SOME RELATIONS OF SUBSEQUENCES IN PERMUTATIONS TO GRAPH THEORY WITH ALGORITHMIC APPLICATIONS

by

D. ROTEM

A thesis submitted to the Faculty of Science of the University of the Witwatersrand in fulfillment of the requirements for the degree of Doctor of Philosophy.

Johannesburg, 1977.

DECLARATION OF CANDIDATE

I hereby declare that this thesis is my own work, and that the material forming the basis of this work has not been incorporated in any thesis submitted by me to any other university for degree purposes.

D. Rotem

D. ROTEM

ACKNOWLEDGEMENTS

I would like to express my deep gratitude to my thesis Supervisors Prof. D.S. Henderson and Prof. D. Hawkins for their interest and encouragement in the research project.

I would also like to thank Dr. Y. Varol and Dr. D. Brown for the invaluable help they gave me in the preparation of this thesis.

TO

Batsheva and Yonatan

CONTENTS

INDEX TO FIGURES

SYMBOL TABLE

$\|A\|$	The cardinality of the set A.
$\lfloor x \rfloor$	Floor of x , the greatest integer which does not exceed x.
$\lceil x \rceil$	Ceiling of x , the least integer which is not smaller than x.
$L_\pi(j)$.............	Elements on the left of j in the permutation Π.
$R_\pi(j)$.............	" " " right " " " " " ".
SS_n	Stack-sortable permutations of order n.
C_n	nth catalan number = $(n+1)^{-1}\binom{2n}{n}$.
$<x_1,x_2,...,x_n>$....	Ordered set.
$R_T(i)$	The right subtree of node i in the binary tree T.
$L_T(i)$	The left subtree of node i in the binary tree T.

ABSTRACT

The representation of some types of graphs as permutations, is utilized in devising efficient algorithms on those graphs. Maximum cliques in permutation graphs and circle graphs are found, by searching for a longest ascending or descending subsequence in their representing permutation.

The correspondence between n-noded binary trees and the set SS_n of stack-sortable permutations, forms the basis of an algorithm for generating and indexing such trees.

The relations between a graph and its representing permutation, are also employed in the proof of theorems concerning properties of subsequences in this permutation. In particular, expressions for the average lengths of the longest ascending and descending subsequence, in a random member of SS_n, and the average number of inversions in such a permutation, are derived using properties of binary trees. Finally, a correspondence between the set SS_n, and the set of permutations of order n with no descending subsequence of length 3, is demonstrated.

CHAPTER ONE

INTRODUCTION

1.1 A GENERAL OUTLINE OF THE RESEARCH

Graph theory provides a natural tool for modeling and solving problems which involve a finite set of objects V , and relationships between pairs of objects in V. In practice , most applications which call for a graph theoretical approach , require graphs of large size for their representation. The successful analysis of such graphs is dependent on the availability of fast computers as well as efficient graph theoretical algorithms. Such algorithms may be required, for example, to determine whether a given graph possesses a required property , or to construct all graphs or subgraphs of a particular kind. The input graph for an algorithm , may be stored in many ways, the choice of a representation method depends on the graph itself, and on the nature of the operations which are performed on it by the algorithm .

In this thesis, we are concerned with some types of graphs which lend themselves to a convenient representation by a permutation on a ~~set or multiset~~ of integers. The existence of a required property of the graph , can then be checked by searching for a particular subsequence in the representing permutation. Also, all distinct graphs with a certain property, can be constructed by generating all their representing permutations.

This approach , of reducing a graph theoretical problem to a search for a pattern in a permutation, leads in many cases to algorithms which outperform their counterparts which are based on other methods of representation such as adjacency or incidence matrices.

In particular two types of graphs are considered;

(a) permutation graphs and graphs related to them,

(b) binary trees.

In either case, theoretical results concerning subsequences in the representing permutations are derived. These results are further scrutinized for their practical algorithmic implications.

In Chapter 2 , we first deal with the problem of deciding whether a given labelled graph is a permutation graph. In case of a positive answer , its representing (defining) permutation is constructed. Next , some practical problems which are solved in [1] by finding a maximum clique in a permutation graph, are attacked here by constructing a representing permutation, and then employing Schensted's algorithm for finding a longest monotonic subsequence in integer sequences[2] . A generalization of this algorithm is given , whereby all longest monotonic subsequences can be generated , this can be used for obtaining a set of alternative solutions to the problem.

A fast algorithm for finding a maximum clique in a circle graph is presented. In this case , a representing sequence is produced for each vertex of the graph,and Schensted's algorithm is then applied to each one of those sequences. In the last section , a generalization of Erdos' theorem [3] on monotonic subsequences is proved by extending the idea of permutation graphs to permutation hypergraphs.

The subject of Chapter 3 is the class SS_n of stack sortable permutations of order n. Those permutations provide a useful device for the representation of binary trees [4, pp: 329]. The behaviour of some types of subsequences in a member of SS_n, and their relation to the corresponding binary tree, are studied in detail. It is observed that members of SS_n tend to be more 'ordered' than ordinary permutations in the sense that on the average they contain less inversions, longer maximum ascending subsequences and shorter maximum descending subsequences.

Characterizations of SS_n are given using permutation graphs and inversion tables. The latter result is utilized in the the algorithms of the next chapter.

In chapter 4, algorithms for generating binary trees and indexing them in a systematic manner are presented. These algorithms can be also used to generate a random binary tree, or store a binary tree of order n as an integer smaller than $C_n=(n+1)^{-1}\binom{2n}{n}$. A comparative evaluation of these algorithms show that they are superior to similar algorithms which are based on Knuth's [4] natural order among binary trees.

In Chapter 5, a connection between the subsequences considered in Chapter 2 and those of Chapters 3 and 4 is pointed out by demonstrating a direct correspondence between SS_n and the set of permutations of order n which do not contain a descending subsequence of length 3. The problem of extending this result to other types of subsequences is also considered.

The last chapter contains a summary of the main results of the thesis as well as some problems for further research which are raised but not answered by this work.

1.2 PRELIMINARIES AND NOTATIONS

Let $\Pi=<p_1,p_2,..,p_n>$ be a permutation on the set $N=\{1,2,..,n\}$. A descending subsequence of length k in Π satisfies,

$$p_{i_1}> p_{i_2}>\ldots>p_{i_k} \quad \text{and} \quad i_1<i_2<\ldots<i_k . \tag{1.1}$$

A descending subsequence is maximal in Π if no element of Π can be added to it without violating its monotonicity. A longest descending subsequence in Π (LDS) contains the maximum number of elements among all descending subsequences in Π. We get the corresponding definitions for ascending subsequences by replacing '>' with '<' in (1.1), where LAS stands for 'longest ascending subsequence'. For $j\in N$, we denote by $R_\pi(j)$ the set of elements to the right of j in Π, and by $L_\pi(j)$ the set of elements to the left of j in Π. Two elements p_i and p_j form an inversion in Π if $(p_i-p_j)(i-j)<0$. The SR (Small to the Right) inversion-table of Π, is a vector $<b_1,b_2,..,b_n>$ such that for $1\le i\le n$ b_i counts the number of elements in $R_\pi(i)$ which are smaller than i. In the same way, the SL (Small to the Left), BR (Big to the Right) and BL (Big to the Left) inversion-table of Π is a vector whose i^{th} entry counts the elements which are related to i as indicated by its name. It is well-known, that an inversion-table (no matter of which type) uniquely determines its corresponding permutation. We denote by Π^{-1} the inverse permutation of Π, if $\Pi=\Pi^{-1}$ it is called an involution.

Example 1.1

Let $\Pi=<3,6,4,5,2,1>$. Then $<3,2,1>$ is a maximal descending subsequence in Π, $<6,4,2,1>$ and $<3,4,5>$ are a LDS and a LAS respectively in Π, $R_\pi(4)=<5,2,1>$ and $L_\pi(6)=<3>$. The BR inversion-table of Π is $<0,0,3,1,0,0>$. □

A <u>Standard Young Tableau</u> (SYT) of 'shape' $<r_1,r_2,..,r_m>$, where for $1\le i\le m-1$ $r_i \ge r_{i+1}$ and $r_m \ge 1$, is an arrangement of $r_1+r_2+...+r_m$ distinct natural numbers in an array $S=\{s_{ij}\}$ where,

(1) S has m rows with r_i elements in its i^{th} row,

(2) the elements in each row and each column of S form an ascending sequence.

<u>Example 1.2</u> The following SYT has the 'shape' $<3,3,2,1>$;

```
1 2 4
3 5 7
6 8
9
```

□

A SYT S(Π), is formed from a permutation $\Pi=<p_1,p_2,..,p_n>$ by the following construction due to Schensted [2],

<u>Construction-S</u>

For $1\le i\le n$, insert p_i into S as follows;

(1) Search in the first row of S for the first element (from the left) which is greater than p_i, if no such element exists place p_i at the end of the first row, else call the element found a 'bumped element' and insert p_i in its place.

(2) If a 'bumped element' is found, repeat the process of (1) on the second row, where the 'bumped element ' plays the role of p_i.

(3) Repeat this process row by row until some 'bumped element' is placed at the end of a row. □

<u>Example 1.3</u> Let $\Pi=<6,3,1,2,5,4>$, S is showed after each insertion;

```
6    3    1    1 2    1 2 5    1 2 4
     6    3    3      3        3 5
          6    6      6        6        □
```

A graph G consists of a vertex set V and an edge set E, such that each edge in E is associated with two vertices in V called its end points. We consider here only graphs which have no two edges with the same two end points (parallel edges), and no edge for which its two end points are the same (self loop). Two vertices are adjacent if they are the end points of the same edge , this is denoted by $v_i \underset{G}{—} v_j$, otherwise they are non-adjacent denoted by $v_i \underset{G}{\not—} v_j$. The complement of G , denoted by G^c , has the same vertex set as G, two vertices are adjacent in G^c iff they are non-adjacent in G.

A set of vertices $C \subseteq V$, is completely connected if every pair of vertices in C are adjacent. If no other vertices of V can be added to C without violating this property, C is a clique of G. A maximum clique is the one with the largest number of vertices of all cliques.

The chromatic number of a graph is the minimal number of colours needed to colour its vertex set , such that no two adjacent vertices are assigned the same colour. The set V of a graph with a chromatic number m, can be partitioned into m disjoint sets $V_1, V_2, .., V_m$ each containing the vertices of one of the m colours, such a partition is called a minimal chromatic decomposition of G . A graph G is γ-perfect if the size of its maximum clique is equal to its chromatic number.

A path of length k in G , is a sequence of edges $e_1, e_2, .., e_k$, such that e_i and e_{i+1} have a common end point, and no vertex is traversed more than once. A path is denoted by the sequence of vertices on it $<v_0, v_1, .., v_k>$, in the order of their traversal.

A direction can be assigned to the edge $v_i \underset{G}{\overline{\quad}} v_j$, this is denoted by $v_i \to v_j$. If all edges of G are assigned a direction, it is called a <u>digraph</u> (directed graph). A digraph is <u>transitive</u> if for $v_i, v_j, v_k \in V$, the existence of $v_i \to v_j$ and $v_j \to v_k$ implies $v_i \to v_k$. A graph G is <u>transitively orientable</u> (TRO) , if it is possible to orient all its edges such that its directed image $\vec{G}$ is transitive. We now present an algorithm given in [1] , which finds a transitive orientation of a given graph G, if G is TRO , otherwise the algorithm terminates as soon as it detects that no transitive orientation exists. We first define two rules which are used by the algorithm.

<u>Rule 1</u> ; If $v_i \to v_j$, $v_j \underset{G}{\overline{\quad}} v_k$ and $v_i \underset{G}{\not\overline{\quad}} v_k$ assign the direction $v_j \leftarrow v_k$.

<u>Rule 2</u> ; If $v_i \leftarrow v_j$, $v_j \underset{G}{\overline{\quad}} v_k$ and $v_i \underset{G}{\not\overline{\quad}} v_k$ assign the direction $v_j \to v_k$.

A contradiction can occur, if the application of one of the rules requires that an edge which is already directed , must be assigned a different direction. This is used by the following algorithm to detect graphs which are not TRO.

<u>ALGORITHM 1.1</u>

Step 1 : Choose an edge and direct it arbitrarily.

Step 2 : Use Rules 1 and 2 as long as they are applicable, if a contradiction occurs , stop , G is not TRO.

Step 3 : Delete all edges which were directed in Steps 1 or 2, if no edges are left , stop , the graph is TRO , and $\vec{G}$ thus obtained is transitive , else go to Step 1.□

Let G(N) be a graph which has its vertices labelled by the set N . Then G(N) has a <u>defining permutation</u> with respect to its labelling, if there is a permutation Π on N such that;

$i \underset{G(N)}{\overline{\quad\quad}} j$ (vertices are called by their labels) iff i and j form an inversion in Π . (1.2)

A graph G is a permutation graph, if at least one of the possible labellings of its vertices with N , gives rise to a defining permutation.

Example 1.2 A permutation graph G,with two labellings and their respective defining permutations , is shown in Fig 1.1.

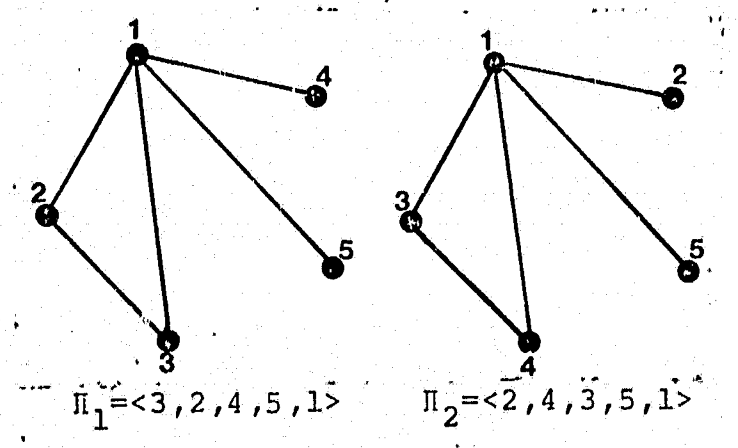


Figure 1.1

$\square$

The next theorem of [1] demonstrates the connection between permutation graphs and transitively-orientable graphs.

Theorem 1.1 A graph G is a permutation graph iff both G and G^c are TRO graphs.

1.3 BIBLIOGRAPHIC NOTES

One of the earliest results on monotonic subsequences in permutations , is due to Erdos and Szekeres [3] . They showed that every permutation of n numbers , has either a descending subsequence of length k+1 or an ascending subsequence of length $\geq \frac{n}{k}$. The combinatorial properties of several types of monotonic subsequences which appear in a permutation , were studied by Brock and Baer [6] . Their research was chiefly inspired by applications in sorting algorithms .The main result of the paper was a counting formula for the permutations of order n , in which the sum of the lengths of the LDS and LAS is n+1.

The results of [6] were significantly extended by Schensted [2] who counted the permutations of order n which contain a LAS and LDS of any prescribed lengths. The enumeration in [2] was based on the construction of a SYT from a permutation (Construction-S), and the correspondence between a permutation and a pair of equally shaped SYT.

In a later paper, Brock and Baer [7] calculated extensive tables for the average lengths of monotonic subsequences in permutations. According to their calculations , the average length of the LDS in a random permutation of order n, shows good agreement with $2\sqrt{n}$. However , the proof of this fact is still an unsolved problem. One result in this direction was given by Dixon [8] who showed that in a random permutation of order n , the probability that the lengths of both the LAS and LDS lie in the range $(e^{-1}\sqrt{n}, e\sqrt{n})$ tends to 1 as $n\to\infty$.

A directed graph which is closely related to a permutation graph , appears in [9 , pp 137] in connection with the proof that the 'permutohedron' is a lattice. A similar graph called 'inversion digraph' , is introduced by Knuth [10] . The inversion digraph' is constructed from a sequence so that its vertices are the elements of the sequence ,and its edges correspond to the non-inversions in the sequence, where the directions are from low to high. It was shown that the SYT of a sequence can be constructed from its 'inversion digraph' by using a topological sort. Permutation graphs, in the context in which we use them, were presented by Pnueli et al in [11], and used for solving some practical problems in [12] .

Properties and characterizations of stack-sortable permutations , are given in [4, pp 239] . A more general model of a network of stacks or queues in parallel , was studied by Even and Itai [5] . They used permutation graphs and circle graphs for characterizing those permutations which can be sorted in such networks. Some other results in this direction were reported by Tarjan [13] , who found relations between the length of monotonic subsequences in permutations,and the number of stacks or queues which are needed to sort it in a network. Some additional bibliographic notes are given in the introductions of the relevant chapters.

CHAPTER TWO

PERMUTATION GRAPHS AND MONOTONIC SUBSEQUENCES IN PERMUTATIONS

2.1 INTRODUCTION

In [1,pp 183] Even presents an algorithm for finding the defining permutation of a labelled graph if one exists.
We seperate this problem into two cases;

(i) the graph is known to have a defining permutation with respect to its labelling,

(ii) no additional information on the graph is known.

The algorithm presented here for case (ii) is shown to be superior to that of Even , it is further shown that utilization of the additional information enables a yet more efficient algorithm to be developed for case (i).

Next, a generalization of Schensted's [2] method for finding a LDS in a permutation is presented. It is shown that this generalized algorithm can be used for generation of all maximum cliques of a permutation graph once its defining permutation is given.

Based on the above algorithms, methods are devised for solving problems of dynamic storage allocation and design of connection boards , those methods are shown to be faster than the existing methods .

The subject of the next section is circle graphs which are closely related to permutation graphs [14]. By using the techniques developed in this chapter, we devise an algorithm which finds a maximum clique in a circle graph in $O(n^2 lg_2 n)$ steps.

To the best of our knowledge, the fastest algorithm to date for this problem is the one given by Gavril [14] which requires $O(n^3)$ steps.

Finally , in the last section a theorem by Erdos and Szekeres is generalized by extending the idea of a permutation graph to that of a permutation hypergraph.

2.2 AN ALGORITHM FOR FINDING THE DEFINING PERMUTATION OF A LABELLED PERMUTATION GRAPH.

Let G(N) be a given graph with $|V| = n$. Suppose that it is known that G(N) has a defining permutation Π with respect to this labelling. In this section we present an algorithm which finds Π. An application of this algorithm is given in Section 2.6.

Let G(N) be represented by its adjacency matrix $A=\{a_{ij}\}$. Since Π is the defining permutation of G(N), $a_{ij}=1$ iff i and j form an inversion in Π . A sequence of distinct integers from N which forms a subsequence of Π is called Π-consistent. The general procedure for finding Π is similar to the well-known 2-way merge sort [16 pp 160]. We start initially with n unit-length sequences which are trivially Π-consistent. In the ith pass, the sequences which were created in the i-1st pass are merged in pairs so that longer Π-consistent sequences are obtained. The algorithm terminates when we are left with one sequence of length n.

We only present here an algorithm for the basic merge operation. In Algorithm 2.1 two Π-consistent sequences are merged using the matrix A. It is then proved in Lemma 2.1 that the output sequence remains Π-consistent. The difference between Algorithm 2.1 and the ordinary merge is that each comparison is followed by a lookup in A in order to decide which of the two compared elements is to be written on the output. Let i and j be the two elements which are currently compared, then max(i,j) is written iff $a_{ij}=1$ and otherwise min(i,j).

ALGORITHM 2.1

Given as input two sequences $X=<x_1,x_2,..,x_m>, Y=<y_1,y_2,..,y_\ell>$ and the matrix A. The sequences X and Y are merged to form an output sequence of length $m+\ell$. The elements of X and Y are distinct.

```
Step 1  :  Set i=1, j=1.
Step 2  :  If $a_{x_i y_j}=0$ set $a_{x_i y_j}=-1$.
Step 3  :  Set $P=(x_i-y_j)a_{x_i y_j}$
Step 4  :  If P>0 go to Step 8.
Step 5  :  Write $y_j$ on output.
Step 6  :  If $j<\ell$ set j=j+1 and go to Step 2.
Step 7  :  Write $x_i,x_{i+1},..x_m$ on output and stop.
Step 8  :  Write $x_i$ on output.
Step 9  :  If i<m set i=i+1 and go to Step 2.
Step 10 :  Write $y_j,y_{j+1},..,y_\ell$ on output and stop. □
```

Example 2.1

We find the defining permutation of the graph of Fig 2.1 which is represented by the adjacency matrix A.

$$A = \begin{array}{c|cccccccc|} & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 2 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 3 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 4 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 5 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 6 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 7 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 8 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 0 \end{array}$$

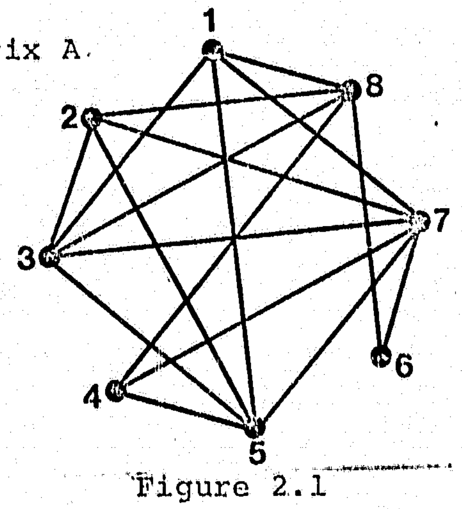


Figure 2.1

We start with the initial sequences:

$S_0^1=<1>$ $S_0^2=<2>$ $S_0^3=<3>$ $S_0^4=<4>$ $S_0^5=<5>$ $S_0^6=<6>$ $S_0^7=<7>$ $S_0^8=<8>$

Pass 1 : $S_1^1=<1,2>$ $S_1^2=<3,4>$ $S_1^3=<5,6>$ $S_1^4=<7,8>$

Pass 2 : $S_2^1=<3,1,2,4>$ $S_2^2=<7,5,8,6>$

Pass 3 : $S_3^1=<7,5,8,3,1,2,4,6>$

□

Lemma 2.1 Let $X=\langle x_1, x_2, \ldots, x_m\rangle$ and $Y=\langle y_1, y_2, \ldots, y_\ell\rangle$ be two Π-consistent sequences with distinct elements which are merged by Algorithm 2.1 and let $z=\langle z_1, \ldots, z_{m+\ell}\rangle$ be the output. Then Z is Π-consistent.

Proof We use the following simple properties of Z.

Property 1 : For $i<j$ if z_i and z_j come from the same input sequence then $z_i \in L_\pi(z_j)$.

Property 2 : If two consecutive elements z_i and z_{i+1} come from different input sequences they are compared in Step 3 and this ensures tha'. $z_i \in L_\pi(z_{i+1})$.

From Properties 1 and 2 it follows that for $1 \le i \le m+\ell-1$ $z_i \in L_\pi(z_{i+1})$ and then by transitivity Z is Π-consistent. □

Running-time

By the nature of 2-way merge sort, if there are k sequences before the i^{th} pass then this number is reduced to $\lceil \frac{k}{2} \rceil$ after the pass. In each pass we make no more than $C \cdot n$ elementary operations where C is a constant which depends mainly on the cost of a matrix lookup. We conclude that the total number of steps which are required to find the defining permutation of a graph of n vertices is $C \lceil \lg_2 n \rceil n$.

2.3 AN ALGORITHM FOR DETECTING WHETHER A LABELLED GRAPH HAS A DEFINING PERMUTATION

Let G(N) be a labelled graph as in the previous section, but assume that it is not known whether a defining permutation exists with respect to this labelling. In this section, we present an algorithm which either finds the defining permutation of G(N) or stops as soon as it detects that no such permutation exists. The algorithm is based on the following theorem which characterizes a graph which has a defining permutation.

Theorem 2.1 Given a graph G(N) which is represented by the adjacency matrix $A=\{a_{ij}\}$, let the matrix $\bar{A}=\{\bar{a}_{ij}\}$ be obtained from A as follows,

$$\begin{array}{lll} \text{For} \quad j<i & a_{ij}=0 & \Rightarrow \quad \bar{a}_{ij}=1 \\ & a_{ij}=1 & \Rightarrow \quad \bar{a}_{ij}=0 \\ \text{Otherwise} & a_{ij}=\bar{a}_{ij} & \end{array}$$

The graph G(N) has a defining permutation iff for $i\neq j$ it follows that $S_i\neq S_j$ where $S_i=\sum_{\ell=1}^{n}\bar{a}_{i\ell}$.

Proof (a) Necessity; Let A be the adjacency matrix of the permutation graph G(N) which has Π as its defining permutation. The term $\sum_{\ell=1}^{i}\bar{a}_{i\ell}$ counts those elements which are smaller than i and do not form an inversion with i in Π . In a similar way , the term $\sum_{\ell>i}^{\ell=n}\bar{a}_{i\ell}$ counts all elements greater than i which form an inversion with i in Π. Therefore $S_i=|L_\pi(i)|$.

We then have,

$$i\neq j \Rightarrow |L_\pi(i)|\neq |L_\pi(j)| \Rightarrow S_i\neq S_j \quad . \tag{2.1}$$

(b) Sufficiency; We use induction on the size of the adjacency matrix. The theorem holds trivially for the two possible adjacency matrices of order 2. Assume that it is true for matrices of order n-1. Let A be the adjacency matrix of the graph G(N). Consider the vector $S=\langle S_1+1, S_2+1,..,S_n+1\rangle$ which we call the S-vector of A. Since all components of S are distinct integers and $0 \le S_i \le n-1$ for $1 \le i \le n$, it follows that S is a permutation on the set $N=\{1,2,..,n\}$. Let $\Pi=\langle p_1,p_2,..,p_n\rangle$ be the inverse permutation of S, we prove our theorem by showing that Π is the defining permutation of G(N). We have to show that for $1 \le i,j \le n$,

i and j form an inversion in Π iff $a_{ij}=1$. (2.2)

Let $p_1=\ell$, we observe that all pairs of the form $\langle \ell,j\rangle$ are inversions in Π iff $j<\ell$. By the definition of Π it follows that $S_\ell=0$, and this implies that $\bar{a}_{\ell j}=0$ for $1 \le j \le n$. We have,

$$\begin{aligned} &\text{for } j<\ell \quad \bar{a}_{\ell j}=0 \Rightarrow a_{\ell j}=1 \quad \text{and} \\ &\text{for } j>\ell \quad \bar{a}_{\ell j}=0 \Rightarrow a_{\ell j}=0 . \end{aligned} \qquad (2.3)$$

We conclude that no violations of condition (2.2) occur in the ℓth row and by symmetry in the ℓth column of A. Consider the ℓth column of A, by (2.3) and the symmetry of A,

$$\begin{aligned} j<\ell &\Rightarrow a_{j\ell}=1 \Rightarrow \bar{a}_{j\ell}=1 \\ j>\ell &\Rightarrow a_{j\ell}=0 \Rightarrow \bar{a}_{j\ell}=1 \end{aligned} \qquad (2.4)$$

Therefore each element of this column contributes a unit to S_i for $1 \le i \le n$ and $i \ne \ell$. Let A' be the matrix which is obtained by eliminating the ℓth row and column from A. We denote by $S'=\langle S_1'+1,S_2'+1,..,S_{n-1}'\rangle$ the S-vector of A'. By the above argument we get the following relation between S and S',

$$\begin{aligned} &\text{for } 1 \le i<\ell \quad S_i'=S_i-1 \quad \text{and} \\ &\text{for } \ell \le i \le n-1 \quad S_i'=S_{i+1}-1 . \end{aligned} \qquad (2.5)$$

By (2.5) for $i \neq j$ we have $S'_i \neq S'_j$, therefore A' satisfies the condition of the theorem. By the induction hypothesis , the permutation $\Pi' = \langle \ell_1, \ell_2, \ldots, \ell_{n-1} \rangle$ such that $\Pi' = S'^{-1}$, is the defining permutation of A'. The elements of Π and Π' are related as follows,

$$\text{for } 1 \leq i \leq n-1 \qquad \begin{array}{ll} \ell_i = p_{i+1} & \text{if } p_{i+1} < p_1 \\ \ell_i = p_{i+1} - 1 & \text{if } p_{i+1} > p_1 \, . \end{array} \tag{2.6}$$

Hence $\langle \ell_i, \ell_j \rangle$ is an inversion in Π' iff $\langle p_{i+1}, p_{j+1} \rangle$ is an inversion in Π .This shows that condition (2.2) is satisfied also by all elements of A which are not in the ℓ^{th} row or column. □

The proof of Theorem 2.1 suggests the following simple algorithm for detecting whether G(N) has a defining permutation, directly from its adjacency matrix A.

ALGORITHM 2.2

Initially we have an array $C = \langle c_1, c_2, .., c_n \rangle$ of n empty cells.

```
Step 1 : Set $i=1$.
Step 2 : Perform a logical 'NOT' operation on all entries
         $a_{ij}$ such that $j<i$.
Step 3 : Set $k = \sum_{j=1}^{n} a_{ij} + 1$ .
Step 4 : If $c_k$ is occupied stop (no defining permutation
         exists), else set $c_k = i$.
Step 5 : If $i=n$ stop, the array C contains the defining permutation
         of G(N), else set $i=i+1$ and go to Step 2. □
```

Example 2.2

Let $N=\{1,2,3,4,5\}$, we consider the graphs $G_1(N)$ and $G_2(N)$ o[illegible] 2.2 , and their respective adjacency matrices A_1 and A_2. The [illegible]ctor of A_1 is $S=<3,4,2,1,5>$, therefore Algorithm 2.2 will te[illegible]inate in Step 5 giving $C=<4,3,1,2,5>$ as the defining permutation of $G_1(N)$. The graph $G_2(N)$ has no defining permutation since in A_2 we have $S_2=S_3=2$.

$$A_1=\begin{array}{c|ccccc} & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & 0 & 0 & 1 & 1 & 0 \\ 2 & 0 & 0 & 1 & 1 & 0 \\ 3 & 1 & 1 & 0 & 1 & 0 \\ 4 & 1 & 1 & 1 & 0 & 0 \\ 5 & 0 & 0 & 0 & 0 & 0 \end{array} \qquad A_2=\begin{array}{c|ccccc} & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & 0 & 1 & 0 & 0 & 0 \\ 2 & 1 & 0 & 1 & 1 & 0 \\ 3 & 0 & 1 & 0 & 0 & 1 \\ 4 & 0 & 1 & 0 & 0 & 1 \\ 5 & 0 & 0 & 1 & 1 & 0 \end{array}$$

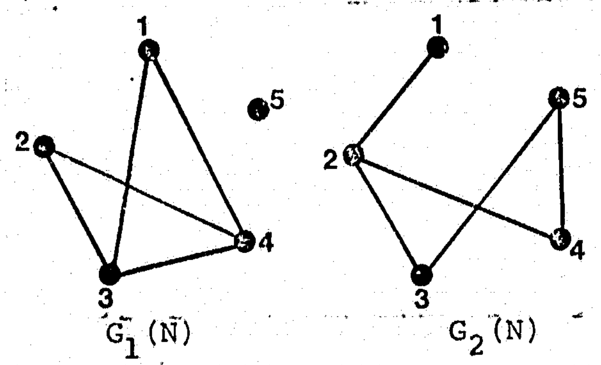


Fig 2.2

Running-time

In case of a positive answer , Algorithm 2.2 requires $\binom{n}{2}$ logical 'NOT' operations and n^2 additions.

Comparison of Algorithm 2.2 with Even's algorithm

In [1, pp 183] Even gives an algorithm for the same purpose as Algorithm 2.2, which has two stages. First, all edges of $G(N)$ and $G^C(N)$ are oriented from low to high and from high to low respectively. In the second stage a topological sort is performed on the vertices of $G(N)$. From the point of view of computation complexity both our algorithm and that of Even are of the same order , i.e. both require $O(n^2)$ steps. However Algorithm 2.2 is superior for the following reasons:

1. Algorithm 2.2 lends itself easier to computer implementation, only two types of operations are needed , logical 'NOT' and counting the number of '1' bits in a storage word ,both operations are very fast on most computers . On the other hand, the edge

orientations and the topological sort which are used in Even's algorithm, require some more involved operations and specia data structures as shown by Knuth in [4 ,pp 258].

2. In case of a negative answer,Algorithm 2.2 performs only a fraction of the total number of operations, depending on how early two S_i's are found to be identical.. In Even's algorithm we always have a fixed cost of $\binom{n}{2}$ orientations , a negative answer is detected only in the second stage when the topological sort procedure fails to find a sink among the vertices of G(N).

orientations and the topological sort which are used in Even's algorithm, require some more involved operations and special data structures as shown by Knuth in [4 ,pp 258].

2. In case of a negative answer,Algorithm 2.2 performs only a fraction of the total number of operations, depending on how early two S_i's are found to be identical. In Even's algorithm we always have a fixed cost of $\binom{n}{2}$ orientations , a negative answer is detected only in the second stage when the topological sort procedure fails to find a sink among the vertices of G(N).

2.4 A GENERALIZATION OF SCHENSTED'S ALGORITHM

Let G(N) be a permutation graph labelled with the set N and let Π be its defining permutation with respect to this labelling. Consider a clique C of G(N) where $|C| = k$. We can order the labels of the vertices of C in descending order and obtain $I_c = \langle i_1, i_2, \ldots, i_k \rangle$ .Since each pair of labels are an inversion in Π ,it follows that I_c is a maximal descending subsequence in Π . In this way a maximum clique of G(N) corresponds to a LDS in Π.

In [2] Schensted gives an algorithm for finding a LDS in a given permutation. In this section we describe a generalization of this algorithm which provides us with the option to generate all LDS's of a permutation. In view of the above connection between maximum cliques and LDS's, this generalized algorithm can be used to generate all maximum cliques of a permutation graph,once its defining permutation is given.

The number of LDS's of a permutation can grow exponentially with its order. For example consider the permutation Π of order n for $n \equiv 0 \pmod 3$,which consists of n/3 trios and has the general form:

$\Pi = \langle n-2, n-1, n, \ldots, (n-2)-3k, (n-1)-3k, n-3k, \ldots, 1, 2, 3 \rangle$.

Then Π has $3^{n/3}$ LDS's. Therefore it is important to avoid repetition of LDS's or generation of intermediate subsequences which eventually do not form a part of a LDS. In order to meet those requirements the algorithm to be described operates in two stages. The first stage reads the input permutation into a storage array and links those elements which are candidates for joining a LDS. In this way each LDS is stored in a form of a chain. In the second stage those chains are traversed and a LDS is generated each time we arrive at the end of a chain.

<u>Informal description — Stage 1</u>

Given a permutation $\Pi = \langle p_1, p_2, \ldots, p_n \rangle$ it is scanned from left to right and an ordered set of queues $Q_1, Q_2, \ldots, Q_n$ is formed from its elements. The construction is described recursively as follows:

1) p_1 is inserted into Q_1,

2) Assume $Q_1, Q_2, \ldots, Q_i$ were formed from $p_1, p_2, \ldots, p_{j-1}$, then p_j is attached to the first queue which has its last element smaller than p_j. If no such queue exists p_j starts a new queue Q_{i+1}.

With each element p_j which is attached to Q_i, two pointers are updated indicating the minimal and maximal elements in Q_{i-1} which are greater than p_j.

<u>Formal description</u>

The set of queues is stored in a 3-dimensional array Q of size n×n×3. If an element is inserted into the jth position of Q_i it is placed in Q[i,j,1] and the two fields Q[i,j,2] and Q[i,j,3] serve as pointer fields for the above mentioned purpose. The variable N[i] counts the number of elements in the ith queue and the variable q counts the number of non-empty queues in Q .

We first describe the procedure ATTACH which inserts an element at the next available place in a specified queue.

ATTACH (p_j, Q_i)

```
AT1 : Set N[i] = N[i] + 1 .
AT2 : Set Q[i,N[i],1] = $p_j$ .
Perform  AT3,AT4 and AT5 only if i>1,
AT3 : Search in $Q_{i-1}$ for the minimal $\ell$ such that Q[i-1,$\ell$,1]
      is greater than $p_j$.
AT4 : Set  Q[i,N[i],2] = $\ell$ .
AT5 : Set  Q[i,N[i],3] = N[i-1].
```

ALGORITHM 2.3. - STAGE 1

```
Step 1 : Set N[1] = 0, ATTACH ($p_1$,$Q_1$), set q = 1.
Repeat Step 2 for  $2 \le i \le n$ :
Step 2 :  If $p_i$ <Q[q,N[q],1] then set q=q+1,  N[q] = 0
          and ATTACH($p_i$,$Q_q$)  else  search the sequence
          Q[1,N[1],1],Q[2,N[2],1],...,Q[q,N[q],1] for the
          first element Q[f,N[f],1] such that $p_i$>Q[f,N[f],1]
          and ATTACH ($p_i$,$Q_f$).
```
□

Example 2.3

Let Π = <7,8,9,4,6,3,1,5,2> . Then after stage 1 there are 4 non-empty queues in Q with the following contents,

Q_1 = 7:-:-,8:-:-,9:-:

Q_2 = 4:1:3,6:1:3

Q_3 = 3:1:2,5:2:2

Q_4 = 1:1:2, 2:1:2

The values of the other variables are:

q=4 and N[1]=3 N[2]= N[3]=N[4]= 2. □

Running-time of Stage 1

The computation of the number of steps required to complete stage 1 is based on some properties of the elements of Q. Those properties are summarized in Lemma 2.2 which is a direct result of [2] and therefore stated without proof.

Lemma 2.2) If q queues are formed from Π in stage 1 then,

(a) The length of a LDS in Π is q.

(b) For $1 \le i \le q$, $Q[i,1,1] < Q[i,2,1] \ldots < Q[i,N[i],1]$.

(c) During the execution of the algorithm the last elements of the queues form a descending sequence, i.e. in Step 2 we have, $Q[1,N[1],1] > Q[2,N[2],1] > \ldots > Q[q,N[q],1]$. $\square$

The running-time is governed by the comparisons made in AT3 and Step 2 of the algorithm. From (b) and (c) of Lemma 2.2 it follows that binary-search can be used in both cases. We recall that the number of comparisons made by a binary-search on a sequence of length m is at most $\lfloor \lg_2 m \rfloor + 1$. Given an input permutation of order n with LDS of length q, the longest queue has no more than n+1-q elements. Therefore the total number of comparisons made in AT3 is bounded by $n(\lfloor \lg_2 (n+1-q) \rfloor + 1)$, we call this term A. Turning our attention to Step 2, we observe that a binary-search is performed only for those n-q elements which do not start a new queue. Therefore the binary-search in Step 2 is bounded by $(n-q)(\lfloor \lg_2 q \rfloor + 1)$ comparisons, we call this term B. In addition to that we have a fixed cost of n-1 comparisons in Step 2. We now calculate an upper bound on A+B as follows,

$$A+B \le n \lg_2 2(n+1-q) + (n-q) \lg_2 2q$$

and since

$$(n+1-q)q \le ((n+1)/2)^2$$

we get

$$A+B \le 2n\lg_2(n+1) - q\lg_2 2q \ .$$

We conclude that the total number of steps required by Stage 1 is proportional to $n\lg_2 n$.

<u>Informal Description of Stage 2</u>

By (a) and (b) of Lemma 2.2 ,it follows that a LDS in Π has exactly one element in each queue of Q, such that the ith element of a LDS is a member of Q_i.

<u>Lemma 2.3</u> Let $L = \langle Q[1,i_1,1], Q[2,i_2,1],\ldots,Q[q,i_q,1]\rangle$ be a sequence constructed from the array Q . Then L is a LDS of Π iff, for $1\le j<q$,

$Q[j+1,i_{j+1},2]\le i_j \le Q[j+1,i_{j+1},3]$.

<u>Proof:</u> By definition L is a LDS in Π iff for $1\le j<q$ the following two conditions are satisfied,

$$Q[j,i_j,1]>Q[j+1,i_{j+1},1] \qquad (2.7)$$

$$Q[j,i_j,1]\in L_\pi(Q[j+1,i_{j+1},1]) \ . \qquad (2.8)$$

(2.7) $\Leftrightarrow$ First element in Q_j which is greater than $Q[j+1,i_{j+1},1]$ is in position i_j or before $\Leftrightarrow$ $i_j \ge Q[j+1,i_{j+1},2]$.

(2.8) $\Leftrightarrow$ Q_j had at least i_j elements when $Q[j+1,i_{j+1},1]$ was attached to Q_{j+1} $\Leftrightarrow$ $i_j \le Q[j+1,i_{j+1},3]$. □

The second stage of Algorithm 2.3 is based directly on Lemma 2.3 The generation scheme is started from $Q[q,1,1]$. By the lemma the possible successors of this element are in Q_{q-1}, in positions which are within the limits specified by the two pointers $Q[q,1,2]$ and $Q[q,1,3]$. Therefore we select the next element in Q_{q-1} which is indicated by

Q[q,1,2] .In this way we move from one queue to the next, selecting one element in each queue, until an element in Q_1 is reached. At this stage the first LDS is completed. We then backtrack to the element in Q_2 which we came from and select its next possible successor. The general rule is as follows, when all successors of an element in Q_j are selected, we backtrack to the element in Q_{j+1} which we came from. Each time we arrive at Q_1 a new LDS is completed. Termination occurs when all the elements of Q_q have been processed.

This backtracking approach requires a nest of iterations of variable depth according to the number of queues generated in Stage 1. A precise description of the process is given by the following ALGOL 68 procedure. This notation provides a more elegant method of describing recursive processes than the semi-formal notation adopted elsewhere in this thesis.

Formal description

Let q be the number of queues formed in Stage 1 , and m the length of the last queue.

```
proc lister = (int m) :
   ( for i to m do access (q,i) od ) ;
proc access = (int j,ℓ) :
   ( buffer [j]= Q[j,ℓ,1] ;
     if j = 1 then print ((buffer,newline ))
              else for k from Q[j,ℓ,2] to Q[j,ℓ,3] do
                   access (j-1,k) od fi ) ;      □
```

Example 2.4

The LDS's of Π of Example 2.3 will be generated in the following order :

(1)	<7,4,3,1>	(6)	<9,6,3,1>	(11)	<8,6,3,2>
(2)	<8,4,3,1>	(7)	<7,4,3,2>	(12)	<9,6,3,2>
(3)	<9,4,3,1>	(8)	<8,4,3,2>	(13)	<7,6,5,2>
(4)	<7,6,3,1>	(9)	<9,4,3,2>	(14)	<8,6,5,2>
(5)	<8,6,3,1>	(10)	<7,6,3,2>	(15)	<9,6,5,2>

□

Remarks

1. Since a LDS in $\Pi = \langle p_1, p_2, .., p_n \rangle$ is a LAS of $\Pi_R = \langle p_n, p_{n-1}, .. p_1 \rangle$ it follows that the same algorithm can be used to generate all LAS's of a given permutation.
2. Given a sequence S which contains repeated elements we may want to generate (a) All longest strictly descending subsequences or (b) All longest non-ascending subsequences in S. It can be proved along the same lines as the proof of Lemma 2.2 in [2] that Stage 1 must be modified as follows:

   Case (a) : Replace '>' by '≥' in Step 2.

   Case (b) : Replace '<' by '≤' in Step 2 and '>' by '≥' in AT3.

2.5 A CONNECTION-BOARD PROBLEM

The problem which is discussed in this section is taken from [17,pp 245] . Consider a connection-board B which consists of two sets of fixed points called X_1 and X_2 and a set E of straight connecting lines such that each line in E connects a point in X_1 to a point in X_2 . Such a connection-board can be represented by a bipartite graph as shown in Fig 2.3 . It is required to decompose B into several parallel planes such that no two lines cross each other in a plane. This situation occurs in the design of printed circuits where the connecting lines are not insulated and therefore may meet each other only in vertices.

The problem is solved in[17] by constructing a new graph G_B in which each vertex corresponds to a line of B and two vertices are adjacent in G_B iff their corresponding lines cross each other in B. It is then observed that a minimal chromatic decomposition of G_B corresponds to a decomposition of B into the least number of planes. Under this correspondence each monochromatic set of G_B represents a set of lines which can be assigned to the same plane. In [1] it is shown that G_B is a permutation-graph. This enables us to employ an efficient algorithm which finds a minimal chromatic decomposition of G_B in $O(|E|^2)$ steps. We present here a method which finds the required decomposition in $O(|E|\lg_2 m)$ steps where m is equal to $\min(|X_1|,|X_2|)$.

Without loss of generality we can assume that $|X_1|= m$ and $|X_2|= n$. We label the points of X_1 by $x_1,x_2,..,x_m$ and those of X_2 by the numbers $1,2,..,n$ from left to right. A line is denoted by (i,x_j) where $i \in X_2$ and $x_j \in X_1$. The lines of B can be represented by m lists $L_1, L_2, .., L_m$ where L_i contains all points of X_2 which are connected to x_i in their order from left to right. We then construct the sequence $L_B = \langle L_1, L_2, .., L_m \rangle$ by catenating those lists.

Example 2.5

Consider the connection-board of Fig 2.3 . It is represented by $L_1 = \langle 1,2,4 \rangle$, $L_2 = \langle 1,3 \rangle$ and $L_3 = \langle 2,4 \rangle$. We then have $L_B = \langle 1,2,4,1,3,2,4 \rangle$.

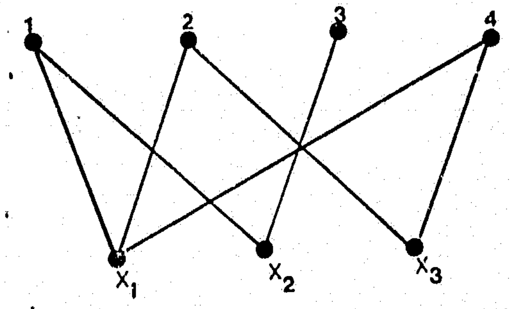


Figure 2.3

□

Lemma 2. 4 The chromatic number of G_B is equal to the length of the longest strictly descending subsequence of L_B.

Proof The vertices of a clique of G_B correspond to a set of lines C_B in B such that every two lines in C_B cross each other. On the other hand, two lines (i,x_k) and (j,x_ℓ) cross each other in B iff $(i-j)(k-\ell) < 0$. Therefore C_B is in 1-1 correspondence with a strictly descending subsequence in L_B. Let M_G be a maximal clique of G_B. Since G_B is γ-perfect [1,§9.1] its chromatic number is $|M_G|$ and by the above correspondence this is also the length of a longest strictly decreasing subsequence in L_B. □

By Lemma 2.4 it follows that we can find a minimal chromatic decomposition of G_B directly from L_B. We apply Algorithm 2.3 Stage 1 with the modification of Remark 2(a) to L_B. Suppose that q non-empty queues are formed in Ω by the algorithm. Then the chromatic number G_B is q. Furthermore, the elements in Q_i for $1 \le i \le q$ correspond to a set of lines in B in which no two cross one another and this set defines a monochromatic set of vertices in G_B.

Example 2.6

We decompose the connection-board of Example 2.5 into planes . The contents of the array Q (ignoring pointers) after Algorithm 2.3 are as follows;

Q_1 = 1,2,4,4

Q_2 = 1,3

Q_3 = 2

Therefore Plane 1 contains the lines: $(1,x_1),(2,x_1),(4,x_1)$ and $(4,x_3)$; Plane 2 : $(1,x_2)$ and $(3,x_2)$;Plane 3 : $(2,x_3)$. □

Running-time

The running-time here is dominated by the number of comparisons made by Algorithm 2.3,which we now compute. For $1 \le i \le m$ the elements in L_i appear in ascending order , this ensures that no more than i queues will be formed in Q from the elements of the lists $L_1, L_2, .., L_i$. Therefore when each element of L_i is attached to a queue, a binary-search on at most i elements is performed. The total number of comparisons is therefore bounded by

$$\sum_{i=1}^{m} |L_i| (\lfloor \lg_2 i \rfloor + 1) \le |E| \lg_2 2m \text{ .}$$

2.6 A DYNAMIC STORAGE ALLOCATION PROBLEM

Let $N=\{1,2,..,n\}$ be a set of programs which reside in memory in starting addresses $x_1,x_2,..,x_n$ where $x_i \le x_{i+1}$ for $1 \le i \le n-1$. During execution time, some programs may change their memory requirements. Let those new requirements be $\ell_1,\ell_2,..,\ell_n$ respectively. In order to meet these requirements, it may be necessary to shift some programs to new storage addresses. We assume that the order of the programs in memory must be preserved and that $\sum_{i=1}^{n}\ell_i$ does not exceed the available memory space. The problem is to find a reallocation scheme which has a minimal cost. If we assume that each program has the same cost of transplantation ,the above problem is equivalent to that of finding a maximal set of programs which can remain in place while the other programs are shifted..
In [12] a model graph $G(N)$ is constructed such that $i \overline{_{G(N)}} j$ iff $\sum_{k=i}^{j-1}\ell_k \le x_j - x_i$. In words, vertices i and j are adjacent in G(N) iff the programs i and j can remain in place under the new requirements. It is then proved that G(N) thus constructed is a permutation graph. The minimization problem is then solved using an algorithm which finds a maximum clique in G(N) in $O(n^2)$ steps.

However, we observe that by using the algorithms of the previous sections, we can find a maximum clique of G(N) in $O(n\lg_2 n)$ steps as follows. We define $z_1=0$ and for $2 \le i \le n$,$z_i = z_{i-1} + \ell_{i-1}$. We then find the defining permutation Π of G(N) with Algorithm 2.1, where Step 2 is replaced by

Step 2^* : If $x_j - x_i \ge z_j - z_i$ set $a_{ij}=1$,otherwise set $a_{ij}=-1$.

A maximum clique of G(N) can now be found from the LDS of Π .

In order to ensure that programs are not shifted out of memory , the programs 1 and n are dummy programs such that $x_1=\ell_1=\ell_2=0$ and x_n is set to the highest address available. We then restrict ourselves to those reallocation schemes where both program 1 and program n are kept in place. We now have to find the longest descending subsequence in Π which includes both 1 and n. We overcome this difficulty by applying some simple modifications to Π which we now proceed to show.

Since the maximal available memory is greater than the total requirements we have,

$x_n \geq z_n \Rightarrow x_n - x_1 \geq z_n - z_1 \Rightarrow a_{1n}=1 \Rightarrow n \in L_\pi(1)$. (2.9)

Let $\Pi=<p_1,p_2,\dots,p_n>$ such that $p_k=n$ and $p_\ell=1$; by (2.9), $k<\ell$. We construct a new sequence Π' which is obtained from Π by repeating p_k k times and p_ℓ $n-\ell+1$ times , leaving all other elements of Π unchanged.

<u>Lemma 2.5</u> Let $\Pi'=<p_1,\dots,\overbrace{p_k,\dots,p_k}^{k},p_{k+1},\dots,\overbrace{p_\ell,\dots,p_\ell}^{n-\ell+1},p_{\ell+1},\dots,p_n>$ be a sequence which is obtained from Π as described above. Then a longest non-ascending subsequence (LNAS) in Π' is equal (disregarding repetitions) to a longest descending subsequence in Π which includes 1 and n.

<u>Proof</u> We first prove that a LNAS in Π' contains both 1 and n. We apply Algorithm 2.3,modified as shown in Remark 2(b), to Π'. A LNAS of Π' is constructed by choosing one element from each non-empty queue in Q. It suffices to show that there

is at least one queue which has p_k as its only element and another which contains only p_ℓ. When p_k is attached to a queue by the algorithm, there can be no more than k-1 queues in Q. Therefore, at least one of the k occurrences of p_k must start a new queue, call this queue Q_j. No other element of Π' can join Q_j, so Q_j contains only p_k. Now , each occurrence of p_ℓ starts a new queue. We therefore have $n-\ell+1$ queues containing p_ℓ . Since there are only $n-\ell$ elements in $R_\pi(p_\ell)$, this ensures that the last queue contains only p_ℓ.

A LNAS in Π' has the form,

$$L_{\Pi'}=\langle p_k,\ldots,p_k,d_1,d_2,\ldots,d_r,p_\ell,\ldots,p_\ell\rangle .$$

Now $L_\Pi=\langle p_k,d_1,d_2,\ldots,d_r,p_\ell\rangle$ is descending in Π . Any descending subsequence in Π which is longer than L_Π and includes both p_k and p_ℓ ,gives rise to a subsequence in Π' which is longer than $L_{\Pi'}$, and is non-ascending, in contradiction to the definition of $L_{\Pi'}$. □

Example 2.7

Consider the reallocation problem which is represented by the following table.

i	x_i	ℓ_i	z_i
1	0	0	0
2	0	300	0
3	400	500	300
4	800	400	800
5	1100	300	1200
6	1600	200	1500
7	1700	0	1700

The defining permutation $\Pi = <6,3,7,4,2,1,5>$ is obtained directly from this table. We then have $\Pi' = <6,3,7,7,7,4,2,1,1,5>$. The LNAS in Π' is $<7,7,7,4,2,1,1>$,therefore the programs $\{7,4,2,1\}$ form a maximum set containing 7 and 1,which can be kept in place while the other programs are shifted.

Remarks

1. During the construction of queues by Algorithm 2.3,if the first p_k is attached to Q_i, then the following p_k's are automatically attached to $Q_{i+1},..,Q_{i+k-1}$. In the same way ,a binary-search is required only for the first occurrence of p_ℓ. We conclude that a binary-search is performed no more than n times in Step 2 of the algorithm. Since the number of queues formed from Π' does not exceed $k+(n-\ell+1)+(\ell-k-1)=n$,it follows that the total number of comparisons made by the algorithm is bounded by $n.(\lfloor \lg_2 n \rfloor +1)$.
2. The number of elements in Π' is $n+(k-1)+(n-\ell)=2n-1+(k-\ell)$. Since $k<\ell$,the maximal number of storage locations required for Π' is $2n-2$.
3. This method of repetition of elements,can be applied in cases where an integral cost of shift w_i is associated with the i^{th} program. A sequence $W(\Pi)$ is constructed from Π by repeating the integer i w_i times. It is easy to prove that the distinct elements in a LNAS of $W(\Pi)$, correspond to a maximum weight clique of $G(N)$.

2.7 AN ALGORITHM FOR FINDING A MAXIMUM CLIQUE IN A CIRCLE GRAPH

Given a set $S=\{c_1,c_2,..,c_n\}$ of chords in a circle, we can construct a graph G_S where each vertex in G_S corresponds to a chord in S , two vertices are adjacent if their corresponding chords intersect. A graph which represents a set of chords in this way is called a circle graph.

Example 2.8

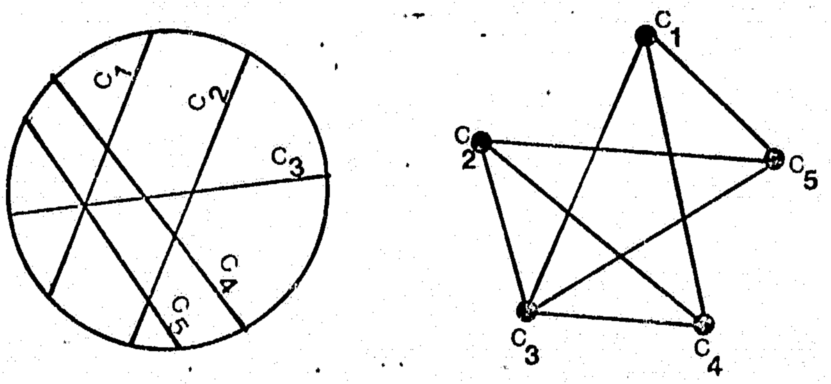


Figure 2.4

□

Circle graphs were used by Even and Itai[5] as a model for finding the least number of parallel stacks which realize a given permutation.

Consider a subset C of S with the property that every pair of chords in C intersect. If no subset of S with cardinality greater than C has this property , C is called a maximum clique of S. Our purpose here is to find a maximum clique of a given set of chords , clearly such a maximum clique corresponds to a maximum clique of G_S. This problem was previously solved by Gavril [14] , his algorithm finds a maximum clique of S from its representing graph G_S ,where as the approach here is to determine the maximum clique directly from S.

Without loss of generality, we can assume that no two chords in S have a common end point on the circle, if this occurs , we can move one chord slightly without changing the intersection structure. Given a circle and a set S such that $|S| = n$, we can label the end points of the chords in the following way. We choose an arbitrary end point on the circle and label it '1' , then we move clockwise along the circle and label each end point by the next integer until the starting point is reached. In this way the end points are numbered from 1 to 2n , each chord is represented by the two numbers on its end points. For each chord ,we order its representing pair such that the smaller number comes first. We then sort the n ordered pairs thus obtained on their first component. The chords are now represented by the ordered set of pairs $\bar{S}=\langle\langle x_1,y_1\rangle,\langle x_2,y_2\rangle,\ldots,\langle x_n,y_n\rangle\rangle$ where for $1\le i\le n-1$ $x_i<x_{i+1}$, and $x_i<y_i$ for $1\le i\le n$. We rename the chords so that c_i is the chord which is represented by $\langle x_i,y_i\rangle$ in $\bar{S}$.

For each chord c_i , a set $E(c_i)$ is defined as follows,

$$E(c_i)=\{c_j \mid x_j>x_i \text{ and } c_j \text{ intersects } c_i\} \quad .$$

From this definition we have,

$$c_j \in E(c_i) \quad \text{iff} \quad x_i<x_j<y_i<y_j. \tag{2.10}$$

Consider the set $E(c_i)\cup c_i$. Let X_i and Y_i be the sets of first components and second components respectively , of all chords in $E(c_i)\cup c_i$. The graph $B_i(X_i,Y_i,E(c_i)\cup c_i)$ is a bipartite graph where each vertex in X_i is adjacent to exactly one vertex in Y_i. We can find a maximum set of intersecting edges in B_i, by using a similar technique to the one used for the connection board problem. Clearly,two edges $\langle x_\ell,y_\ell\rangle$ and $\langle x_k,y_k\rangle$

intersect in B_i iff $(x_\ell - x_k)(y_\ell - y_k) > 0$. Assume that the chords in $E(c_i) \cup c_i$ are arranged in ascending order according to their first component. Let $\bar{Y}_i$ be the sequence of second components of these chords , then a maximum set of intersecting edges in B_i corresponds to a LAS in $\bar{Y}_i$.

Example 2.9

Consider the set of chords of Fig 2.5 . After numbering the end points and sorting we obtain the ordered set

$\bar{S}$= <1,14>,<2,11>,<3,8>,<4,10>,<5,13>,<6,12>,<7,16>,<9,15> .

The chord c_3=<3,8> defines the set (arranged in ascending order)

$E(c_3) \cup c_3$= <3,8>,<4,10>,<5,13>,<6,12>,<7,16> .

$\bar{Y}_3$=<8,10,13,12,16> , a LAS in $\bar{Y}_3$ is <8,10,13,16> .

Therefore a maximum set of intersecting edges in B_3 is

{<3,8>,<4,10>,<5,13>,<7,16>}

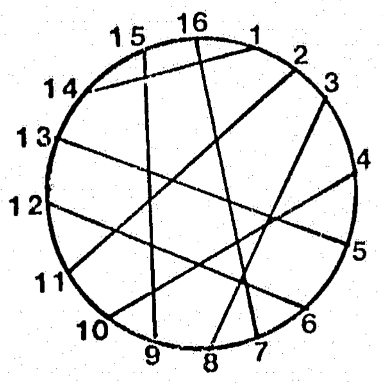


Figure 2.5

Consider a maximum clique C in S where |C|=k , and let $\bar{C}=<c_{i_1}, c_{i_2}, \ldots, c_{i_k}>$ be obtained from C by ordering its chords as previously shown. The set $E(c_{i_1}) \cup c_{i_1}$ contains all the chords of C , therefore C forms a maximum set of intersecting edges in the bipartite graph $B_{i_1}(X_{i_1}, Y_{i_1}, E(c_{i_1}) \cup c_{i_1})$. This implies that the sequence $<y_{i_1}, y_{i_2}, \ldots, y_{i_k}>$ is a LAS in $\bar{Y}_{i_1}$. We are now in a position to describe the following algorithm for finding a maximum clique in S.

ALGORITHM 2.4

We assume that the set S is given as n pairs of distinct integers between 1 and 2n.

Step 1 : Order each pair so that its smaller member comes first.

Step 2 : Arrange the pairs in ascending order according to their first component to obtain the ordered set $\bar{S}$.

Step 3 : Given the set $\bar{S}=\langle x_1,y_1\rangle,\langle x_2,y_2\rangle,\ldots,\langle x_n,y_n\rangle$, for $i\le i\le n-1$ form a list L_i as follows;

(a) the first element in L_i is $\langle x_i,y_i\rangle$,

(b) for $j=i+1,i+2,\ldots,n$. add the pair $\langle x_j,y_j\rangle$ to L_i iff it satisfies (2.10).

Step 4 : For $1\le i\le n-1$, find a LAS in $\bar{Y}_i$ of the list L_i, the longest LAS found defines a maximum clique of S. □

Example 2.10

Consider the set $\bar{S}$ of the previous example. The lists L_i which are formed in Step 3 and the LAS in $\bar{Y}_i$ are:

$L_1=\langle 1,14\rangle,\langle 7,16\rangle,\langle 9,15\rangle$;	LAS in $\bar{Y}_1=\langle 14,16\rangle$
$L_2=\langle 2,11\rangle,\langle 5,13\rangle,\langle 6,12\rangle,\langle 7,16\rangle,\langle 9,15\rangle$;	" " $\bar{Y}_2=\langle 11,13,16\rangle$
$L_3=\langle 3,8\rangle\ \langle 4,10\rangle,\langle 5,13\rangle,\langle 6,12\rangle,\langle 7,16\rangle$;	" " $\bar{Y}_3=\langle 8,10,13,16\rangle$
$L_4=\langle 4,10\rangle,\langle 5,13\rangle,\langle 6,12\rangle,\langle 7,16\rangle,\langle 9,15\rangle$;	" " $\bar{Y}_4=\langle 10,13,16\rangle$
$L_5=\langle 5,13\rangle,\langle 7,16\rangle,\langle 9,15\rangle$;	" " $\bar{Y}_5=\langle 13,16\rangle$
$L_6=\langle 6,12\rangle,\langle 7,16\rangle,\langle 9,15\rangle$;	" " $\bar{Y}_6=\langle 12,16\rangle$
$L_7=\langle 7,16\rangle$;	" " $\bar{Y}_7=\langle 16\rangle$

Since $\bar{Y}_3$ has the longest LAS, we conclude that the chords $\{\langle 3,8\rangle,\langle 4,10\rangle,\langle 5,13\rangle,\langle 7,16\rangle\}$ form a maximum clique in S. □

Running-time

We compute upper bounds on the number of elementary operations made in each step of Algorithm 2.4.

Step 1 : n comparisons and interchanges.

Step 2 : Since the first components are integers between 1 and 2n, we can use radix sort which requires O(n) elementary operations.

Step 3 : When the list L_i is formed , n-i chords must be checked to find out if they **satisfy** condition (2.10). Each check requires 3 comparisons , therefore a total of $3\binom{n}{2}$ comparisons are made in this step.

Step 4 : Finding a LAS in a list of length n , requires about $n\lg_2 n$ steps . Since $|L_i|=n-i$, the total number of steps required for finding the LAS's in all lists ,is bounde. by $\sum_{i=1}^{n} i\lg_2 i = S(n)$. By using Euler-Maclaurin's formula ,

$$S(n)= 0.72n^2\lg_2 n - 0.36n^2 + O(n\lg_2 n).$$

We conclude that the total number of steps required by Algorithm 2.4 grows as $n^2\lg_2 n$.

2.8 A GENERALIZATION OF A THEOREM BY ERDOS AND SZEKERES

The concept of a permutation graph can be generalized to that of a permutation hypergraph in the following way. Given a permutation Π on the set N , the hypergraph $H_k(\Pi)$ has the vertex set N and edge set E. An edge $E_i \in E$ iff (a) $|E_i|=k+1$, and ~~its~~-elements form a descending subsequence in Π , or (b) $|E_i|=1$ and its element is not in any descending subsequence of length k+1 in Π . From this definition it follows that $H_1(\Pi)$ is the permutation graph defined by Π . It may be interesting to investigate the possible applications of hypergraphs in the study of subsequences in permutations. One attempt in this direction is given in this section.

Following Greene [18], we denote by $d_k(\Pi)$ the length of the longest subsequence in Π which does not contain an ascending subsequence of length k+1 , similarly, $a_k(\Pi)$ denotes the length of the longest subsequence in Π which does not contain a descending subsequence of length k+1. Using this notation , $d_1(\Pi)$ and $a_1(\Pi)$ stand for the lengths of the LDS and LAS in Π respectively.

A famous theorem of Erdos and Szekeres [3] states that

$$d_1(\Pi)\cdot a_1(\Pi) \geq n \qquad (2.11)$$

where n is the order of Π . There are many known proofs of this theorem (see [6] ,[19] for example) , it can also be derived using the fact that $d_1(\Pi)$ and $a_1(\Pi)$ are the lengths of the first column and first row in the SYT of Π (as proved by Schensted [2]). Our purpose here is to prove the following generalization of (2.11).

<u>Theorem 2.2</u> Let Π be a permutation of order n , then

$$a_k(\Pi)\cdot\left\lceil \frac{d_1(\Pi)}{k} \right\rceil \geq n \ . \qquad (2.12)$$

We shall need the next lemma in the proof.

<u>Lemma 2.6</u> Let $X(H_k(\Pi))$ be the chromatic number of $H_k(\Pi)$, then

$$X(H_k(\Pi)) = \left\lceil \frac{d_1(\Pi)}{k} \right\rceil . \qquad (2.13)$$

<u>Proof</u> By definition, the chromatic number of a hypergraph is the smallest number of colours which are required to colour its vertices such that no edge E_i with $|E_i|>1$ has all its

vertices assigned the same colour. Since each set of k+1 vertices which appear in the LDS of Π form an edge of E,

$$X(H_k(\Pi)) \geq \left\lceil \frac{d_1(\Pi)}{k} \right\rceil . \qquad (2.14)$$

In order to prove the inequality in the other direction, we construct a chromatic decomposition of the vertices of $H_k(\Pi)$ by a generalization of Schensted's algorithm of Chapter 2.

Given $\Pi = \langle p_1, p_2, \ldots, p_n \rangle$, a set of r queues $Q_1, Q_2, \ldots, Q_r$ are formed from Π · 'ollows;

1) p_1 is inserted into Q_1,
2) Assume $Q_1, Q_2, \ldots, Q_i$ were formed from $p_1, p_2, \ldots, p_{j-1}$, then p_j is attached to the first queue such that p_j does not form a descending subsequence of length k+1 :ith its elements. If no such queue exists, p_j starts a new queue Q_{i+1}.

Clearly , this procedure provides an r colouration of $H_k(\Pi)$, since the elements of each queue do not form an edge $E_i \epsilon E$ of cardinality greater than 1. Assume that the last queue Q_r contains a descending subsequence of length m $(m \leq k)$. Then Π must contain a descending subsequence of length ℓ where $\ell = (r-1)k+m$. The following inequalities are then satisfied,

$$d_1(\Pi) \geq \ell$$

$$\left\lceil \frac{d_1(\Pi)}{k} \right\rceil \geq \left\lceil \frac{\ell}{k} \right\rceil \geq r \geq X(H_k(\Pi)). \qquad (2.15)$$

□

A set of vertices in a hypergraph H is stable if it does not contain an edge E_i with $\lceil E_i \rceil > 1$. The stability number of H $\beta(H)$ is the cardinality of the maximum stable set in H.

The following inequality is proved in [15, pp. 429].

Lemma 2.7 In a hypergraph H of order n

$$\beta(H)\chi(H)\geq n \ . \qquad \square \quad (2.16)$$

Proof of Theorem 2.2 By our definitions the maximum stable set in $H_k(\Pi)$ corresponds to the longest subsequence in Π which does not contain a descending subsequence of length k+1. Hence ,

$$\beta(H_k(\Pi))=a_k(\Pi) \ . \qquad (2.17)$$

By substituting (2.13) and (2.17) into (2.16) our result is proved. $\square$

CHAPTER THREE

PERMUTATIONS WHICH ARE SORTABLE WITH A STACK

3.1 INTRODUCTION

Given a permutation $\Pi=<p_1,p_2,\ldots,p_n>$ and an empty stack, the elements of Π can be passed through the stack using two elementary operations coded 'S' and 'X'. The operation 'S' denotes ' put the next element of Π on top of stack' and 'X' stands for ' transfer the element on top of stack to the output'. A sequence L of the above mentioned operations , is called a valid operation sequence (or simply an operation sequence) iff (1) all elements of Π are transferred to the output and (2) the operation 'X' is never specified when the stack is empty. Conditions (1) and (2) imply that an operation sequence must consist of 2n operations , n of each kind, the number of 'X' operations may never exceed the number of 'S' operations when L is scanned from left to right.

We denote by $L(\Pi)$ the output permutation which results from passing Π through a stack . For example if $\Pi=<1,3,2,4>$ and $L=<S,X,S,X,S,S,X,X>$ then $L(\Pi)=<1,3,4,2>$. A permutation Π is sortable with a stack iff there exists an operation sequence $\bar{L}$ such that $\bar{L}(\Pi)=<1,2,\ldots,n>$, it is realizable with a stack iff an operation sequence $\bar{R}$ exists such that $\bar{R}(<1,2,\ldots,n>)=\Pi$.

We denote by SS_n the class of permutations of order n which are sortable with a stack, and by SR_n the class of permutations of the same order which are realizable with a stack. Those two classes are related as follows,

$\Pi\epsilon SS_n$ iff $\Pi^{-1}\epsilon SR_n$. (3.1)

The class SR_n is characterized by Knuth [4, pp 239] by the following theorem,

<u>Theorem 3.1</u> The permutation $\Pi=<p_1,p_2,\ldots,p_n>$ is a member of SR_n iff does not contain a subsequence

$<p_i,p_j,p_k>$ such that $p_i>p_k>p_j$. (3.2)

From this theorem and the relation (3.1) we obtain a characterization of SS_n as follows,

<u>Theorem 3.1*</u> $\Pi\epsilon SS_n$ iff it does not contain a subsequence

$<p_i,p_j,p_k>$ such that $p_j>p_i>p_k$. (3.3)

There is a well-known one-to-one correspondence between the class SS_n (or SR_n) and the class of binary trees of n nodes [4, pp 329]. This correspondence serves as a basis to many of the results obtained here , it is therefore presented in Section 3.2 and proved in a different method to the one which is found in the literature. In section 3.3 , a new characterization of SS_n is given in terms of inversion tables , this is utilized in the next chapter for generating and indexing binary trees.

In the rest of the chapter,we study some more properties of stack sortable permutations such as; runs, readings , standard Young tableaux and associated permutation graphs. In particular, surprisingly simple expressions are obtained for the expected length of the LAS in a random permutation in SS_n , and for the number of involutions in this class.

3.2 A CORRESPONDENCE BETWEEN SS_n AND BINARY TREES

Two binary trees T and T' are similar ($T \simeq T'$) if they have the same 'shape', formally, they both have the same number of nodes, with the left subtree of T similar to the left subtree of T' and the same holds true for right subtrees. For a node j in T, we denote by $L_T(j)$ and $R_T(j)$ the left and right subtrees of j respectively.

A permutation $\Pi = \langle p_1, p_2, \ldots, p_n \rangle$ can be mapped into a labelled binary tree using the following well known construction.

Construction-T

We start initially with an empty tree T. For convenience, we shall call the nodes of T by the labels from Π which are assigned to them.

Given $\Pi = \langle p_1, p_2, \ldots, p_n \rangle$, assign p_1 to the root of tree T; for each p_k, $k=2,3,\ldots,n$, apply the rule -- if p_k is to be inserted into a non-empty subtree rooted by p_i, it must be on the left subtree of p_i iff $p_k < p_i$, otherwise p_k must appear to the right of p_i -- until en empty subtree is reached, and then create a root to that subtree and assign the label p_k to it.

We denote by T_π the tree into which Π is mapped by Construction-T. For a node p_k in T_π, Construction-T assigns labels smaller than p_k to nodes of $L_{T_\pi}(p_k)$ and bigger than p_k to nodes of $R_{T_\pi}(p_k)$. Such a labelling scheme of a binary tree is said to have the SLBR (Small to the Left Big to the Right) property.

Lemma 3.1 There is exactly one SLBR labelling scheme of a binary tree T of order n with the set N={1,2,..,n}.

Proof We can draw the tree T, using a line of length 2^{-i} units to connect a node in level i with a node in level i+1 (the root is considered in level 0). We then project the nodes of T on a horizontal line L as shown in Fig 3.1. Let i be a node of T and i_p its corresponding projection point on L. From obvious geometrical considerations, the members of $L_T(i)$ and $R_T(i)$ have their projection points on the left and on the right of i_p respectively (no two nodes are projected on the same point in L).

All the possible labellings of T can be obtained by assigning a permutation on N to the projection points, then copying the numbers from the points of L to the corresponding nodes in T. Assume that the permutation Π_L is assigned to the projection points and let $<j,i>$ be an inversion in Π_L. This will give rise to a labelling of T with one of the following situations;

(a) $j \in L_T(i)$,

(b) j and i belong to a subtree rooted by k where $j \in L_T(k)$ and $i \in R_T(k)$.

In both cases this labelling does not have the SLBR property. Hence, only the identity permutation <1,2,..,n> will produce a SLBR labelling of T. □

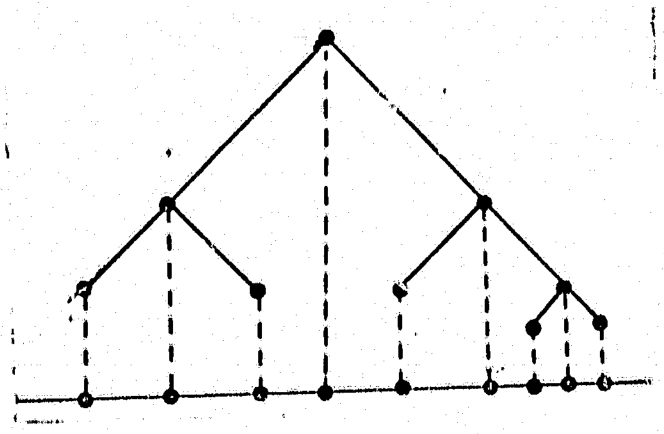


Figure 3.1

Given a binary tree T, we can construct Π such that $T=T_{\Pi}$ by the following recursive procedure:

1) Assign a SLBR labelling to T as shown in Lemma 3.1. Let the label at the root of T be p_1.

2) The first element of Π is p_1, the labels of a root p_k and its two subtrees $L_T(p_k)$ and $R_T(p_k)$ are written in Π in the order $<p_k, L_T(p_k), R_T(p_k)>$.

It is easy to prove by induction on the order of T that Π thus formed does not contain a subsequence (3.3) and therefore $\Pi \in SS_n$, furthermore for two non-similar trees T and T', their respective permutations Π and Π' satisfy $\Pi \neq \Pi'$.

The one-to-one correspondence between SS_n and the class of binary trees of order n will be shown by proving the converse of the above statement. We need the following lemma.

<u>Lemma 3.2</u> Let $<b_1, b_2, .., b_n>$ be the SR inversion-table of Π such that $\Pi \in SS_n$, then for a node labelled k in T_{Π}; $|L_{T_{\Pi}}(k)| = b_k$.

<u>Proof</u> We show that the elements which are counted by b_k are exactly the ones which are inserted into $L_{T_{\Pi}}(k)$ by Construction-T. Clearly, only an element j such that j<k and $j \in L_{\Pi}(k)$ can be inserted into $L_{T_{\Pi}}(k)$. If no such element exists in Π then $b_k=0$ and the subtree $L_{T_{\Pi}}(k)$ is empty. Assume $b_k>0$. Since $\Pi \in SS_n$, all elements in $L_{\Pi}(k)$ are either bigger or smaller than both k and j, any other possibility will create a subsequence (3.3) in Π. Therefore, application of Rule 1 will force j to be inserted into the same subtrees as k, finally j must be compared with k and since j<k it follows that $j \in L_{T_{\Pi}}(k)$. □

Theorem 3.2 Let $\Pi,\Pi' \in SS_n$, then if $\Pi \neq \Pi'$ it follows that $T_\pi \neq T_{\pi'}$.

Proof Given the tree T_π , the SR inversion-table of Π can be formed by writing next to each node the size of its left subtree , then copying those numbers to the corresponding projection points of T_π on L . Suppose $T_\pi = T_{\pi'}$, then this will imply that Π and Π' have identical SR inversion-tables thus contradicting $\Pi \neq \Pi'$.

3.3 THE INVERSION-TABLES

In this section we show a characte_ization of the members of SS_n in terms of their BR inversion-tables.

Theorem 3.3 The entries of the BR inversion-table $<b_1,b_2,\dots,b_n>$ of the permutation $\Pi=<p_1,p_2,\dots,p_n>$ satisfy:

$$b_i \geq b_{i+1} \quad \text{for } 1 \leq i \leq n-1 , \tag{3.4}$$

iff $\Pi \in SS_n$.

Proof Assume that $\Pi \notin SS_n$, by Theorem 3.1, Π contains a subsequence

$$p_j > p_i > p_k \quad \text{and} \quad i<j<k . \tag{3.5}$$

We choose p_k to be the rightmost element in Π which can be a member of such a subsequence; in other words there is no element in $R_\pi(p_k)$ which is smaller than p_k. Let there be m elements in $R_\pi(p_k)$ (m can be zero). All elements in $R_\pi(p_k)$ are greater than both p_i and p_k by the choice of p_k, therefore we have in the inversion-table of Π

$$b_{p_i} \geq b_{p_k} . \tag{3.6}$$

BuBut $p_j \in R_\pi(p_i) - R_\pi(p_k)$, and so p_j is counted only in b_{p_i} and the following inequality holds ;

$$b_{p_i} \geq b_{p_k}+1 \qquad (3.7)$$

Therefore (3.4) does not hold.

Conversely , let $<b_1,b_2,..,b_n>$ be an inversion-table of $\Pi=<p_1,p_2,..,p_n>$ for which (3.4) is not satisfied. We will show that Π has an occurrence of a subsequence (3.3).

Since condition (3.4) is not satisfied we must have two entries b_ℓ and $b_{\ell+1}$ such that $b_\ell<b_{\ell+1}$. By the definition of a BR inversion-table $\ell\in R_\pi(\ell+1)$ and there is at least one element between ℓ and $\ell+1$ in Π which is greater than $\ell+1$. Call this element g. Then ,the three elements $<\ell+1,g,\ell>$ form the desired subsequence (3.3) in Π. □

<u>**Remark**</u> : In [20] Knott proves that the BL inversion-table of Π satisfies $b_i-b_{i+1}<2$ iff $\Pi\in SS_n$. This is equivalent to Theorem 3.3 since the BL inversion-table of Π is equal to the vector subtraction of its BR inversion-table from the vector $<n-1,n-2,..,0>$. The proof in [20] is longer and employs induction, it was brought to my attention after this work has been completed independently.

Sequences $<b_1,b_2,..,b_n>$ which satisfy (a) $b_i\geq b_{i+1}$ and (b) $b_i\leq n-i$, may be called ballot-sequences since they are directly related to the classical ballot problem [4,pp531]. It follows from Theorem 3.3 that the BR inversion-table of members of SS_n is a ballot-sequence. Our next theorem shows a connection between the sequence of operations $\bar{\Sigma}$ which sorts a permutation Π of SS_n and its BR inversion-table. The proof is based on the natural correspondence between ballot-sequences and lattice-paths.

Consider the lattice-diagram in Fig 3.2 . We assign the weight i to the line that joins the points (i,j) and (i,j+1). The set of paths from (o,n) to (n,O) which pass through points (i,j) where $i+j \leq n$ and consist of unit steps from left to right or downwards , are in one-to-one correspondence with the set of ballot-sequences of length n , with respect to the sequence of weights in the path.

Example 3.1

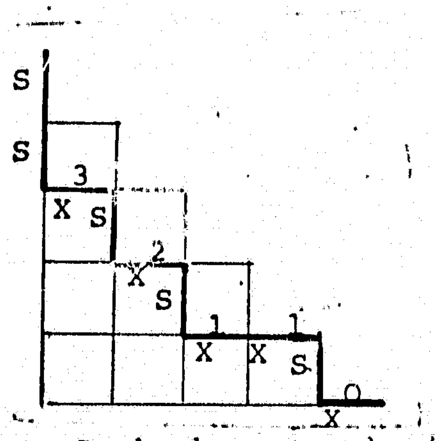


Figure 3.2

Theorem 3.4 Let $B=<b_1,b_2,..b_n>$ be the BR inversion-table of $\Pi \epsilon SS_n$ and let P be its corresponding lattice-path. The operation sequence $\bar{L}$ can be constructed by reading P from left to right denoting each vertical step by 'S' and horizontal step by 'X' .

Illustration In Example 3.1 , the lattice-path P is constructed from the inversion-table B=<3,2,1,1,0> of the permutation Π=<4,1,2,3,5> . Reading P as described in the theorem will produce L =<S,S,X,S,X,S,X,X,S,X> , since $L(\Pi)$=<1,2,..,n> it follows that $L=\bar{L}$.

Proof We construct the sequence of differences $D=<d_1,d_2,..,d_n>$ from B such that

$$d_1=n-b_1 \quad \text{and for } 2\leq i\leq n \quad d_i=b_{i-1}-b_i \ .$$

In P , d_1 counts the number of vertical steps preceding the first horizontal step ,for $2 \le i \le n$ d_i counts the number of vertical steps between the i-1st and ith horizontal steps.

We prove our theorem by showing:

(a) $\bar{L}$ begins with d_1 'S' operations,

(b) there are d_i 'S' operations between the i-1st and ith 'X' operations in $\bar{L}$. Here we have two cases,(1)$d_i=0$ and (2) $d_i>0$.

Proofs of (a) and (b) :

(a) By the definitions of D and B , d_1 gives the position of the integer 1 in Π , therefore sorting Π with a stack requires that d_1 elements are pushed into the stack before 1 can be removed from stack to the output.

(b) Case 1 ; $d_i=0 \Rightarrow b_{i-1}=b_i \Rightarrow i \in L_\pi(i-1)$, therefore no 'S' operations are required between the removal of i-1 and i from the stack during the sorting process.

Case 2 ; $d_i>0 \Rightarrow i \in R_\pi(i-1)$, in this case d_i counts the elements which are greater than i-1 in $R_\pi(i-1)-R_\pi(i)$. These elements are exactly the ones which are pushed into the stack between the removal of i-1 and i to the output. Hence , also in this case $\bar{L}$ has d_i 'S' operations between the i-1st and ith 'X' operations. □

3.4 MONOTONIC SUBSEQUENCES IN MEMBERS OF SS_n AND THEIR RELATION TO THE CORRESPONDING BINARY TREE

The first theorem in this section places $\Pi \in SS_n$ in a unique position,with respect to its LDS and LAS, among all other permutations which are mapped by Construction-T into a tree similar to T_π .

Theorem 3.5 Let T be a binary tree of n nodes and $G=\{\Pi_0,\Pi_1,\ldots,\Pi_\ell\}$ the set of all permutations such that $T_{\Pi_i}=T$ for $0\le i\le\ell$. Let d_i and a_i denote the lengths of the LDS and LAS in $\Pi_i\in G$. Without loss of generality we assume that $\Pi_0\in SS_n$. We then have

$$d_0\le d_i \quad\text{and}\quad a_0\ge a_i \text{ for } 1\le i\le\ell .$$

The proof is based on the following algorithm which gradually rearranges the elements of Π_i where $1\le i\le\ell$ until Π_0 is obtained. We first define a procedure called BT which is employed by this algorithm.

Procedure BT Given an ordered set of distinct elements $P=<p_1,p_2,\ldots p_k>$ where $k>2$;

BT1 : Create two ordered sets P_2 and P_3 where $P=P_1\cup P_2\cup P_3$, such that $P_1=\{p_1\}$, $P_2=\{p_i|p_i\in P$ and $p_i<p_1\}$ and $P_3=\{p_j|p_j\in P$ and $p_j>p_1\}$. The order of elements in P_2 and P_3 is consistent with their order in P.

BT2 : Place the sets created in Step 1 in the order $<P_1,(P_2),(P_3)>$ enclosing each set in brackets only if it contains more than two elements.

ALGORITHM 3.1 Initially a permutation Π enclosed in brackets is given.

Step 1 : Apply BT from left to right on every set enclosed in brackets, then delete those brackets.

Step 2 : If no set created in Step 1 has more than two elements stop, else go to step 1.

Example 3.2

Let $\Pi=<4,2,6,3,8,9,7,1,5>$, we give the sets which are created after each application of Step 1.

Initially : (4,2,6,3,8,9,7,1,5)

Pass 1 : 4,(2,3,1),(6,8,9,7,5)

Pass 2 : 4,2,1,3,6,5,(8,9,7)

Pass 3 : 4,2,1,3,6,5,8,7,9 □

Lemma 3.3 Let $\Pi \epsilon G-\{\Pi_0\}$ be the input to Algorithm 3.1. Then,

(1) Step 1 is performed no more than k times,where k is the height of T_π ,

(2) if $S=<s_1,s_2,..,s_n>$ is the permutation obtained after the algorithm terminates,then $S=\Pi_0$.

Proof Let $P=<p_1,p_2,..,p_n>$ be an input sequence to procedure BT and $\bar{P}$ its output (brackets are disregarded). Then , T_p and $T_{\bar{p}}$ have the same root p_1 , the elements of P_2 are inserted into both $L_{T_p}(p_1)$ and $L_{T_{\bar{p}}}(p_1)$ and those of P_3 into $R_{T_p}(p_1)$ and $R_{T_{\bar{p}}}(p_1)$. By definition we then have $T_p=T_{\bar{p}}$, furthermore when $T_{\bar{p}}$ is produced by Construction-T from $\bar{P}$, $L_{T_{\bar{p}}}(p_1)$ is completed before $R_{T_{\bar{p}}}(p_1)$ is started.

In the ith pass of the algorithm , procedure BT creates sets which form left and right subtrees of roots at level i-1 in T_π. Therefore the algorithm can have no more than k passes and (1) is proved.

Consider the permutation S, since it was obtained by applications of BT to sets that form subtrees in T_π, it follows that $T_s=T_\pi$. Suppose that $S\not\in SS_n$. Then S contains a subsequence

$$s_j>s_i>s_k \quad \text{and} \quad i<j<k \ .$$

This implies that when T_s is produced by Construction-T from S, there exists a root s_h (possibly $s_h=s_i$) such that s_j is inserted into $R_{T_s}(s_h)$ before s_k is inserted into $L_{T_s}(s_h)$. This is impossible by the definition of BT and Algorithm 3.1. We therefore conclude that $S \in SS_n$ and by the one-to-one correspondence of Section 3.2 $S=\Pi_0$. □

Proof of Theorem 3.5 Let $\Pi_i \in G-\{\Pi_0\}$ and let $<p_i, p_j>$ be a non-inversion pair in Π_i. We apply Algorithm 3.1 to Π_i . By the nature of this algorithm, p_j cannot be moved to the left of p_i, hence all the non-inversions in Π_i are also non-inversions in the final permutation $S=\Pi_0$.

Let $A=<p_{i_1}, p_{i_2}, .., p_{i_r}>$ be a LAS in Π_i $(r=a_i)$, then all the $\binom{r}{2}$ non-inversions defined by A are present in Π_0, therefore $a_0 \geq a_i$. On the other hand , let $D=<s_{i_1}, s_{i_2}, .., s_{i_q}>$ be a LDS in Π_0 $(q=d_0)$, then since Algorithm 3.1 does not form any new inversions it follows that D is also a descending subsequence in Π_i and $d_i \geq d_0$. □

We now demonstrate a connection between the LDS in $\Pi \in SS_n$ and the traversal of T_π in symmetric order. Traversal methods of binary trees can be classified according to the sequence in which a root and its two subtrees are visited, in the symmetric order the sequence is: (1) left subtree (2) root (3) right subtree . It follows from the construction of Lemma 3.1, that given a tree T with its SLBR labelling , the labels of T listed in their symmetric order will form an ascending sequence. In [4,pp 317] Knuth gives an algorithm for traversing a binary tree in symmetric order using a stack. For reference,we first present Algorithm 3.2 which is basically equivalent to Knuth's algorithm, then the following theorem is proved.

Theorem 3.6 Let T be a given binary tree and let $\Pi \in SS_n$ such that $T_\pi = T$, then the maximum depth of stack achieved during the traversal of T by Algorithm 3.2 is equal to the length of the LDS in Π .

ALGORITHM 3.2 Given a labelled binary tree T, T is traversed and the labels are listed in symmetric order. The following notations are used : ST- top of stack, RS(L) and LS(L)- right and left son respectively of the node in L (nodes are called by their labels). Let T be a left subtree of a dummy root called '∞'.

```
Step 1  (Initialize) : Set L='∞'.
Step 2  (Insert into stack): Set ST=L.
Step 3  (Check for left son): If LS(L) does not exist go to Step 5.
Step 4  : Set L=LS(L), go to Step 2.
Step 5  (Remove top of stack):  Set OUT=ST
Step 6  (Check for end ): If OUT='∞' stop, else write OUT on output.
Step 7  (Check for right son): If RS(OUT) exists set L=RS(OUT)
        and go to Step 2, else go to Step 5.  □
```

Proof of Theorem 3.6 We observe that the sequence of insertions (Step 2) and removals (Step 5) from stack,made by Algorithm 3.2 while traversing T , is equivalent to the sequence of operations needed to sort Π with a stack (we ignore the insertion of '∞' to the stack). Therefore it is sufficient to prove that the maximum depth of stack achieved while sorting Π is equal to the length of its LDS.

Let $D = \langle d_{i_1}, d_{i_2}, \ldots, d_{i_\ell} \rangle$ be a LDS in Π. Since no member of D can be removed from the stack before d_{i_ℓ},the stack must have at least ℓ entries.

Conversely, assume that in one stage of the sorting process m entries of the stack are occupied where $m>\ell$. Let $B=\langle b_{i_1}, b_{i_2}, \ldots, b_{i_m}\rangle$ be the elements present in the stack at that stage, then B is a descending subsequence in Π, a contradiction to the definition of D. □

Remarks

(1) Considering all binary trees of n nodes equally probable, empirical results for the expected maximal depth of stack achieved by Algorithm 3.2, show agreement with $\sqrt{\pi n}-1.6$ [4,Ex 11,pp 329]. By Theorem 3.6, this is also the expected length of the LDS in a random permutation in SS_n. It is interesting to compare this value with the corresponding result $(2\sqrt{n})$ observed in [7] for ordinary permutations.

(2) A sequence of numbers S can be sorted by a method called Tree Insertion Sort [16,pg 428] as follows;

Stage 1 : map S into a binary tree T_s using Construction-T,

Stage 2 : traverse T_s using Algorithm 3.2.

By theorems 3.5 and 3.6, it follows that the number of storage locations needed for the stack in Stage 2, is bounded by the length of the LDS in S.

3.5 THE SORTING OPERATION-SEQUENCE

Definitions

(1) Given a permutation $\Pi=\langle p_1,p_2,\ldots,p_n\rangle$, a descending run in Π is a sequence of successive elements $p_i,p_{i+1},\ldots,p_{i+k}$ such that $p_{i-1}<p_i>p_{i+1}>\ldots>p_{i+k}<p_{i+k+1}$.

(2) The permutation Π has k readings if k scans of Π from left to right are required to read off all its elements such that ;

(a) each scan reads off the maximum number of elements which appear in Π in their natural order,

(b) call the sequence of elements read off in the i^{th} scan R_i, then R_1 has the integer 1 as its first element , the first element in R_{i+1} is greater by 1 than the last element of R_i.

(3) A <u>composition</u> of a whole number n into m parts is a vector $C=<c_1,c_2,..,c_m>$ such that $c_i>0$ for $1\le i\le m$ and $\sum_{i=1}^{m} c_i = n$. Following[21] , a composition C of n can be represented as a zig-zag graph, this graph contains m rows with c_i dots in the i^{th} row, for i>1 the first dot in the i^{th} row is written under the last dot in row i-1. Given a composition C , we obtain its <u>conjugate</u> composition $\bar{C}=<\bar{c}_1,\bar{c}_2,..,\bar{c}_{n+1-m}>$ such that for $1\le i\le n+1-m$, $\bar{c}_i$ is equal to the number of dots in the i^{th} column (from left) of the zig-zag graph of C. For example let C=<3,2,4,1> be a composition of the integer 10. The zig-zag graph of C is

```
...
  ..
   ....
      .
```

therefore $\bar{C}$=<1,1,2,2,1,1,2>

(4) Given a permutation $\Pi\in SS_n$, let $\bar{L}$ be the sequence of operations which sorts Π with a stack. Scanning $\bar{L}$ from left to right , we call each sequence of consecutive 'S' operation an <u>S-group</u> and such a sequence of 'X' operations an <u>X-group.</u> Clearly, the number of X-groups is equal to the number of S-groups, two S-groups are seperated by an X-group and vice versa. The <u>S-specification</u> and the <u>X-specification</u> of $\bar{L}$ are vectors $<s_1,s_2,..,s_\ell>$ and $<x_1,x_2,..,x_\ell>$respectively, where for $1\le i\le \ell$ s_i denotes the size of the i^{th} S-group and x_i the size of the i^{th} X-group.

Theorem 3.7 Let $\Pi=<p_1,p_2,..,p_n>\epsilon SS_n$ have a sorting sequence of operations $\bar{L}$, then

(1) $\bar{L}$ has the S-specification $<s_1,s_2,..,s_\ell>$ iff Π has ℓ consecutive descending runs where the size of the ith run is s_i,

(2) $\bar{L}$ has the X-specification $<x_1,x_2,..,x_\ell>$ iff $\Pi^R=<p_n,p_{n-1},..,p_1>$ has ℓ readings where the size of R_i is x_i.

Illustration Let $\Pi=<9,3,2,1,8,5,4,7,6,10>$. The descending runs in Π from left to right are ;

<9,3,2,1>,<8,5,4> ,<7,6>,<10> .

The readings of Π^R are ;

$R_1=<1,2,3>$, $R_2=<4,5>$, $R_3=<6,7,8,9>$ $R_4=<10>$.

The sorting sequence of Π is ;

$\bar{L}$=<S,S,S,S,X,X,X,S,S,S,X,X,S,S,X,X,X,X,S,X>.

Its S-specification is <4,3,2,1> and the X-specification <3,2,4,1>.□

Proof of Theorem 3.7 (1) We put next to each 'S' and 'X' operation the element of Π which it puts or removes from stack.

Case (a) : p_i and p_{i+1} are in the same descending run <=> $p_i>p_{i+1}$ <=> $S(p_i)$ is followed immediately by $S(p_{i+1})$ <=> $S(p_i)$ and $S(p_{i+1})$ belong to the same S-group.

Case (b) : p_i and p_{i+1} are in different runs <=> $p_i<p_{i+1}$ <=> $X(p_i)$ appears in $\bar{L}$ between $S(p_i)$ and $S(p_{i+1})$.

We conclude that for each descending run in Π, we have a corresponding S-group which contains as many 'S' operations as elements in this run.

(2) Each X-group in $\bar{L}$ removes a maximum number of elements from stack which are stored in their reverse natural order.

Furthermore , if the last element removed by the i^{th} X-group is p_i , the $i+1^{st}$ X-group will start by removing p_j where $p_j=p_i+1$ and $p_j \epsilon R_{\pi}(p_i)$. Hence , by definition, the i^{th} X-group removes from stack exactly the elements of R_i in Π^R. □

<u>Theorem 3.8</u> Let $\Pi \epsilon SS_n$,then the number of descending runs in Π is equal to the length of its LAS.

<u>Proof</u> Let $A=<a_{i_1},a_{i_2},..,a_{i_\ell}>$ be a LAS in Π and let $D_1,D_2,..,D_k$ be its descending runs. Since no two members of A can belong to the same descending run,we have $k \geq \ell$. Let $D=<d_1,d_2,..,d_k>$ be a subsequence of Π such that for $1 \leq i \leq k$ d_i is the last element in D_i. We now show that D is an ascending subsequence. Suppose that for two successive elements in D we have $d_i > d_{i+1}$, since d_i and d_{i+1} are in different runs we must have an element $d \epsilon D_{i+1}$ and $d \epsilon L_{\pi}(d_{i+1})$. The three elements d ,d_i and d_{i+1} appear in Π in the order $<d_i,d,d_{i+1}>$ and they satisfy

$$d > d_i > d_{i+1} \text{ .}$$

This contradicts $\Pi \epsilon SS_n$. Therefore D is an ascending subsequence in Π and $k \leq \ell$. We conclude that $k=\ell$ and D is a LAS in Π . □

Let Π and Π_{RF} be two members of SS_n (not necessarily distinct) such that their corresponding trees T_{π} and $T_{\pi_{RF}}$ are reflections of each other about the vertical axis. In Lemma 3.4 we show an interesting relation between the operation-sequences $\bar{L}$ and $\bar{L}_{RF}$ which sort Π and Π_{RF} with a stack. This relation is then utilized in deriving a simple expression for the expected length of the LAS in members of SS_n.

<u>Lemma 3.4</u> Let $X=\langle x_1,x_2,\ldots,x_k\rangle$ and $X_{RF}=\langle x_1',x_2',\ldots,x_m'\rangle$ be the X-specifications of $\bar{L}$ and $\bar{L}_{RF}$ respectively. Then the vectors $X^R=\langle x_k,x_{k-1},\ldots,x_1\rangle$ (the reverse of X) and X_{RF} are conjugate compositions of n.

<u>Illustration</u> Consider the permutations $\Pi=\langle 6,3,2,1,4,5,8,7\rangle$ and $\Pi_{RF}=\langle 3,1,2,6,5,4,7,8\rangle$. The corresponding binary trees are shown in Fig 3.3 (a) and (b) respectively.

The X-specification of $\bar{L}$ is $X=\langle 3,1,2,2\rangle$ and $X^R=\langle 2,2,1,3\rangle$

The zig-zag graph of X^R is

```
..
..
 .
 ...
```

and therefore its conjugate is $\bar{X}^R=\langle 1,2,3,1,1\rangle$, we then have $\bar{X}^R=X_{RF}$.

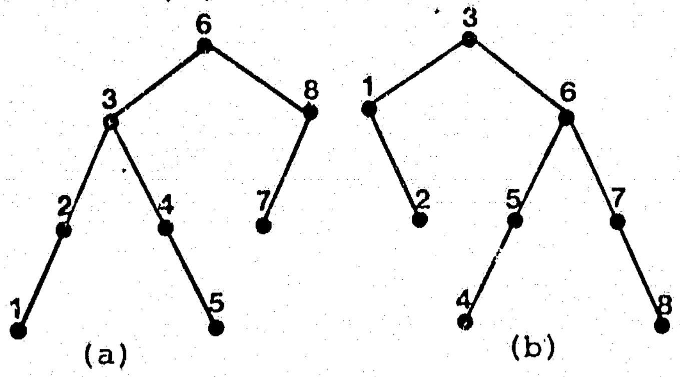


Figure 3.3

<u>Proof</u> A binary tree is traversed in reverse symmetric order if a root and its two subtrees are visited in the order (1) right subtree (2) root (3) left subtree . We observe that the operations which are required in order to traverse T_π in reverse symmetric order are equivalent to those necessary for traversing $T_{\pi_{RF}}$ in symmetric order. Therefore $\bar{L}$ and $\bar{L}_{RF}$ specify the stack operations for traversing T_π in symmetric and reverse symmetric order respectively. For two consecutive labels i and i-1 we can have (a) $i\in R_{T_\pi}(i-1)$ or (b) $i-1\in L_{T_\pi}(i)$. While traversing T_π in symmetric order , (a) implies that i must be stacked after i-1 is written on output and therefore X(i-1) and X(i) are in different X-groups ,(b) implies that i is present in stack

when i-1 is written ,hence X(i) and X(i-1) are in the same X-group. It is easy to see that in the reverse symmetric order traversal of T_π we have exactly the converse, i.e. the labels i and i-1 are written on output by the same X-group in $\bar{L}_{RF}$ in case (a) and by different X-groups in case (b).

We can represent X as a zig-zag graph in which the i^{th} row contains the elements written by the i^{th} X-group in $\bar{L}$. By the above argument , it follows that the i^{th} X-group in $\bar{L}_{RF}$ will write out elements of the i^{th} column in this graph,where counting starts from the rightmost column. For example in the above illustration the graph is

```
123
  4
  56
   78
```

and the X-groups of $\bar{L}_{RF}$ write out <8>,<7,6>,<5,4,3>,<2>,<1> , where brackets enclose elements of the same X-group ∴ Therefore X^R and X_{RF} are conjugate compositions and k=n+1-m . □

<u>Theorem 3.9</u> The expected length of the LAS in a random permutation in SS_n is $\frac{n+1}{2}$.

<u>Proof</u> We define a mapping $RF:SS_n \rightarrow SS_n$ such that $\Pi \in SS_n$ is mapped into Π_{RF} by RF. Suppose that the length of the LAS in Π is equal to k. By Theorems 3.7 and 3.8 this is also the number of components in the S-specification and X-specification of the sorting sequence $\bar{L}$. From Lemma 3.4 ,the length of the LAS in Π_{RF} is n+1-k. Since RF is a one-to-one correspondence our result follows. □

3.6 PROPERTIES OF THE PERMUTATION GRAPHS ASSOCIATED WITH SS_n

Definition

Let $P=\langle v_1,v_2,\ldots,v_k\rangle$ be a path in the graph G, P has a triangular chord if for some index i ($1\le i\le k-2$) $v_i \underset{G}{—} v_{i+2}$.

Theorem 3.10 Let G be a permutation graph with $|V|=n$. Then the following conditions are equivalent;

(1) at least one of the labellings of G with N, gives rise to a defining permutation $\Pi\in SS_n$,

(2) G does not contain a path of length 3 without a triangular chord.

Proof (1)=>(2) ; Let G(N) be the graph which is obtained by labelling G with N, such that $\Pi\in SS_n$ is its defining permutation. Suppose that G(N) contains a path $P=\langle i,j,k,\ell\rangle$ without a triangular chord. There are two possible transitive orientations of P, (a) $i\leftarrow j, j\rightarrow k, k\leftarrow \ell$ and (b) $i\rightarrow j, j\leftarrow k$ and $k\rightarrow \ell$. This implies that either

$\langle i,k,j\rangle$ with $k>i>j$ or $\langle j,\ell,k\rangle$ with $\ell>j>k$ (3.8)

are subsequences in Π , thus contradicting $\Pi\in SS_n$.

(2)=> (1) ; We show that G which satisfies condition (2) can be transitively oriented without using Rule 1 of Algorithm 1. . This implies that for every three vertices x,y and z such that $x \underset{G}{—} y$, $y \underset{G}{—} z$ and $x \underset{G}{\not—} z$ we assign directions $x\leftarrow y$ and $y\rightarrow z$. (3.9)

Clearly , the defining permutation which results from such an orientation is a member of SS_n irrespective of the orientation assigned to the edges of G^C.

Let $G_1,G_2,\ldots,G_k$ be subgraphs of G such that $G_1=G$ and for $1 \le i \le k-1$ G_{i+1} is obtained from G_i by removing the edges which were oriented by the i^{th} pass of Algorithm 1.1. (G_k is empty).

We prove our theorem by showing;

(a) if Algorithm 1.1 is forced to use Rule 1 in the i^{th} pass then G_i violates condition (2),

(b) if G_i violates condition (2) so does G_{i-1}.

Proofs of (a) and (b).

(a) If G_i is a union of disjoint complete graphs then neither rules are used, otherwise we can find a vertex b such that $a \underset{G_i}{—} b, c \underset{G_i}{—} b$ and $a \underset{G_i}{\not—} c$ we then start the pass by orienting $a \leftarrow b$ and $c \leftarrow b$. Assume that we have to switch from Rule 2 to Rule 1 during this pass. This implies that we have three vertices e, f and g in G_i such that

$$e \rightarrow f \;,\; f \underset{G_i}{—} g \text{ and } e \underset{G_i}{\not—} g \;. \tag{3.10}$$

Since this is the first switch, $e \rightarrow f$ was oriented by Rule 2. Therefore there is a vertex d in G_i where

$$e \rightarrow d \text{ and } d \underset{G_i}{\not—} f \;. \tag{3.11}$$

By (3.10) and (3.11) it follows that the path $<d,e,f,g>$ in G_i does not contain a triangular chord.

(b) Let $P=<q,r,s,t>$ be a path without a triangular chord in G_i. Then P had at most one triangular chord in G_{i-1}, since the i-1st pass can orient only edges with a common end point. Without loss of generality, let this chord be $q \underset{G_{i-1}}{—} s$ and assume it was directed $q \leftarrow s$. Then a vertex u must exist in G_{i-1} where

$$s \rightarrow u \text{ and } q \underset{G_{i-1}}{\not—} u \quad \text{(as shown in Fig 3.4)} \tag{3.12}$$

From the fact that $s \underset{G_i}{—} t$ (and was not removed after the i-1st pass) we have

$$t \underset{G_{i-1}}{—} u \text{ and } t \underset{G_{i-1}}{—} q \;. \tag{3.13}$$

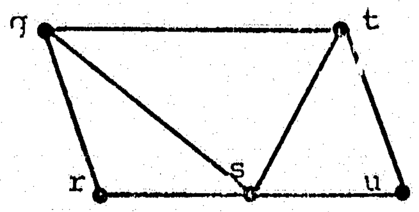


Figure 3.4

By (3.12) and (3.13) it follows that $P'=<u,t,q,r>$ is a path without a triangular chord in G_{i-1}. In a similar way, the assumption that the chord $q \overline{}_{G_{i-1}} s$ was directed $s \to q$ will also lead to a violation of condition (2) in G_{i-1}. □

<u>Theorem 3.11</u> Let $G(N)$ be a permutation graph which has Π as its defining permutation. Let $C_1, C_2, \ldots, C_k$ be the cliques of G where s_i is the smallest label in C_i. Then $\Pi \in SS_n$ iff $i \neq j \Rightarrow s_i \neq s_j$.

<u>Proof</u> Let C_i and C_j be two distinct cliques in $G(N)$, suppose $s_i = s_j = \ell$. There must be two vertices $c_i \in C_i$ and $c_j \in C_j$ such that $c_i \overline{\not}_{G(N)} c_j$. Let $n = \max(c_i, c_j)$ and $m = \min(c_i, c_j)$. We then have $\langle m, n, \ell \rangle$ as a subsequence in Π, and since $n > m > \ell \Rightarrow \Pi \notin SS_n$.

Conversely, assume that $\Pi \notin SS_n$, let $\langle c_1, c_2, c_3 \rangle$ be a forbidden subsequence (3.3) in Π. We choose c_3 to be the smallest element in Π which can join such a subsequence with c_1 and c_2. Since $c_1 \overline{\not}_{G(N)} c_2$, c_1 and c_2 belong to two different cliques C_i and C_j respectively, c_3 is adjacent to both c_1 and c_2 and therefore $c_3 \in C_i \cap C_j$.

Assume that $c_3 > s_i$, then a transitive orientation in $G(N)$ from high to low gives,

$c_1 \to s_i$ (both in the same clique), $c_2 \to c_3, c_3 \to s_i \Rightarrow c_2 \to s_i$.

Therefore $\langle c_1, c_2, s_i \rangle$ is also a forbidden subsequence in Π, in contradiction to the minimality of c_3. Similarly $c_3 > s_j$ gives rise to the forbidden subsequence $\langle c_1, c_2, s_j \rangle$. Hence $c_3 = s_i = s_j$. □

<u>Remarks</u>

(1) By Theorem 3.11, it follows that if $\Pi \in SS_n$ is a defining permutation of a graph $G(N)$, then $G(N)$ can have no more than n cliques. This could also be proved from the first part of Theorem 3.10 since condition (2) of the theorem implies that G is

chordal[22], it is well known that a chordal graph can have no more than n cliques .

(2) Since each clique in $G(N)$ corresponds to a maximal descending subsequence in its defining permutation Π, Theorem 3.11 can also be stated in terms of descending subsequences.
We call the last element in a maximal descending subsequence its end point. The following theorem is equivalent to Theorem 3.11.

Theorem 3.11* Let Π be a permutation on the set N. Then $\Pi\in SS_n$ iff no element in Π is an end point of more than one maximal descending subsequence in Π.

3.7 STANDARD YOUNG TABLEAUX

Given a sequence of distinct elements Π, let $D(\Pi)$ be a subsequence of Π which is constructed from the last elements of the descending runs in Π (as shown in the proof of Theorem 3.8). We can construct a progression of sequences $\Pi_1,\Pi_2,..,\Pi_k$ in the following way. Let $\Pi_1=\Pi$, for $2\le i\le k$ $\Pi_i= \Pi_{i-1}-D(\Pi_{i-1})$ and $\Pi_k=D(\Pi_k)$. In words ,the sequence Π_i is a subsequence of Π_{i-1} which is obtained by eliminating the elements of $D(\Pi_{i-1})$ from Π_{i-1}. The next theorem provides a simple method for constructing the SYT of $\Pi\in SS_n$.

Theorem 3.12 Let Π be a stack sortable sequence , then the jth row in the SYT of Π is equal to $D(\Pi_j)$.

Illustration

Let $\Pi=<2,1,6,3,5,4>$. Then $D(\Pi)=<1,3,4>$, $\Pi_2=<2,6,5>$,

$D(\Pi_2)=\langle 2,5\rangle$, $\Pi_3=\langle 6\rangle$ and $D(\Pi_3)=\langle 6\rangle$. Let $S(\Pi)$ be the SYT of Π.

Then $S(\Pi)=\begin{matrix}1 & 3 & 4\\ 2 & 5 & \\ 6 & & \end{matrix}$.

<u>Proof</u> We use induction on the length of Π. It is easy to verify the correctness of the theorem for sequences of length 2 , assume that it holds for sequences of length $k<n$. Let Π be a stack-sortable sequence of length n. Let Π' be the sequence of elements which are bumped from the first row of $S(\Pi)$ during its construction, such that elements appear in Π' in the same order in which they were bumped.It is well known that rows 2,3,.. of $S(\Pi)$ form a SYT which is equal to $S(\Pi')$ [16,pp 54].

We now show that the elements of $D_i(\Pi)$ (i^{th} run in Π) go into the i^{th} column in the first row of $S(\Pi)$. This is clearly true for $D_1(\Pi)$, let ℓ be the first index such that $y\epsilon D_\ell(\Pi)$ goes into the j^{th} column,where $j<\ell$ and $s_{1j}=x$. Then $x>y$ and $x\epsilon L_\pi(y)$. Since x and y belong to two different runs,there is an element z between x and y in Π such that $z>x$. Then Π contains a subsequence $\langle x,z,y\rangle$ with $z>x>y$, a contradiction. Therefore, the first row of $S(\Pi)$ consists of the elements of $D(\Pi)$. Furthermore the sequence of bumped elements Π' is equal to $\Pi-D(\Pi)=\Pi_2$. Since Π_2 is a subsequence of Π it is sortable with a stack , our result follows by applying the induction hypothesis to $S(\Pi_2)$. □

Given a valid sequence of operations $\bar{L}$, another valid sequence can be obtained by reading $\bar{L}$ from right to left, replacing 'X' by 'S' and vice versa. We denote the sequence thus constructed by $\bar{L}_C$, and call the sequences $\bar{L}$ and $\bar{L}_C$ conjugate operation sequences. Two conjugate operation sequences correspond to the two modes in which a lattice-path (as in Fig 3.2) can be read.

Theorem 3.13 Let $\bar{L}$ and $\bar{L}_C$ be two conjugate operation sequences of length 2n, and let Π and Π_C be the members of SS_n which are sorted by those sequences respectively. Then $S(\Pi)$ and $S(\Pi_C)$ have the same shape.

Illustration Consider the permutation Π =<2,1,6,3,5,4> of the previous illustration . Its sorting sequence is;

$\bar{L}$=<S,S,X,X,S,S,X,S,S,X,X,X> , $\bar{L}_C$=<S,S,S,X,X,S,X,X,S,S,X,X>

Π_C=<4,2,1,3,6,5> and $S(\Pi_C)= \begin{matrix} 1 & 3 & 5 \\ 2 & 6 & \\ 4 & & \end{matrix}$ which has the same shape as $S(\Pi)$ in that illustration.

Proof The proof is by induction on the length of $\bar{L}$. The theorem holds trivially for sequences of length 4. Assume correctness for k<2n. Since $\bar{L}$ and $\bar{L}_C$ have the same number of X-groups (or S-groups) it follows by theorems 3.7 and 3.12 that the first rows in $S(\Pi)$ and $S(\Pi_C)$ have an equal length.

Let Π' be the sequence of bumped elements in $S(\Pi)$ (as in Theorem 3.12) and let $\bar{L}'$ be the sequence which sorts Π' with a stack. Then $\bar{L}'$ can be obtained from $\bar{L}$ by eliminating the last 'S' in each S-group and the first 'X' in each X-group, since those are the operations which put and remove from stack the last elements of runs in Π. Let Π'_C be the sequence of bumped elements in $S(\Pi_C)$, then its sorting sequence $\bar{L}'_C$ can be obtained from $\bar{L}_C$ in the same way as $\bar{L}'$ was obtained from $\bar{L}$. Now , each last 'S' in an S-group and first 'X' in an X-group of $\bar{L}$, correspond to a first 'X' and last 'S' in X-groups and S-groups of $\bar{L}_C$ respectively. Hence , $\bar{L}'$ and $\bar{L}'_C$ are conjugate operation sequences, by the induction hypothesis $S(\Pi')$ and $S(\Pi'_C)$ have the same shape. □

3.8 THE NUMBER OF INVOLUTIONS IN SS_n

It is well known[16] that a permutation is an involution iff it does not contain a cycle with more than two elements. Using this fact , we prove in Lemma 3.5 that the set of involutions in SS_n is equal to $SS_n \cap SR_n$. A simple expression for the cardinality of this set is then calculated in Theorem 3.14.

Lemma 3.5 Let $\Pi \in SS_n$, then Π is an involution iff $\Pi \in SS_n \cap SR_n$

Proof The 'only if' part follows directly from the definitions. We prove the 'if' part by showing that a permutation which is a non-involution must contain at least one of the subsequences (3.2) or (3.3) , therefore it is not a member of $SS_n \cap SR_n$.

Let Π be a non-involution , then Π contains a cycle of length $k \geq 3$. Let this cycle be $[a_1, a_2, .., a_k]$ where a_1 is the smallest element in this cycle. We can arrange the elements of the cycle according to their original order in Π in the following way. First we sort the cycle into ascending order, then write under each element its right successor in the cycle, the second line thus obtained forms a subsequence of Π . For example if Π contains the cycle $[1,4,3,6,5]$ then the above operations will give $\begin{bmatrix} 1 & 3 & 4 & 5 & 6 \\ 4 & 6 & 3 & 1 & 5 \end{bmatrix}$ and $\langle 4,6,3,1,5 \rangle$ is a subsequence of Π (a_1 is considered to be the right successor of a_k).

We distinguish between two cases:

Case 1 ; $a_2 < a_3$. Let $k=3$, then after sorting the cycle we get $\begin{bmatrix} a_1 & a_2 & a_3 \\ a_2 & a_3 & a_2 \end{bmatrix}$, and $\langle a_2, a_3, a_1 \rangle$ forms a subsequence (3.3) in Π.

Assume that $k>3$. We sort the cycle by placing a_2 on the right of a_1, then inserting the elements $a_k, a_{k-1}, \ldots, a_3$ one by one into their correct positions. We write under each element its right successor when it is inserted. If $a_k > a_2$ then a_k is inserted on the right of a_2 and we get the same result as in the case k=3, a_k playing the role of a_3. Assume that $a_k < a_2$. We insert $a_{k-1}, a_{k-2}, \ldots$ into their positions until an element a_{k-i} is found such that $a_{k-i} > a_2$, the existence of such an element is guaranteed since $a_3 > a_2$. The element a_{k-i+1} is smaller than a_2 , hence after inserting a_{k-i} we have the following configuration

$$\begin{bmatrix} a_1 \cdots\cdots a_{k-i+1} \cdots\cdots a_2 & a_{k-i} \\ a_2 \cdots\cdots a_{k-i+2} \cdots\cdots a_3 & a_{k-i+1} \end{bmatrix} \qquad (3.14)$$

and $\langle a_2, a_3, a_{k-i+1} \rangle$ forms a subsequence (3.3) in Π .

Case 2 ; $a_2 > a_3$. If k=3 we have the configuration $\begin{bmatrix} a_1 & a_3 & a_2 \\ a_2 & a_1 & a_3 \end{bmatrix}$ after sorting the cycle, and $\langle a_2, a_1, a_3 \rangle$ forms a subsequence (3.2) in Π. Assume k>3. If $a_k < a_2$ we obtain the same subsequence, we therefore consider the case $a_k > a_2$. We use the same procedure as in Case 1, this time we search for the first element a_{k-i} such that $a_{k-i+1} > a_2$ and $a_{k-i} < a_2$. We then have the configuration

$$\begin{bmatrix} a_1 & a_{k-i} & a_2 \cdots\cdots a_{k-i+1} \cdots \\ a_2 & a_{k-i+1} & a_3 \cdots\cdots a_{k-i+2} \cdots \end{bmatrix} . \qquad (3.15)$$

and $\langle a_2, a_{k-i+1}, a_3 \rangle$ is a subsequence (3.3) in Π. $\square$

<u>Theorem 3.14</u> The number of involutions in SS_n is equal to 2^{n-1}.

<u>Proof</u> By Lemma 3.5 ,we have to show that there are 2^{n-1} permutations of length n which do not contain subsequences (3.2) or (3.3). A permutation $\Pi \in SS_n \cap SR_n$ can be characterized by the following property of its maximal descending subsequences.

Let $D=<d_{i_1},d_{i_2},\dots,d_{i_k}>$ be a maximal descending subsequence in a permutation Π of order n, then $\Pi\epsilon SS_n\cap SR_n$ iff for $i\le j\le k-1$,

$d_{i_j}=d_{i_{j+1}}+1$ (elements of D appear in reverse natural order). (3.16)

Proof ;

Clearly every permutation which satisfies condition (3.16) is a member of $SS_n\cap SR_n$, since each of the forbidden subsequences (3.2) and (3.3) have at least one pair of elements which belong to a descending subsequence and are not in reverse natural order. We now show that if any violations of condition (3.16) occur in Π then $\Pi\not\epsilon SS_n\cap SR_n$.

Suppose that for some index m ,$(1\le m\le k-1)$ $d_{i_m}\ne d_{i_{m+1}}+1$. Let $d_{i_{m+1}}+1=\ell$. Then ℓ can not appear between d_{i_m} and $d_{i_{m+1}}$ in Π , since it is not a member of D. Therefore one of the two subsequences $<\ell,d_{i_m},d_{i_{m+1}}>$ or $<d_{i_m},d_{i_{m+1}},\ell>$ must appear in Π , thus contradicting $\Pi\epsilon SS_n\cap SR_n$.

For each permutation $\Pi\epsilon SS_n\cap SR_n$, we can generate two permutations Π_1 and Π_2 of order n+1 as follows;

(a) generate Π_1 by inserting n+1 one position to the left of n in Π ,

(b) generate Π_2 by putting n+1 after the rightmost element in Π .

Clearly , condition (3.16) is not violated in Π_1 and Π_2 thus generated. Furthermore , inserting n+1 in any other position of Π, generates a maximal descending subsequence (with n+1 as its first element) which does not satisfy condition (3.16). Therefore Π_1 and Π_2 belong to $SS_{n+1}\cap SR_{n+1}$. Since all the elements of $SS_{n+1}\cap SR_{n+1}$ are generated in this way, we have

$$|SS_{n+1}\cap SR_{n+1}|=2|SS_n\cap SR_n| \quad . \qquad (3.17)$$

Our result follows from the fact that $SS_3\cap SR_3$ contains 4 elements , namely, <1,2,3>,<1,3,2>,<2,1,3>,<3,2,1> . □

3.9 THE AVERAGE NUMBER OF INVERSIONS

Theorem 3.15 The average number of inversions in a random permutation of SS_n is

$$\frac{1}{2}\left(\frac{4^n}{C_n} - 3n - 1\right).$$

Proof Let $i(\Pi)$ denote the number of inversions in a permutation Π and $int(T)$ the internal path length of the tree T. The sum of sizes of all subtrees in a binary tree (or any other tree) is equal to $int(T)$. This follows from the fact that in a tree T, the distance of vertex i from the root is equal to the number of subtrees in which i participates.

Let $\langle s_1, s_2, \ldots, s_n \rangle$ be the SR inversion-table of a permutation $\Pi \in SS_n$, then by definition

$$\sum_{i=1}^{n} s_i = i(\Pi). \tag{3.18}$$

By Lemma 3.2 , $i(\Pi)$ is the sum of sizes of all left subtrees in T_π . Hence, by the symmetry of left and right subtrees

$$\sum_{\Pi \in SS_n} int(T_\pi) = 2 \sum_{\Pi \in SS_n} i(\Pi) . \tag{3.19}$$

The value of the left member of (3.19) is given in [4, pp 404] as

$$\sum_{\Pi \in SS_n} int(T_\pi) = 4^n - (3n+1)C_n , \tag{3.20}$$

from which the result follows . □

It is interesting to note that on the average a random permutation of SS_n contains $O(n^{1.5})$ inversions, whereas the corresponding value for a random permutation of order n is $O(n^2)$.

CHAPTER FOUR

AN ALGORITHM FOR GENERATING AND INDEXING BINARY TREES

4.1 INTRODUCTION

In this chapter we describe an algorithm which generates all 'shapes' of n-noded binary trees. Such algorithms are used in investigating and comparing various deletion schemes in binary trees [20] and can be effectively employed for systematic generation of combinatorial objects which are in 1-1 correspondence with such trees [23,pp·154] .

The algorithm is based on a correspondence between binary trees and the class SS_n of stack-sortable permutations, together with a representation of such permutations as ballot-sequences (Theorem 3.3). Initially, a ballot-sequence of length n is generated. This is then used to construct a binary tree. It is shown that if a ballot-sequence is an inversion table of $\Pi \in SS_n$, then the algorithm generates T_π. Thus , by generating all ballot-sequences of length n , all C_n binary trees of n nodes are obtained.

A unique integer , $num_n(B)$, between 1 and C_n , is associated with each ballot-sequence B of length n , by using some combinatorial properties of such sequences. The lexicographic order is preserved by this association , namely , for any two ballot-sequences B and B' , if B precedes B' then $num_n(B) < num_n(B')$.

A simple recursion relation , is used both in computing $num_n(B)$ from a given B , and its inverse $num_n^{-1}(m)$ from a given integer $m < C_n$. This provides the capability of storing a binary tree of n nodes as an integer smaller than C_n , as well as efficient generation of a random binary tree.

Definitions

Given a binary tree T , one of the following relations can hold between two adjacent nodes i and j in T;

(a) iLSj (i is the left son of j).

(b) iRSj (i is the right son of j).

(c) iRFj (jRSi).

(d) iLFj (jLSi).

A path $<a_o,a_1,\dots,a_\ell>$ between nodes u and k in T, where $a_o = u$ and $a_\ell = k$, is denoted by $P_T(u,k)$. It can be characterized as a product of relations $\alpha_1\cdot\alpha_2\cdot,\dots,\cdot\alpha_\ell$ where α_i is the relation between a_{i-1} and a_i in the path. Two paths $P_T(u,k)= \alpha_1\cdot\alpha_2\cdot,\dots,\cdot\alpha_\ell$ and $P_T(r,s)=\alpha'_1\cdot\alpha'_2\cdot,\dots,\cdot\alpha'_\ell$ are similar iff for $1\le i\le \ell$, $\alpha_i=\alpha'_i$. Similarity between two binary trees can be now defined in terms of paths as follows; two binary trees T and T' with the roots r and r' respectively , are similar, if there is a one-to-one correspondence between their nodes, such that if $u\in T$ corresponds to $u'\in T'$,then $P_T(r,u)=P_{T'}(r',u')$. This definition is used in the proof of Theorem 4.1, it is easy to check that it is equivalent to the definition which was given in Section 3.2.

4.2 GENERATION

To simplify the presentation,the generation of the ballot-sequence , and the construction of the binary tree , are given seperately in two procedures. Procedure BALLOT generates the next ballot-sequence from the previous one in an array B , and then calls procedure TREE to construct the corresponding binary tree. Semi-formal descriptions with some explanatory notes are given below.

ALGORITHM 4.1 (BALLOT)

The application of this procedure will generate all ballot-sequences of length n in B, in their lexicographic order, where the rightmost digit is the most significant one. Note that by definition B[n] is always 0, and the values of an entry B[i] range from B[i+1] up to n-i. The algorithm is optimal in the sense that the new sequence is generated by setting only those entries which differ from the previous one's.

```
Step 1 :  Set B[n]=0.
Step 2 :
                 2.1 : Set B[n-1]=B[n].
                 2.2 : Set B[n-2]=B[n-1].
                 .
                 .
                 .
                 .
                 .
                 2.n-1:Set B[1]=B[2].
                 2.n  :Set m=1.
Step 3 : ( The next ballot-sequence is ready in B) Call TREE.
Step 4 :  Set B[m]=B[m]+1 . If B[m]>n-m go to Step 6.
Step 5 :  If m≠1 go to Step 2.n-(m-1) else go to Step 3.
Step 6 :  Set m=m+1 . If m=n stop , else go to Step 4.  □
```

TREE - Informal description

Given a ballot-sequence in B, we create its difference-sequence in the array D, as shown in the proof of Theorem 3.4. By the definitions of D and B, it follows that for $1 \le i \le n$, $D[i] \ge 0$, and $D[1]+D[2]+,..,+D[n]=n$.

The array D is scanned from D[1] to D[n], and a portion of the tree is constructed as each entry is processed, as described below. During the construction, the nodes are labelled from 1 to n in symmetric order. This labelling may be required in some cases, and it also serves in proving the validity of the algorithm, it can be omitted if we are only concerned with the 'shape' of the tree. The algorithm also uses a stack.

Initially, a pointer P points to a dummy root labelled 0, and in general, P always points to a node labelled i-1 when D[i] is processed. We distinguish between two cases;

case 1 - $D[i]>0$: Let $D[i] = j$, then j new nodes are created $g_1, g_2, .., g_j$ such that $(i-1)RFg_1$ and for $1 \le \ell \le j-1$ $g_\ell LFg_{\ell+1}$. Each node is pushed into the stack after it is created, finally, the last node g_j is removed from the stack and the label i is assigned to it (all the other nodes are not labelled at this stage). The pointer P now points to the node i. Thus, the path $P_T(i-1,i)$ is constructed, and it has the following 'shape',

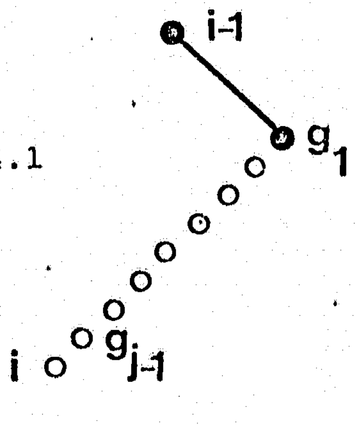


Figure 4.1

case 2 - D[i]=0: The node on top of stack is removed and assigned the label i , again , P now points to i.

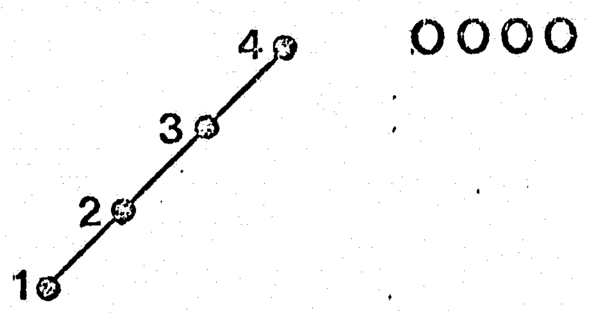
In this way, all the entries of D are processed to obtain a labelled binary tree of n nodes. If only the 'shape' matters, we may stop even earlier, after n nodes are generated. Finally, the dummy root is removed and P points to its right son.

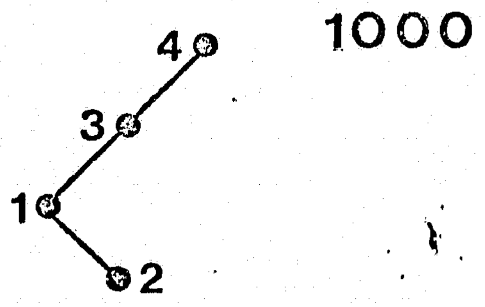
ALGORITHM 4.1 (TREE)

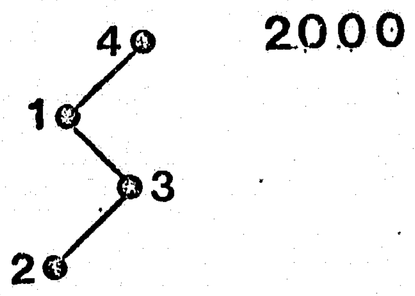
Initially, P points to a dummy root; ST is the top of an empty stack.

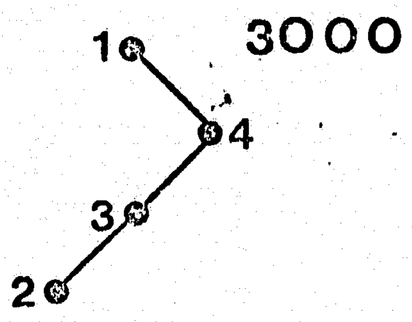
```
Step 1  : Set D[1]= n-B[1] , and for 2≤i≤n  set D[i]=B[i-1]-B[i].
          Set i=1.
Step 2  : Assign the label i-1 to the node which is indicated
          by P.
Step 3  : Set j=D[i] . If j=0 go to Step 7.
Step 4  : Create a node as a right son of the node indicated
          by P, place the new node in ST.
Step 5  : Set j=j-1. If j=0 go to Step 7.
Step 6  : Create a node as the left son of the node which is in
          ST , place the new node in ST , then go to Step 5.
Step 7  : If i=n  go to Step 10 .
Step 8  : Set i=i+1 .
Step 9  : Set P to point at the node in ST, remove that node
          from ST , then go to Step 2.
Step 10 : Assign the label n to the node in ST. Remove the
          dummy root , and let P point to its right son, then
          stop. □
```

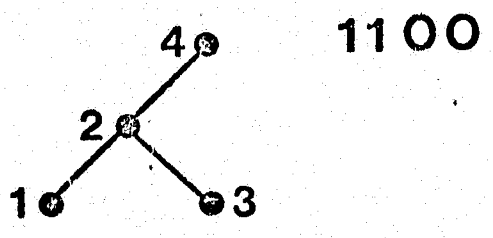
As an example, all binary trees of order 4 are shown in Figure 4.2 in their order of generation.

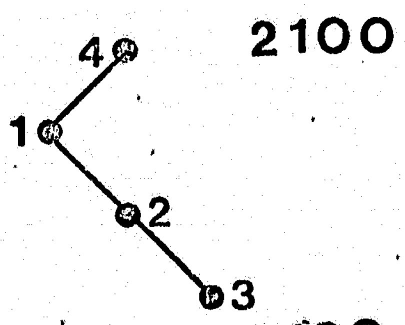
1


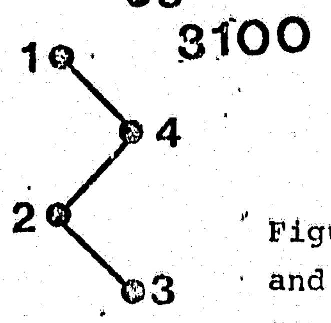
2


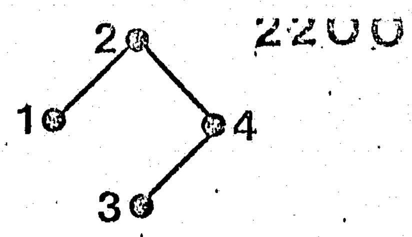
3


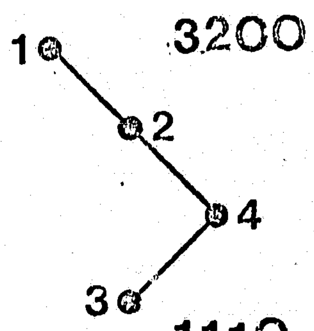
4


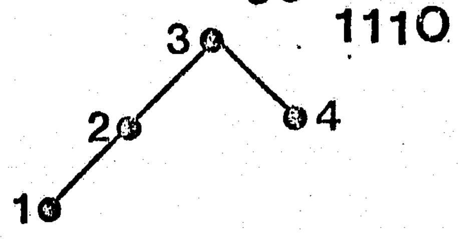
5


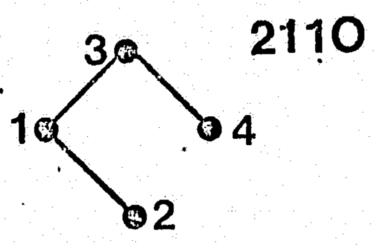
6


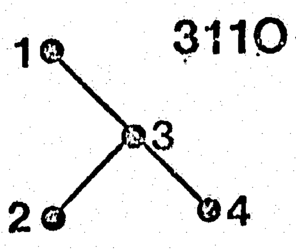
7


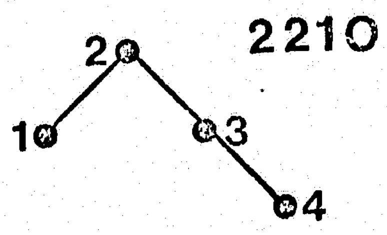
8


9


10


11


12


13


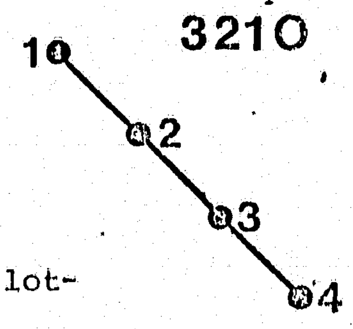
14


Figure 4.2- The binary trees and their corresponding ballot-sequences in their order of generation by procedure BALLOT.

<u>Theorem 4.1</u> Let $\Pi\in SS_n$ have an inversion-table B . Then TREE constructs from B a binary tree T_b , such that $T_b=T_\pi$.

<u>Pf.</u> : For convenience let the number 0 be added to Π as its leftmost member, and let us consider the generated tree T_b with its dummy root. We have to prove that $P_{T_b}(0,i)=P_{T_\pi}(0,i)$ for $1\le i\le n$. First let us show that $P_{T_b}(i-1,i)=P_{T_\pi}(i-1,i)$ for the case when i-1 is to the left of i in Π. According to the construction of T_π , any element between i and i-1 which is smaller than i-1 will not appear on the path $P_{T_\pi}(i-1,i)$. Therefore ,consider $G=<g_1,g_2,\ldots,g_j=i>$, the subsequence of all elements greater than i-1 which appear between them in Π. If $j\le 2$, then G is trivially a decreasing subsequence . For $j>2$, suppose that G is non-decreasing , then we can find two elements $g_k<g_\ell$ with $k<\ell$, and this in turn implies that Π contains a subsequence

$$g_\ell<g_k<g_j=i \quad \text{and} \quad k<\ell<j \tag{4.1}$$

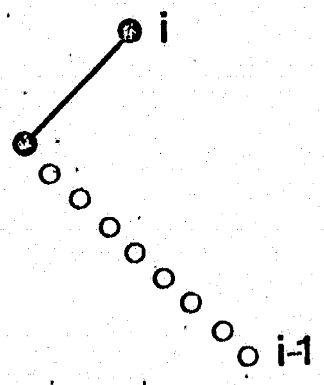
which would contradict the fact that $\Pi\in SS_n$ (Theorem 3.1*).Thus G is always decreasing. Therefore $P_{T_\pi}(i-1,i)$ will have the 'shape' shown in Figure 4.1,which is also the 'shape' of $P_{T_b}(i-1,i)$, since by definition $j=D[i]$.

The proof now proceeds by induction on i. As a special case $P_{T_\pi}(0,1)=P_{T_b}(0,1)$. Assume that

$$P_{T_\pi}(0,i-1)=P_{T_b}(0,i-1) \ . \tag{4.2}$$

When i-1 is to the left of i in Π , then $P_{T_\pi}(i-1,i)=P_{T_b}(i-1,i)$.

This together with the induction hypothesis(4.2) gives the desired result. When i-1 is to the right of i in Π, we need to show that node i is placed at corresponding points on the paths fr O to i-1 by both constructions. Clearly, in T_π, i-1 is the ri cmost node in the left subtree of i. Namely , the path P_{T_π} (i-1,1) has the shape given in Figure 4.3. Note that in this case D[i]=O . Therefore i is assigned to the node on top of stack, which is the last one generated before i-1, and having a left son. This observation combined with(4.2) completes the proof.



Fi 4.3

$\sqcap$

4.3 INDEXING BINARY TREES

To store a binary tree T as an integer, we need to know its index with respect to the generation scheme of procedure TREE. This can be achieved by solving the following two problems:

i) find the ballot-sequence B which is the inversion-table of $\Pi \in SS_n$, such that $T_\pi = T$,

ii) find the number $num_n(B)$, of ballot-sequences of length n which precede B in the lexicographic order of procedure BALLOT.

We shall later see that the difference-sequence D is generated in the solution of both problems as an intermediary step, and therefore B does not need to be explicitly derived. However, to simplify the presentation we shall solve both problems as posed. The solution to the first one is an algorithm which is the inverse of procedure TREE, and will be illustrated by an example. Consider the tree T given in Figure 4.4 . T is traversed in symmetric order, and the number of new nodes which are pushed into the stack before the removal of the i'th node from it is recorded as D[i]. This results in the sequence D = <2,0,3,0,1,0,1,1> which in fact is the difference-sequence of the ballot-sequence corresponding to T. Hence, we find B = <6,6,3,3,2,2,1,0>, . The justification of this algorithm uses the same arguments as in the proof of Theorem 4.1.

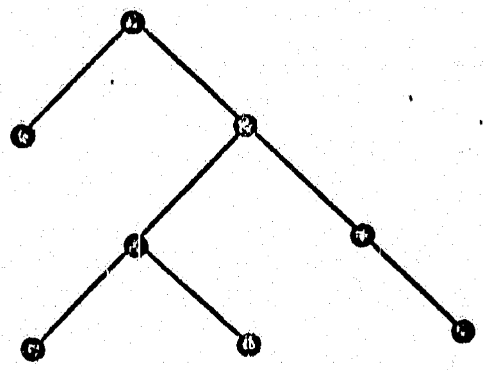


Figure 4.4

We now turn to the problem of finding $num_n(B)$ for a given ballot-sequence. Let S_{nk} be the set of all ballot-sequences of order n, with exactly k non-zero digits. A sequence $U_{nk} \epsilon S_{nk}$ will be called a unit-sequence if its k non-zero digits are all equal to 1.

Lemma 4.1

$$|S_{nk}| = \begin{cases} 1 & \text{for } k=0 & (4.3a) \\ |S_{(n-1)k}| + |S_{n(k-1)}| & \text{for } 0<k\le n-1 & (4.3b) \\ 0 & \text{for } k=n & (4.3c) \end{cases}$$

Proof The relations (4.3a) and (4.3c) follow directly from the definitions. To prove (4.3b) we first show that

$$|S_{nk}| = \sum_{i=1}^{k} |S_{(n-1)i}| \quad , \qquad (4.4)$$

by constructing a correspondence between S_{nk} and $J = \bigcup_{i=0}^{k} S_{(n-1)i}$.
Given a sequence $B=\langle b_1,b_2,\ldots,b_n\rangle \epsilon SS_n$, we define $B'=\langle b_1',b_2',..,b_{n-1}'\rangle$ as follows ,

$$\begin{array}{lll} b_i'=b_i-1 & \text{for} & 1\le i\le k \; , \\ b_i'=b_i=0 & \text{for} & k<i\le n-1 \; . \qquad (4.5) \end{array}$$

Clearly, every member of S_{nk} is mapped by (4.5) into a unique member of J. Conversely , given any $B'\epsilon J$, we can find the unique $B\epsilon S_{nk}$ associated with it using ,

$$\begin{array}{lll} b_i=b_i'+1 & \text{for} & 1\le i\le k \\ b_i=0 & \text{for} & k<i\le n \; . \qquad (4.6) \end{array}$$

This correspondence implies (4.4) from which (4.3b) is readily derived. □

In the sequel the subscript of 'num' is omitted when it is applied to unit-sequences.

<u>Lemma 4.2</u> $num(U_{nk}) = |S_{(n+1)(k-1)}|$ for $0<k<n$. (4.7)

<u>Proof</u> By definition the sequence U_{nk} is the first ballot-sequence in the lexicographic which has k non-zero digits. Hence , the sequences which precede U_{nk} in this order , are exactly the ones which have fewer than k non-zero digits. The number of such sequences is $\sum_{i=0}^{k-1} |S_{ni}|$. The result now follows from (4.4). □

<u>Theorem 4.2</u> For a ballot-sequence $B=\langle b_1,b_2,\ldots,b_n\rangle$, let g_i $(1\le i\le b_1)$ be the number of elements in B which are not smaller than i. Then

$$num_n(B) = \sum_{i=1}^{b_1} num(U_{(n+1-i)g_i}) . \qquad (4.8)$$

<u>Proof</u> The proof is by induction on the length of B. The result is easily verified for sequences of length 2. Assume that it holds for ballot-sequences of length n-1. Let S(X) denote the set of sequences which are generated before X by procedure BALLOT.

Since B has g_1 non-zero digits , it follows that S(B) can be partitioned into two disjoint sets $S(U_{ng_1})$ and $S(B)-S(U_{ng_1}) = G$. Therefore ,

$$num_n(B) = |S(U_{ng_1})| + |G| = num(U_{ng_1}) + |G| . \qquad (4.9)$$

Consider the ballot-sequence B' of length n-1 , which is obtained according to (4.5). Using the mappings of (4.5) and (4.6) , a 1-1 correspondence between G and S(B') can easily be established. Hence,

$$num_n(B) = num(U_{ng_1}) + num_{n-1}(B') . \qquad (4.10)$$

Let there be g_i' digits which are not smaller than i in B' , then by the construction of B' and the induction hypothesis

$$g_i' = g_{i+1} \tag{4.11}$$

$$\mathrm{num}_{n-1}(B') = \sum_{i=1}^{b_1-1} \mathrm{num}(U_{(n-1)g_i'}) \; . \tag{4.12}$$

The result follows by substituting (4.11) and (4.12) into (4.10). □

The two algorithms for computing $\mathrm{num}_n(B)$ from B and its inverse $\mathrm{num}_n^{-1}(m)$ from an integer $m<C_n$, are both based on Theorem 4.2. However, they are considerably simplified by observing that in the difference-sequence D[1],...,D[n] of B, an entry D[k] represents the number of times that g_i is equal to k-1. The entry D[1] is irrelevant in both cases. For fast performance, the algorithms make use of a pre-computed table TAB containing the values of $\mathrm{num}(U_{ik})$ $1 \le i \le n$ and $1 \le k \le n-1$, which can be computed directly using the results of Lemmas 4.1 and 4.2.

The first procedure, NUM, does not require any explanation since it is a straightforward application of (4.8). The computation of the inverse in procedure INVNUM, is based on a recursive application of (4.10). Observe that in row n of TAB, $\mathrm{num}(U_{ng_1})$ is the maximal value smaller than m. Similarly, in row n-1, $\mathrm{num}(U_{(n-1)g_2})$ is the maximal element smaller than $m-\mathrm{num}(U_{ng_1})$, and so on. Thus, after $\mathrm{num}(U_{ik})$ is found in row i, and subtracted from the argument, row i-1 is searched for the index of the maximal unit-sequence of order i-1. This process is continued until all the terms of (4.8) are found. Note that in any row i of TAB, the first i-1 elements constitute a non-decreasing sequence suitable for binary search, and that this search can be restricted to fewer than i-1 elements since the g_i's are non-increasing.

ALGORITHM 4.2 (NUM)

Given the difference-sequence D of a ballot-sequence B, the algorithm computes the value of $num_n(B)$.

```
Step 1 : Set i=0, set k=n ,set $num_n(B)=0$.
Step 2 : If D[k]> 0 perform Step 4 D[k] times.
Step 3 : Set k=k-1. If k=1 stop , else go to Step 2.
Step 4 : Set $num_n(B)= num_n(B) + num(U_{(n-i)(k-1)})$,
         set i=i+1.                                        □
```

ALGORITHM 4.3 (INVNUM)

Given an integer $m<C_n$, and the vector D of n entries, the difference-sequence of $B=num_n^{-1}(m)$ is constructed in D (D[1] is not computed since it is not required for the construction of B).

```
Step 1   (Initialize): Set D[i] = 0 for $1\le i\le n$, set i=n ,
                       set k=n-1 , set ARG=m.
Step 2 : If ARG=0 stop ( the ballot-sequence B can now be
         constructed from D[2],D[3],..,D[n]).
Step 3 : Search in row i of TAB between columns 1 and k, for
         the maximal element which is not greater than ARG
         ('-'='$\infty$' for this purpose).
Step 4 : Suppose that this element is found in column j,
         set D[j+1]=D[j+1]+1 , set k=j,
         set ARG=ARG-num($U_{ij}$).
Step 5 : Set i=i-1 and go to Step 2.        □
```

Example

Given $B=<5,3,1,1,0,0>$ find $num_6(B)$. We first derive $D=<1,2,2,0,1,0>$. Hence, by procedure NUM

$$num_6(B) = num(U_{64}) + num(U_{52}) + num(U_{42}) + num(U_{31}) + num(U_{21})$$

$$= 48+5+4+1+1 = 59.$$

Conversely, consider the construction of the ballot-sequence B of length 6 whose index is 86. In this case, five successive rows of the table (given below) containing the values of $num(U_{ik})$, are searched. The contents of array D after each search are listed below.

i) The maximal element not exceeding 86 in row 6 is 48, found in column 4, $D=<0,0,0,0,1,0>$.

ii) $m=86-48=38$, the required number in row 5 is $num(U_{54})=28$, $D=<0,0,0,0,2,0>$.

iii) $m=38-28=10$, $D=<0,0,0,1,2,0>$.

iv) $m=10-9=1$, $D=<0,1,0,1,2,0>$.

v) $m=1-1=0$ and no more searches are needed.

The required sequence B is constructed from D (in iv) as

$$B=<4,3,3,2,0,0> \quad . \qquad \square$$

Table of the $num(U_{ik})$ values

i \ k	1	2	3	4	5
1	-	-	-	-	-
2	1	-	-	-	-
3	1	3	-	-	-
4	1	4	9	-	-
5	1	5	14	28	-
6	1	6	20	48	90

4.4 COMPARATIVE EVALUATION

In this chapter , the approach taken for generating and indexing of all 'shapes' of binary trees , is based on a relation between such trees and ballot-sequences of the same order.

A different approach,based directly on stack-sortable permutations and Construction-T , has been adopted by Knott [20] to solve the same problems. It may be argued that the indexing we propose is not a natural one , while Knott uses the natural indexing [4 , pp 331] for binary trees. However , the suggested algorithms are considerably more efficient than their counterparts in similar works known to the authors.

Let us first consider (a) Construction-T as opposed to (b) procedure TREE. The actual creation of the nodes of the tree is the same in both cases . In creating the 'shape' , it is well known that (a) requires $O(n^2)$ comparisons for worst case , and $O(n\lg_2 n)$ for best case. The amount of assignment statements have similar bounds. In (b) , the extra memory space for n-1 pointers which may have to be stacked is insignificant. Comparisons are applied to the value $j=D[i]$, and since $\sum_{i=1}^{n} D[i] = n$, it is clear that exactly n comparisons are performed , while the number of assignment statements is only $O(n)$. The superiority of (b) is highlighted especially for those applications where all C_n 'shapes' need to be generated.

If the 'shape' of the tree is given , generating its corresponding permutation Π or ballot-sequence B is straightforward and of equivalent complexity. However , if only the index of the tree is given , as in the case of random tree generation , the cost of procedure INVNUM is governed by the binary search which may require

$$\sum_{i=1}^{n} (\lfloor lg_2(i-1) \rfloor + 1) \approx O(n lg_2 n)$$

comparisons between entries of a two dimensional array TAB and the argument m of the procedure. In addition there is a fixed initial cost of $O(n^2)$ additions in computing all the values in TAB , but this is insignificant since in many applications it is required to generate random trees in large numbers.

Similar algorithms for generating Π from a given index , rely on the definition of natural order among binary trees and the well known recursion relation

$$G_{n+1} = \sum_{j=0}^{n} G_j G_{n-j} \quad ;$$

where G_j is the number of distinct binary trees of order j. The relation between the complexities of the recursive procedure given by Knott [20] for this purpose, and INVNUM seems to be similar to the relation between Construction-T and procedure TREE.

Finally, let us consider the case where k trees having consecutive index numbers q,q+1,...,q+k-1 , are to be generated. Using INVNUM , we find the sequence $B=num_n^{-1}(q-1)$. Then we apply procedure BALLOT with a slight modification so that only k sequences are generated , starting with the initial sequence B. Note that we compute a ballot-sequence from a given index only once, and each consecutive sequence is generated from its predecessor at minimal cost. On the other hand, to generate k trees corresponding to consecutive permutations would require the transformation of k indices, since there is no simple way of deriving stack-sortable permutations in their order corresponding to the natural order of trees.

CHAPTER FIVE

A CORRESPONDENCE BETWEEN BINARY TREES AND 2-PERMUTATIONS *

5.1 INTRODUCTION

In this chapter we deal with the special case of permutations which do not contain a descending subsequence of length three. This class of permutations is called 2-permutations , and the set of 2-permutations of order n is denoted by P_n^2.

In [16, pp 64] , Knuth proves that

$$|P_n^2| = C_n = (n+1)^{-1}\binom{2n}{n} \ . \qquad (5.1)$$

The proof in employs an indirect method involving the enumerative theory of SYT. It is based on a correspondence of MacMahon [21, pp 130] between an SYT of shape <n,n> and a pair of SYT's of order n which have the same shape and contain at most two rows. Knuth also states that " Curiously there seems to be no apparent way to establish a correspondence between such permutations and binary trees ...," , and compares this situation to the direct correspondence which exists between the class SS_n and binary trees of n nodes.

* Some of the contents of this chapter were published by the author [24].

5.2 THE CORRESPONDENCE

Let $\Pi=\langle p_1,p_2,\ldots,p_n\rangle$ be a permutation on the set $N=\{1,2,..,n\}$, an element $j\varepsilon N$ is a left-to-right maximum in Π , iff $j>x$ for all x in $L_\pi(j)$. Let M_π denote the set of all left-to-right maxima in Π, and let N_π be the set of all other elements of Π .

Example 5.1

If $\Pi=\langle 3,4,2,6,1,5\rangle$, we have $M_\pi=\langle 3,4,6\rangle$ and $N_\pi=\langle 2,1,5\rangle$. Since there is a descending subsequence $4>2>1$, Π is not a 2-permutation. □

The basis of the correspondence to be described here , is the following simple characterization of 2-permutations.

Lemma 5.1 Π is a 2-permutation iff the elements of N_π form an ascending subsequence in Π .

Proof If Π is not a 2-permutation , it contains a descending subsequence of length three ,

$$p_i>p_j>p_k \quad \text{and } i<j<k . \tag{5.2}$$

Then , both p_j and p_k are members of N_π and therefore N_π is not an ascending subsequence.

Conversely , suppose that N_π is not an ascending subsequence , then it contains two elements ,

$$p_j>p_k \quad \text{and } j<k . \tag{5.3}$$

By the definition of N_π there is an element $p_i\varepsilon L_\pi(p_j)$ such that $p_i>p_j$. Hence , Π contains a subsequence (5.2) and it is not a 2-permutation. □

We now define a mapping from $\Pi \in P_n^2$ to an n-tuple of integers $B(\Pi) = \langle b_1, b_2, \ldots, b_n \rangle$ in the following way:

<u>Mapping B</u>

Let $\Pi = \langle p_1, p_2, \ldots, p_n \rangle$ then

(1) $b_1 = 0$;

(2) for $2 \le i \le n$

(a) $b_i = b_{i-1}$ iff $p_i \in M_\pi$;

(b) $b_i = p_i$ iff $p_i \in N_\pi$. $\square$

<u>Example 5.2</u>

Given $\Pi = \langle 1,4,2,3,5 \rangle$ we construct $B(\Pi)$. In this case $M_\pi = \langle 1,4,5 \rangle$ and $N_\pi = \langle 2,3 \rangle$;

$b_1 = 0$

$4 \in M_\pi \Rightarrow b_2 = b_1 = 0$

$2 \in N_\pi \Rightarrow b_3 = 2$

$3 \in N_\pi \Rightarrow b_4 = 3$

$5 \in M_\pi \Rightarrow b_5 = b_4 = 3$.

Hence $B(\Pi) = \langle 0,0,2,3,3 \rangle$. $\square$

This mapping preserves the elements of N_π and their relative position in Π , therefore it follows that for $\Pi, \Pi' \in P_n^2$, $\Pi \ne \Pi'$ implies that $B(\Pi) \ne B(\Pi')$.

<u>Theorem 5.1</u> If $\Pi = \langle p_1, p_2, \ldots, p_n \rangle \in P_n^2$ and $B(\Pi) = \langle b_1, b_2, \ldots, b_n \rangle$ is its corresponding n-tuple then

$$b_1 \le b_2 \le \ldots \le b_n \quad ; \tag{5.4}$$

$$b_j \le j-1 \quad \text{for } j=1,2,\ldots,n \quad . \tag{5.5}$$

Proof

(1) Assume that (5.4) does not hold, then, for some index j such that $1<j<n$ we have $b_j>b_{j+1}$. By the rules of Mapping B,

$$b_{j+1}=p_{j+1}\in N_\pi \tag{5.6}$$

$$b_j=p_\ell\in N_\pi \quad \text{where} \quad \ell\le j\ . \tag{5.7}$$

But $p_\ell>p_{j+1}$ contradicts Lemma 5.1.

(2) Let j be the first index such that

$$b_j>j-1\ . \tag{5.8}$$

Then $j>1$. Since j is the first index for which (5.8) holds, it follows that $b_j\ne b_{j-1}$. By Rule 2(a) of Mapping B,

$$b_j=p_j\in N_\pi\ . \tag{5.9}$$

By the definition of N_π, there exists an element p_i such that

$$p_i>p_j \quad \text{and} \quad p_i\in L_\pi(p_j). \tag{5.10}$$

All p_j-1 elements which are smaller than p_j, must be members of $L_\pi(p_j)$. For if there is an element p_ℓ such that

$$p_\ell<p_j \quad \text{and} \quad p_\ell\notin L_\pi(p_j)\ , \tag{5.11}$$

we can show a descending subsequence of length three in Π,

$$p_i>p_j>p_\ell \quad \text{and} \quad i<j<\ell\ . \tag{5.12}$$

From (5.10) and the contradiction of (5.12), we find that

$$|L_\pi(p_j)|\ge p_j\ , \tag{5.13}$$

and from (5.8)

$$|L_\pi(p_j)|\ge b_j>j-1. \tag{5.14}$$

This contradicts the fact that $L_\pi(p_j)$ contains exactly $j-1$ elements. Hence (5.8) cannot hold and (5.5) is proved. □

Given an n-tuple of integers $B=\langle b_1,b_2,\ldots,b_n\rangle$ which satisfies (5.4) and (5.5), the reversed sequence $B^R=\langle b_n,b_{n-1},\ldots,b_1\rangle$ is a ballot-sequence (as defined in Section 3.3). Hence, a 2-permutation is mapped into a reversed ballot-sequence by Mapping B. We now define Mapping T, which maps a reversed ballot-sequence into a 2-permutation. It is then proved that Mapping T is the inverse of Mapping B.

<u>Mapping T</u>

Given a reversed ballot-sequence $B=\langle b_1,b_2,\ldots,b_n\rangle$ and n empty cells $p_1,p_2,\ldots,p_n$, the sequence T(B) is constructed as follows.

Step 1 : For $1<j\leq n$ put all b_j such that $b_j\neq b_{j-1}$ into the corresponding empty cell p_j.

Step 2 : Take all elements of $\{1,2,\ldots,n\}$ which do not appear in B and put them in ascending order in the non-occupied cells. □

<u>Example 5.3</u>

Given the reversed ballot-sequence $B=\langle 0,1,1,2,3,3,6,6\rangle$;

Step 1 : $\boxed{}\boxed{1}\boxed{}\boxed{2}\boxed{3}\boxed{}\boxed{6}\boxed{}$

Step 2 : $\boxed{4}\boxed{1}\boxed{5}\boxed{2}\boxed{3}\boxed{7}\boxed{6}\boxed{8}$

Hence $T(B)=\langle 4,1,5,2,3,7,6,8\rangle$. □

<u>Lemma 5.2</u> Given a reversed ballot-sequence B of length n , then $T(B)\epsilon P_n^2$.

<u>Proof</u> Since each of the integers $\{1,2,\dots,n\}$ is inserted into the cells in one of the two steps , $\Pi=T(B)$ is a permutation on $\{1,2,\dots,n\}$.

Assume that $\Pi\not\epsilon P_n^2$. Then it contains a descending subsequence of length three $S=<p_i,p_j,p_\ell>$. At least two members of S are inserted into the cells in Step 1 or Step 2. But in each step , the elements are inserted in ascending order. Therefore S cannot be a subsequence of Π . □

<u>Lemma 5.3</u> Let B be a reversed ballot-sequence and $\Pi=T(B)$. Then the kth cell is filled in Step 1 iff $p_k \epsilon N_\pi$.

<u>Proof</u> Let $p_k=j$ be inserted in Step 1 into the kth cell , i.e.

$$p_k=j=b_k\le k-1 \ . \qquad (5.15)$$

There are k-1 elements in $L_\pi(p_k)$ but only j-1 integers which are smaller than p_k . Therefore there is at least one member in $L_\pi(p_k)$ which is greater than p_k , and $p_k\epsilon N_\pi$.

Conversely , assume that there is an element p_ℓ in Π such that $p_\ell\epsilon N_\pi$ and p_ℓ was inserted in Step 2. Then , an element p_j exists in Π where

$$p_j\epsilon L_\pi(p_\ell) \text{ and } p_j>p_\ell . \qquad (5.16)$$

Now , p_j was inserted in Step 1 because two elements which are inserted in the same step cannot form an inversion. By the first part of the lemma , p_j must belong to N_π thus violating the fact that in a 2-permutation, N_π is an ascending subsequence (Lemmas 5.1 and 5.2). □

Theorem 5.2 The mappi·'s T and B yield a one-to-one correspondence between reversed ballot sequences and 2 permutations.

Proof We show that B and T are inverses of each other. Let $\Pi \epsilon P_n^2$, and let $B(\Pi)$ be its corresponding n-tuple under Mapping B. By the definitions, every $b_i \epsilon B(\Pi)$ such that $b_i \neq b_{i-1}$ is an element of N_π, which appears in the i^{th} position in Π. Furthermore the elements of M_π appear in increasing order within Π. Hence $T(B(\Pi)) = \Pi$, by the definition of the Mapping T.

Conversely, let S be a reversed ballot-sequence such that $\Pi = T(S)$. Then $B(\Pi) = S$ by Lemma 5.3 and the definition of the Mapping B. □

By using Theorems 5.2 and 3.3, the required correspondence between P_n^2 and binary trees is established.

5.3 PATTERNS IN PERMUTATIONS

Following Tarjan [13], we say that a permutation Π on $N=\{1,2,..,n\}$ contains the pattern $P=\langle p_1,p_2,..,p_k\rangle$ $(k\leq n)$ where P is a permutation on $\{1,2,..,k\}$, if there is a 1-1 mapping Φ from P to Π such that $\langle \Phi(p_1),\Phi(p_2),..,\Phi(p_k)\rangle$ is a subsequence of Π and $p_i<p_j$ iff $\Phi(p_i)<\Phi(p_j)$.

In terms of this definition, we proved in the previous section that the number of permutations of order n which do not contain the pattern <2,3,1> is equal to the number of those which do not contain the pattern <3,2,1>. By using the fact that Π contains P iff Π^{-1} contains P^{-1} and Π^R contains P^R, it follows that for any given permutation on $\{1,2,3\}$, the number of permutations of order n which do not contain it as a pattern is C_n (or the number of those which contain it is $n!-C_n$).

The question is asked whether for any two permutations P and P' of order k, the number of permutations of order n which contain P as a pattern is equal to those which contain P'. The many cases that were attempted by a computer support a positive answer, but no proof of this fact is known to the author. We conclude this chapter by proving the special case $k=n-1$.

Theorem 5.3 Given a permutation P of order n-1 , the number of permutations of order n which contain P as a pattern is n^2-2n+2.

Proof Let S(P) denote the set of permutations of length n which contain the pattern P. We construct the sequence P^i from P for $1\leq i\leq n$, by adding 1 to all elements of P which are not smaller than i ($P^n=P$ by this definition). It follows from the definitions that if Π contains P^i as a subsequence then $\Pi\in S(P)$. It is easy to show that the converse is also true, namely , if $\Pi\in S(P)$ it must contain at least one of $P^1,P^2,\dots,P^n$ as a subsequence. For example , when n=4 and $P=<3,1,2>$, $P^1=<4,2,3>$, $P^2=<4,1,3>$ and $P^3=<4,\ \ 2>$, the permutation $\Pi=<4,1,2,3>$ contains the pattern P , in this case P^1 , P^2 and P^3 appear as subsequences.

Given a subsequence $P^i=<p_1,p_2,\dots,p_{n-1}>$ for $i>1$, observe that;

1) P^i does not contain the integer i ,

2) if $p_j=i-1$ then in $P^1,P^2,\dots,P^{i-1}$ the integer i appears in the j^{th} position,

3) there are n members of $S(P)$ which contain P^i , those members can be generated by placing i before p_1, between p_i and p_{i+1} for $i=1,2,..,n-2$ and after p_{n-1} .

Let us generate the set $S(P)$ in the following manner, first , all permutations which contain P^1 then those which contain P^2 and so on. Clearly , a permutation Π' which is generated from P^i is a repetition of a previously generated permutation , iff Π' contains $P^{i'}$ where $i'<i$. Our purpose is to show that exactly two such repetitions occur during the above generation scheme for each P^i where $i>1$. But this follows from observation 2 , since when i moves from left to right while generating $S(P)$ from P^i, a previous subsequence $P^{i'}$ is formed when i assumes the j^{th} position with respect to the elements of $P^{i'}$. This can happen exactly when i is placed immediately to the left or to the right of $p_j=i-1$ in P^i.

Since P^1 introduces n new permutations of $S(P)$, and each of $P^2,P^3,..,P^n$ only n-2 new permutations,

$$|S(P)|=(n-1)(n-2)+n = n^2-2n+2 \quad . \quad \square$$

CHAPTER SIX

CONCLUSIONS

6.1 SUMMARY OF RESULTS

In this thesis we were concerned with the study of subsequences in permutations. The research was motivated by applications in graph theoretic algorithms. However , following some initial work , it was found that the relations of a graph to its representing permutation, may lead to combinatorial results on subsequences which are of interest in their own right.

A theorem of this nature can be found in Chapter 2 , where properties of hypergraphs were used in generalizing a result concerning lengths of monotonic subsequences in a permutation. In Chapter 3 , the class SS_n was studied from various aspects, such as average lengths of monotonic subsequences , average number of inversions , the number of involutions and others. Again , the correspondence of this class and n-noded binary trees was employed in many of the derivations. In particular, properties of binary trees such as symmetry between left and right subtrees , or the connection between sizes of subtrees and internal path length , played an important role in the proofs. It was also shown that the permutation graphs which correspond to members of SS_n , do not contain a path of length 3 without a triangular chord.

In Chapter 5, it was proved that for any given permutation on {1,2,3} , the number of permutations of order n which do not contain it as a pattern is C_n. This was shown by establishing a one-to-one correspondence between SS_n and permutations of order n with no descending subsequence of length 3. It was also conjectured that the above result can be generalized to patterns of order k , for any $k \le n$. A proof was given for the special case $k=n-1$.

Some of the above mentioned results were implemented in algorithms. Those were shown to be faster than their counterparts which are based on other techniques.

The connection-board and dynamic storage allocation problems , were both formulated in terms of permutations on a multiset. The solutions to both problems were then obtained by finding maximum monotonic subsequences in those permutations.
In the case of circle graphs , integer sequences were produced for each vertex. It was then proved that the longest ascending subsequence over all sequences thus produced , represented a maximum clique in the circle graph.

In Chapter 4 , algorithms for generating and indexing binary trees were presented. The correspondence between SS_n and n-noded binary trees, as well as a representation of members of SS_n as ballot-sequences, formed the basis of these algorithms.

6.2 PROBLEMS FOR FURTHER RESEARCH

The results obtained in this work , demonstrate the utility of the approach of using graph theoretic representations for studying subsequences in permutations , and conversely , of applying results on such subsequences to the construction of algorithms on graphs.

Some specific problems for further investigation are:

(a) Given two permutations of order n , is there an efficient algorithm which decides whether they define isomorphic permutation graphs ?

(b) In chapter 4 , a method for coding a binary tree as an integer was given. It may be useful to find out which properties of a binary tree can be determined by manipulations on its representing integer.

(c) Can the results of Chapter 5 concerning patterns in permutations , be extended to any $k \le n$? It seems that the techniques which were used to prove the cases $k=3$ and $k=n-1$, are not applicable in the general case.

In general it might prove fruitful to consider whether additional problems which are intaractable on arbitrary graphs, can be solved in polynomial time on permutation graphs.

REFERENCES

[1] S. Even , " Algorithmic Combinatorics " , The Macmillan Company 1973.

[2] C. Schensted , " Longest Increasing and Decreasing Subsequences " , Canadian. J. Math 13(1961) 179-191.

[3] P. Erdos and G. Szekeres , "A Combinatorial Problem in Geometry " , Compositio Math 2(1935) 463 - 470.

[4] D. E . Knuth , "The Art of Computer Programming " , Vol 1, Addison-Wesley 1968.

[5] S. Even and A. Itai, " Theory of Machines and Computations " , Ed. Z. Kohavi and A. Paz , Academic press 1971.

[6] R.M. Baer and Brock , "Natural Sorting " , J. Soc. Indust. Appl. Math. 10(1962) 284-304.

[7] R.M. Baer and P. Brock , "Natural Sorting over Permutation Spaces " , Math Comp 22(1968) 385-410.

[8] J. D. Dixon , "Monotonic Subsequences in Random Sequences ", Discrete Maths 12(1975) 139-142

[9] C. Berge , "Principals of Combinatorics " , Academic Press 1971.

[10] D. E. Knuth , " Permutations , Matrices , and Generalized Young Tableaux " , Pacific. J. Math 34(1970). 709-727.

[11] A. Pnueli , A. Lempel and S. Even , " Transitive Orientation of Graphs and Identification of Permutation Graphs " , Canadian. J. Math 23(1971) 160-175.

[12] S. Even , A. Pnueli and A. Lempel , " Permutation Graphs and Transitive Graphs " , J. ACM 19(1972) 400-410.

[13] R. E. Tarjan , "Sorting Using Networks of Queues and Stacks " J. ACM. 19(1972) 341-346 .

[14] F. Gavril , " Algorithms for a Maximum Clique and a Maximum Independent Set of a Circle Graph " , Networks 3(1973) 261-273.

[15] C. Berge , " G[illegible] and Hypergraphs " , Academic Press 1973.

[16] D. E. Knuth , " The Art of Computer Programming " , Vol 3 , Addison-wesley 1973.

[17] C. L. Liu , " Introduction to Combinatorial Mathematics" , Mcgraw-Hill 1968.

[18] C. Greene , " An Extension of Schensted's Theorem", Advances in Math . 14(1974) 254-265.

[19] A. Seidenberg , " A Simple Proof of a Theorem of Erdos and Szekeres " , J. London Math. Soc. 34(1959) 352.

[20] G. D. Knott , " Deletion in Binary Storage Trees " , Ph.D Thesis , Stanford University 1975 .

[21] P. A. Macmahon , " Combinatory Analysis " , Vol 1,2 , Cambridge Univ. Press 1915.

[22] F. Gavril , " Algorithms for Minimum Coloring , Maximum Clique , Minimum Covering by Cliques , and Maximum Independent Set of a Chordal Graph " , Siam J. Comput 1(1972) 180-187.

[23] M. B. Wells " Elements of Combinatorial Computing" , Pergamon Press 1971.

[24] D. Rotem , " On a Correspondence Between Binary Trees and a Certain Type of Permutation " , Inf. Proc. Letters 4(1975) 58-61.