  i = 0; i <= k ; i++ ) {
            m = new Message((3 + 4*i),0," ",(column_n + 7 + 10*i), str2);
            (*v).Sent_Messages.Add(m);
            }
```

```
      for (  i = 0; i <= k ; i++ ) {
            m = new Message((3 + 4*i),0," ",(column_n + 15 + 10*i), str3);
            (*v).Sent_Messages.Add(m);
            }

// Messages (4 + 4k)

      //Receives
      for (  i = 0; i <= k ; i++ ) {
            m = new Message((4 + 4*i),(column_n + 18 + 10*i), str4,0, " " );
            (*v).Received_Messages.Add(m);
            }

// Messages (0)

      //Receives
      m = new Message(0,(column_n + 8), str4, 0, " ");
      (*v).Received_Messages.Add(m);

      m = new Message(0,(column_n + 2 + 10*(k+1)), str3,0, " " );
      (*v).Received_Messages.Add(m);

      //Sends
      m = new Message(0,0," ",(column_n + 5),str3);
      (*v).Sent_Messages.Add(m);

      m = new Message(0,0," ",(column_n + 3 + 10*(k+1)),str4);
      (*v).Sent_Messages.Add(m);

      str1 = "";
      str2 = "";
      str3 = "";
      str4 = "";


////////////////////////////////////////////////////////////////////////////////

// Vertex v(3,2)

v = vertices.Find_Vertex(3,2);

                  str1.append("v(2,2)");
                  str2.append("v(4,2)");
                  str3.append("v(3,1)");
                  str4.append("v(3,3)");

// Messages (1 + 4k)

      //Receives
      for ( i = 0; i <= k ; i++ ) {
            m = new Message((1+4*i),(4 + 10*i),str1, 0, " ");
            (*v).Received_Messages.Add(m);
            }

      //Sends
      for (  i = 0; i <= k ; i++ ) {
            m = new Message((1+4*i),0," ",(5 + 10*i), str2);
            (*v).Sent_Messages.Add(m);
            }

      for (  i = 0; i <= k ; i++ ) {
            m = new Message((1+4*i),0," ",(6 + 10*i), str3);
            (*v).Sent_Messages.Add(m);
            }
```

```
// Messages (2 + 4k)

      //Receives
      for (  i = 0; i <= k ; i++ ) {
            m = new Message((2+4*i),(17 + 10*i),str4, 0, " ");
            (*v).Received_Messages.Add(m);
            }

// Messages (3 + 4k)

      //Receives
      for (  i = 0; i <= k ; i++ ) {
            m = new Message((3 + 4*i),(21 + 10*i),str2,0, " ");
            (*v).Received_Messages.Add(m);
            }

// Messages (4 + 4k)

      //Receives
      for (  i = 0; i <= k ; i++ ) {
            m = new Message((4 + 4*i),(12 + 10*i), str3,0, " " );
            (*v).Received_Messages.Add(m);
            }

      //Sends
      for (  i = 0; i <= k ; i++ ) {
            m = new Message((4+4*i),0," ",(13 + 10*i), str4);
            (*v).Sent_Messages.Add(m);
            }

      for (  i = 0; i <= k ; i++ ) {
            m = new Message((4+4*i),0," ",(18 + 10*i), str1);
            (*v).Sent_Messages.Add(m);
            }

// Messages (0)

      //Receives
      m = new Message(0,11, str2, 0, " ");
      (*v).Received_Messages.Add(m);

      m = new Message(0,(4 + 10*(k+1)), str1,0, " " );
      (*v).Received_Messages.Add(m);

      //Sends
      m = new Message(0,0," ",8,str1);
      (*v).Sent_Messages.Add(m);

      m = new Message(0,0," ",(5 + 10*(k+1)),str2);
      (*v).Sent_Messages.Add(m);

      str1 = "";
      str2 = "";
      str3 = "";
      str4 = "";

////////////////////////////////////////////////////////////////////////////////

// Vertex v(i,2) (4 <= i <= (row_m - 3)

for ( i = 4; i <= (row_m - 3); i++)
```

```
{
v = vertices.Find_Vertex(i,2);

                    str1.append("v(");
                    str_index = itos(i-1);
                    str1.append(str_index);
                    str1.append(",");
                    str_index = itos(2);
                    str1.append(str_index);
                    str1.append(")");

                    str2.append("v(");
                    str_index = itos(i+1);
                    str2.append(str_index);
                    str2.append(",");
                    str_index = itos(2);
                    str2.append(str_index);
                    str2.append(")");

                    str3.append("v(");
                    str_index = itos(i);
                    str3.append(str_index);
                    str3.append(",");
                    str_index = itos(1);
                    str3.append(str_index);
                    str3.append(")");

                    str4.append("v(");
                    str_index = itos(i);
                    str4.append(str_index);
                    str4.append(",");
                    str_index = itos(3);
                    str4.append(str_index);
                    str4.append(")");


if ( i%2 == 0 )  // i is EVEN
{

// Messages (1 + 4k)

      //Receives
      for ( j = 0; j <= k ; j++ ) {
            m = new Message((1+4*j),(i + 1 + 10*j),str1, 0, " ");
            (*v).Received_Messages.Add(m);
            }

      //Sends
      for (  j = 0; j <= k ; j++ ) {
            m = new Message((1+4*j),0, " ", (i + 2 + 10*j),str2);
            (*v).Sent_Messages.Add(m);
            }

// Messages (2 + 4k)

      //Receives
      for (  j = 0; j <= k ; j++ ) {

            m = new Message((2+4*j),(i + 3 + 10*j),str3, 0, " ");
            (*v).Received_Messages.Add(m);
            }

      //Sends
```

```
      for (  j = 0; j <= k ; j++ ) {
            m = new Message((2+4*j),0, " ", (i + 4 + 10*j),str4);
            (*v).Sent_Messages.Add(m);
            }

// Messages (3 + 4k)

      //Receives
      for (  j = 0; j <= k ; j++ ) {
            m = new Message((3 + 4*j),(i + 16 + 10*j),str4,0, " ");
            (*v).Received_Messages.Add(m);
            }

      //Sends
      for (  j = 0; j <= k ; j++ ) {
            m = new Message((3 + 4*j),0," ",(i + 17 + 10*j), str1);
            (*v).Sent_Messages.Add(m);
            }

      for (  j = 0; j <= k ; j++ ) {
            m = new Message((3 + 4*j),0," ",(i + 20 + 10*j), str3);
            (*v).Sent_Messages.Add(m);
            }

// Messages (4 + 4k)

      //Receives
      for (  j = 0; j <= k ; j++ ) {
            m = new Message((4 + 4*j),(i + 18 + 10*j), str2,0, " " );
            (*v).Received_Messages.Add(m);
            }

// Messages (0)

      //Receives
      m = new Message(0,(i + 8), str2, 0, " " );
      (*v).Received_Messages.Add(m);

      m = new Message(0,(i + 6), str4, 0, " ");
      (*v).Received_Messages.Add(m);

      m = new Message(0,(i + 1 + 10*(k+1)), str1, 0, " ");
      (*v).Received_Messages.Add(m);

      m = new Message(0,(i + 3 + 10*(k+1)), str3, 0, " ");
      (*v).Received_Messages.Add(m);

      //Sends
      m = new Message(0,0," ",(i + 7),str1);
      (*v).Sent_Messages.Add(m);

      m = new Message(0,0, " ", (i + 10), str3);
      (*v).Sent_Messages.Add(m);

      m = new Message(0,0, " ", (i + 4 + 10*(k+1)), str4);
      (*v).Sent_Messages.Add(m);

      m = new Message(0,0, " ", (i + 2 + 10*(k+1)), str2);
      (*v).Sent_Messages.Add(m);

      }
      else {  // i is ODD
```

```
// Messages (1 + 4k)

      //Receives
      for ( j = 0; j <= k ; j++ ) {
            m = new Message((1+4*j),(i + 1 + 10*j),str1, 0, " ");
            (*v).Received_Messages.Add(m);
            }

      //Sends
      for (  j = 0; j <= k ; j++ ) {
            m = new Message((1 + 4*j),0," ",(i + 2 + 10*j), str2);
            (*v).Sent_Messages.Add(m);
            }
      for (  j = 0; j <= k ; j++ ) {
            m = new Message((1 + 4*j),0," ",(i + 4 + 10*j), str3);
            (*v).Sent_Messages.Add(m);
            }

// Messages (2 + 4k)

      //Receives
      for (  j = 0; j <= k ; j++ ) {
            m = new Message((2+4*j),(i + 16 + 10*j),str4, 0, " ");
            (*v).Received_Messages.Add(m);
            }


// Messages (3 + 4k)

      //Receives
      for (  j = 0; j <= k ; j++ ) {
            m = new Message((3 + 4*j),(i + 18 + 10*j),str2,0, " ");
            (*v).Received_Messages.Add(m);
            }

// Messages (4 + 4k)

      //Receives
      for (  j = 0; j <= k ; j++ ) {
            m = new Message((4 + 4*j),(i + 9 + 10*j), str3,0, " " );
            (*v).Received_Messages.Add(m);
            }
      //Sends
      for (  j = 0; j <= k ; j++ ) {
            m = new Message((4 + 4*j),0," ",(i + 10 + 10*j), str4);
            (*v).Sent_Messages.Add(m);
            }

      for (  j = 0; j <= k ; j++ ) {
            m = new Message((4 + 4*j),0," ",(i + 17 + 10*j), str1);
            (*v).Sent_Messages.Add(m);
            }

// Messages (0)

      //Receives
      m = new Message(0,(i + 8), str2, 0, " ");
      (*v).Received_Messages.Add(m);

      m = new Message(0,(i + 1 + 10*(k+1)), str1, 0, " ");
      (*v).Received_Messages.Add(m);

      //Sends
```

```
      m = new Message(0,0," ",(i + 7), str1);
      (*v).Sent_Messages.Add(m);

      m = new Message(0,0, " ", (i + 2 + 10*(k+1)), str2);
      (*v).Sent_Messages.Add(m);


} //end of ELSE

      str1 = "";
      str2 = "";
      str3 = "";
      str4 = "";

} //end of i FOR loop


////////////////////////////////////////////////////////////////////////////////

//Vertex v(row_m-2,2) FROM ODD CASE


v = vertices.Find_Vertex(row_m-2,2);

calculate_vertex_row_minus_2_2(v);

////////////////////////////////////////////////////////////////////////////////

// Vertex v(row_m-1,2) FROM ODD CASE

v = vertices.Find_Vertex(row_m-1,2);

                  str1.append("v(");
                  str_index = itos(row_m - 2);
                  str1.append(str_index);
                  str1.append(",");
                  str_index = itos(2);
                  str1.append(str_index);
                  str1.append(")");

                  str2.append("v(");
                  str_index = itos(row_m);
                  str2.append(str_index);
                  str2.append(",");
                  str_index = itos(2);
                  str2.append(str_index);
                  str2.append(")");

                  str3.append("v(");
                  str_index = itos(row_m - 1);
                  str3.append(str_index);
                  str3.append(",");
                  str_index = itos(1);
                  str3.append(str_index);
                  str3.append(")");

                  str4.append("v(");
                  str_index = itos(row_m - 1);
                  str4.append(str_index);
                  str4.append(",");
                  str_index = itos(3);
                  str4.append(str_index);
                  str4.append(")");
```

```
// Messages (1 + 4k)

      //Receives
      for ( j = 0; j <= k ; j++ ) {
            m = new Message((1+4*j),(row_m + 10*j),str1, 0, " ");
            (*v).Received_Messages.Add(m);
            }

      //Sends
      for (  j = 0; j <= k ; j++ ) {
            m = new Message((1 + 4*j),0," ",(row_m + 1 + 10*j), str2);
            (*v).Sent_Messages.Add(m);
            }

// Messages (2 + 4k)

      //Receives
      for (  j = 0; j <= k ; j++ ) {
            m = new Message((2+4*j),(row_m + 2 + 10*j),str3, 0, " ");
            (*v).Received_Messages.Add(m);
            }

      //Sends
      for (  j = 0; j <= k ; j++ ) {
            m = new Message((2 + 4*j),0," ",(row_m + 3 + 10*j), str4);
            (*v).Sent_Messages.Add(m);
            }

// Messages (3 + 4k)

      //Receives
      for (  j = 0; j <= k ; j++ ) {
            m = new Message((3 + 4*j),(row_m + 15 + 10*j),str4,0, " ");
            (*v).Received_Messages.Add(m);
            }

      //Sends
      for (  j = 0; j <= k ; j++ ) {
            m = new Message((3 + 4*j),0," ",(row_m + 19 + 10*j), str3);
            (*v).Sent_Messages.Add(m);
            }

      for (  j = 0; j <= k ; j++ ) {
            m = new Message((3 + 4*j),0," ",(row_m + 24 + 10*j), str1);
            (*v).Sent_Messages.Add(m);
            }

// Messages (4 + 4k)

      //Receives
      for (  j = 0; j <= k ; j++ ) {
            m = new Message((4 + 4*j),(row_m + 16 + 10*j), str2,0, " " );
            (*v).Received_Messages.Add(m);
            }

// Messages (0)

      //Receives
      m = new Message(0,(row_m + 6), str2, 0, " ");
      (*v).Received_Messages.Add(m);

      m = new Message(0,(row_m + 5), str4, 0, " ");
      (*v).Received_Messages.Add(m);
```

```
        m = new Message(0,(row_m + 2 + 10*(k+1)), str3, 0, " ");
        (*v).Received_Messages.Add(m);

        m = new Message(0,(row_m + 10*(k+1)), str1, 0, " ");
        (*v).Received_Messages.Add(m);

        m = new Message(0,(row_m + 10*(k+2)), str1, 0, " ");
        (*v).Received_Messages.Add(m);

        //Sends
        m = new Message(0,0," ",(row_m + 4), str1);
        (*v).Sent_Messages.Add(m);

        m = new Message(0,0," ",(row_m + 14), str1);
        (*v).Sent_Messages.Add(m);

        m = new Message(0,0," ",(row_m + 9), str3);
        (*v).Sent_Messages.Add(m);

        m = new Message(0,0, " ", (row_m + 1 + 10*(k+1)), str2);
        (*v).Sent_Messages.Add(m);

        m = new Message(0,0, " ", (row_m + 3 + 10*(k+1)), str4);
        (*v).Sent_Messages.Add(m);

        str1 = "";
        str2 = "";
        str3 = "";
        str4 = "";


////////////////////////////////////////////////////////////////////////////////

// Vertex v(row_m-1,3) FROM ODD CASE

v = vertices.Find_Vertex(row_m-1,3);

                    str1.append("v(");
                    str_index = itos(row_m - 2);
                    str1.append(str_index);
                    str1.append(",");
                    str_index = itos(3);
                    str1.append(str_index);
                    str1.append(")");

                    str2.append("v(");
                    str_index = itos(row_m);
                    str2.append(str_index);
                    str2.append(",");
                    str_index = itos(3);
                    str2.append(str_index);
                    str2.append(")");

                    str3.append("v(");
                    str_index = itos(row_m - 1);
                    str3.append(str_index);
                    str3.append(",");
                    str_index = itos(2);
                    str3.append(str_index);
                    str3.append(")");

                    str4.append("v(");
```

```
                  str_index = itos(row_m - 1);
                  str4.append(str_index);
                  str4.append(",");
                  str_index = itos(4);
                  str4.append(str_index);
                  str4.append(")");

// Messages (1 + 4k)

      //Receives
      for ( j = 0; j <= k ; j++ ) {
            m = new Message((1+4*j),(row_m + 11 + 10*j),str2, 0, " ");
            (*v).Received_Messages.Add(m);
            }

// Messages (2 + 4k)

      //Receives
      for (  j = 0; j <= k ; j++ ) {
            m = new Message((2+4*j),(row_m + 3 + 10*j),str3, 0, " ");
            (*v).Received_Messages.Add(m);
            }

      //Sends
      for (  j = 0; j <= k ; j++ ) {
            m = new Message((2 + 4*j),0," ",(row_m + 4 + 10*j), str4);
            (*v).Sent_Messages.Add(m);
            }

      for (  j = 0; j <= k ; j++ ) {
            m = new Message((2 + 4*j),0," ",(row_m + 12 + 10*j), str1);
            (*v).Sent_Messages.Add(m);
            }

// Messages (3 + 4k)

      //Receives
      for (  j = 0; j <= k ; j++ ) {
            m = new Message((3 + 4*j),(row_m + 7 + 10*j),str1,0, " ");
            (*v).Received_Messages.Add(m);
            }

      //Sends
      for (  j = 0; j <= k ; j++ ) {
            m = new Message((3 + 4*j),0," ",(row_m + 8 + 10*j), str2);
            (*v).Sent_Messages.Add(m);
            }

      for (  j = 0; j <= k ; j++ ) {
            m = new Message((3 + 4*j),0," ",(row_m + 15 + 10*j), str3);
            (*v).Sent_Messages.Add(m);
            }

// Messages (4 + 4k)

      //Receives
      for (  j = 0; j <= k ; j++ ) {
            m = new Message((4 + 4*j),(row_m + 20 + 10*j), str4,0, " " );
            (*v).Received_Messages.Add(m);
            }

// Messages (0)

      //Receives
```

```
      m = new Message(0,(row_m + 10), str4, 0, " ");
      (*v).Received_Messages.Add(m);

      m = new Message(0,(row_m + 3 + 10*(k+1)), str3, 0, " ");
      (*v).Received_Messages.Add(m);

      //Sends
      m = new Message(0,0," ",(row_m + 5), str3);
      (*v).Sent_Messages.Add(m);

      m = new Message(0,0, " ", (row_m + 4 + 10*(k+1)), str4);
      (*v).Sent_Messages.Add(m);

      str1 = "";
      str2 = "";
      str3 = "";
      str4 = "";


////////////////////////////////////////////////////////////////////////////////

// Vertex v(row_m-1,j) (4 <= j <= (column_n - 4)

for ( j = 4; j <= (column_n-4); j++)

{
v = vertices.Find_Vertex(row_m-1,j);

                  str1.append("v(");
                  str2.append("v(");
                  str3.append("v(");
                  str4.append("v(");

                  str_index = itos(row_m - 2);
                  str1.append(str_index);

                  str_index = itos(row_m);
                  str2.append(str_index);

                  str1.append(",");
                  str2.append(",");

                  str_index = itos(j);

                  str1.append(str_index);
                  str2.append(str_index);
                  str1.append(")");
                  str2.append(")");

                  str_index = itos(row_m - 1);

                  str3.append(str_index);
                  str4.append(str_index);

                  str3.append(",");
                  str4.append(",");

                  str_index = itos(j-1);
                  str3.append(str_index);

                  str_index = itos(j+1);
                  str4.append(str_index);
```

```
                  str3.append(")");
                  str4.append(")");


if ( j%2 == 0 )  // j is EVEN
{

// Messages (1 + 4k)

      //Receives
      for ( i = 0; i <= k ; i++ ) {
            m = new Message((1+4*i),(row_m + j - 2 + 10*i),str1, 0, " ");
            (*v).Received_Messages.Add(m);
            }

      //Sends
      for (  i = 0; i <= k ; i++ ) {
            m = new Message((1+4*i),0," ",(row_m + j - 1 + 10*i), str2);
            (*v).Sent_Messages.Add(m);
            }

// Messages (2 + 4k)

      //Receives
      for (  i = 0; i <= k ; i++ ) {
            m = new Message((2+4*i),(row_m + j + 10*i),str3, 0, " ");
            (*v).Received_Messages.Add(m);
            }

      //Sends
      for (  i = 0; i <= k ; i++ ) {
            m = new Message((2+4*i),0," ",(row_m + j + 1 + 10*i), str4);
            (*v).Sent_Messages.Add(m);
            }

// Messages (3 + 4k)

      //Receives
      for (  i = 0; i <= k ; i++ ) {
            m = new Message((3 + 4*i),(row_m + j + 7 + 10*i),str4,0, " ");
            (*v).Received_Messages.Add(m);
            }

      //Sends
      for (  i = 0; i <= k ; i++ ) {
            m = new Message((3+4*i),0," ",(row_m + j + 12 + 10*i), str1);
            (*v).Sent_Messages.Add(m);
            }

// Messages (4 + 4k)

      //Receives
      for (  i = 0; i <= k ; i++ ) {
            m = new Message((4 + 4*i),(row_m + j + 14 + 10*i), str2,0, " " );
            (*v).Received_Messages.Add(m);
            }

      //Sends
      for (  i = 0; i <= k ; i++ ) {
            m = new Message((4 + 4*i),0," ",(row_m + j + 16 + 10*i), str3);
            (*v).Sent_Messages.Add(m);
            }

// Messages (0)
```

```
      //Receives
      m = new Message(0,(row_m + j + 4), str2, 0, " " );
      (*v).Received_Messages.Add(m);

      m = new Message(0,(row_m + j + 10*(k+1)), str3, 0, " ");
      (*v).Received_Messages.Add(m);

      m = new Message(0,(row_m + j - 2 + 10*(k+1)), str1, 0, " ");
      (*v).Received_Messages.Add(m);

      //Sends
      m = new Message(0,0," ",(row_m + j + 6),str3);
      (*v).Sent_Messages.Add(m);

      m = new Message(0,0," ",(row_m + j + 2),str1);
      (*v).Sent_Messages.Add(m);

      m = new Message(0,0, " ", (row_m + j - 1 + 10*(k+1)), str2);
      (*v).Sent_Messages.Add(m);
}
else {  // j is ODD


// Messages (1 + 4k)

      //Receives
      for ( i = 0; i <= k ; i++ ) {
            m = new Message((1+4*i),(row_m + j + 8 + 10*i),str2, 0, " ");
            (*v).Received_Messages.Add(m);
            }

// Messages (2 + 4k)

      //Receives
      for (  i = 0; i <= k ; i++ ) {
            m = new Message((2+4*i),(row_m + j + 10*i),str3, 0, " ");
            (*v).Received_Messages.Add(m);
            }

      //Sends
      for (  i = 0; i <= k ; i++ ) {
            m = new Message((2+4*i),0," ",(row_m + j + 1 + 10*i), str4);
            (*v).Sent_Messages.Add(m);
            }

      for (  i = 0; i <= k ; i++ ) {
            m = new Message((2+4*i),0," ",(row_m + j + 2 + 10*i), str1);
            (*v).Sent_Messages.Add(m);
            }

// Messages (3 + 4k)

      //Receives
      for (  i = 0; i <= k ; i++ ) {
            m = new Message((3 + 4*i),(row_m + j + 4 + 10*i),str1,0, " ");
            (*v).Received_Messages.Add(m);
            }

      //Sends
      for (  i = 0; i <= k ; i++ ) {
            m = new Message((3 + 4*i),0," ",(row_m + j + 6 + 10*i), str3);
            (*v).Sent_Messages.Add(m);
```

```
            }

      for (  i = 0; i <= k ; i++ ) {
            m = new Message((3 + 4*i),0," ",(row_m + j + 5 + 10*i), str2);
            (*v).Sent_Messages.Add(m);
            }

// Messages (4 + 4k)

      //Receives
      for (  i = 0; i <= k ; i++ ) {
            m = new Message((4 + 4*i),(row_m + j + 17 + 10*i), str4,0, " " );
            (*v).Received_Messages.Add(m);
            }

// Messages (0)

      //Receives
      m = new Message(0,(row_m + j + 7), str4, 0, " " );
      (*v).Received_Messages.Add(m);

      //Sends
      m = new Message(0,0," ",(row_m + j + 1 + 10*(k+1)),str4);
      (*v).Sent_Messages.Add(m);


} //end of ELSE

      str1 = "";
      str2 = "";
      str3 = "";
      str4 = "";

} //end of j FOR loop


////////////////////////////////////////////////////////////////////////////

// Vertex v(row_m-1,column_n - 3)

v = vertices.Find_Vertex(row_m-1,column_n - 3);
calculate_vertex_row_minus_1_column_minus_3(v);

////////////////////////////////////////////////////////////////////////////

// Vertex v(row_m-1,column_n - 2)

v = vertices.Find_Vertex(row_m-1,column_n - 2);
calculate_vertex_row_minus_1_column_minus_2(v);

////////////////////////////////////////////////////////////////////////////

// Vertex v(row_m-1,column_n - 1)

v = vertices.Find_Vertex(row_m-1,column_n - 1);
calculate_vertex_row_minus_1_column_minus_1(v);

////////////////////////////////////////////////////////////////////////////

// Vertex v(3,column_n - 2)

v = vertices.Find_Vertex(3,column_n - 2);

                  str1.append("v(2,");
```

```
                    str2.append("v(4,");
                    str3.append("v(3,");
                    str4.append("v(3,");

                    str_index = itos(column_n - 2);

                    str1.append(str_index);
                    str2.append(str_index);
                    str1.append(")");
                    str2.append(")");

                    str_index = itos(column_n - 3);
                    str3.append(str_index);

                    str_index = itos(column_n - 1);
                    str4.append(str_index);

                    str3.append(")");
                    str4.append(")");


// Messages (1 + 4k)

      //Receives
      for ( i = 0; i <= k ; i++ ) {
            m = new Message((1+4*i),(column_n + 10*i),str1, 0, " ");
            (*v).Received_Messages.Add(m);
            }

      //Sends
      for (  i = 0; i <= k ; i++ ) {
            m = new Message((1+4*i),0," ",(column_n + 1 + 10*i), str2);
            (*v).Sent_Messages.Add(m);
            }

      for (  i = 0; i <= k ; i++ ) {
            m = new Message((1+4*i),0," ",(column_n + 2 + 10*i), str3);
            (*v).Sent_Messages.Add(m);
            }


// Messages (2 + 4k)

      //Receives
      for (  i = 0; i <= k ; i++ ) {
            m = new Message((2+4*i),(column_n + 15 + 10*i),str4, 0, " ");
            (*v).Received_Messages.Add(m);
            }

// Messages (3 + 4k)

      //Receives
      for (  i = 0; i <= k ; i++ ) {
            m = new Message((3 + 4*i),(column_n + 17 + 10*i),str2,0, " ");
            (*v).Received_Messages.Add(m);
            }

// Messages (4 + 4k)

      //Receives
      for (  i = 0; i <= k ; i++ ) {
            m = new Message((4 + 4*i),(column_n + 8 + 10*i), str3,0, " " );
            (*v).Received_Messages.Add(m);
```

```
            }

      //Sends
      for (  i = 0; i <= k ; i++ ) {
            m = new Message((4 + 4*i),0," ",(column_n + 9 + 10*i), str4);
            (*v).Sent_Messages.Add(m);
            }

      for (  i = 0; i <= k ; i++ ) {
            m = new Message((4 + 4*i),0," ",(column_n + 14 + 10*i), str1);
            (*v).Sent_Messages.Add(m);
            }

// Messages (0)

      //Receives
      m = new Message(0,(column_n + 7), str2, 0, " " );
      (*v).Received_Messages.Add(m);

      m = new Message(0,(column_n + 10*(k+1)), str1, 0, " ");
      (*v).Received_Messages.Add(m);

      //Sends
      m = new Message(0,0," ",(column_n + 4),str1);
      (*v).Sent_Messages.Add(m);

      m = new Message(0,0, " ", (column_n + 1+ 10*(k+1)), str2);
      (*v).Sent_Messages.Add(m);


      str1 = "";
      str2 = "";
      str3 = "";
      str4 = "";

////////////////////////////////////////////////////////////////////////////////

// Vertex v(3,column_n - 1)

v = vertices.Find_Vertex(3,column_n - 1);


                  str1.append("v(2,");
                  str2.append("v(4,");
                  str3.append("v(3,");
                  str4.append("v(3,");

                  str_index = itos(column_n - 1);

                  str1.append(str_index);
                  str2.append(str_index);
                  str1.append(")");
                  str2.append(")");

                  str_index = itos(column_n - 2);
                  str3.append(str_index);

                  str_index = itos(column_n);
                  str4.append(str_index);

                  str3.append(")");
                  str4.append(")");

// Messages (1 + 4k)
```

```
      //Receives
      for ( i = 0; i <= k ; i++ ) {
            m = new Message((1+4*i),(column_n + 3 + 10*i),str4, 0, " ");
            (*v).Received_Messages.Add(m);
            }

      //Sends
      for (  i = 0; i <= k ; i++ ) {
            m = new Message((1+4*i),0," ",(column_n + 4 + 10*i), str1);
            (*v).Sent_Messages.Add(m);
            }

// Messages (2 + 4k)

      //Receives
      for (  i = 0; i <= k ; i++ ) {
            m = new Message((2+4*i),(column_n + 12 + 10*i),str2, 0, " ");
            (*v).Received_Messages.Add(m);
            }

      //Sends
      for (  i = 0; i <= k ; i++ ) {
            m = new Message((2+4*i),0," ",(column_n + 15 + 10*i), str3);
            (*v).Sent_Messages.Add(m);
            }

// Messages (3 + 4k)

      //Receives
      for (  i = 0; i <= k ; i++ ) {
            m = new Message((3 + 4*i),(column_n + 7 + 10*i),str1,0, " ");
            (*v).Received_Messages.Add(m);
            }

      //Sends
      for (  i = 0; i <= k ; i++ ) {
            m = new Message((3 + 4*i),0," ",(column_n + 8 + 10*i), str2);
            (*v).Sent_Messages.Add(m);
            }

// Messages (4 + 4k)

      //Receives
      for (  i = 0; i <= k ; i++ ) {
            m = new Message((4 + 4*i),(column_n + 9 + 10*i), str3,0, " " );
            (*v).Received_Messages.Add(m);
            }

      //Sends
      for (  i = 0; i <= k ; i++ ) {
            m = new Message((4 + 4*i),0," ",(column_n + 10 + 10*i), str4);
            (*v).Sent_Messages.Add(m);
            }

// Messages (0)

      //Receives

      //Sends

      str1 = "";
      str2 = "";
      str3 = "";
```

```
      str4 = "";

////////////////////////////////////////////////////////////////////////////////

// Vertex v(i,column_n-1) (4 <= i <= (row_m - 2)

for ( i = 4; i <= (row_m - 2); i++)

{
v = vertices.Find_Vertex(i,column_n-1);

                  str1.append("v(");
                  str2.append("v(");
                  str3.append("v(");
                  str4.append("v(");

                  str_index = itos(i-1);
                  str1.append(str_index);

                  str_index = itos(i+1);
                  str2.append(str_index);

                  str1.append(",");
                  str2.append(",");

                  str_index = itos(column_n - 1);

                  str1.append(str_index);
                  str2.append(str_index);
                  str1.append(")");
                  str2.append(")");

                  str_index = itos(i);

                  str3.append(str_index);
                  str4.append(str_index);

                  str3.append(",");
                  str4.append(",");

                  str_index = itos(column_n - 2);
                  str3.append(str_index);

                  str_index = itos(column_n);
                  str4.append(str_index);

                  str3.append(")");
                  str4.append(")");

if ( i%2 == 0 )  // i is EVEN
{

// Messages (1 + 4k)

      //Receives
      for ( j = 0; j <= k ; j++ ) {
            m = new Message((1+4*j),(i + column_n + 9 + 10*j),str2, 0, " ");
            (*v).Received_Messages.Add(m);
            }
```

```
// Messages (2 + 4k)

      //Receives
      for (  j = 0; j <= k ; j++ ) {
            m = new Message((2+4*j),(i + column_n + 10*j),str3, 0, " ");
            (*v).Received_Messages.Add(m);
            }

      //Sends
      for (  j = 0; j <= k ; j++ ) {
            m = new Message((2+4*j),0, " ", (i + column_n + 1 + 10*j),str4);
            (*v).Sent_Messages.Add(m);
            }

      for (  j = 0; j <= k ; j++ ) {
            m = new Message((2+4*j),0, " ", (i + column_n + 8 + 10*j),str1);
            (*v).Sent_Messages.Add(m);
            }

// Messages (3 + 4k)

      //Receives
      for (  j = 0; j <= k ; j++ ) {

            m = new Message((3 + 4*j),(i + column_n + 4 + 10*j),str1,0, " ");
            (*v).Received_Messages.Add(m);
            }

      //Sends
      for (  j = 0; j <= k ; j++ ) {
            m = new Message((3 + 4*j),0," ",(i + column_n + 5 + 10*j), str2);
            (*v).Sent_Messages.Add(m);
            }

      for (  j = 0; j <= k ; j++ ) {
            m = new Message((3 + 4*j),0," ",(i + column_n + 12 + 10*j), str3);
            (*v).Sent_Messages.Add(m);
            }

// Messages (4 + 4k)

      //Receives
      for (  j = 0; j <= k ; j++ ) {
      m = new Message((4 + 4*j),(i + column_n + 16 + 10*j), str4,0, " " );
            (*v).Received_Messages.Add(m);
            }


// Messages (0)

      //Receives
      m = new Message(0,(i + column_n + 6), str4, 0, " ");
      (*v).Received_Messages.Add(m);

      m = new Message(0,(i + column_n + 10*(k+1)), str3, 0, " ");
      (*v).Received_Messages.Add(m);

      //Sends
      m = new Message(0,0, " ", (i + column_n + 2), str3);
      (*v).Sent_Messages.Add(m);

      m = new Message(0,0, " ", (i + column_n + 1 + 10*(k+1)), str4);
      (*v).Sent_Messages.Add(m);
```

```
}
else {  // i is ODD


// Messages (1 + 4k)

      //Receives
      for ( j = 0; j <= k ; j++ ) {
            m = new Message((1+4*j),(i + column_n + 10*j),str4, 0, " ");
            (*v).Received_Messages.Add(m);
            }

      //Sends
      for (  j = 0; j <= k ; j++ ) {
            m = new Message((1 + 4*j),0," ",(i + column_n + 8 + 10*j), str1);
            (*v).Sent_Messages.Add(m);
            }


// Messages (2 + 4k)

      //Receives
      for (  j = 0; j <= k ; j++ ) {
            m = new Message((2+4*j),(i + column_n + 9 + 10*j),str2, 0, " ");
            (*v).Received_Messages.Add(m);
            }

      //Sends
      for (  j = 0; j <= k ; j++ ) {
            m = new Message((2 + 4*j),0," ",(i + column_n + 12 + 10*j), str3);
            (*v).Sent_Messages.Add(m);
            }

// Messages (3 + 4k)

      //Receives
      for (  j = 0; j <= k ; j++ ) {
            m = new Message((3 + 4*j),(i + column_n + 4 + 10*j),str1,0, " ");
            (*v).Received_Messages.Add(m);
            }

      //Sends
      for (  j = 0; j <= k ; j++ ) {
            m = new Message((3 + 4*j),0," ",(i + column_n + 5 + 10*j), str2);
            (*v).Sent_Messages.Add(m);
            }

// Messages (4 + 4k)

      //Receives
      for (  j = 0; j <= k ; j++ ) {
      m = new Message((4 + 4*j),(i + column_n + 6 + 10*j), str3,0, " " );
            (*v).Received_Messages.Add(m);
            }

      //Sends
      for (  j = 0; j <= k ; j++ ) {
            m = new Message((4 + 4*j),0," ",(i +  column_n + 7 + 10*j), str4);
            (*v).Sent_Messages.Add(m);
            }
```

```
// Messages (0)

      //Receives

      //Sends

} //end of ELSE

      str1 = "";
      str2 = "";
      str3 = "";
      str4 = "";

} //end of i FOR loop


////////////////////////////////////////////////////////////////////////////////

// Vertex v(3,j) (3 <= j <= (column_n - 3)

for ( j = 3; j <= (column_n-3); j++)

{
v = vertices.Find_Vertex(3,j);

                  str1.append("v(2,");
                  str2.append("v(4,");
                  str3.append("v(3,");
                  str4.append("v(3,");

                  str_index = itos(j);

                  str1.append(str_index);
                  str2.append(str_index);
                  str1.append(")");
                  str2.append(")");

                  str_index = itos(j-1);
                  str3.append(str_index);

                  str_index = itos(j+1);
                  str4.append(str_index);

                  str3.append(")");
                  str4.append(")");

if ( j%2 == 0 )  // j is EVEN
{


// Messages (1 + 4k)

      //Receives
      for ( i = 0; i <= k ; i++ ) {
            m = new Message((1+4*i),(j + 2 + 10*i),str1, 0, " ");
            (*v).Received_Messages.Add(m);
            }

      //Sends
      for (  i = 0; i <= k ; i++ ) {
            m = new Message((1+4*i),0," ",(j + 3 + 10*i), str2);
            (*v).Sent_Messages.Add(m);
            }
```

```
      for (  i = 0; i <= k ; i++ ) {
            m = new Message((1+4*i),0," ",(j + 4 + 10*i), str3);
            (*v).Sent_Messages.Add(m);
            }

// Messages (2 + 4k)

      //Receives
      for (  i = 0; i <= k ; i++ ) {
            m = new Message((2+4*i),(j + 15 + 10*i),str4, 0, " ");
            (*v).Received_Messages.Add(m);
            }

// Messages (3 + 4k)

      //Receives
      for (  i = 0; i <= k ; i++ ) {
            m = new Message((3 + 4*i),(j + 19 + 10*i),str2,0, " ");
            (*v).Received_Messages.Add(m);
            }


// Messages (4 + 4k)

      //Receives
      for (  i = 0; i <= k ; i++ ) {
            m = new Message((4 + 4*i),(j + 10 + 10*i), str3,0, " " );
            (*v).Received_Messages.Add(m);
            }

      //Sends
      for (  i = 0; i <= k ; i++ ) {
            m = new Message((4 + 4*i),0," ",(j + 16 + 10*i), str1);
            (*v).Sent_Messages.Add(m);
            }

      for (  i = 0; i <= k ; i++ ) {
            m = new Message((4 + 4*i),0," ",(j + 11 + 10*i), str4);
            (*v).Sent_Messages.Add(m);
            }


// Messages (0)

      //Receives
      m = new Message(0,(j + 9), str2, 0, " " );
      (*v).Received_Messages.Add(m);

      m = new Message(0,(j + 2 + 10*(k+1)), str1, 0, " ");
      (*v).Received_Messages.Add(m);

      //Sends
      m = new Message(0,0," ",(j + 6),str1);
      (*v).Sent_Messages.Add(m);

      m = new Message(0,0, " ", (j + 3 + 10*(k+1)), str2);
      (*v).Sent_Messages.Add(m);
      }

else {  // j is ODD


// Messages (1 + 4k)
```

```
      //Receives

      for ( i = 0; i <= k ; i++ ) {
           m = new Message((1+4*i),(j + 5 + 10*i),str4, 0, " ");
            (*v).Received_Messages.Add(m);
            }

      //Sends
      for (  i = 0; i <= k ; i++ ) {
            m = new Message((1+4*i),0," ",(j + 6 + 10*i), str1);
            (*v).Sent_Messages.Add(m);
            }

// Messages (2 + 4k)

      //Receives
      for (  i = 0; i <= k ; i++ ) {
            m = new Message((2+4*i),(j + 13 + 10*i),str2, 0, " ");
            (*v).Received_Messages.Add(m);
            }

      //Sends
      for (  i = 0; i <= k ; i++ ) {
            m = new Message((2+4*i),0," ",(j + 14 + 10*i), str3);
            (*v).Sent_Messages.Add(m);
            }

// Messages (3 + 4k)

      //Receives
      for (  i = 0; i <= k ; i++ ) {
            m = new Message((3 + 4*i),(j + 8 + 10*i),str1,0, " ");
            (*v).Received_Messages.Add(m);
            }

      //Sends
      for (  i = 0; i <= k ; i++ ) {
            m = new Message((3 + 4*i),0," ",(j + 9 + 10*i), str2);
            (*v).Sent_Messages.Add(m);
            }

// Messages (4 + 4k)

      //Receives
      for (  i = 0; i <= k ; i++ ) {
            m = new Message((4 + 4*i),(j + 10 + 10*i), str3,0, " " );
            (*v).Received_Messages.Add(m);
            }

      //Sends
      for (  i = 0; i <= k ; i++ ) {
            m = new Message((4 + 4*i),0," ",(j + 11 + 10*i), str4);
            (*v).Sent_Messages.Add(m);
            }

// Messages (0)

      //Receives

      //Sends


} //end of ELSE
```

```
      str1 = "";
      str2 = "";
      str3 = "";
      str4 = "";

} //end of j FOR loop

////////////////////////////////////////////////////////////////////////////////

// Vertex v(i,3) (4 <= i <= (row_m - 3)

for ( i = 4; i <= (row_m - 3); i++)

{
v = vertices.Find_Vertex(i,3);


                  str1.append("v(");
                  str2.append("v(");
                  str3.append("v(");
                  str4.append("v(");

                  str_index = itos(i-1);
                  str1.append(str_index);

                  str_index = itos(i+1);
                  str2.append(str_index);

                  str1.append(",");
                  str2.append(",");

                  str_index = itos(3);

                  str1.append(str_index);
                  str2.append(str_index);
                  str1.append(")");
                  str2.append(")");

                  str_index = itos(i);

                  str3.append(str_index);
                  str4.append(str_index);

                  str3.append(",");
                  str4.append(",");

                  str_index = itos(2);
                  str3.append(str_index);

                  str_index = itos(4);
                  str4.append(str_index);

                  str3.append(")");
                  str4.append(")");


if ( i%2 == 0 )  // i is EVEN
{

// Messages (1 + 4k)
```

```
      //Receives
      for ( j = 0; j <= k ; j++ ) {
            m = new Message((1+4*j),(i + 13 + 10*j),str2, 0, " ");
            (*v).Received_Messages.Add(m);
            }

// Messages (2 + 4k)

      //Receives
      for (  j = 0; j <= k ; j++ ) {
            m = new Message((2+4*j),(i + 4 + 10*j),str3, 0, " ");
            (*v).Received_Messages.Add(m);
            }

      //Sends
      for (  j = 0; j <= k ; j++ ) {
            m = new Message((2+4*j),0, " ", (i + 5 + 10*j),str4);
            (*v).Sent_Messages.Add(m);
            }

      for (  j = 0; j <= k ; j++ ) {
            m = new Message((2+4*j),0, " ", (i + 12 + 10*j),str1);
            (*v).Sent_Messages.Add(m);
            }

// Messages (3 + 4k)

      //Receives
      for (  j = 0; j <= k ; j++ ) {
            m = new Message((3 + 4*j),(i + 8 + 10*j),str1,0, " ");
            (*v).Received_Messages.Add(m);
            }

      //Sends
      for (  j = 0; j <= k ; j++ ) {
            m = new Message((3 + 4*j),0," ",(i + 9 + 10*j), str2);
            (*v).Sent_Messages.Add(m);
            }

                  for (  j = 0; j <= k ; j++ ) {

                        m = new Message((3 + 4*j),0," ",(i + 16 + 10*j),
str3);
                        (*v).Sent_Messages.Add(m);

                  }

// Messages (4 + 4k)

      //Receives
      for (  j = 0; j <= k ; j++ ) {
            m = new Message((4 + 4*j),(i + 21 + 10*j), str4,0, " " );
            (*v).Received_Messages.Add(m);
           }


// Messages (0)

      //Receives
      m = new Message(0,(i + 11), str4, 0, " ");
      (*v).Received_Messages.Add(m);

      m = new Message(0,(i + 4 + 10*(k+1)), str3, 0, " ");
```

```
      (*v).Received_Messages.Add(m);

      //Sends
      m = new Message(0,0, " ", (i + 6), str3);
      (*v).Sent_Messages.Add(m);

      m = new Message(0,0, " ", (i + 5 + 10*(k+1)), str4);
      (*v).Sent_Messages.Add(m);

}

else {  // i is ODD

// Messages (1 + 4k)

      //Receives
      for ( j = 0; j <= k ; j++ ) {
            m = new Message((1+4*j),(i + 5 + 10*j),str4, 0, " ");
            (*v).Received_Messages.Add(m);
            }

      //Sends
      for (  j = 0; j <= k ; j++ ) {
            m = new Message((1 + 4*j),0," ",(i + 12 + 10*j), str1);
            (*v).Sent_Messages.Add(m);
            }

// Messages (2 + 4k)

      //Receives
      for (  j = 0; j <= k ; j++ ) {
            m = new Message((2+4*j),(i + 13 + 10*j),str2, 0, " ");
            (*v).Received_Messages.Add(m);
            }

      //Sends
      for (  j = 0; j <= k ; j++ ) {
            m = new Message((2 + 4*j),0," ",(i + 16 + 10*j), str3);
            (*v).Sent_Messages.Add(m);
            }

// Messages (3 + 4k)

      //Receives
      for (  j = 0; j <= k ; j++ ) {
            m = new Message((3 + 4*j),(i + 8 + 10*j),str1,0, " ");
            (*v).Received_Messages.Add(m);
            }

      //Sends
      for (  j = 0; j <= k ; j++ ) {
            m = new Message((3 + 4*j),0," ",(i + 9 + 10*j), str2);
            (*v).Sent_Messages.Add(m);
            }

// Messages (4 + 4k)

      //Receives
      for (  j = 0; j <= k ; j++ ) {
            m = new Message((4 + 4*j),(i + 10 + 10*j), str3,0, " " );
            (*v).Received_Messages.Add(m);
            }

      //Sends
```

```
      for (  j = 0; j <= k ; j++ ) {
            m = new Message((4 + 4*j),0," ",(i + 11 + 10*j), str4);
            (*v).Sent_Messages.Add(m);
            }

// Messages (0)

      //Receives

      //Sends

} //end of ELSE

      str1 = "";
      str2 = "";
      str3 = "";
      str4 = "";

} //end of i FOR loop

////////////////////////////////////////////////////////////////////////////////

// Vertex v(row_m - 2,3)

v = vertices.Find_Vertex(row_m - 2,3);


                  str1.append("v(");
                  str2.append("v(");
                  str3.append("v(");
                  str4.append("v(");

                  str_index = itos(row_m - 3);
                  str1.append(str_index);

                  str_index = itos(row_m - 1);
                  str2.append(str_index);

                  str1.append(",");
                  str2.append(",");

                  str_index = itos(3);

                  str1.append(str_index);
                  str2.append(str_index);
                  str1.append(")");
                  str2.append(")");

                  str_index = itos(row_m - 2);

                  str3.append(str_index);
                  str4.append(str_index);

                  str3.append(",");
                  str4.append(",");

                  str_index = itos(2);
                  str3.append(str_index);

                  str_index = itos(4);
```

```
                  str4.append(str_index);

                  str3.append(")");
                  str4.append(")");


// Messages (1 + 4k)

      //Receives
      for ( j = 0; j <= k ; j++ ) {
            m = new Message((1+4*j),(row_m + 3 + 10*j),str4, 0, " ");
            (*v).Received_Messages.Add(m);
            }

      //Sends
      for (  j = 0; j <= k ; j++ ) {
            m = new Message((1+4*j),0, " ", (row_m + 10 + 10*j),str1);
            (*v).Sent_Messages.Add(m);
            }


// Messages (2 + 4k)

      //Receives
      for (  j = 0; j <= k ; j++ ) {
            m = new Message((2+4*j),(row_m + 12 + 10*j),str2, 0, " ");
            (*v).Received_Messages.Add(m);
            }

      //Sends
      for (  j = 0; j <= k ; j++ ) {
            m = new Message((2+4*j),0, " ", (row_m + 21 + 10*j),str3);
            (*v).Sent_Messages.Add(m);
            }


// Messages (3 + 4k)

      //Receives
      for (  j = 0; j <= k ; j++ ) {
            m = new Message((3 + 4*j),(row_m + 6 + 10*j),str1,0, " ");
            (*v).Received_Messages.Add(m);
            }

      //Sends
      for (  j = 0; j <= k ; j++ ) {
            m = new Message((3 + 4*j),0," ",(row_m + 7 + 10*j), str2);
            (*v).Sent_Messages.Add(m);
            }


// Messages (4 + 4k)

      //Receives
      for (  j = 0; j <= k ; j++ ) {
            m = new Message((4 + 4*j),(row_m + 8 + 10*j), str3,0, " " );
            (*v).Received_Messages.Add(m);
            }

      //Sends
      for (  j = 0; j <= k ; j++ ) {
            m = new Message((4 + 4*j),0," ",(row_m + 9 + 10*j), str4);
            (*v).Sent_Messages.Add(m);
```

```
            }

// Messages (0)

      //Receives
      m = new Message(0,(row_m + 8 + 10*(k+1)), str3, 0, " ");
      (*v).Received_Messages.Add(m);


      //Sends
      m = new Message(0,0, " ", (row_m + 11), str3);
      (*v).Sent_Messages.Add(m);

      str1 = "";
      str2 = "";
      str3 = "";
      str4 = "";


////////////////////////////////////////////////////////////////////////////////

// Vertex v(row_m - 2,j) (4 <= j <= (column_n - 3)  FROM ODD CASE


for ( j = 4; j <= (column_n-3); j++)

{
v = vertices.Find_Vertex(row_m - 2,j);

                  str1.append("v(");
                  str2.append("v(");
                  str3.append("v(");
                  str4.append("v(");


                  str_index = itos(row_m - 3);
                  str1.append(str_index);

                  str_index = itos(row_m - 1);
                  str2.append(str_index);

                  str1.append(",");
                  str2.append(",");

                  str_index = itos(j);

                  str1.append(str_index);
                  str2.append(str_index);
                  str1.append(")");
                  str2.append(")");

                  str_index = itos(row_m - 2);

                  str3.append(str_index);
                  str4.append(str_index);

                  str3.append(",");
                  str4.append(",");

                  str_index = itos(j-1);
                  str3.append(str_index);

                  str_index = itos(j+1);
```

```
                  str4.append(str_index);


                  str3.append(")");
                  str4.append(")");


if ( j%2 == 0 )  // j is EVEN
{


// Messages (1 + 4k)

      //Receives
      for ( i = 0; i <= k ; i++ ) {
            m = new Message((1+4*i),(row_m + j - 3 + 10*i),str1, 0, " ");
            (*v).Received_Messages.Add(m);
            }


      //Sends
      for (  i = 0; i <= k ; i++ ) {
            m = new Message((1+4*i),0," ",(row_m + j - 2 + 10*i), str2);
            (*v).Sent_Messages.Add(m);
            }

      for (  i = 0; i <= k ; i++ ) {
            m = new Message((1+4*i),0," ",(row_m + j - 1 + 10*i), str3);
            (*v).Sent_Messages.Add(m);
            }

// Messages (2 + 4k)

      //Receives
      for (  i = 0; i <= k ; i++ ) {
            m = new Message((2+4*i),(row_m + j + 10 + 10*i),str4, 0, " ");
            (*v).Received_Messages.Add(m);
            }

// Messages (3 + 4k)

      //Receives
      for (  i = 0; i <= k ; i++ ) {
            m = new Message((3 + 4*i),(row_m + j + 12 + 10*i),str2,0, " ");
            (*v).Received_Messages.Add(m);
            }


// Messages (4 + 4k)

      //Receives
      for (  i = 0; i <= k ; i++ ) {
            m = new Message((4 + 4*i),(row_m + j + 5 + 10*i), str3,0, " " );
            (*v).Received_Messages.Add(m);
            }

      //Sends
      for (  i = 0; i <= k ; i++ ) {
            m = new Message((4 + 4*i),0," ",(row_m + j + 6 + 10*i), str4);
            (*v).Sent_Messages.Add(m);
            }


      for (  i = 0; i <= k ; i++ ) {
```

```
            m = new Message((4 + 4*i),0," ",(row_m + j + 13 + 10*i), str1);
            (*v).Sent_Messages.Add(m);
            }

// Messages (0)

      //Receives
            m = new Message(0,(row_m + j + 2), str2, 0, " " );
            (*v).Received_Messages.Add(m);

            m = new Message(0,(row_m + j - 3 + 10*(k+1)), str1, 0, " ");
            (*v).Received_Messages.Add(m);

      //Sends
            m = new Message(0,0," ",(row_m + j + 3),str1);
            (*v).Sent_Messages.Add(m);

            m = new Message(0,0, " ", (row_m + j - 2 + 10*(k+1)), str2);
            (*v).Sent_Messages.Add(m);
      }
else {  // j is ODD



// Messages (1 + 4k)

      //Receives
      for ( i = 0; i <= k ; i++ ) {
            m = new Message((1+4*i),(row_m + j + 10*i),str4, 0, " ");
            (*v).Received_Messages.Add(m);
            }

      //Sends
      for (  i = 0; i <= k ; i++ ) {
            m = new Message((1+4*i),0," ",(row_m + j + 7 + 10*i), str1);
            (*v).Sent_Messages.Add(m);
            }


// Messages (2 + 4k)

      //Receives
      for (  i = 0; i <= k ; i++ ) {
            m = new Message((2+4*i),(row_m + j + 2 + 10*i),str2, 0, " ");
            (*v).Received_Messages.Add(m);
            }

      //Sends
      for (  i = 0; i <= k ; i++ ) {
            m = new Message((2+4*i),0," ",(row_m + j + 9 + 10*i), str3);
            (*v).Sent_Messages.Add(m);
            }


// Messages (3 + 4k)

      //Receives
      for (  i = 0; i <= k ; i++ ) {
            m = new Message((3 + 4*i),(row_m + j + 3 + 10*i),str1,0, " ");
            (*v).Received_Messages.Add(m);
            }

      //Sends
      for (  i = 0; i <= k ; i++ ) {
```

```
            m = new Message((3 + 4*i),0," ",(row_m + j + 4 + 10*i), str2);
            (*v).Sent_Messages.Add(m);
            }


// Messages (4 + 4k)

      //Receives
      for (  i = 0; i <= k ; i++ ) {
            m = new Message((4 + 4*i),(row_m + j + 5 + 10*i), str3,0, " " );
            (*v).Received_Messages.Add(m);
            }


      //Sends
      for (  i = 0; i <= k ; i++ ) {
            m = new Message((4 + 4*i),0," ",(row_m + j + 6 + 10*i), str4);
            (*v).Sent_Messages.Add(m);
            }


// Messages (0)

      //Receives

      //Sends


} //end of ELSE

      str1 = "";
      str2 = "";
      str3 = "";
      str4 = "";

} //end of j FOR loop


////////////////////////////////////////////////////////////////////////////////


// Vertex v(i,column_n - 2) (4 <= i <= (row_m - 2)

for ( i = 4; i <= (row_m - 2); i++)

{
v = vertices.Find_Vertex(i,column_n - 2);


                  str1.append("v(");
                  str2.append("v(");
                  str3.append("v(");
                  str4.append("v(");

                  str_index = itos(i-1);
                  str1.append(str_index);

                  str_index = itos(i+1);
                  str2.append(str_index);

                  str1.append(",");
                  str2.append(",");

                  str_index = itos(column_n - 2);
```

```
                    str1.append(str_index);
                    str2.append(str_index);
                    str1.append(")");
                    str2.append(")");

                    str_index = itos(i);

                    str3.append(str_index);
                    str4.append(str_index);

                    str3.append(",");
                    str4.append(",");

                    str_index = itos(column_n - 3);
                    str3.append(str_index);

                    str_index = itos(column_n - 1);
                    str4.append(str_index);


                    str3.append(")");
                    str4.append(")");


if ( i%2 == 0 )  // i is EVEN
      {

// Messages (1 + 4k)

      //Receives
      for ( j = 0; j <= k ; j++ ) {
            m = new Message((1+4*j),(column_n + i - 3 + 10*j),str1, 0, " ");
            (*v).Received_Messages.Add(m);
            }

      //Sends
      for (  j = 0; j <= k ; j++ ) {
            m = new Message((1+4*j),0, " ", (column_n + i - 2 + 10*j),str2);
            (*v).Sent_Messages.Add(m);
            }

// Messages (2 + 4k)

      //Receives
      for (  j = 0; j <= k ; j++ ) {
            m = new Message((2+4*j),(column_n + i - 1 + 10*j),str3, 0, " ");
            (*v).Received_Messages.Add(m);
            }

      //Sends
      for (  j = 0; j <= k ; j++ ) {
            m = new Message((2+4*j),0, " ", (column_n + i + 10*j),str4);
            (*v).Sent_Messages.Add(m);
            }


// Messages (3 + 4k)

      //Receives
      for (  j = 0; j <= k ; j++ ) {
            m = new Message((3 + 4*j),(column_n + i + 12 + 10*j),str4,0, " ");
            (*v).Received_Messages.Add(m);
            }
```

```
      //Sends
      for (  j = 0; j <= k ; j++ ) {
            m = new Message((3 + 4*j),0," ",(column_n + i + 13 + 10*j), str1);
            (*v).Sent_Messages.Add(m);
            }


// Messages (4 + 4k)

      //Receives
      for (  j = 0; j <= k ; j++ ) {
      m = new Message((4 + 4*j),(column_n + i + 14 + 10*j), str2,0, " " )
            (*v).Received_Messages.Add(m);
            }

      for (  j = 0; j <= k ; j++ ) {
            m = new Message((4 + 4*j),0," ",(column_n + i + 15 + 10*j), str3);
            (*v).Sent_Messages.Add(m);
            }

// Messages (0)

      //Receives
      m = new Message(0,(column_n + i + 2), str4, 0, " ");
      (*v).Received_Messages.Add(m);

      m = new Message(0,(column_n + i + 4), str2, 0, " ");
      (*v).Received_Messages.Add(m);

      m = new Message(0,(column_n + i - 1 + 10*(k+1)), str3, 0, " ");
      (*v).Received_Messages.Add(m);

      m = new Message(0,(column_n + i - 3 + 10*(k+1)), str1, 0, " ");
      (*v).Received_Messages.Add(m);

      //Sends
      m = new Message(0,0, " ", (column_n + i + 3), str1);
      (*v).Sent_Messages.Add(m);

      m = new Message(0,0, " ", (column_n + i + 5), str3);
      (*v).Sent_Messages.Add(m);

      m = new Message(0,0, " ", (column_n + i + 10*(k+1)), str4);
      (*v).Sent_Messages.Add(m);

      m = new Message(0,0, " ", (column_n + i - 2 + 10*(k+1)), str2);
      (*v).Sent_Messages.Add(m);
   }
      else {  // i is ODD


// Messages (1 + 4k)

      //Receives
      for ( j = 0; j <= k ; j++ ) {
            m = new Message((1+4*j),(column_n + i - 3 + 10*j),str1, 0, " ");
            (*v).Received_Messages.Add(m);
            }

      //Sends
      for (  j = 0; j <= k ; j++ ) {
            m = new Message((1 + 4*j),0," ",(column_n + i - 2 + 10*j), str2);
            (*v).Sent_Messages.Add(m);
```

```
            }

      for (  j = 0; j <= k ; j++ ) {
            m = new Message((1 + 4*j),0," ",(column_n + i - 1 + 10*j), str3);
            (*v).Sent_Messages.Add(m);
            }

// Messages (2 + 4k)

      //Receives
      for (  j = 0; j <= k ; j++ ) {
            m = new Message((2+4*j),(column_n + i + 12 + 10*j),str4, 0, " ");
            (*v).Received_Messages.Add(m);
            }

// Messages (3 + 4k)

      //Receives
      for (  j = 0; j <= k ; j++ ) {
            m = new Message((3 + 4*j),(column_n + i + 14 + 10*j),str2,0, " ");
            (*v).Received_Messages.Add(m);
            }

// Messages (4 + 4k)

      //Receives
      for (  j = 0; j <= k ; j++ ) {
      m = new Message((4 + 4*j),(column_n + i + 5 + 10*j), str3,0, " " );
      (*v).Received_Messages.Add(m);
      }

                  //Sends
      for (  j = 0; j <= k ; j++ ) {
            m = new Message((4 + 4*j),0," ",(column_n + i + 6 + 10*j), str4);
            (*v).Sent_Messages.Add(m);
            }

      for (  j = 0; j <= k ; j++ ) {
            m = new Message((4 + 4*j),0," ",(column_n + i + 13 + 10*j), str1);
            (*v).Sent_Messages.Add(m);
            }

// Messages (0)

            //Receives
            m = new Message(0,(column_n + i + 4), str2, 0, " ");
            (*v).Received_Messages.Add(m);

            m = new Message(0,(column_n + i - 3 + 10*(k+1)), str1, 0, " ");
            (*v).Received_Messages.Add(m);

            //Sends
            m = new Message(0,0, " ", (column_n + i + 3), str1);
            (*v).Sent_Messages.Add(m);

            m = new Message(0,0, " ", (column_n + i - 2 + 10*(k+1)), str2);
            (*v).Sent_Messages.Add(m);

      } //end of ELSE

      str1 = "";
      str2 = "";
      str3 = "";
      str4 = "";
```

```
} //end of i FOR loop


////////////////////////////////////////////////////////////////////////////////

      // Vertex v(i,j) (4 <= i <= (row_m - 3), 4 <= j <= (column_n - 3)


      for (i = 4; i <= (row_m - 3); i++) {

            for ( j = 4; j <= (column_n-3); j++)
            {
            v = vertices.Find_Vertex(i,j);
                  calculate_vertex_i_j ( v, i, j);
            }
      }

vertices.Report_Vertex();
vertices.Report_Max_time();

int max_rec_t = vertices.Max_received_t();
fout<< "\nMaximum received time for grid is " << max_rec_t;
cout<< "\nMaximum received time for grid is " << max_rec_t;

int max_sent_t = vertices.Max_sent_t();
fout<< "\nMaximum sent time for grid is " << max_sent_t<<"\n";
cout<< "\nMaximum sent time for grid is " << max_sent_t<<"\n";

fout.close();

} //end of main
```