Supplementary Notes for Graph Theory 1

Including solutions for selected weekly exercises

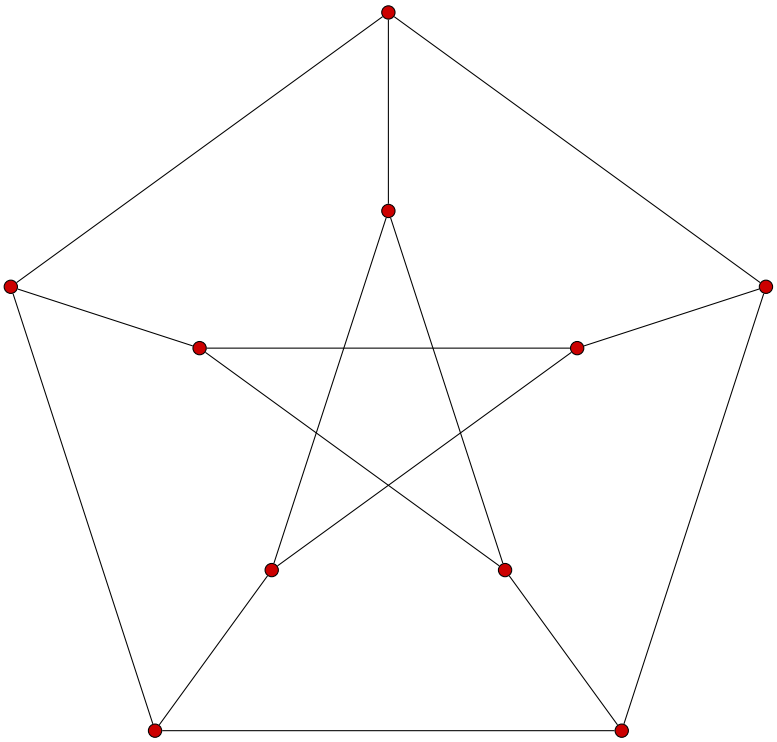
Wind, David Kofoed; Wind, David Kofoed

Publication date:
2014

Document Version
Publisher's PDF, also known as Version of record

Supplementary Notes for Graph Theory I

Including solutions for selected weekly exercises



First Edition

Authored by Hjalte Wedel Vildhøj and David Kofoed Wind

DTU Mathematics

CONTENTS

1

INTRODUCTION

These notes are written for the course 01227 Graph Theory at the Technical University of Denmark, taught by Professor Carsten Thomassen. The notes are meant solely as a supplement to the course curriculum and can under no circumstances replace the weekly lectures or group exercises. The focus is on applications and the aim is to improve the problem solving skills of the students through numerous well-explained examples. Consequently proofs are only given exceptionally, but this does not diminish the importance of understanding the proofs. A major part of the course concerns proving theorems. In that connection the lectures provide an essential resource of insight and we urge each and every student to participate.

The authors of this document take absolutely no responsibility for the usage of its contents. It is very likely that there are errors in the material. If you have found an error, have a better solution for something or wish to contribute in some constructive way please send a message to 01227notes@gmail.com.

Solutions for selected weekly exercises are included in the appendices. It is important that you try hard to solve the exercises on your own. Use the solutions only as a last resort.

We thank the following people for their contributions to these notes: *Christine Lindeblad, Tanja Søndergaard, Søren Vind, Anders Schlichtkrull* and *Helge Munk Jacobsen*.

Have fun with Graph Theory!

2

GENERAL THEORY

Most graphs can be viewed as some dots on paper, with some lines joining them, but we want to take a more formal approach. A graph G can be defined as a pair (V, E), where V is a set of vertices, and E is a set of edges. $E \subseteq V^2$, meaning that an edge $e \in E$ is a 2-element subset $\{v_i, v_j\}$ of V. So an edge is just defined by the vertices at its ends. When talking about the vertex set or edge set of a graph G, the sets will be denoted $V(G)$ and $E(G)$, and the size of the sets $v(G)$ and $e(G)$. Normally when illustrated, a graph is depicted in diagram-form as a set of dots for the vertices, joined by lines or curves for the edges. The geometric positioning of the vertices does not matter. Figure 2.1 shows an example of a graph.

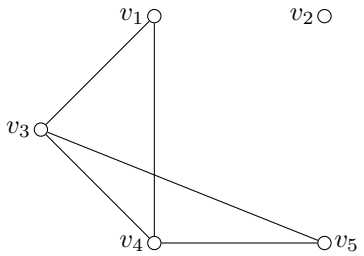


Figure 2.1: A graph with the vertex set $V = \{v_1, v_2, v_3, v_4, v_5\}$ and edge set $E = \{\{v_1, v_3\}, \{v_1, v_4\}, \{v_3, v_4\}, \{v_3, v_5\}, \{v_4, v_5\}\}$. In this graph, the vertex v_2 is completely isolated from the other vertices.

Directed Edges Until now, an edge $e = \{v_i, v_j\}$ did not point in any particular direction, but extending our definition we can allow an edge to have a direction. We denote a directed edge e as a pair of vertices (v_i, v_j) instead of a set $\{v_i, v_j\}$ of vertices. In an embedding, the edges are drawn with an arrow showing their direction.

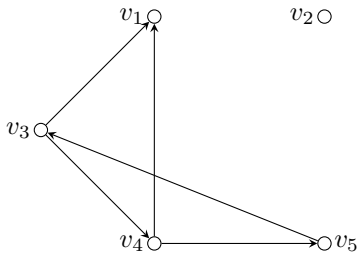


Figure 2.2: A directed graph with the vertex set $V = \{v_1, v_2, v_3, v_4, v_5\}$ and edge set $E = \{(v_3, v_1), (v_3, v_4), (v_5, v_3), (v_4, v_5), (v_4, v_1)\}$.

A graph with directed edges is called a directed graph, or a digraph. Figure 2.2 shows an example of a directed graph.

Multigraph Usually, we do not allow two vertices to be connected by more than one edge, but if we allow this, the resulting graph is called a multigraph. Figure 2.3 shows an example of a multigraph.

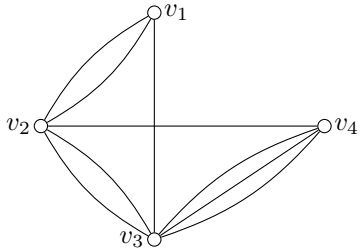
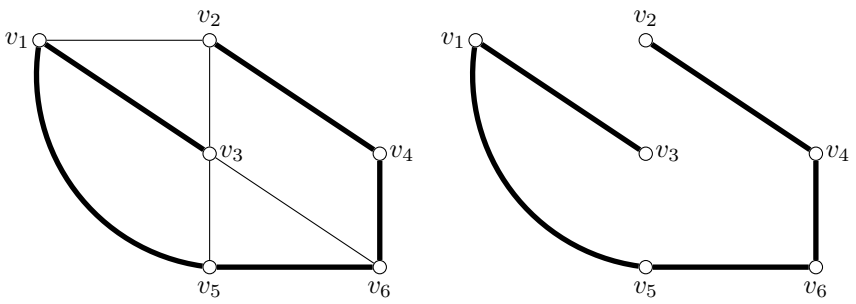


Figure 2.3: A multigraph.

A graph which is not a multigraph, is called a simple graph.

Walks and Paths A walk P is a graph with the vertex set $V = \{v_1, v_2, \ldots, v_k\}$, and with an edge set of the form $E = \{(v_1, v_2), (v_2, v_3), \ldots, (v_{k-1}, v_k)\}$. We will normally denote a walk the vertices in the order they appear on the path, e.g. $P = v_1 v_2 \ldots v_k$. If a walk contains no repeated vertices, it is called a path.



(a) A path in a graph (marked in bold). (b) The same path isolated from the graph.

Figure 2.4: A path $P = v_3 v_1 v_5 v_6 v_4 v_2$, which is also a spanning subgraph of G.

Neighbour For a vertex v, we define the neighbors $N(v)$ of v as the verticies joined to v by an edge.

Degree For a vertex v and an edge $e = (v_i, v_j)$, we call e incident to v if $v = v_i$ or $v = v_j$. The degree $d(v)$ of a vertex v, is defined as the number of edges incident to v. An isolated vertex has degree 0. For example, the vertex v_4 in Figure 2.1 has degree 3. The in-degree $d_{\text{in}}(v)$ of v is the number of directed edges going into v, and the out-degree $d_{\text{out}}(v)$ of v is the number of directed edges going out from v. For vertex v_4 in Figure 2.2 the in-degree is 1 and the out-degree is 2.

The minimum degree in a graph G is denoted $\delta(G)$ and the maximum degree is denoted $\Delta(G)$. In Figure 2.1 the minimum degree is 0 and the maximum degree is 3.

Theorem 2.1 *The sum of all degrees in a graph G is equal to twice the number of edges*

$$\sum_{v \in V(G)} d(v) = 2e(G)$$

Theorem 2.2 *The number of vertices of odd degree is even.*

Degree Sequence For a graph G, we define its degree sequence as the sequence of vertex degrees $d(v_1), d(v_2), \ldots, d(v_n)$. Sometimes we present the degree sequence sorted. See Figure 2.5 for an example.

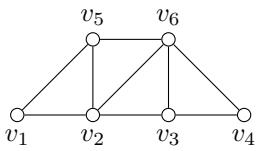
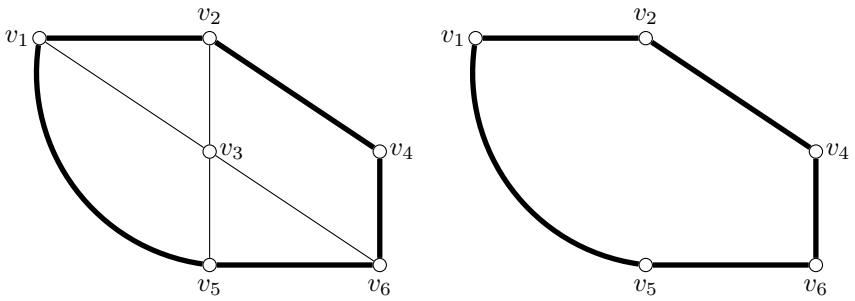


Figure 2.5: A graph with degree sequence $2, 4, 3, 2, 3, 4$ or sorted $2, 2, 3, 3, 4, 4$.

Theorem 2.3 *A sequence is the degree sequence of a graph with multiple loops allowed, if and only if the number of odd numbers in the sequence is even.*

Theorem 2.4 *A sequence is the degree sequence of a graph with multiple edges (but no loops) allowed, if and only if the number of odd numbers in the sequence is even and, in addition, the largest number in the sequence is less than or equal to the sum of the others.*

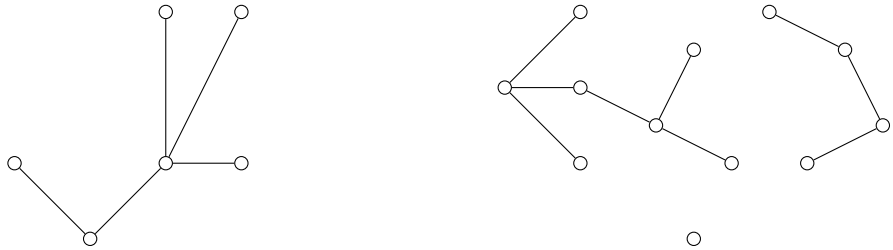
Cycles A cycle is a path $C = v_1, \ldots, v_i, \ldots, v_1$ in which all vertices except v_1 are distinct. An example of a cycle can be seen in Figure 2.6(a) and Figure 2.6(b).



(a) A cycle in a graph (marked in bold). (b) The same cycle isolated from the graph.

Figure 2.6: A cycle $C = v_1v_2v_4v_6v_5v_1$ in a graph.

Trees and Forests A connected graph containing no cycles is called a tree. A graph consisting of multiple disjoint trees is called a forest. See Figure 2.7 for an example.



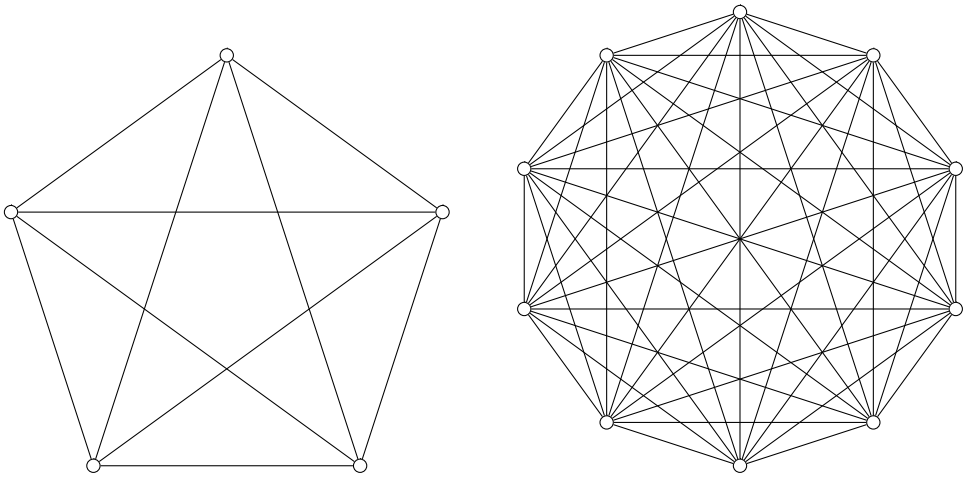
(a) A tree on 6 vertices. (b) A forest consisting of 3 trees.

Figure 2.7: A tree and a forest.

A tree on n vertices has exactly $n - 1$ edges, and any two vertices in a tree are connected by a unique simple path. Consequently, adding an edge between any two vertices in a tree will create a unique cycle.

Trees are widely used in many situations and lots of results have been proven for trees. A special kind of trees are *spanning trees* introduced in Chapter 5.

Complete Graph A complete graph is a graph in which every pair of distinct vertices is connected by an edge. We denote the complete graph on n vertices as K_n. Figure 2.8 shows an example of complete graphs.



(a) K_5 (b) K_{10}

Figure 2.8: Three complete graphs.

K_1 is just a single vertex, K_2 is a single edge, and K_3 is the triangle.

Theorem 2.5 *The number of edges in K_n is $\binom{n}{2} = \frac{n(n-1)}{2}$.*

Theorem 2.6 *K_n where $n \geq 5$ can not be drawn in the plane without edges crossing.*

Empty Graph An empty graph, is a graph on n vertices and with no edges. The special graph with no vertices and no edges is called the *null-graph*.

k-Regular Graph A k-regular graph is a graph where all vertices have degree k. 3-regular graphs are also called cubic graphs. The Petersen Graph is an example of a cubic graph (3-regular) .

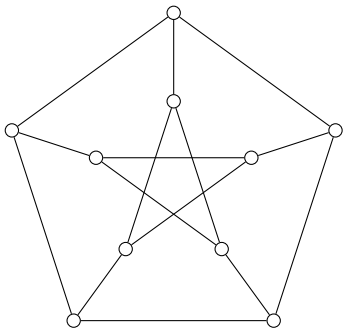


Figure 2.9: The Petersen Graph is 3-regular (also called cubic) since all vertices have degree 3.

Bipartite Graph A graph G is said to be bipartite, if it is possible to partition the vertices of G into two disjoint sets A and B, such that there are no edges between vertices inside A and no edges between vertices inside B.

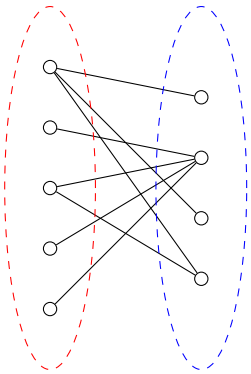


Figure 2.10: A bipartite graph, where the partitions are highligted by a red and a blue ellipsis.

It is easy to determine if a graph is bipartite by the following theorem

Theorem 2.7 *A graph is bipartite if and only if it contains no odd cycle.*

In a more general setting, we can define a k-partite graph as a graph where you can partition the vertices into k disjoint sets, such that there are no edges internally in a set. Determining if a graph is k-partite for $k > 2$ is a hard problem.

Diameter The diameter of a graph is the longest shortest path. To find the diameter, consider the shortest distance between all pairs of vertices u and v in G. The diameter is then the longest of these distances. The diameter of the Petersen Graph is 2, the diameter of a path on n vertices is $n - 1$ and the diameter of a n-cycle is $\lfloor n/2 \rfloor$.

Girth The girth of a graph, is the length of a shortest cycle in the graph. If the graph contains no cycles, the girth is defined to be infinity. For example, the Petersen Graph has girth 5 since there are no triangles or 4-cycles in the Petersen Graph.

Subgraph In a graph $G = (V, E)$, a subgraph H is a pair (V', E') where $V' \subseteq V$ and $E' \subseteq E$ and any edge in E' joins two vertices of V'. So a subgraph of G is just some part of G.

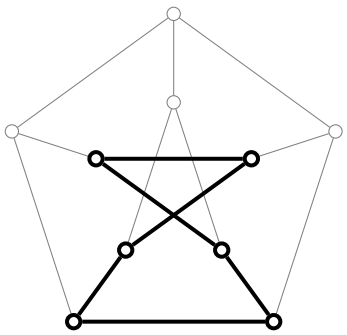


Figure 2.11: The Petersen Graph, and a subgraph highlighted.

Connectivity and Connected Components A graph is said to be connected if all pairs of vertices are connected by a path. In a graph G, a connected component is a connected subgraph of G that can not be made bigger. Figure 2.12 shows an example of a graph with three connected components.

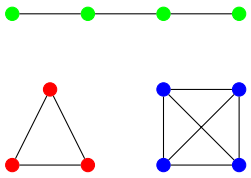


Figure 2.12: A graph with three connected components. Each connected component is marked with a unique color.

A bridge or cut-edge e in a graph G, is an edge, such that if e is removed, the number of connected components of G increases. In the same way, a cut-vertex v in G is a vertex such that a removal of v will increase the number of connected components of G.

Definition 2.1 *If k vertices need to be removed to disconnect a non-complete graph G, then G is said to have vertex-connectivity k, and we write $\kappa(G) = k$. We define $\kappa(K_{k+1}) = k$.*

Definition 2.2 *If k edges need to be removed to disconnect a graph G, then G is said to have edge-connectivity k and we write $\lambda(G) = k$.*

There is the following relation

Theorem 2.8 *In a graph G*

$$\delta(G) \geq \lambda(G) \geq \kappa(G)$$

where $\delta(G)$ is the minimum degree of G, $\lambda(G)$ is the edge-connectivity of G and $\kappa(G)$ is the vertex-connectivity of G.

Isomorphism Two graphs G and H are said to be isomorphic if there exists a bijection between the vertex sets of the graphs. In less formal terms, this means that G and H are isomorphic, if it is possible to draw G and H in the same way. If two graphs G and H are isomorphic, we write $G \simeq H$. For example, the two graphs shown in Figure 2.13 are isomorphic, even though their drawings look very different.

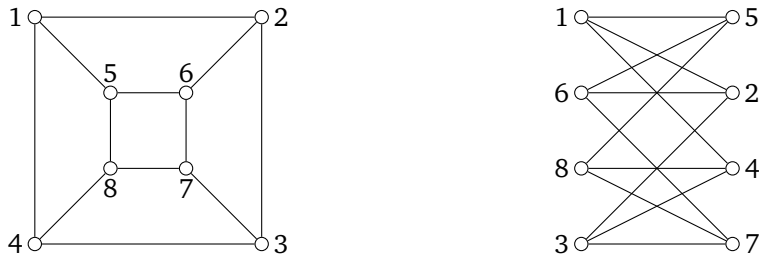


Figure 2.13: Two isomorphic graphs with different embeddings.

Planarity A graph G is said to be planar, if it is possible to draw the graph in the plane, with no edges crossing. A planar graph drawn in the plane with no crossings, is called plane. A drawing of a graph is also denoted an *embedding*.

Figure 2.14: A plane and a non-plane embedding of a planar graph.

Not all graphs are planar, and thus there exist graphs which can not be drawn in the plane without edge-crossings.

Figure 2.15: Two graphs, which does not have a plane embedding.

In a planar graph, we define a *face* as a region bounded by the edges. One special face is the outer face, which should also be counted.

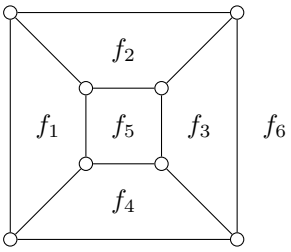


Figure 2.16: A plane embedding of the cube. Each face is denoted by f_i. Here f_6 is the outer face.

In planar graphs, there is a simple formula connecting the number of vertices $v(G)$, the number of edges $e(G)$ and the number of faces $f(G)$ in a graph called Euler's formula.

Theorem 2.9 *For a planar graph G*

$$v(G) - e(G) + f(G) = 2$$

Dual Graph For a plane embedding of a planar graph G, we can construct its dual graph G' in the following way: For each face in G, create a vertex in G'. If two faces in G are adjacent, connect the corresponding vertices in G'. Note that a planar graph can have different dual graphs, depending on the plane embedding.

A graph is said to be *self-dual*, if it is isomorphic to its dual graph. An example of a self-dual graph is K_4.

Blocks A graph is said to be biconnected or 2-connected, if it does not contain a cut-vertex. A *block* is a maximal biconnected subgraph of a given graph G. Note that maximal does not mean biggest. It means that the subgraph cannot be extended to a larger 2-connected subgraph. If a graph G is biconnected, then G itself is called a block.

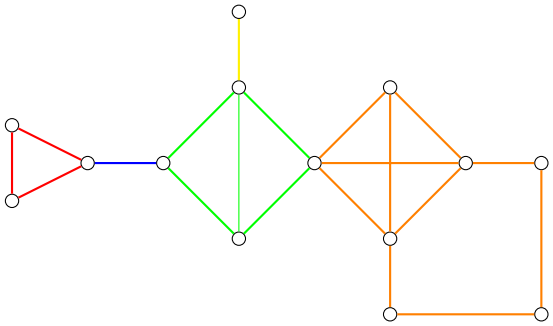


Figure 2.17: A graph, and its blocks. Every block is colored with its own color.

3

SHORTEST PATHS

It is not hard to find applications for shortests paths. Every day people travel and goods are transported, and since we have limited time and money, we need to move as fast and effective as possible. We will look at shortest paths in graphs, and most real-life situations can be transformed into graph theory. One of the most fundamental algorithms for finding shortests paths is Dijkstra's Algorithm, which for a given vertex s, finds the shortest path to all other vertices.

3.1 Dijkstra's Algorithm

Given a connected weighted graph G, where $w(v, u)$ denotes the weight of the edge vu, and a starting vertex s, we want to find the shortest path distance from s to all other nodes in G. Start by assigning all vertices the distance ∞, except s, which gets distance 0. We will denote the distance from s to a vertex v by $l(v)$. The distance assigned to a vertex is the current, best known distance to the vertex from s. At the beginning we say that s has been *visited*, and that no other vertices have been *visited*.

Algorithm 1 Dijkstra's Algorithm

repeat
  Choose a vertex, which is not yet visited, and has the lowest distance assigned.
  Denote this vertex v.
  for all neighbours u of v **do**
    If $l(v) + w(v, u)$ is lower than $l(u)$, assign $l(v) + w(v, u)$ to u.
  Mark v as visited.
until all vertices are visited

In this way, we mark a new vertex in every step. When all vertices are marked visited, the distance $l(v)$ at some vertex v, is the distance of the shortest path from s to v.

Directed Graph If the graph is directed, instead of considering all neighbours of u, only consider those to which you can go directly from u.

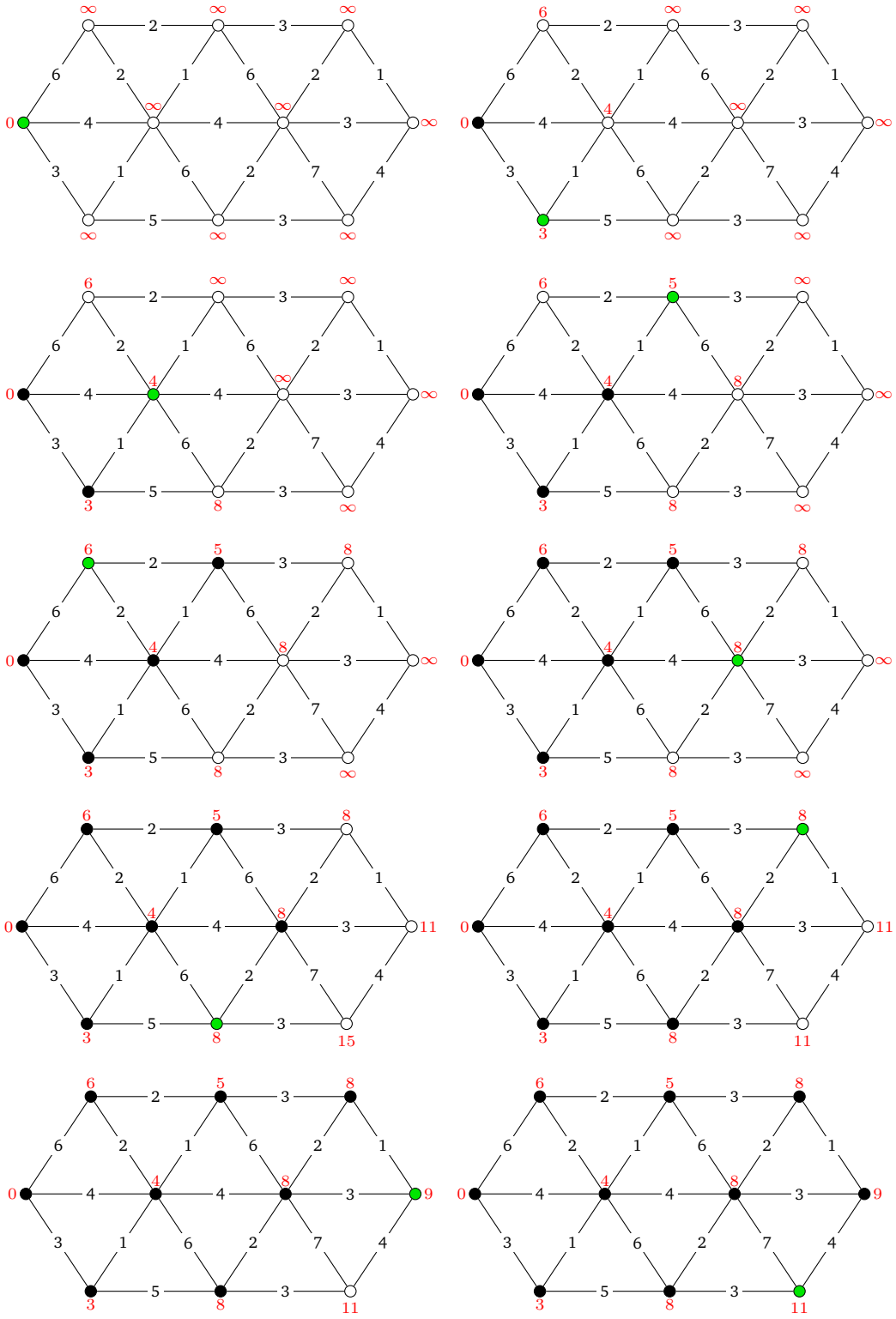


Figure 3.1: An example of Dijkstra's Algorithm run on a graph.

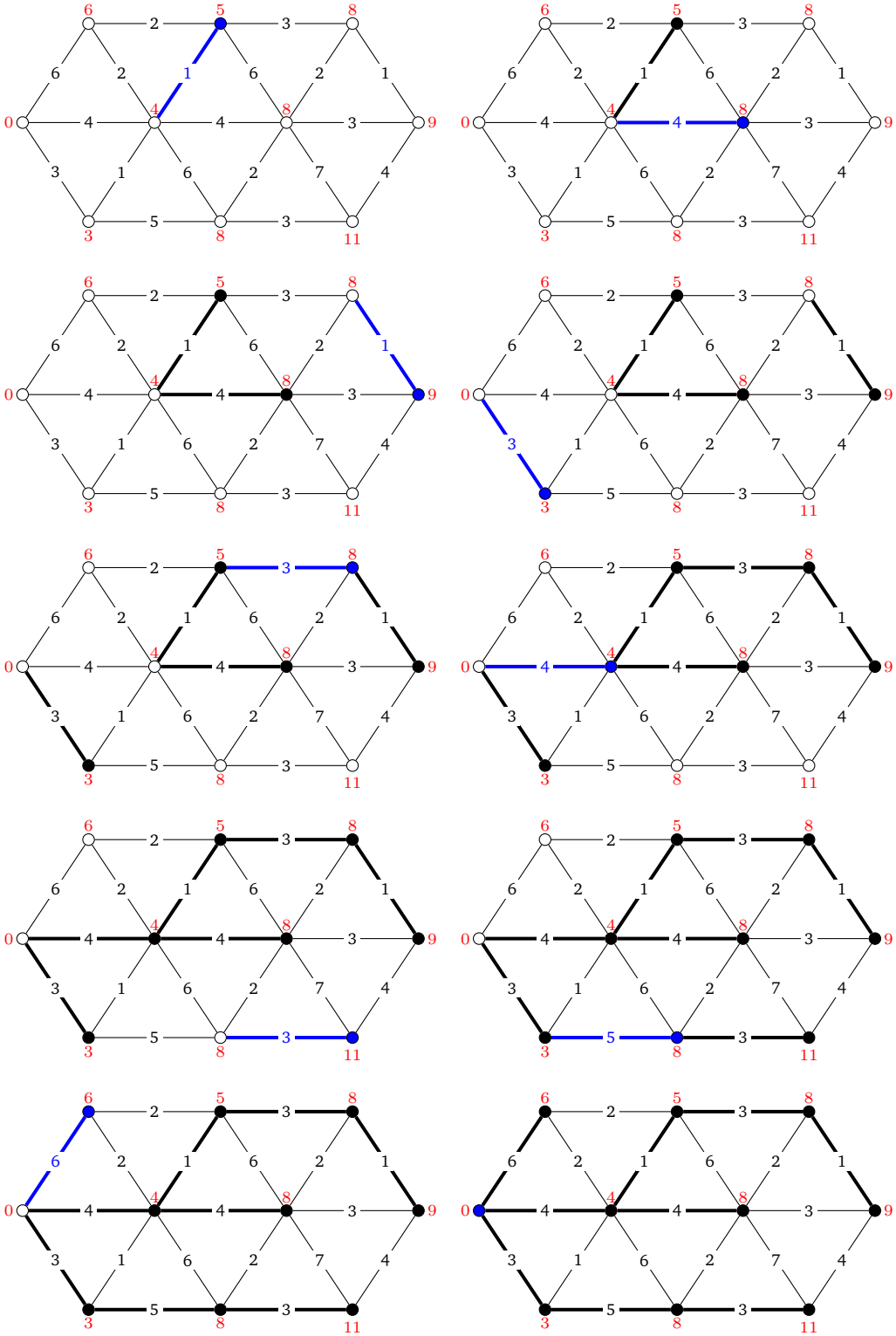
3.1.1 Finding the Actual Paths

When Dijkstra's Algorithm is run on a graph G, with starting vertex s, all vertices get a distance from s, but the actual shortest path is not found. Luckily, a small trick can be used to find the actual shortests paths very fast.

Algorithm 2 Finding the actual paths

for every vertex v **do**
Find a neighbour u of v such that $l(v) - w(v, u) = l(u)$.
Mark the edge between v and u as being part of a shortest path.

This procedure, will mark a spanning tree in G, in this case, a shortest path tree. Now the shortest path from s to a vertex v, is simply to follow the marked path from s to v.



3.2 Number of Shortest Paths

Suppose we do not just want to compute the length of a shortest path between two vertices, but also the number of different shortest paths. For an unweighted graph, this is a simple task. Start by running Dijkstra on the graph from some start node s, to get the distance labels on all nodes

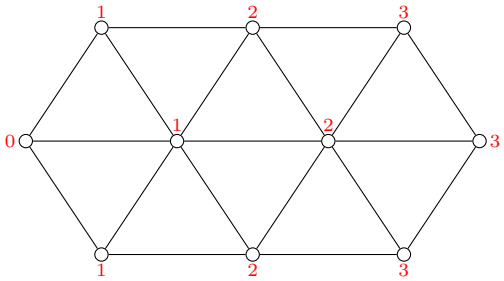


Figure 3.2: The label of node v is $l(v)$.

Now the graph is partitioned into so-called distance classes. A distance class is a set of vertices, with the same distance from the start vertex s. For example, all the vertices labeled 2 in Figure 3.3 is in distance class 2 since they all have distance 2 to s. Now we can find the number of shortest paths from s to a vertex v in the following way. We will let $N(v)$ denote the number of shortest paths from s to v.

Algorithm 3 Finding the number of shortest paths

Look at each distance class D in ascending order
for every vertex v in D **do**
Set $N(v)$ to be the sum of $N(w)$ where w are all the neighbours of v in a lower distance class

This will result in the following labeling

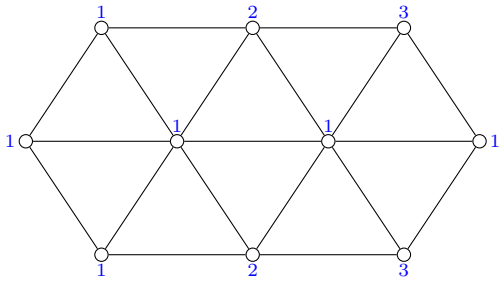


Figure 3.3: The label of vertex v is $N(v)$. Note that a vertex far from s does not always have a large number of shortest paths.

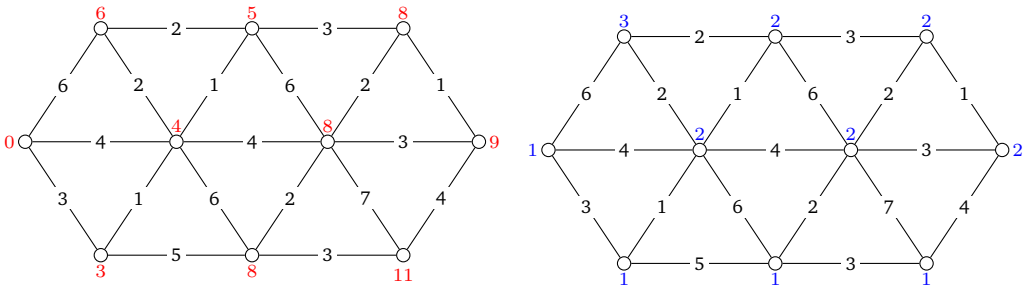
3.2.1 Weighted version

Maybe we want to do the same thing, but for a weighted graph. This can be achieved by changing some details in the algorithm

Algorithm 4 Finding the number of shortest paths

Look at the vertices in the order they are marked *visited* in Dijkstra's algorithm
for every vertex v in D **do**
Set $N(v)$ to be the sum of $N(u)$ where u are all the neighbours of v such that $l(u)+w(v,u) = l(v)$

This looks a lot like the algorithm for computing the distance labels in Dijkstra's Algorithm on a weighted graph.
If we apply this algorithm to the graph in Figure 3.4(a), we get the labeling in Figure 3.4(b).



(a) The distance labeling (b) The number of shortest paths labeling

Figure 3.4: A distance labelling of a weighted graph, and the number of shortest paths for each vertex in the same graph.

4

EULER TOURS AND THE CHINESE POSTMAN PROBLEM

A tour in a connected graph G is a closed walk, visiting every edge in G at least once, and returning to the starting point.

4.1 Euler Tours

An euler tour in a graph, is a tour visiting all edges exactly once. The problem of determining whether or not a graph has an euler tour, was one of the founding problems of Graph Theory considered by Euler. If a graph has an euler tour, the graph is said to be eulerian. Figure 4.1 shows an example of an eulerian graph.

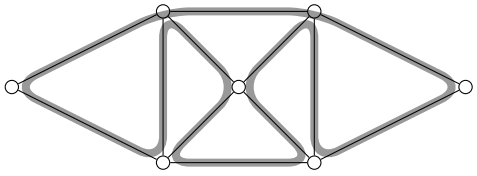


Figure 4.1: An euler tour in a graph (marked in gray).

It is very easy to determine if a graph is eulerian. The following theorem states a necessary and sufficient condition, which is very easy to check.

> **Theorem 4.1** *A graph G is eulerian, if and only if it is connected and all vertices have even degree.*

Sometimes, we might want to start and end in different vertices of the graph. A walk starting in a vertex s, ending in vertex t, and visiting every edge in the graph exactly once, is called an euler trail.

> **Theorem 4.2** *A graph G has an euler trail, if and only if it is connected and exactly two of the vertices in G have odd degree.*

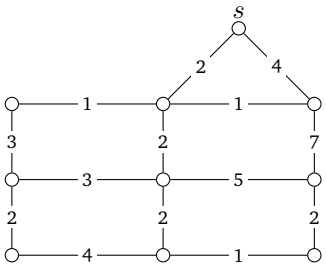
4.2 The Chinese Postman Problem

Consider a Chinese postman, he needs to bring out the mail to every house in a big Chinese city, so he must walk down every street. But to minimize time, he wants to find the shortest tour of the city, visiting every edge at least once. The Chinese Postman Problem is to find such a tour.

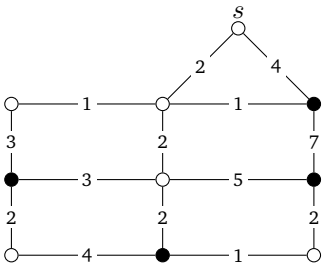
If the city is eulerian, the shortest tour is of course an euler tour, since an euler tour visits every edge exactly once, thus wasting no time. However, if the city is not eulerian, the postman must traverse some edges more than once. When an edge is traversed a second time, we can think of this as doubling the edge, and we call the new edge a *waste edge*. The graph consisting of waste edges is called the waste graph W, and the graph $G + W$ is Eulerian. Hence, by finding a waste graph W of minimum weight, we can solve the Chinese Postman Problem.

4.2.1 An Algorithm for Chinese Postman Problem

Given a connected graph G with weights on the edges, and a starting vertex s, start by marking all the vertices of odd degree. It is a general property that the number of vertices of odd degree is even.



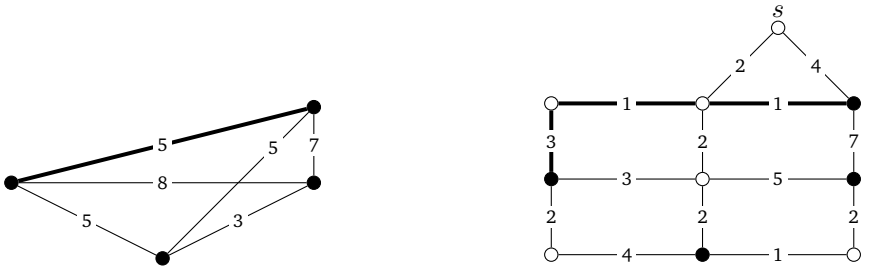
(a) The weighted graph G.



(b) G where all vertices of odd degree are marked.

The reason for finding the vertices of odd degree, is that we want to make the graph eulerian by adding edges, such that the degrees of these vertices become even.

Now create an auxiliary graph consisting of the marked vertices. We will call this graph the *auxiliary Graph* and denote it H. For each pair of vertices (v, w) in H, connect them by an edge where the weight is to the distance between v and w in G. Since G is connected, H will be a complete graph. Figure 4.2 illustrates the auxiliary graph of G.

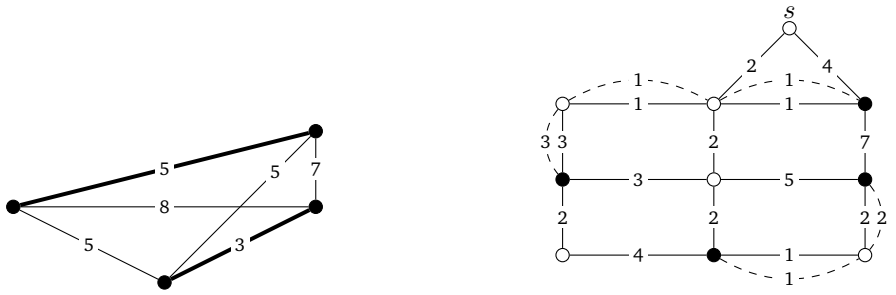


(a) The auxiliary graph H.

(b) The corresponding shortest path in G.

Figure 4.2: An edge in H corresponds to a shortest path in G. Here, an edge in H is marked, and the corresponding path in G is also marked. The union of these graphs is a waste graph of minimum weight.

Find a minimum perfect matching in H, that is, a perfect matching in H of lowest weight-sum. It is always possible to create a perfect matching since H is complete and has an even number of vertices.



(a) A minimum matching in H.

(b) G with the edges from the matching in W inserted.

Figure 4.3: Each edge in the minimum matching corresponds to a shortest path in G. Adding these paths to G, causes all vertices in G to have even degree.

Let M be the minimum perfect matching found in H. As illustrated in Figure 4.3, adding the edges of M to G (or more precisely: The corresponding paths in G), causes G to become eulerian. And because M is minimum one can show that this method produces a waste graph of minimum weight, thus this is the cheapest way to make G eulerian. It is important to remember that the added edges are not in the original graph, so traversing one of them really corresponds to traversing the edge in G twice! This is illustrated in Figure 4.4 that shows a minimum tour in G that traverses each edge at least once.

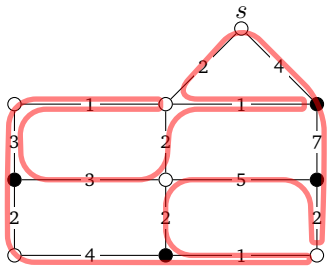


Figure 4.4: A minimum walk in G that traverses each edge at least once. The tour is obtained by finding an euler tour in Figure 4.3(b). Note that there are many such tours.

4.2.2 Starting and Ending in Distinct Vertices

Sometimes, the Chinese Postman needs to end a different place than where he started. So given a graph G, a starting vertex s and an ending vertex t, we want to find a shortest trail from s to t. Fortunately, we can use the algorithm described for the general Chinese Postman problem, with only a few changes. When creating H, we mark vertices in G with odd degree, but now we need to handle s and t in the opposite way. If s or t has odd degree, it should **not** be in W, and if s or t has even degree, it should be in W. When H is created, the rest of the algorithm is the same. The reason is, that we want to ensure that s and t get odd degree, and all other vertices get even degree.

5

SPANNING TREES

A *spanning subgraph* of a graph G is a subgraph H such that $V(H) = V(G)$. A *spanning tree* in a graph $G = (V, E)$ is a spanning subgraph of G, which is a tree. We define a spanning tree formally as follows.

Definition 5.1 *A spanning tree T of a graph $G = (V, E)$ is a graph $T = (V, E')$ such that T is a tree and $E' \subseteq E$.*

Figure 5.1 shows an example of a spanning tree in a graph.

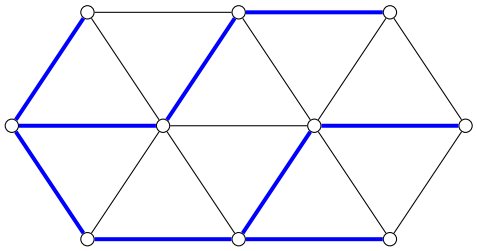


Figure 5.1: A spanning tree (shown in blue) in a graph G.

If G is a weighted graph, T will also be a weighted graph. We define the *total edge-weight* of T as the sum of the edge weights in G.

Definition 5.2 *A minimum spanning tree T of G, is a spanning tree of G, such that no other spanning tree of G has lower total edge-weight.*

Figure 5.2 shows an example of a minimum spanning tree in a graph.

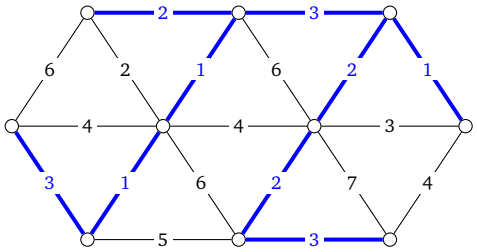


Figure 5.2: A minimum spanning tree (shown in blue) having total edge-weight 18.

There are multiple practical applications of spanning trees, and especially of minimum spanning trees. Finding a minimum spanning tree is not a hard problem, and many algorithms exist for this task. The most commonly used is Kruskal's Algorithm, which we describe in the next section.

5.1 Kruskal's Algorithm

The algorithm proposed by Kruskal is a so-called Greedy Algorithm. Here greedy means that every choice taken by the algorithm, is what seems to be the best at the given time. This approach is often very fast, but rarely works, but luckily it does for finding a minimum spanning tree.

Algorithm 5 Kruskal's Algorithm

Label the edges in non-decreasing order of weight $e_1, \ldots, e_m$
for $i = 1 \ldots m$ **do**
  Mark e_i unless this creates a marked cycle

When this algorithm terminates, the marked edges in G correspond to a minimum spanning tree.

5.1.1 An example of Kruskal's Algorithm

Here we run Kruskal's Algorithm on the same graph as in Figure 5.2, note that the resulting spanning tree is not the same as in Figure 5.2.

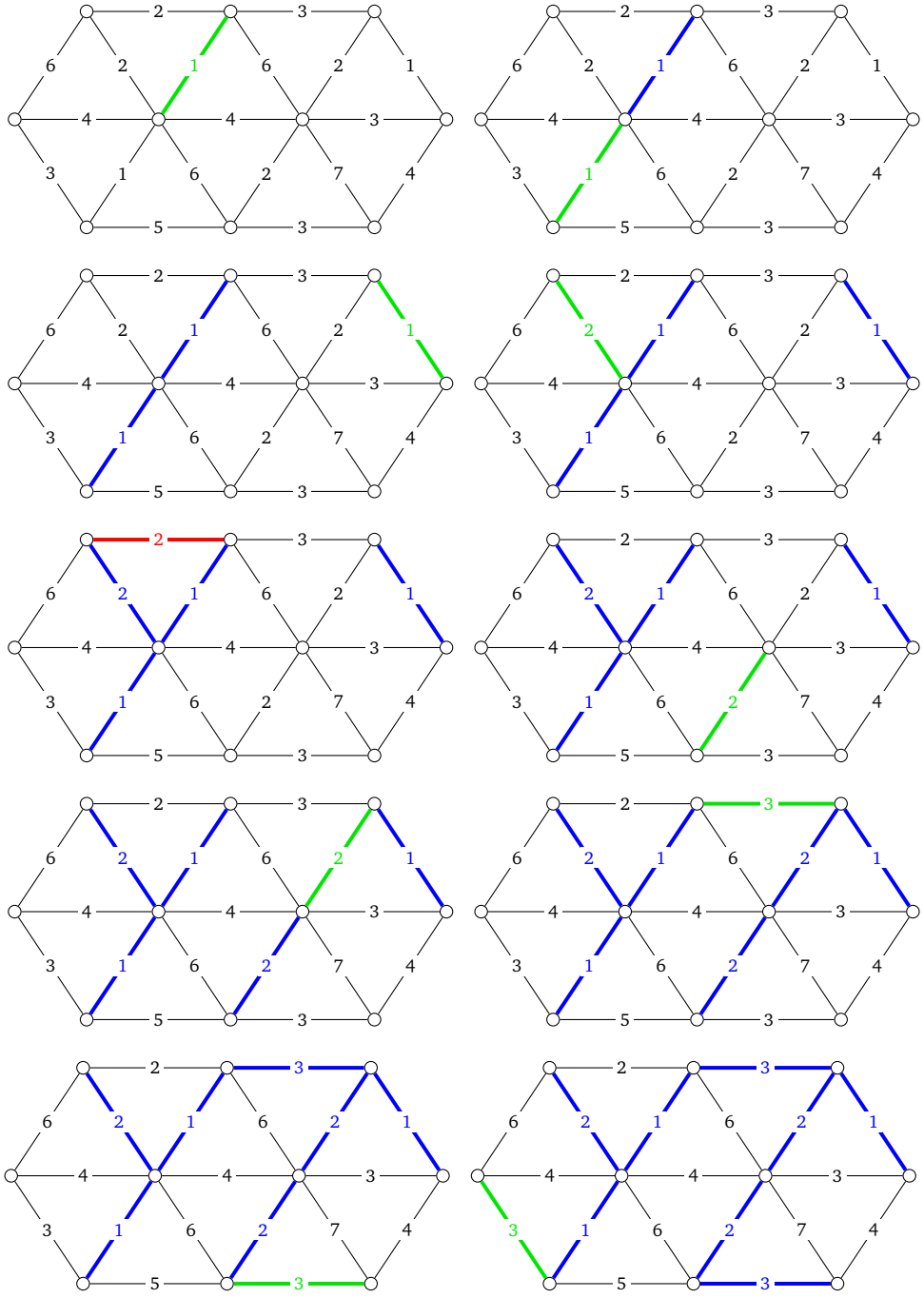


Figure 5.3: Blue edges are edges of the spanning tree. Green edges are the edges just inserted into the spanning tree. Red edges are edges which are not inserted into the spanning tree since they would create a cycle.

5.2 Counting Spanning Trees

The number of different spanning trees in a graph G turns out to be a very useful number with many applications. We define this number as $\tau(G)$ and in this section, we will investigate different methods for calculating this number. If G is itself a tree, we have that $\tau(G) = 1$.

5.2.1 Contraction-Deletion

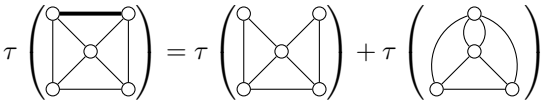
One way to compute the number of spanning trees, is to recursively degenerate the graph by deleting and contracting edges. In this way, the number of spanning trees of an unknown graph G is reduced to a sum of the number of spanning trees in small graphs which we know. The formula for counting spanning trees is the following

Theorem 5.1 *In any graph G,*

$$\tau(G) = \tau(G/e) + \tau(G - e)$$

where e is an arbitrary edge in G. G/e denotes the graph obtained by contracting e and $G - e$ is the graph obtained by deleting e.

Applying the Contraction-Deletion theorem can be done as follows.



The approach is to take a graph G and then recursively remove and contract edges until you get a lot of small graphs. Appendix D contains a catalog of spanning tree numbers for some small graphs, and can be useful when using the Contraction-Deletion theorem.
Theorem 5.1 is due to the following observation about the number of spanning trees containing or avoiding an edge

Theorem 5.2 *The number of spanning trees in a graph G containing an edge e is equal to the number of spanning trees in G/e. The number of spanning trees in a graph G* ***not*** *containing an edge e is equal to the number of spanning trees in $G - e$.*

5.2.2 Complete Graphs

There is a general counting formula for the number spanning trees in the complete graph K_n. Counting the number of spanning trees in K_n corresponds to counting the number of labeled trees on n vertices.

Theorem 5.3 (Cayley's Formula) $\tau(K_n) = n^{n-2}$

There are also formulas for the number of spanning trees in a complete graph on n vertices, where an arbitrary edge has been removed or contracted.

Theorem 5.4 *For any edge in e in a complete graph K_n,*

$$\tau(K_n - e) = (n-2)n^{n-3} \qquad \textit{and} \qquad \tau(K_n/e) = 2n^{n-3}$$

Let K_n and K_m denote two complete graphs. Then $K_n \ominus K_m$ denotes the graph obtained by joining K_n and K_m by identifying an edge from K_n with an edge from K_m, and then removing that edge. Likewise, we define $K_n \oplus K_m$ as the graph obtained by the same procedure, but where the shared edge is not deleted. Figure 5.4 shows an example of graphs obtained by this construction.

(a) $K_4 \ominus K_5$ (b) $K_4 \oplus K_5$

Figure 5.4: Illustrating the constructions $K_4 \ominus K_5$ and $K_4 \oplus K_5$.

Counting the spanning trees in $K_n \ominus K_m$ and $K_n \oplus K_m$ is not that difficult, since we can partition the spanning trees into groups, that we can count using what we already know. This results in the following theorem.

Theorem 5.5

$$\tau(K_n \ominus K_m) = \tau(K_n - e)\tau(K_m/e) + \tau(K_n/e)\tau(K_m - e) \quad = 2(n+m-4)n^{n-3}m^{m-3}$$
$$\tau(K_n \oplus K_m) = \tau(K_n/e)\tau(K_m/e) + \tau(K_n \ominus K_m) \quad = 2(n+m-2)n^{n-3}m^{m-3}$$

Complete Bipartite Graphs

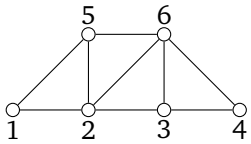
We can also count the number of spanning trees in a complete bipartite graph $K_{n,m}$.

Theorem 5.6 (Scoin's formula) $\tau(K_{n,m}) = n^{m-1}m^{n-1}$

5.2.3 The Matrix-Tree Theorem

Another way to compute the number of spanning trees in a graph is using linear algebra. It has been shown that the number of spanning trees can be computed by finding the determinant of a special matrix called the Laplacian matrix or conductance matrix.

Degree Matrix The degree matrix D of a graph, is a diagonal matrix with the vertex degrees in the diagonal. The graph



has the degree matrix

$$\begin{bmatrix} 2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 4 & 0 & 0 & 0 & 0 \\ 0 & 0 & 3 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 & 0 & 4 \end{bmatrix}$$

Adjacency Matrix The adjacency matrix A of a graph, is a $(0,1)$-matrix, where 1's represent adjacent vertices. A has a row for each vertex, and a column for each vertex. If a vertex v is connected to a vertex u, the entry in row v, column u is a one, and so is the entry in row u, column v. The same graph as before has adjacency matrix:

$$\begin{bmatrix} 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 & 1 & 0 \end{bmatrix}$$

Laplacian Matrix The Laplacian Matrix Q of a graph G is the difference between its degree matrix D and its adjacency matrix A.

$$Q = D - A$$

Which for the above example gives

$$Q = \begin{bmatrix} 2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 4 & 0 & 0 & 0 & 0 \\ 0 & 0 & 3 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 & 0 & 4 \end{bmatrix} - \begin{bmatrix} 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 2 & -1 & 0 & 0 & -1 & 0 \\ -1 & 4 & -1 & 0 & -1 & -1 \\ 0 & -1 & 3 & -1 & 0 & -1 \\ 0 & 0 & -1 & 2 & 0 & -1 \\ -1 & -1 & 0 & 0 & 3 & -1 \\ 0 & -1 & -1 & -1 & -1 & 4 \end{bmatrix}$$

Theorem 5.7 (Kirchoff's Matrix-Tree Theorem) *The number of spanning trees in a simple graph G, is the absolute value of any cofactor of the Laplacian Matrix of G.*

A cofactor of Q can be obtained by removing an arbitrary row, and an arbitrary column from Q and then computing the determinant. In the example above, we could find a cofactor could be by removing the last row and column,

$$\begin{bmatrix} 2 & -1 & 0 & 0 & -1 \\ -1 & 4 & -1 & 0 & -1 \\ 0 & -1 & 3 & -1 & 0 \\ 0 & 0 & -1 & 2 & 0 \\ -1 & -1 & 0 & 0 & 3 \end{bmatrix}$$

and then taking the determinant

$$\tau(G) = \det\left(\begin{bmatrix} 2 & -1 & 0 & 0 & -1 \\ -1 & 4 & -1 & 0 & -1 \\ 0 & -1 & 3 & -1 & 0 \\ 0 & 0 & -1 & 2 & 0 \\ -1 & -1 & 0 & 0 & 3 \end{bmatrix}\right) = 55$$

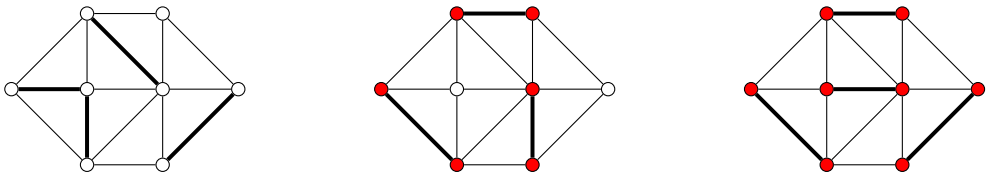
6

MATCHINGS AND COVERINGS

Matchings and coverings are fundamental and important concepts in graph theory with many applications. In this chapter we introduce the basic definitions and study some applications of matchings and coverings.

6.1 Basic Definitions

Matching A *matching* in a graph G is a set of edges $M \subseteq E(G)$, such that no two edges in M are incident to the same vertex. The size of a matching is the number of edges in the matching. We say that M *saturates* a vertex v in G if one of the edges in M is incident to v. Otherwise v is unsaturated by M. These definitions are illustrated in Figure 6.1.



(a) Not a matching, since two of the edges are incident to the same vertex.

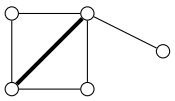
(b) A matching of size 3 saturating the 6 red vertices.

(c) A perfect matching saturating all vertices.

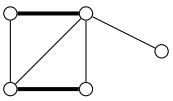
Figure 6.1: Illustrating the concept of matchings in a graph.

Perfect Matching A matching is called *perfect* if it saturates all vertices in G. A necessary (but not sufficient) condition for a graph to have a perfect matching, is that the number of vertices is even. Figure 6.1(c) shows an example of a perfect matching.

Maximal and Maximum Matchings If a matching M can not be extended by adding more edges from G to M, the matching is said to be *maximal*. Figure 6.2(a) shows an example of a maximal matching. A *maximum* matching is a matching with as many edges as possible. Figure 6.2(b) shows a maximum matching.



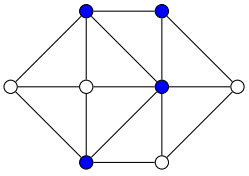
(a) A maximal matching.



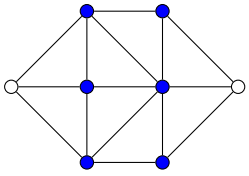
(b) A maximum matching.

Figure 6.2: Illustrating maximal and maximum matchings.

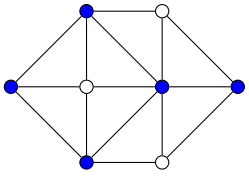
Covering A *covering* in G is a set of vertices $C \subseteq V(G)$, such that every edge in G is adjacent to at least one vertex from C. We say that C covers all the edges of G, so removing the vertices C from G will leave a graph with no edges. The size of a covering C is the number of vertices in C. Figure 6.3 illustrates the definition of coverings.



(a) Not a covering, since some edges are uncovered.



(b) A covering of size 6.



(c) A minimum covering.

Figure 6.3: Illustrating the concept of coverings in a graph.

Matchings and coverings are related by the following lemma.

Lemma 6.1 *Let M be a matching and C a covering in a graph G. Then the number of vertices in C is at least as big as the number of edges in M. That is,*

$$e(M) \leq v(C)$$

This is because C must include at least one of the end-vertices x or y for every edge xy in the matching, otherwise C would not cover the edge xy.

The above definitions give rise to two important problems. Given a graph G,

1. How do we find a maximum matching M^* in G?
2. How do we find a minimum covering C^* in G?

The first problem turns out to be easy, but the second is very difficult (an NP-hard problem). Notice that if we can find a matching M and a covering C of the same size, then it follows from Lemma 6.1 that M is a maximum matching and C is a minimum covering. This is a very useful way to prove the optimality of a matching and a covering: Just highlight them in the graph and observe they have the same size! See Figure 6.4 for an example.

(a) A graph G.

(b) A matching and a covering in G of size 3.

Figure 6.4: Proving the optimality of a matching and a covering in a graph. The the matching M (shown in bold) and the covering C (shown in blue) have the same size, so by Lemma 6.1 M is a maximum matching and C a minimum covering in G.

Unfortunately some graphs do not have a matching and a covering of the same size. This is for instance the case for the graph in Figure 6.1. This graph has a maximum matching of size 4 (see Figure 6.1(c)), but the minimum covering is of size 5 (see Figure 6.3(c)). It is easy to see that the matching is maximum, since it is perfect, but it takes some effort to verify that the covering is minimum, since one has to check that no covering of size 4 exists.
In the next section we will investigate matchings and coverings in bipartite graphs and we need not to be concerned about this issue, as it turns out that these graphs always have a minimum covering of size equal to that of a maximum matching.

6.2 Matchings in Bipartite Graphs

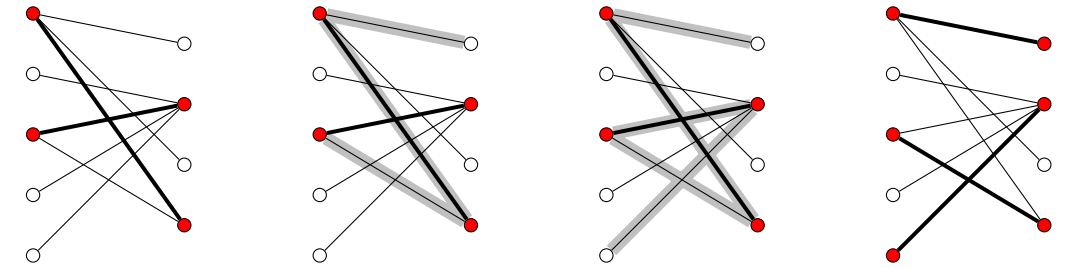
Recall that a graph G is bipartite if the vertices of G can be partitioned into two sets A and B, such that all edges are between the two sets. As mentioned in the previous section, the following important theorem holds for bipartite graphs.

Theorem 6.1 (Kőnig-Egerváry Theorem) *In a bipartite graph G the number of edges in a maximum matching M^* is equal to the number of vertices in a minimum covering C^*.*

We will now consider how to solve the two problems stated in the previous section for bipartite graphs. If G is bipartite, how can we find a maximum matching and a minimum covering in G? As previously stated, finding a minimum covering is a difficult problem in general, but due to Theorem 6.1 there is an algorithm that finds a maximum matching and a minimum covering at the same time. This algorithm is commonly known as the Hungarian Algorithm (or Method) due to the two Hungarian mathematicians Dénes Kőnig and Jenő Egerváry who independently discovered Theorem 6.1 in 1931.

6.2.1 The Hungarian Algorithm

Before introducing the algorithm, we will need to define the notion of M-alternating and M-augmenting paths in G. Given a matching M in G, an *Alternating path* is a path in G consisting of edges alternately in M and not in M. An *M-alternating path* is an M-alternating path in which both end-points are unsaturated by M. Figure 6.5 illustrates these definitions.



(a) A matching M in a bipartite graph G.
(b) An M-alternating path in G.
(c) An M-augmenting path in G.
(d) A bigger matching in G.

Figure 6.5: Illustrating the concept of M-alternating and M-augmenting paths in a bipartite graph G.

Having found an M-augmenting path such as the one shown in Figure 6.5(c), we can use it to augment M as follows: Simply swap the matched and the unmatched edges on the path. This operation will increase M by one edge as illustrated in Figure 6.5(d). M-augmenting paths can be used to increase the matching in arbitrary graphs – not only bipartite ones.
The following theorem by Claude Berge states a useful connection between maximum matchings and augmenting paths.

Theorem 6.2 (Berge's Theorem) *A matching M in a graph G is maximum if and only if G contains no M-augmenting path.*

Notice that the theorem is not restricted to bipartite graphs – it is valid for arbitrary graphs. The theorem suggests a natural algorithm for finding a maximum matching: Start with any matching M in G. If M is not maximum, then there is an M-augmenting path in G, find it and use it to increase the matching. Repeat until there is no M-augmenting path, and then M is a maximum matching. This is the Hungarian Algorithm.

Algorithm 6 The Hungarian Algorithm

Find a maximal matching M in G.
Repeat: Find an M-augmenting path in G and increase the matching.

The algorithm also works for non-bipartite graphs, but in that case finding an M-augmenting path is quite complicated. We therefore restrict our attention to bipartite graphs, since in that case finding an M-augmenting path can be done using a simple marking procedure described in the following section.

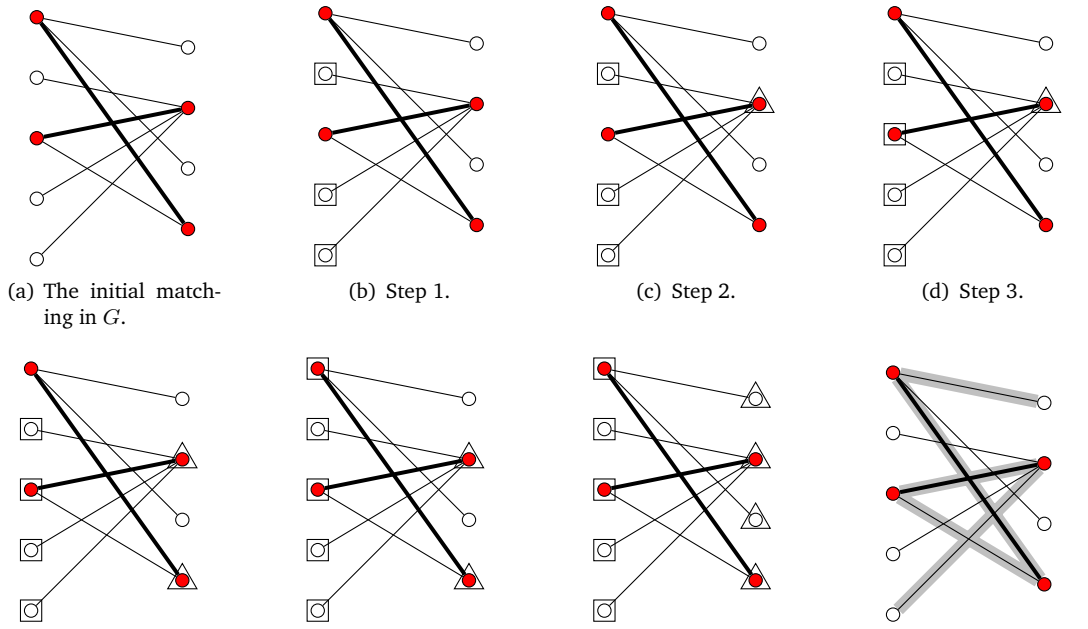
Finding M-augmenting Paths in Bipartite Graphs

To find an M-augmenting path in a bipartite graph $G = (A \cup B, E)$ we use a marking procedure. This procedure will either produce an M-augmenting path or terminate with a covering of the same size as M. If the latter happens, we can conclude (by Lemma 6.1) that M is a maximum matching.
The marking procedure is performed as follows: Start by marking all unsaturated vertices in A as squares. Then continue expanding the marking using these two rules repeatedly.

1. In every square: Follow all edges to B and mark the neighbours as triangles.
2. In every triangle: Follow the matched edge to A and mark the neighbour as a square.

If at some point an unsaturated vertex in B is marked as a triangle, we have found an M-augmenting path ending in that vertex, and we stop the marking procedure. If the marking procedure stops without having marked an unsaturated vertex in B, then this is because no M-augmenting path exists in G, and hence M is a maximum matching. In that case, the unmarked vertices in A and the marked vertices in B constitute a covering in G of the same size as M.

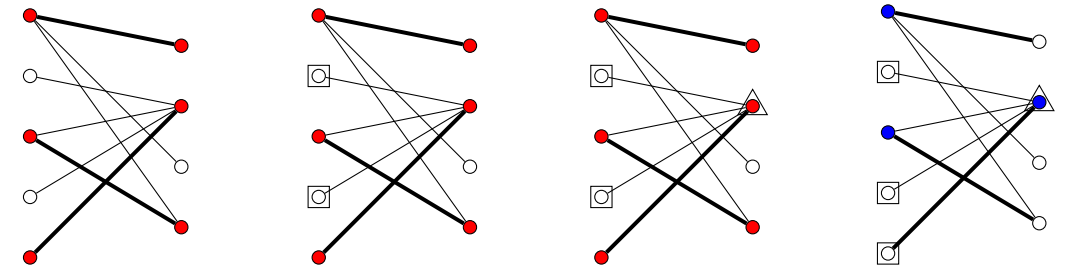
The marking procedure is illustrated in Figure 6.6 and Figure 6.7. The first figure shows how the marking procedure can used on Figure 6.5(a) to find the M-augmenting path shown in Figure 6.5(c). The second figure illustrates how the marking procedure is performed on the increased matching, this time not finding a M-augmenting path, but instead yielding a minimum covering.



(a) The initial matching in G. (b) Step 1. (c) Step 2. (d) Step 3.

(e) Step 4. (f) Step 5. (g) Step 6. (h) An M-augmenting path in G.

Figure 6.6: Performing the marking procedure on the matching shown in Figure 6.5(a). The marking procedure stops after six steps having marked both unsaturated vertices on the right as triangles. By backtracking we get the M-augmenting path shown in Figure 6.6(h). Notice that there are many other M-augmenting paths.



(a) The new matching in G.

(b) Step 1.

(c) Step 2.

(d) Step 3 and a minimum covering in G.

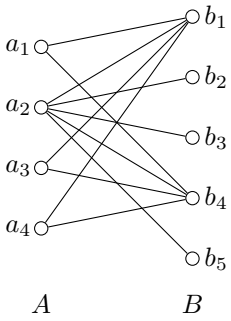
Figure 6.7: Performing the marking procedure on the new matching obtained from the M-augmenting path in Figure 6.6(h). The marking procedure stops after three steps, but no unsaturated vertex on the right has been marked, so M is a maximum matching. The unmarked vertices in A and the marked vertices in B constitute a covering of size 3 (shown in blue).

6.2.2 Perfect Matchings in Bipartite Graphs

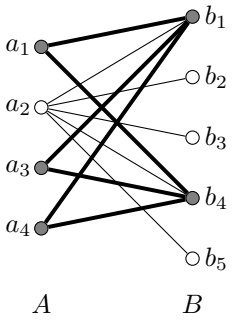
Consider a bipartite graph $G = (A \cup B, E)$ such as the one shown in Figure 6.8(a). Sometimes we are interested in finding a matching that saturates all vertices in A, but this is not always possible. A necessary and sufficient condition for the existence of such a matching is given by Hall's Theorem.

Theorem 6.3 (Hall's Theorem) *A bipartite graph $G = (A \cup B, E)$ has a matching that saturates all vertices in A if and only if every subset S of A has at least $|S|$ neighbours in B.*

Figure 6.8(b) shows how to use Hall's Theorem to conclude that no matching saturating all vertices A exists in a graph.



(a) Does this bipartite graph have a matching that saturates all vertices in A?



(b) No, because the set $S = \{a_1, a_3, a_4\}$ has only 2 neighbours in B (b_1 and b_4).

From Hall's Theorem it is possible to derive the following two corollaries about perfect matchings in bipartite graphs. The first gives a necessary and sufficient condition for a bipartite graph to have a perfect matching. The later only gives a sufficient condition.

Corollary 6.1 *A bipartite graph $G = (A \cup B, E)$ has a perfect matching if and only if $|A| = |B|$ and every subset S of A has at least $|S|$ neighbours in B.*

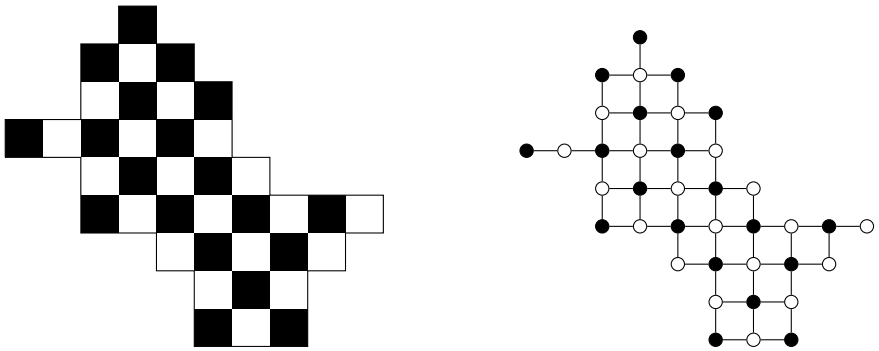
Corollary 6.2 *If G is k-regular (all vertices have degree k) and bipartite, then G has a perfect matching.*

6.3 Applications

In this section we study some of the applications of matchings and coverings in bipartite graphs.

6.3.1 Carving Out Dominoes

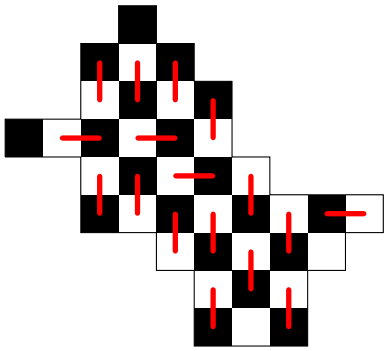
Suppose you have a wooden board tiled in black and white as shown in Figure 6.8(a). Your task is to cut the board into as many domino pieces as possible (this is a domino piece: ▯■).



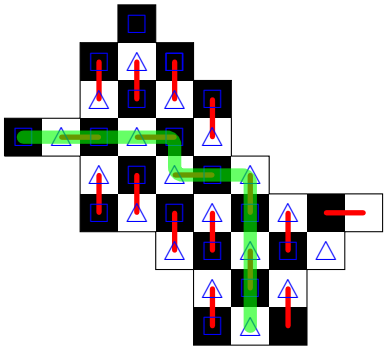
(a) A wooden board. (b) The underlying bipartite graph G.

Figure 6.8: A wooden board and the underlying bipartite graph.

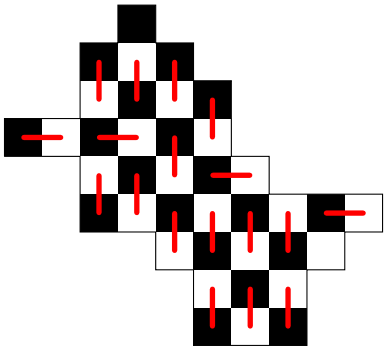
The wooden board can be seen as a bipartite graph G as shown in Figure 6.8(b). Solving the problem of cutting out as many dominoes as possible, corresponds to finding a maximum matching in G. To do this, we use the Hungarian Algorithm. First we find a matching and then use the marking procedure to increase it, until we have a maximum matching. When executing the algorithm, we will use the board representation shown in Figure 6.8(a), but bear in mind that we are actually working on the underlying bipartite graph. Figure 6.9 illustrates the execution of the algorithm and shows that at most 18 domino pieces can be carved from the wooden board.



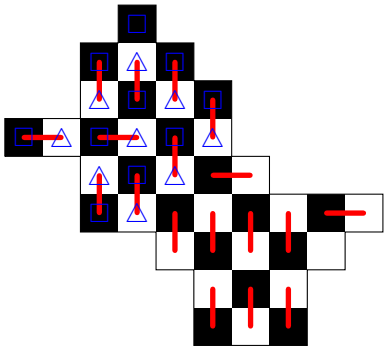
(a) A matching in G.



(b) Finding an an augmenting path using the marking procedure.



(c) Using the augmenting path to increase the matching.

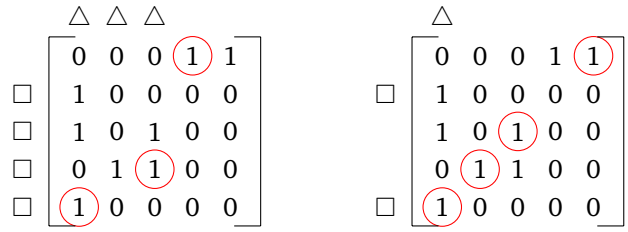


(d) Performing the marking procedure to see that no augmenting path can be found. Hence this matching is maximum.

Figure 6.9: Using the Hungarian Algorithm to find a maximum matching in G.

6.3.2 Structure Rank

The *structure rank* of a $(0,1)$-matrix A is the maximum number of 1's that can be selected in A such that none of the selected entries are in the same row or column. This definition of structure rank is equivalent to the definition of a maximum matching in the bipartite graph represented by A in the following way: The i'th row corresponds to a vertex a_i and the j'th column corresponds to a vertex b_j, and there is an edge between vertex a_i and b_j if there is a 1 in $A[i,j]$. Figure 6.10 shows an example of this definition.



(a) A selection of 1's which is not largest possible.

(b) A largest possible selection of entries. This indicates that the structure rank of the matrix is 4.

(c) The bipartite graph correspondng to the matrix. A maximum matching is emphasized.

Figure 6.10: Here two different selections of 1-elements from a matrix is shown. The elements are selected in such a way, that no two elements share a row or a column. The selection of entries on the right is as large as possible, showing that the structure rank of the matrix is 4. The marking process described in Section 6.2.1 has been performed on the matrices to show that the left selection can be extended, and that the right selection can not.

Another way to think of the structure rank, is the following. We define a *line of a matrix* as a general term for a row or a column. Now the structure rank is equal to the minimum number of lines needed to contain all 1's of a matrix. Such a set of lines, corresponds to a covering of the bipartite graph corresponding to the matrix. This implies that the unmarked rows, together with the marked colums form such a covering. In Figure 6.11 such a covering is shown on the same matrix as in Figure 6.10. Such covering of elements in a matrix, is equivalent to an edge covering in the corresponding bipartite graph.

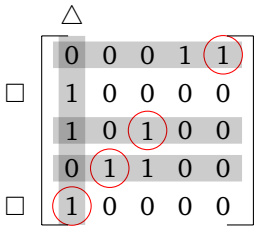


Figure 6.11: When the selection of 1's in the matrix is largest possible, the unmarked rows together with the marked colums form a covering of the 1's in the matrix. The number of lines in such a covering is equal to the structure rank of the matrix, thus in this case 4.

One of the reasons to consider structure rank directly on matrices instead of working with the corresponding bipartite graph, is that the matrix is easier to work with when the number of vertices grows large. After performing the marking process on the matrix, it is also possible to find an augmenting path in the corresponding graph by only considering the matrix entries, but this is a rather tedious job.

6.3.3 The Job Assignment Problem

Consider n workers and n jobs. Every worker has a skill level for each job, and we want to assign the workers to the jobs so the overall skill level is as good as possible. We can formulate this as a matching problem. You have a weighted bipartite graph G (see Figure 6.12) and want to find a matching in G that maximizes the sum of the weight of all edges in the matching.

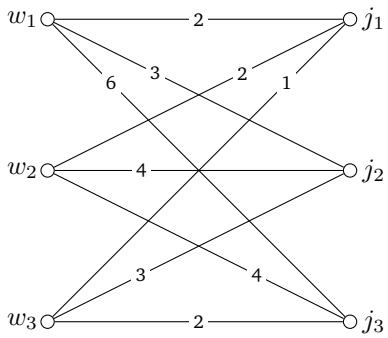


Figure 6.12: An instance of the Job Assignment problem shown as a bipartite graph G.

An Algorithm for Job Assignment

Create an Assignment Matrix We create a matrix where every row corresponds to a worker, and every column corresponds to a job. In every cell, we put the efficiency of that worker on that job.

	j_1	j_2	j_3
w_1	2	3	6
w_2	2	4	4
w_3	1	3	2

Initialize Auxiliary Numbers We give every row and every column an auxiliary number. These numbers will be a tool when we find the maximal matching. We start by giving every column the auxiliary number 0 and every row the auxiliary number equal to the largest number in the row as shown in Table 6.1(a).

	j_1	j_2	j_3	
w_1	2	3	6	6
w_2	2	4	4	4
w_3	1	3	2	3
	0	0	0	

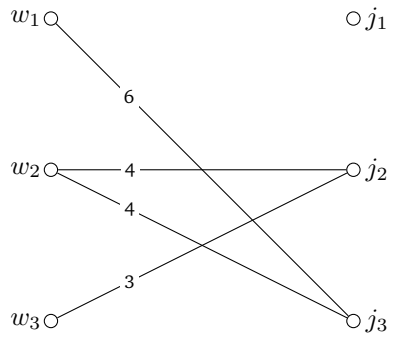
(a) Initializing the auxiliary numbers.

	j_1	j_2	j_3	
w_1	2	3	$\underline{6}$	6
w_2	2	$\underline{4}$	$\underline{4}$	4
w_3	1	$\underline{3}$	2	3
	0	0	0	

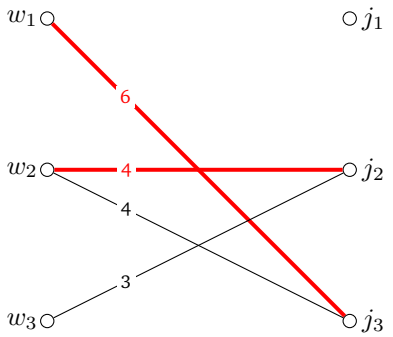
(b) The cells corresponding to edges in the equality graph L.

Table 6.1: Initializing the auxiliary numbers and identifying the edges in the equality graph.

Create Equality Graph We then create a bipartite graph L (called the equality graph) with the workers to the left and the jobs to the right (see Figure 6.13(a)). We create an edge between a worker-node w_i and a job-node j_j if and only if the number in cell (i, j) (the efficiency of worker i on job j) is equal to the sum of the auxiliary numbers in the corresponding row and column.
In Table 6.1(b) the underlined numbers are the ones corresponding to the edges in equality graph. Notice that each underlined number is the sum of the auxiliary numbers in the same column and row.



(a) The equality graph L.



(b) A maximal matching in the equality graph

Figure 6.13: Creating the equality graph and finding a maximal matching in it.

Find a Matching in L We now find a maximal matching in L (see Figure 6.13(b)). We try to create a matching with high weights since this will make the algorithm terminate faster.
As long as possible, extend the matching by using augmenting paths. At some point, the matching is maximal (can not be increased).

Perform the Marking Procedure Perform the marking procedure that we previously used to find an M-augmenting path. We state the procedure on the context of workers and jobs. Start by marking all unmatched workers-nodes with squares. Then continue expanding the marking using these two rules.

1. In every square: Follow all edges to the right and mark the neighbours as triangles.
2. In every triangle: Follow the matched edge to the left and mark the neighbour as a square.

This stops when none of the rules can be applied. The resulting graph is shown in Figure 6.14.

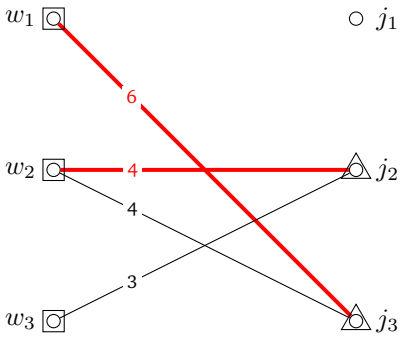


Figure 6.14: The final marking, when the marking process is done on Figure 6.13(b). Initially w_3 is marked as a square.

Update Auxiliary Numbers on Marked Vertices We now change the auxiliary numbers by $\varepsilon \in \mathbb{N}$. If ε is too large, edges will be lost and the matching will become smaller. The optimal value of ε can be found by considering all numbers in unmarked columns and all numbers

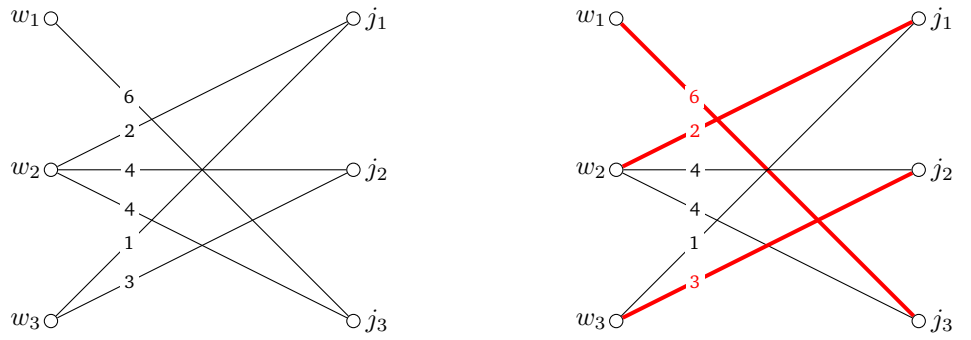
in marked rows (i.e. the bold numbers in the left table below). For each of these numbers x we add the corresponding row- and column auxiliary numbers and subtract x from that. The smallest difference we get is then our value for ε. When ε is found, we change the auxiliary numbers. For every row corresponding to a marked vertex, we lower the auxiliary number by ε, and for every column corresponding to a marked vertex, we add ε to the auxiliary number.
We look at the bold numbers to find ε to be equal to 2 since the smallest difference is $(3+0)-1 = 2$ and $(4+0)-2 = 2$.

			△	△	
		j_1	j_2	j_3	
□	w_1	**2**	3	6	6↓
□	w_2	**2**	4	4	4↓
□	w_3	**1**	3	2	3↓
		0	0↑	0↑	

$\xrightarrow{\varepsilon=2}$

	j_1	j_2	j_3	
w_1	2	3	$\underline{6}$	4
w_2	$\underline{2}$	$\underline{4}$	$\underline{4}$	2
w_3	$\underline{1}$	$\underline{3}$	2	1
	0	2	2	

Repeat Until the Equality Graph Has a Perfect Matching With the new auxiliary numbers in the last table, we create a new equality graph (see Figure 6.15(a)) and repeat the procedure. We continue to find matchings in the equality graph and change the auxiliary numbers until the matching we find in L is perfect.



(a) The new equality graph L.

(b) A maximal matching in the equality graph

Figure 6.15: Creating the new equality graph and finding a maximal matching in it.

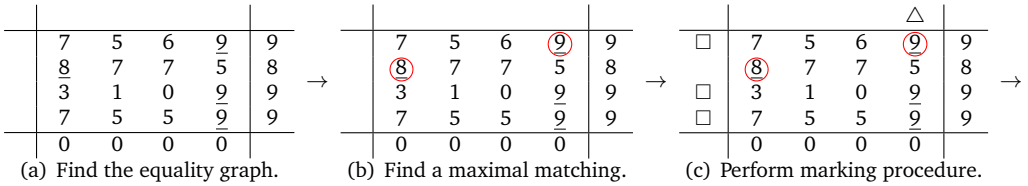
The matching in Figure 6.15(b) is perfect, which means that we are done. The vertices matched together represent an optimal assignment of a workers to jobs. If L had no perfect matching, we should repeat the marking procedure, find a new ε-value and update the auxiliary numbers.

Piece of Advice When solving this kind of exercise, there is no need to construct the equality graphs explicitly. Instead just construct the tables and underline the cells corresponding to edges in the equality graph. Then find a maximal matching, perform the marking process in the table itself and finally find ε. Doing this enables you to solve these exercises much faster. For example consider the following job assignment problem.

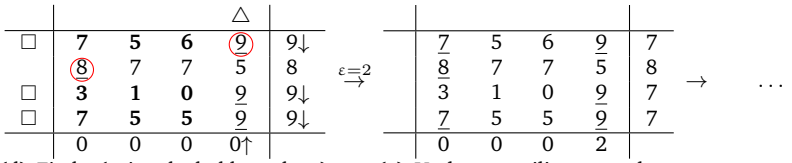
	j_1	j_2	j_3	j_4
w_1	7	5	6	9
w_2	8	7	7	5
w_3	3	1	0	9
w_4	7	5	5	9

Table 6.2: An instance of the job assignment problem.

Solving this instance by table manipulations only would begin by the steps indicated in Table 6.3.



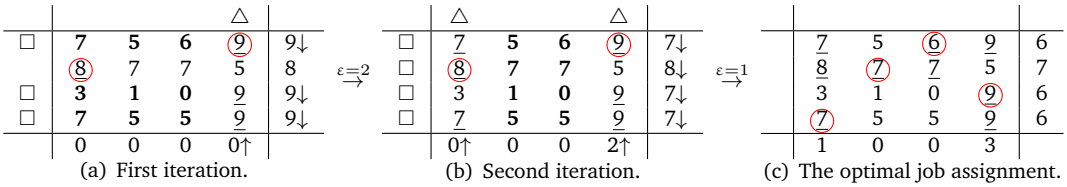
(a) Find the equality graph. (b) Find a maximal matching. (c) Perform marking procedure.



(d) Find ε (using the bold numbers). (e) Update auxiliary numbers. Find the new equality graph.

Table 6.3: The first iteration in solving the job assignment problem shown in Table 6.2.

By performing the above steps in the same table, a complete solution including all intermediate calculations can be represented using only the three tables shown in Table 6.4. The total cost of the optimal job assignment is $6 + 7 + 9 + 7 = 29$.



(a) First iteration. (b) Second iteration. (c) The optimal job assignment.

Table 6.4: Complete solution for the job assignment problem shown in Table 6.2.

7

BENZENOIDS

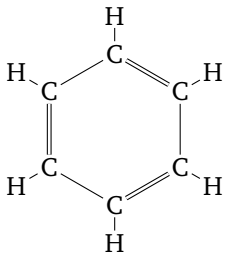
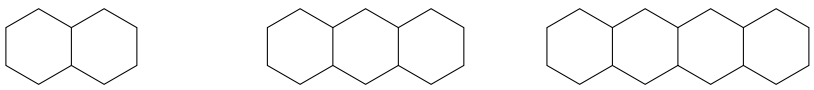


Figure 7.1: Structural formula for Benzene C_6H_6.

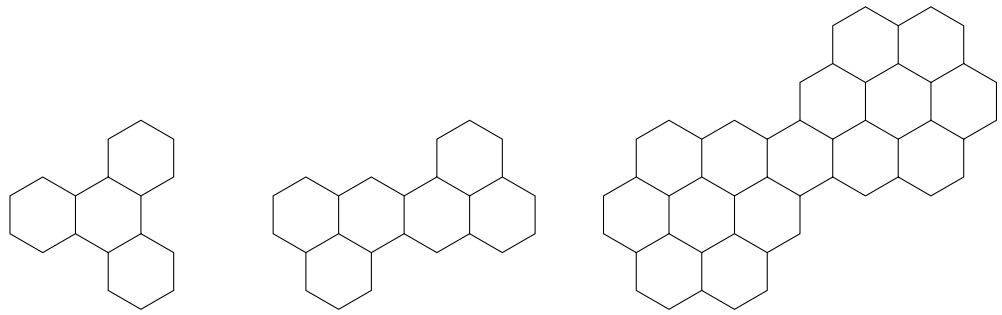
Benzene or C_6H_6 is a well-studied organic chemical compound with a hexagonal structure as shown in Figure 7.1. The bond order between two atoms is the number of chemical bonds between them. It was first believed that the bonds in the Benzene ring were alternating single and double bonds as depicted in the figure above. This turned out to be false. The bonds between the carbon atoms are all the same and have bond order 1.5.
Benzenoids are compounds consisting of one or more Benzene rings joined together. Chemists also refer to benzenoids as polycyclic aromatic hydrocarbons. Besides Benzene some simple benzenoids are shown in Figure 7.2.



(a) Naphthalene $C_{10}H_8$ (b) Anthracene $C_{14}H_{10}$ (c) Tetracene $C_{18}H_{12}$

Figure 7.2: Some simple benzenoids.

Benzenoids can also have more complicated non-linear structures as the ones shown in Figure 7.3.



(a) Triphenylene $C_{18}H_{12}$ (b) Zethrene $C_{24}H_{14}$ (c) Dicoronylene $C_{48}H_{20}$

Figure 7.3: More complicated benzenoids.

Given the structure of a benzenoid B, we want to investigate the following two questions.

1. Does B exist in nature?
2. And if so, what are the bond orders between the individual carbon atoms in B?

It turns out that both of these questions can be answered using Graph Theory. If we think of the benzenoid B as a graph, where the carbon atoms are the vertices and the bonds are the edges, we say that

B can be *realized in nature* if and only if it has a perfect matching.

This is a mathematical definition, and in nature it is possible to find benzenoids violating this rule. For example, Phenalene (depicted in Figure 7.4 does exist in nature, but has no perfect matching, since it contains an odd number of carbon atoms. However, apart from a few exceptions, the rule seems to hold in nature.

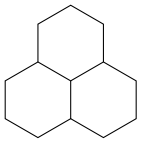


Figure 7.4: Phenalene $C_{13}H_{10}$ is an exception.

For two neighbouring carbon atoms x and y in B, we define the Pauling Bond as follows.

Definition 7.1 *The Pauling Bond of the edge xy is the number $1 + \frac{p(xy)}{p(t)}$, where $p(xy)$ is the number of perfect matching in B containing the edge xy, and $p(t)$ is the total number of perfect matching in B.*

The Pauling Bond of the edge xy turns out to be a very good approximation to the bond order between the carbon atoms x and y.

7.1 Finding the Pauling Bonds

We now describe a fast method for finding the Pauling Bonds in benzenoids where the dual graph (omitting the vertex in the exterior region) is a tree T. This is the case for all the benzenoids in Figure 7.2 and for the first benzenoid in Figure 7.3.

1. Pick an endvertex w in T and orient all edges in T towards w. Also add an edge from w into the exterior region.
2. Each vertex v in T is assigned two numbers a, a'. Let $B(v)$ be the subbenzenoid of B consisting of only the vertices in T from which there is a directed path to v and let e denote the edge in $B(v)$ intersected by the outgoing edge of v. Then, these numbers are defined as follows:

   a is the number of perfect matchings in $B(v)$ containing e.

   a' is the number of perfect matching in $B(v)$ not containing e.

Hence $a + a'$ will be the total number of perfect matchings in $B(v)$. Also notice that $B(w) = B$, so we can find the total number of perfect matchings in B by computing the two numbers a and a' for the end-vertex w. We can do this by using the three rules shown in Figure 7.5.

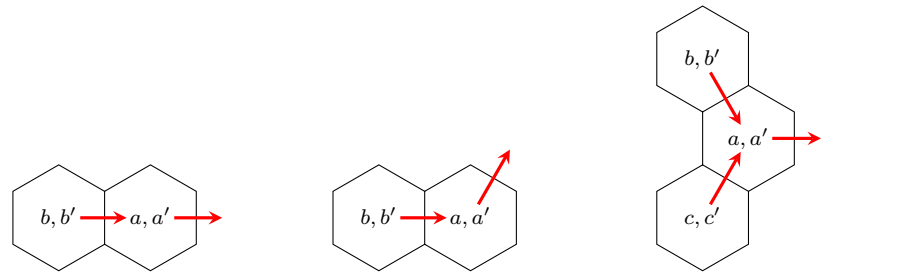


Figure 7.5: The three possible cases.

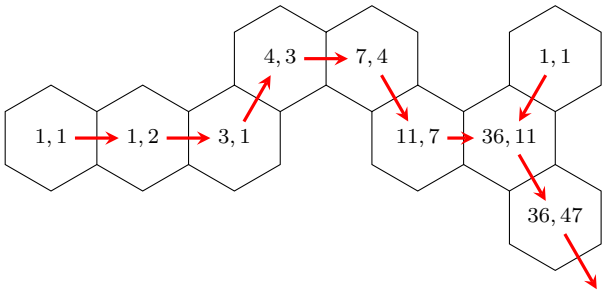


Figure 7.6: Example showing how to determine the number of perfect matchings in a benzenoid.

8

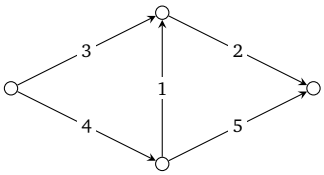
NETWORK FLOW

8.1 Basic Definitions

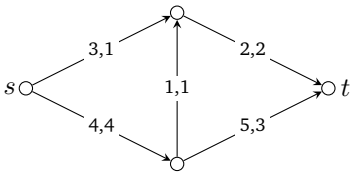
A *flow network*, or simply a *network*, is a directed graph $N = (V, E)$ in which each edge $e \in E$ is assigned a number $c(e)$, called the *capacity* of the edge. Figure 8.1(a) shows an example of a network. Let s and t be two special vertices in V, called the *source* and the *sink*, respectively. We define a *flow* from s to t, also called an st*-flow*, as a function $f : E \mapsto \mathbb{R}$ satisfying

1. For every vertex v in $V \setminus \{s, t\}$ $\quad \sum_{e \text{ into } v} f(e) - \sum_{e \text{ out of } v} f(e) = 0$
2. For every edge e in E $\quad 0 \leq f(e) \leq c(e)$.

The first requirement is known as Kirchhoff's law of flow conservation, and says that for all vertices except s and t, the sum of flow into the vertex must equal the sum of the flow out of the vertex. The second requirement ensures that the flow along every edge is non-negative and does not exceed the capacity of the edge. If the flow on an edge is equal to the capacity of the edge, the edge is said to be saturated. Figure 8.1(b) shows an example of an st-flow in a network. It is easy to verify that the flow indicated in the figure satisfies both of the above requirements.



(a) Example of a network N. Each edge e is labeled by its capacity $c(e)$.



(b) Example of an st-flow in N. Each edge e is labeled by the numbers $c(e), f(e)$.

Figure 8.1: Illustrating the definitions of a network and an st-flow in a network.

We define the *value* of an st-flow f, denoted $|f|$, as

$$|f| = \sum_{e \text{ into } t} f(e) - \sum_{e \text{ out of } t} f(e) \ .$$

Intuitively, the value of an st-flow is the value of flow that originates in s and disappears in t, i.e., the value of the flow indicated in Figure 8.1(b) is $|f| = 2 + 3 = 5$.

To motiviate the above definitions, suppose that the network shown in Figure 8.1(b) models two cities s and t connected by one-way roads. The flow along an edge xy could then be the number of cars travelling on the road between x and y in one minute. The capacity of an edge would then be the maximum number of cars that can travel from x to y in one minute. This number could for example depend on the number of lanes of the actual road and the speed limit. The value of the indicated flow means that currently 5 cars arrive at t each minute. A natural question is then: Is this the maximum number of cars that can arrive at t per minute, or can we increase the value of the flow, if the cars travel differently? In the next sections we will consider this problem in more detail.

8.2 Maximum Flows and Minimum Cuts

A *maximum st-flow* is a flow f such that $|f|$ is at least as large as the value of all other st-flows. We denote a maximum flow in a network N as $f_{\max}(N)$. The flow in Figure 8.1(b) is not a maximum st-flow, since the st-flow indicated in Figure 8.2 has a greater value. In fact, this flow is a maximum st-flow in N.

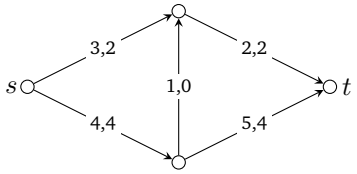
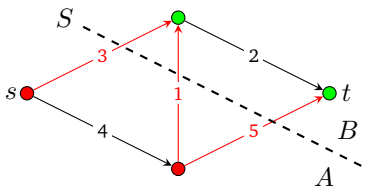


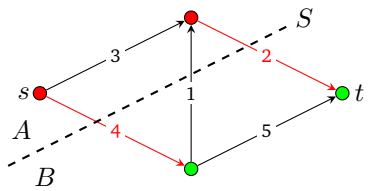
Figure 8.2: A maximum st-flow in the network N of value $|f| = 6$.

In Section 8.3, we describe an algorithm for finding a maximum flow in a network. Before doing that, we will investigate a very important relationship between flows and cuts in a network. We first introduce some necessary definitions.

An *st-cut* $S = (A, B)$ in a network N is a partition of the vertices of N into two disjoint sets A and B such that $s \in A$ and $t \in B$. The *cut edges* of S are the edges directed from a vertex in A to a vertex in B. An st-cut separates s from t in the following sense: If the cut edges are deleted, it is impossible to find a path from s to t in N. Figure 8.3(a) shows an example of an st-cut.



(a) Example of an st-cut $S = (A, B)$. The red vertices belong to A and the green to B. The cut edges are marked in red.



(b) A minimum st-cut in N. The capacity of this cut is $c(S) = 4 + 2 = 6$.

Figure 8.3: Illustrating the definition of an st-cut in the network from Figure 8.1(a).

For an st-cut S we define the *capacity* of S, denoted $c(S)$, as the sum of the capacities of the cut edges, that is, $c(S) = \sum_{e \in S} c(e)$. The capacity of the cut shown in Figure 8.3(a) is

$c(S) = 3 + 1 + 5 = 9$. A cut S is a *minimum st-cut* if there is no st-cut with a smaller capacity. The cut in Figure 8.3(a) is not a minimum st-cut, because the cut shown in Figure 8.3(b) has a smaller capacity. In fact, the cut in Figure 8.3(b) is a minimum st-cut.

We now consider the problem of finding a minimum st-cut in a network. A naive algorithm could go through every st-cut and find one with minimum capacity. Unfortunately, there are 2^{n-2}, i.e., an exponential number of st-cuts in a network with n vertices, rendering such an algorithm useless in practice. It turns out that finding a minimum cut is closely related to finding a maximum flow. To make this connection more precise, we need the following lemma

Lemma 8.1 *Let N be a network. For any st-cut S and any st-flow f in N,*

$$|f| \leq c(S) \ .$$

This lemma is not hard to show. Intuitively, one can think of an st-cut as a bottleneck of the flow, since all the flow from s to t must pass through the cut edges. The lemma implies that if we can find a flow f and a cut S in a network N such that $|f| = c(S)$, then f must be a maximum flow and S must be a minimum cut. For example, consider the flow f of value 6 in Figure 8.2 and the cut S with capacity 6 in Figure 8.3(b). Since $|f| = c(S)$, we can immediately conclude that f is a maximum st-flow and S a minimum st-cut. This leads to the MaxFlow-MinCut Theorem, which is the main theorem in the subject of Network Flow.

Theorem 8.1 (MaxFlow-MinCut Theorem) *In any network N the value of a maximum st-flow equals the capacity of a minimum st-cut.*

The relationship between maximum flows and minimum cuts is very similar to that between maximum matchings and minimum coverings, which we studied in Chapter 6. There, we learned that not all graphs had a matching and a covering of the same size, and this is essentially the reason why finding a minimum covering in an arbitrary graph is a very hard problem. Fortunately, the same is not true for flows and cuts.

In the next section we consider an efficient algorithm for finding a maximum st-flow in a network, and thanks to the MaxFlow-MinCut Theorem, the same algorithm will also be able to find a minimum st-cut.

8.3 Finding a Maximum Flow

To solve the problem of finding a maximum-flow in a network, we introduce an algorithm first described by L. R. Ford, Jr. and D. R. Fulkerson, namely the Ford-Fulkerson algorithm. The algorithm works by finding augmenting paths between s and t, that is, paths where the flow can be increased, and then increase the flow on that path. This is repeated until a flow of maximal value is reached.

Augmenting paths An augmenting path, is a path in N where we allow traversing directed edges backwards, and where the following constraints hold

1. For each forward edge e in P $\qquad f(e) < c(e)$
2. For each backward edge e in P $\qquad 0 < f(e)$.

We can augment extra flow over an augmenting path, by increasing the flow on the forward edges by some number δ, and decreasing the flow on the backwards edges by the same number δ.

The Ford-Fulkerson Algorithm Here we describe the original algorithm proposed by Ford and Fulkerson: To find the maximum flow in a st-network N, find an augmenting path from s to t and augment as much flow along the path as possible. Continue doing this until the value of the flow in N is maximal.

Algorithm 7 Ford-Fulkerson algorithm for computing a maximum flow

for every edge e in N **do**
  $f(e) = 0$
while there exists an augmenting path P from s to t in N **do**
  Let δ be the bottleneck when increasing the flow over P
  for all edges $xy \in E(P)$ **do**
    if xy is directed from s to t in P **then**
      $f(xy) = f(xy) + \delta$
    else
      $f(xy) = f(xy) - \delta$

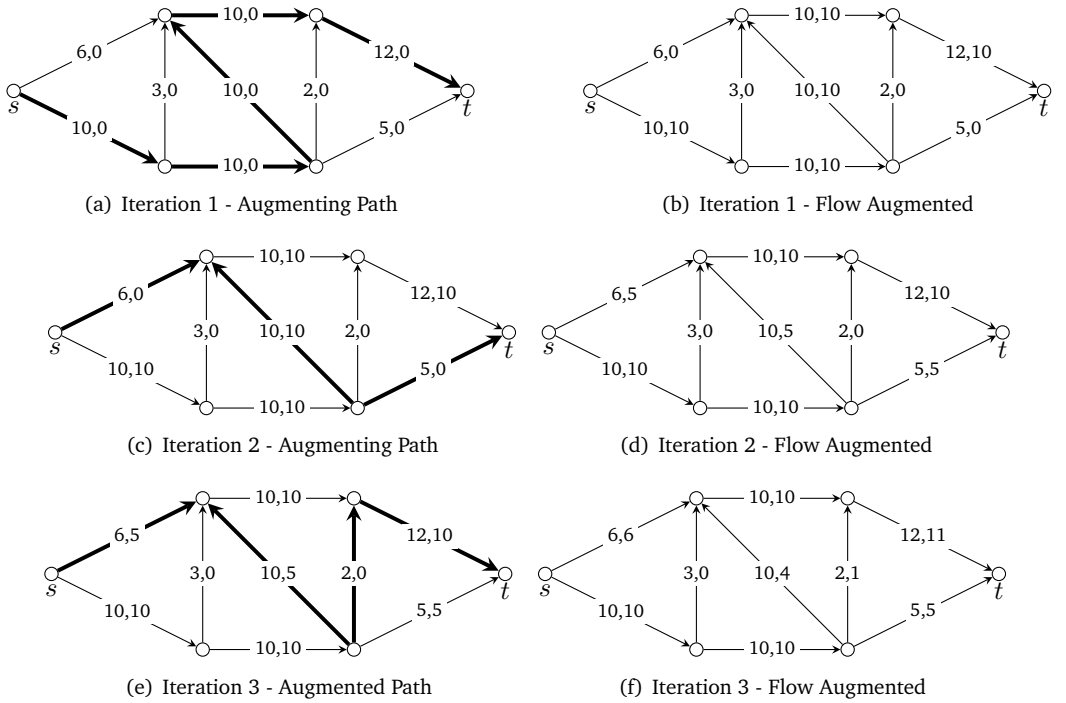


Figure 8.4: 3 complete iterations of the Ford-Fulkerson algorithm. Augmenting paths are marked as bold.

When the Ford-Fulkerson algorithm terminates, it is because there is no augmenting path to be found.

Theorem 8.2 *Let N be a network. If f is a st-flow in N such that there is no augmenting path then there exists an st-cut (A, B) in N for which $|f| = c(A, B)$. Consequently f is a maximum flow, and (A, B) is a minimum cut.*

Another useful theorem is the following, which shows another way of determining whether a cut is in fact minimum.

Theorem 8.3 *Let N be a network and f be a maximum st-flow in N, then an st-cut $C = (A, B)$ is minimum if and only if all edges from A to B are fully saturated by f, and all edges from B to A have zero flow with respect to f.*

We also state the following theorem for general directed graphs

Theorem 8.4 (Menger's Theorem) *Let s and t be vertices in a directed graph G. Now we consider the following numbers*

$a =$ *maximum number of edge-disjoint directed paths from s to t*

$b =$ *the smallest number of edges to delete in order to destroy all directed paths from s to t*

$c =$ *the smallest number of edges in an st-cut*

$d =$ *the largest value of a flow from s to t when all capacities are* 1

Then $a = b = c = d$.

8.4 Finding a minimum cut

When we have found a maximum flow in a network using the algorithm of Ford-Fulkerson, often we are also interested in finding a minimum cut. An easy way to do this, is by using a marking process, assigning colors to the vertices.

8.4.1 Performing the marking process

Let N be a network, where we have found some maximum st-flow f. Start by assigning the color red to s, the color green to t, and the color white to the rest of the vertices in N. Then assign the color red, to all vertices which can be reached by an augmenting path starting in s. At last, assign the color green to all vertices, from where an augmenting path to t can begin.

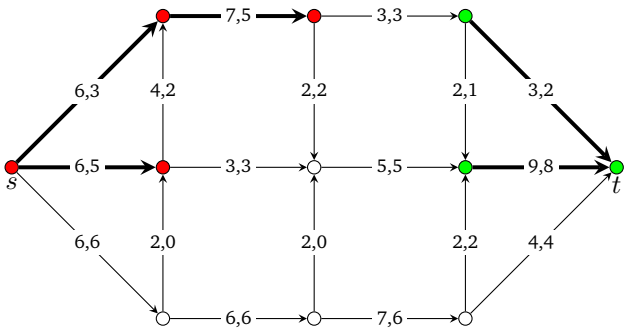


Figure 8.5: The marking process performed on a graph with a maximum flow. The bold edges show the augmenting paths leading to the colouring.

8.4.2 Optimal and critical edges

We now introduce two new concepts, namely *critical edges* and *optimal edges*.

Definition 8.1 *An edge e is **critical**, if any change of its capacity changes the value of the maximal flow. Equivalently, an edge is critical if and only if it belongs to all minimal cuts.*

Definition 8.2 *An edge e is **optimal**, if any reduction of its capacity reduces the value of the maximal flow, but no increase of its capacity increases the value of the maximal flow. Equivalently, an edge is optimal if and only if it belongs to at least one but not every minimal cut.*

To determine if an edge is critical, optimal or neither, there are some rules to follow. These rules can be applied after the marking process has be carried out:

1. The critical edges are precisely those from red to green.
2. All edges from red to white are optimal.
3. All edges from white to green are optimal.
4. If there are more optimal edges, they must be saturated and go from white to white.

To find the optimal edges from white to white, we need to consider all edges xy where both x and y are white. Now we make x red and continue the (red) labelling procedure from x. In this way some white vertices become red. (But no green vertex changes color.) Now there are two possibilities: Either y becomes red. In this case it is easy to see that xy cannot be in any minimal cut. Therefore, xy is not optimal. Or else y does not become red. In this case, the original red vertices together with the new red vertices form the left hand side of a minimal cut which contains xy and which shows that xy is optimal.

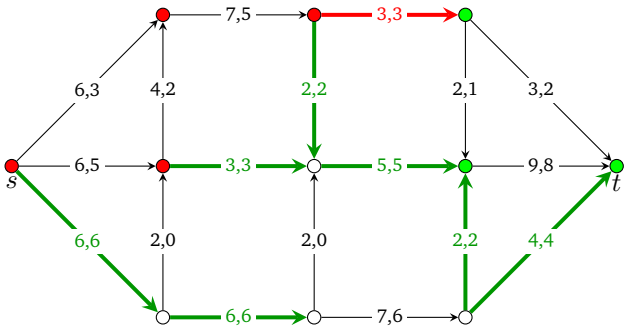


Figure 8.6: The red edges are critical, and the green edges are optimal. The edges which are neither critical nor optimal are black.

9

ELECTRICAL NETWORKS

In this chapter we use graph theory to study properties of electrical networks. We consider an electrical network as a graph N, where the edges are either resistors, current or voltage generators. We first introduce some important definitions in Section 9.1 and Section 9.2. We then show how to use Kirchhoff's Rule to compute the current through any resistor in an electrical network and how to calculate the driving point resistance between any two vertices. Finally, in Section 9.5 we describe a remarkable connection between random walks in graphs and electrical networks.

9.1 Co-tree Product

For an electrical network N and some spanning tree $\mathcal{T}$ in N, we define the *co-tree product* $\pi(\mathcal{T})$ as follows.

Definition 9.1 (Co-tree Product) *Let $\mathcal{R} = \{R_1, R_2, \ldots, R_q\}$ be the set of resistors in the network N and let $\mathcal{T}$ be a spanning tree in N. The co-tree product of $\mathcal{T}$, $\pi(\mathcal{T})$, is defined as the product of the resistors* not *in $\mathcal{T}$. That is,*

$$\pi(\mathcal{T}) = \prod_{\substack{R \in \mathcal{R} \\ R \notin \mathcal{T}}} R$$

We will use the co-tree product to calculate different properties of electrical networks. Figure 9.1 illustrates how to calculate the co-tree product of a spanning tree.

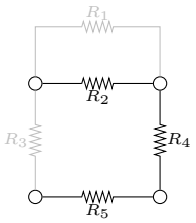


Figure 9.1: A spanning tree $\mathcal{T}$ with co-tree product $\pi(\mathcal{T}) = R_1 R_3$, since these are the resistors outside of the tree.

9.2 Network Determinant

For an electrical network N consisting of resistors, voltage and current generators, we define the network determinant ΔN. The network determinant is interesting as it can be used to reason about many properties of the network N.

Definition 9.2 (Network Determiant) *Let $\mathcal{R} = \{R_1, R_2, \ldots, R_q\}$ denote the set of resistors, $\mathcal{V}$ the set of voltage generators and $\mathcal{I}$ the set of current generators in a network N. The network determinant of N is defined as*

$$\Delta N = \sum_{\mathcal{T}} \pi(\mathcal{T})$$

where the sum is the co-tree products of all spanning trees $\mathcal{T}$ in $(N - \mathcal{I})/\mathcal{V}$, i.e. the spanning trees of N which contain all the voltage generators $\mathcal{V}$, but none of the current generators $\mathcal{I}$.

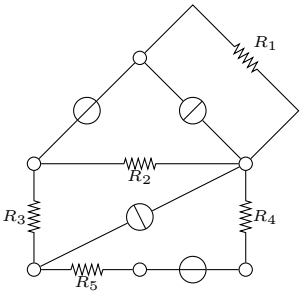
9.2.1 Calculating the Network Determinant

In concordance with the definition, the calculation of the network determinant can be done with the following algorithm.

Algorithm 8 Calculating the Network Determinant

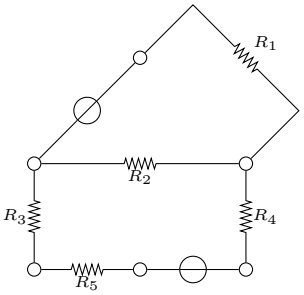
Remove all current generators in N.
Contract all voltage generators in N (Now N only consists of resistors).
for Each spanning tree $\mathcal{T}$ in the graph **do**
  Find the co-tree product $\pi(\mathcal{T})$
Take the sum of all these products, this is the network determinant.

Figure 9.2 shows how to calculate the network determinant using the above algorithm.



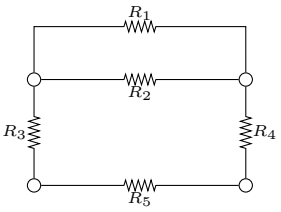
N

(a) A regular network.



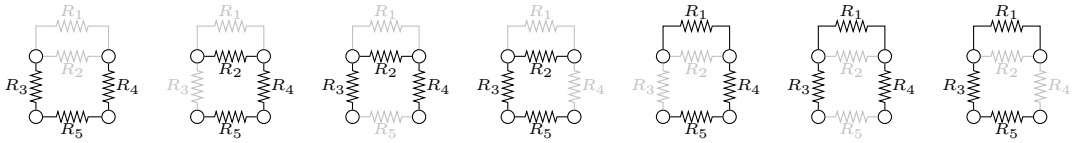
$N - \mathcal{I}$

(b) Step 1: Removing all current generators.



$(N - \mathcal{I})/\mathcal{V}$

(c) Step 2: Contracting all voltage generators.



R_1R_2 R_1R_3 R_1R_5 R_1R_4 R_2R_3 R_2R_5 R_2R_4

(d) Step 3: Finding the product of the resistors outside each of the seven spanning trees in $(N - \mathcal{I})/\mathcal{V}$. The network determinant is the sum of these products: $\Delta N = R_1R_2 + R_1R_3 + R_1R_5 + R_1R_4 + R_2R_3 + R_2R_5 + R_2R_4 = R_1R_2 + (R_1 + R_2)(R_3 + R_4 + R_5)$

Figure 9.2: Example showing how to calculate the network determinant of a regular network.

Useful Tip: If the network only contains resistors with resistance 1, the network determinant is the number of spanning trees in $(N - \mathcal{I})/\mathcal{V}$.

9.3 Finding the Current Using Kirchhoff's Rule

If N is a electrical network consisting of only resistors and a single generator (either voltage or current), then we can use *Kirchhoff's Rule* to find the current through any of the resistors in the network.

Theorem 9.1 (Kirchhoff's Rule) *Let N be a network consisting only of resistors and a single generator of value G. Then the current through the resistor R_k is*

$$i_{R_k} = \frac{\pm G}{\Delta N} \sum_{\mathcal{T}_{R_k}} \pm \pi(\mathcal{T}_{R_k})$$

where sum is over all R_k-trees in N. The first $\pm$ is $+$ if G is a current generator and $-$ if G is a voltage generator. The second $\pm$ is the sign of the R_k-tree $\mathcal{T}_{R_k}$ (refer to the section about R_k-trees).

9.3.1 R_k-trees and their Sign

We still consider a network of resistors $\mathcal{R}$ and a single generator G. For a fixed resistor $R_k \in \mathcal{R}$, a R_k-tree is a special spanning tree of $N - G$. More precisely,

Definition 9.3 (R_k-tree) *Let $\mathcal{T}$ be a spanning tree of $N - G$, then $\mathcal{T}$ is a R_k-tree if and only if R_k is contained in the fundamental cycle created when adding G to $\mathcal{T}$.*

Consider a R_k-tree $\mathcal{T}_{R_k}$ and let C be the fundamental cycle created when adding G to $\mathcal{T}_{R_k}$. If we have fixed the direction on G and R_k in N, we will distinguish between *positive* and *negative* R_k-trees. $\mathcal{T}_{R_k}$ is positive if the direction of G and R_k agrees in the fundamental cycle C, and negative if G and R_k has opposite directions in C.

9.3.2 Example of How to Use Kirchhoff's Rule

Consider the network shown in Figure 9.3. In this example, we show how to use Kirchhoff's Rule to determine the current through the resistor R_7 in the direction indicated.

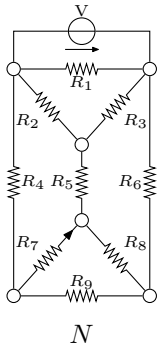


Figure 9.3: A network N consisting of nine resistors and a single voltage generator.

Calculating ΔN. First we need to determine the network determinant ΔN. To do this, we contract the voltage generator and consider the spanning trees in the resulting graph. Since this graph has 40 spanning trees, the network determinant will have 40 terms! However, if most of the resistors have value 1, the expression for the ΔN is likely to be a lot simpler. Notice that each of the terms in ΔN will have exactly 4 factors, since there are 4 edges outside each spanning tree in N/V.
Since a spanning tree $\mathcal{T}$ in N/V can not contain both R_2 and R_3, it will fall into one of the following three categories.

(a) $\mathcal{T}$ contains neither R_2 nor R_3. In this case, we can further partition the spanning trees into those which contain R_9 and those that do not. In this way, the combined contribution of these spanning trees to ΔN becomes

$$R_1R_2R_3\Big(\overbrace{(R_4+R_6)(R_7+R_8)}^{R_9\in\mathcal{T}}+\overbrace{R_9(R_4+R_6+R_7+R_8)}^{R_9\notin\mathcal{T}}\Big)\,.$$

(b) $\mathcal{T}$ contains R_2 but not R_3. Among these trees consider first the ones that contain R_5. Their contribution to ΔN is

$$\begin{aligned}
&R_1R_3\Big(\overbrace{R_4R_6R_7+R_4R_6R_8+R_4R_7R_8+R_6R_7R_8}^{R_9\in\mathcal{T}}+\overbrace{R_9(R_4+R_7)(R_6+R_8)}^{R_9\notin\mathcal{T}}\Big)=\\
&R_1R_3\Big(R_6R_8(R_4+R_7)+R_4R_7(R_6+R_8)+R_9(R_4+R_7)(R_6+R_8)\Big)=\\
&R_1R_3\Big(R_4R_7(R_6+R_8)+(R_4+R_7)\big(R_6R_8+R_9(R_6+R_8)\big)\Big)
\end{aligned}$$

Likewise those not containing R_5 contribute a total of

$$R_1R_3R_5\Big(\overbrace{(R_4+R_6)(R_7+R_8)}^{R_9\in\mathcal{T}}+\overbrace{R_9(R_4+R_6+R_7+R_8)}^{R_9\notin\mathcal{T}}\Big)\ .$$

(c) $\mathcal{T}$ contains R_3 but not R_2. Due to symmetry, this case is identical to the previous, when R_3 is replaced by R_2.

Recapitulating, the network determinant is

$$\begin{aligned}\Delta N = \ & R_1R_2R_3\Big((R_4+R_6)(R_7+R_8)+R_9(R_4+R_6+R_7+R_8)\Big)+\\ & R_1(R_2+R_3)\Big(\Big(R_4R_7(R_6+R_8)+(R_4+R_7)\big(R_6R_8+R_9(R_6+R_8)\big)\Big)+\\ & R_1R_5\Big((R_4+R_6)(R_7+R_8)+R_9(R_4+R_6+R_7+R_8)\Big)\Big)\end{aligned}$$

Calculating the co-tree products of the R_7-trees. Figure 9.4 shows the seven R_7-trees and their contribution to the sum of co-tree products. The total sum becomes

$$\sum_{\mathcal{T}_{R_7}}\pm\pi(\mathcal{T}_{R_7})=-R_1\Big(R_2\big(R_6(R_8+R_9)-R_9(R_3+R_5+R_8)\big)-R_3(R_5R_9-R_4R_8)\Big)$$

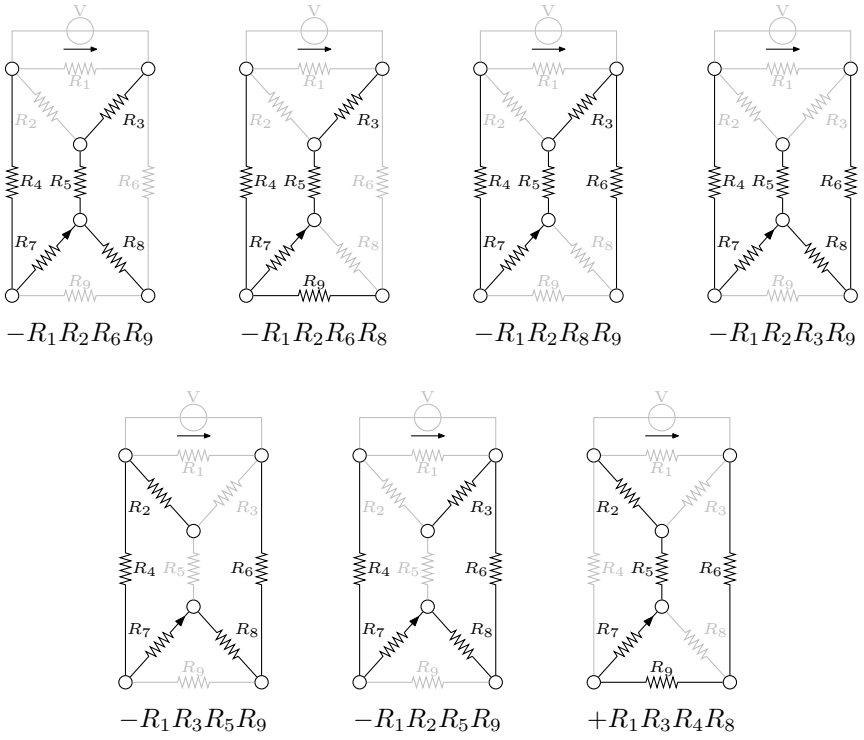


Figure 9.4: The seven R_7-trees in N and their contribution to the sum of co-tree products.

The Current in R_7. Inserting into Kirchhoff's Rule, we find the current through R_7 in the direction indicated to be

$$i_{R_7} = R_1\Big(R_2\big(R_6(R_8+R_9) - R_9(R_3+R_5+R_8)\big) - R_3(R_5R_9 - R_4R_8)\Big)\frac{V}{\Delta N}\ .$$

This expression reveals that the direction of the current through R_7 depends on the values of the resistors. In case all resistors have value 1, then the current through R_7 simply becomes

$$i_{R_7} = \frac{-V}{\tau(N/V)}(-6+1) = \frac{-V}{40}(-5) = \frac{1}{8}V\ .$$

9.4 Finding the driving point resistance between two vertices

In a network N, we can find the driving point resistance between two vertices p and q by the formula

$$R_{p,q} = \frac{\Delta(N/pq)}{\Delta N}$$

where $\Delta(N)$ is the network determinant of N.

9.5 Random Walks

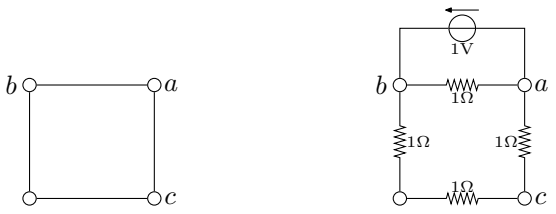
Let G be an undirected graph. A random walk in G is a way of visiting the vertices of G in random fashion. The walk starts in a vertex v of G. We then select a neighbour of v uniformly at random and walk to this vertex, where we repeat. The walk never stops and hence eventually it will visit all the vertices of G. When the walk visits a vertex v each neighbour of v has probability $\frac{1}{d(v)}$ of being chosen as the next vertex to be visited, where $d(v)$ is the degree of v. It turns out that there is a remarkable connection between random walks in undirected graphs and electrical networks. To make this connection more precise, we consider the following problem.

> **Problem** Let G be an undirected graph and let a and b be two distinct vertices of G. If a random walk starts in some vertex c, what is the probability that the walk will reach a before it reaches b?

Obviously, if $c = a$ (i.e., the walk starts in a) this probability is 1 and if $c = b$, the probability is 0. But for any other vertex in G it is not clear how we can determine this probability. Fortunately, the following theorem allows us to determine this probability for any vertex in G by considering the node voltages in an electrical network.

> **Theorem 9.2** *The probability that a random walk starting in c reaches a before it reaches b is equal to the node voltage of c in the electrical network obtained from G as follows: Replace each edge in G with a 1Ω resistor and connect a $1V$ voltage generator between a and b such that the voltage drop is from a to b.*

The construction described by the theorem is illustrated in Figure 9.6.



(a) An undirected graph G.

(b) The electrical network obtained from G as described by Theorem 9.2.

Figure 9.5: Illustrating the construction described by Theorem 9.2.

9.5.1 Computing Node Voltages

We now consider the problem of determining the node voltages in the electrical network described by Theorem 9.2. Observe that the voltage drop over a resistor equals the current through it since all resistances are 1Ω (this follows from Ohm's Law). Hence we could calculate the network determinant and use Kirchhoff's Rule (Theorem 9.1) to compute the current (and voltage) through each of the resistors. However, this approach is quite intricate and impractical. In this section we will show how to determine the current/voltage through any resistor by solving a number of simple equations. We will formulate these equations using Kirchhoff's Circuit Laws.

Theorem 9.3 (Kirchhoff's Circuit Laws) *In any electrical network N,*

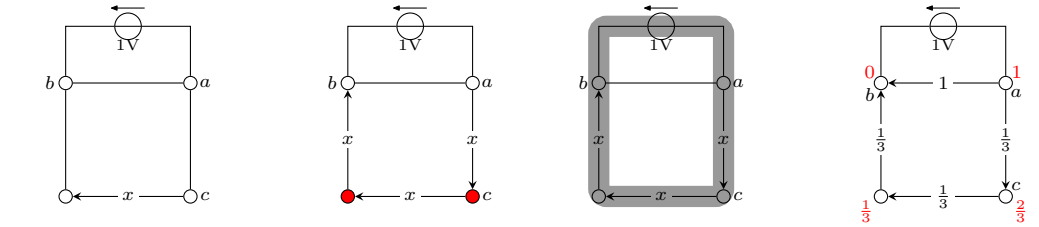
1. *For every node, the sum of currents flowing into the node is equal the sum of currents flowing out of the node. (Kirchoff's Current Law)*
2. *The sum of directed voltage drops along any cycle in N is zero. (Kirchoff's Voltage Law)*

Example

We now show how to find the node voltages in the network shown in Figure 9.5(b).
First we introduce a variable x to denote the unknown current flowing through the lower horizontal resistor in the direction indicated in Figure 9.6(a). We then apply Kirchhoff's Current Law on the two bottom nodes marked in red in Figure 9.6(b). The law implies that current flowing through the two vertical resistors also must be x in the directions indicated. Next we use Kirchhoff's Voltage Law on the cycle shown in Figure 9.6(c) in clockwise direction. This gives the equation

$$x + x + x - 1 = 0 \quad \Leftrightarrow \quad x = \frac{1}{3}\,.$$

Thus the current and voltage through each of these three resistors are $\frac{1}{3}$. The current through the upper horizontal resistor can easily be found to be 1, also by using Kirchhoff's Voltage Law. Having found the currents and voltages in all edges, the node voltage $U(v)$ in a vertex v is found by adding the directed voltage drops on a path from v to b. Hence $U(c) = \frac{1}{3} + \frac{1}{3} = \frac{2}{3}$. The final solution is shown in Figure 9.6(d) with the node voltages marked in red.
Recapitulating, we have found that the probability that a random walk starting in node c gets to a before it gets to b is $\frac{2}{3}$. If the walk starts in the vertex to the left of c, the probability is $\frac{1}{3}$.



(a) Introducing a variable.

(b) Using Kirchhoff's Current Law.

(c) Using Kirchhoff's Voltage Law.

(d) The final solution.

Figure 9.6: Illustrating how to compute the current and voltages in a network using Kirchhoff's Circuit Laws.

The above is a simple example. For larger networks, we might need to introduce more variables and solve more equations. It is often possible to exploit the symmetry of the network to reduce the number of unknowns needed. We refer the reader to Section A.12 for exercises on calculating probabilities for random walks. Beware though, when solving these exercises, a very common mistake is often made by using Kirchhoff's Current Law on a or b:

> **Caveat:** Contrary to the resistors, the current through the voltage generator is *not necessarily* equal to the voltage drop. Consequently, one should be very careful if Kirchhoff's Current Law is used on a or b.

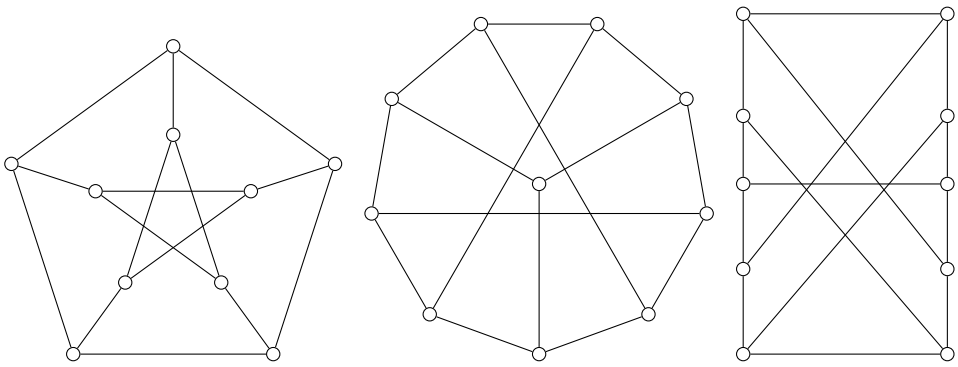
Appendices

A

WEEKLY EXERCISES

A.1 Week 1

E1.1 (1.11 FN) Are the following graphs isomorphic?



E1.2 (1.18 FN) Can an eulerian graph contain a bridge?

E1.3 On a 3 by 3 chess board are 4 different knights (Danish: springere) as indicated in Table A.1(a) below.

1		2
3		4

(a)

4		3
2		1

(b)

1		3
2		4

(c)

Table A.1: Three different configurations of knights on a 3 by 3 chessboard.

Is it possible to move them such that we obtain the configuration in Table A.1(b)? Or in Table A.1(c)? If so, how many moves are needed? (Two knights are not allowed to be at the same square at the same time.)

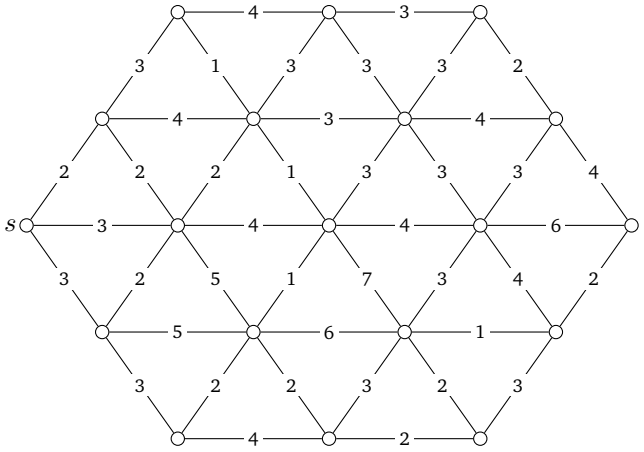
E1.4 Read an understand the proof of Theorem 4.1 in Bondy and Murty.

E1.5 Does there exist a connected graph in which all edges are cut edges? Does there exist a connected graph in which all vertices are cut vertices?

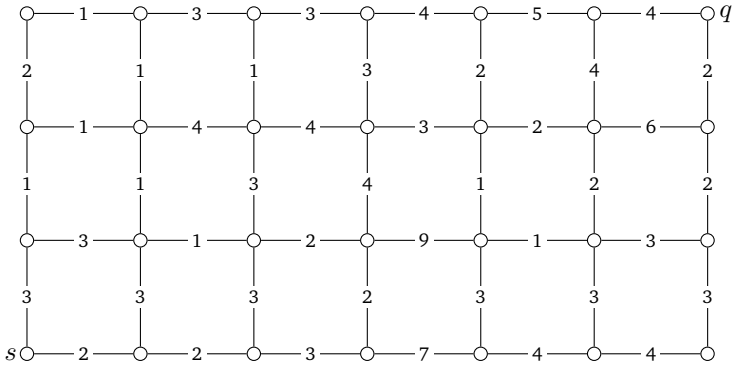
E1.6 (1.13 FN) Show that a simple graph G with n vertices, is connected if $d(v) \geq \frac{1}{2}(n-1)$ for all $v \in V(G)$. Is this the best possible bound?

A.2 Week 2

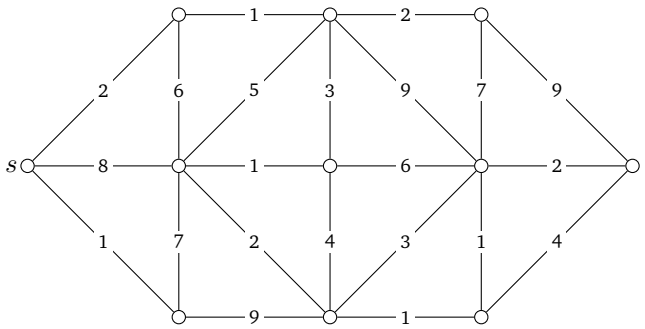
E2.1 (4.1 FN) Run Dijkstra's Algorithm on the graph below, and find the distance from s to every vertex. Find and mark the shortest paths from s to all vertices.



E2.2 (4.6 FN) Run Dijkstra's Algorithm on the graph. Find and mark the shortest path from s to q.



E2.3 (1.8.1 BM) Run Dijkstra's Algorithm on the graph below. Find and mark the shortest path from s to all other vertices.



E2.4 (1.8.2 BM) What additional instructions to Dijkstra's Algorithm are needed in order that Dijkstra's Algorithm determine shortest paths rather than merely distances?

E2.5 (1.8.6 BM) Describe a good algorithm for determining

(a) the components of a graph.

(b) the girth (size of smallest cycle) of a graph.

How good are your algorithms?

E2.6 (If you have time). This exercise (1.5.7 BM) is related to the home work exercise FN 1.16. Note that in the following we require that the graphs are simple (that is, they have no loops and no multiple edges).

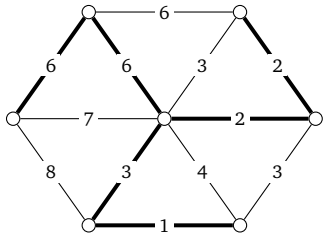
A sequence $\boldsymbol{d} = (d_1, d_2, \ldots, d_n)$ is *graphic* is there is a simple graph with degree sequence $\boldsymbol{d}$. Let $\boldsymbol{d} = (d_1, d_2, \ldots, d_n)$ be a non-increasing sequence of non-negative integers, and denote the sequence $(d_2 - 1, d_3 - 1, \ldots, d_{d_1+1} - 1, d_{d_1+2}, \ldots, d_n)$ by $\boldsymbol{d'}$.

(a) Show that $\boldsymbol{d}$ is graphic if and only if $\boldsymbol{d'}$ is graphic.

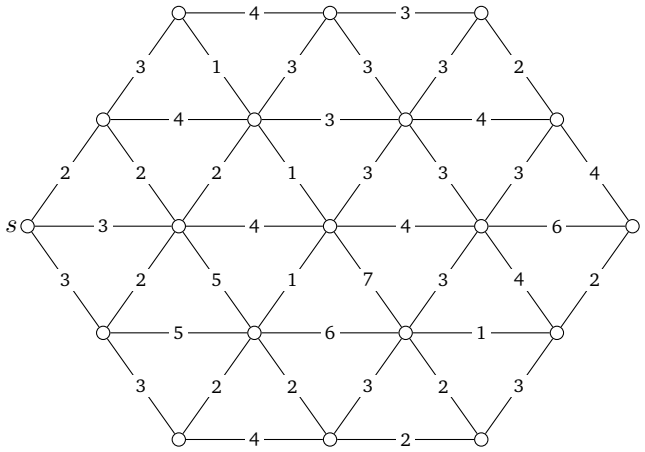
(b) Using that, describe an algorithm for constructing a simple graph with degree sequence $\boldsymbol{d}$, if such a graph exists.

A.3 Week 3

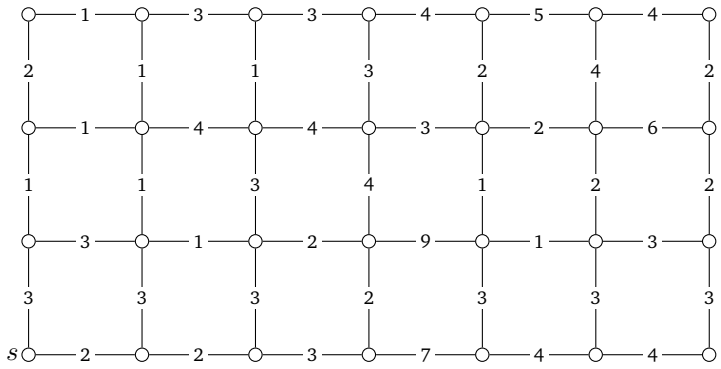
E3.1 (2.5.1 BM) Show, by applying Kruskal's Algorithm, that the tree indicated in this figure is indeed optimal (i.e. the total cost is minimum).



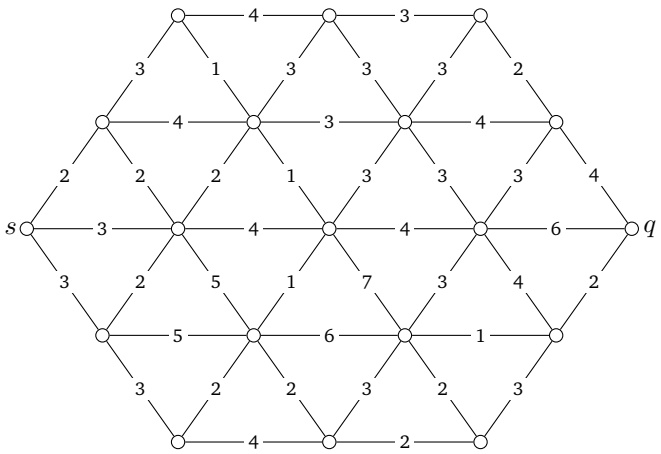
E3.2 Run The Chinese Postman's Algorithm on the following graph and find the postman's walk.



E3.3 Run The Chinese Postman's Algorithm on the following graph and find the postman's walk.



E3.4 Run The Chinese Postman's Algorithm on the following graph. This time the walk should start in s and end in q. How long is the walk? More generally, if the walk starts and ends in distinct vertices, how should the algorithm for the Chinese Postman problem walk be modified?



E3.5 Kruskal's Algorithm finds a spanning tree in which the **sum** of the weights of the edges is smallest possible. If all weights are positive, then how do we find a spanning tree such that the **product** of the edges is smallest possible? Can we use Kruskal's Algorithm?

E3.6 Kruskal's Algorithm grows a forest. Suppose we start at a vertex x and insist that we want to grow a tree. Let us each time choose a smallest edge joining a vertex in the tree with a vertex not in the tree. Is this the same as Kruskal's Algorithm? Does it always give a spanning tree of smallest total weight? This modification of Kruskal's Algorithm is sometimes called Prim's Algorithm.

A.4 Week 4

E4.1 (5.2.1 BM) Show that it is impossible, using 1×2 rectangles, to exactly cover an 8×8 square from which two opposite corner squares have been removed.

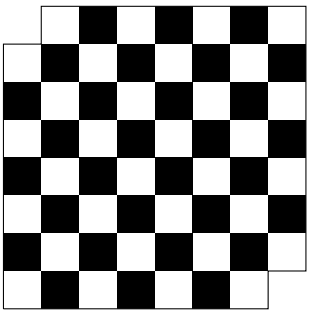


Figure A.1: The domino square.

E4.2 (5.1.3 BM) For each $k > 1$, find an example of a k-regular simple graph that has no perfect matching.

E4.3 (5.1.2 BM) Show that a tree has at most one perfect matching.

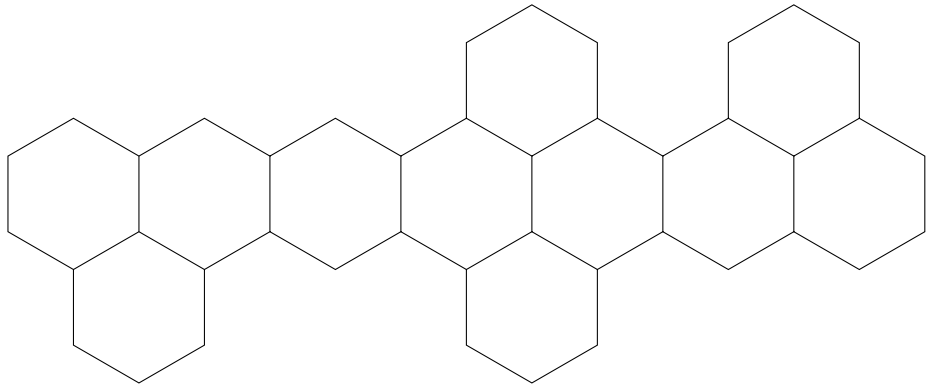
E4.4 (5.2.2 BM) Show that a bipartite graph G has a perfect matching if and only if $|N(S)| \geq |S|$ for all $S \subseteq V$. Give an example to show that the above statement does not remain valid if the condition that G be bipartite is dropped.,

E4.5 (5.2.5 BM) A *line* of a matrix is a row or a column of the matrix. Show that the minimum number of lines containing all the 1's of a $(0,1)$-matrix is equal to the maximum number of 1's, no two of which are in the same line.

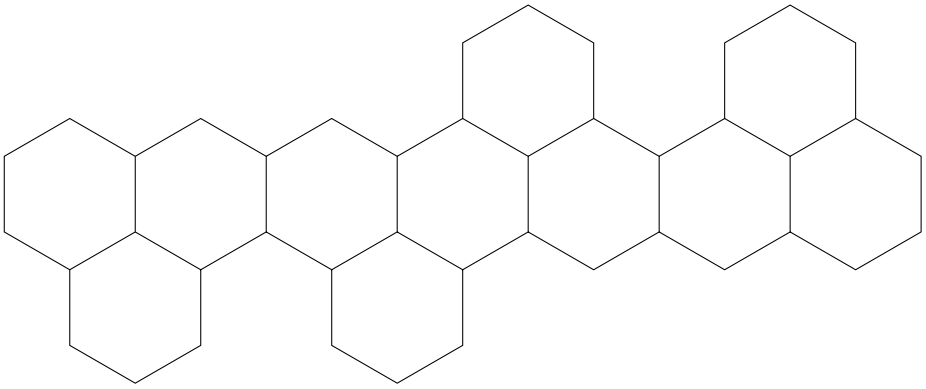
E4.6 (5.1.4 BM) Two people play a game on a graph G by alternately selecting distinct vertices $v_0, v_1, v_2, \ldots$ such that, for $i > 0$, v_i is adjacent to v_{i-1}. The last player able to select a vertex wins. Show that the first player has a winning strategy if and only if G has no perfect matching.

A.5 Week 5

E5.1 Which of the benzoids shown in Figure A.2 can be realized? Find the Pauling bonds of all edges.



(a)



(b)

Figure A.2:

E5.2 Find the total number of perfect matchings and all the Pauling bonds of the red hexagon in Figure A.3.

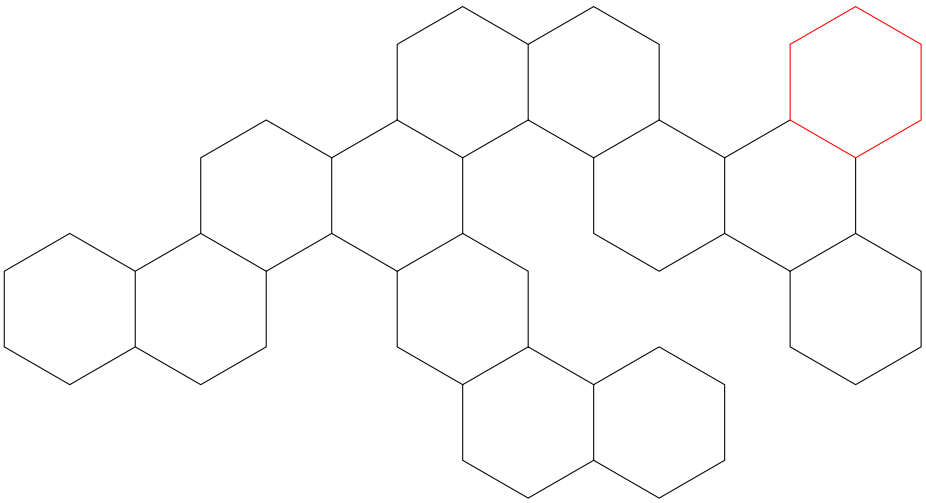
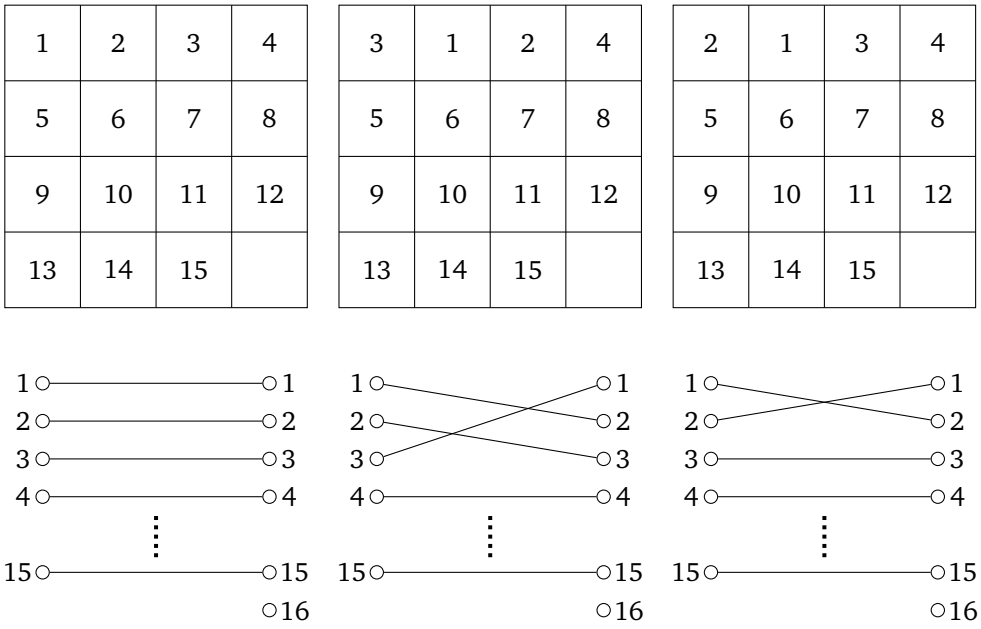


Figure A.3:

E5.3 (5.3.3 BM) Show that a tree T has a perfect matching if and only if $o(T - v) = 1$ for all $v \in V(T)$. Remember that $o(G)$ is the number of odd components in G, that is, the number of connected components in G with an odd number of vertices.

E5.4 The leftmost figure below shows what we shall call a 15-game. It consists of 15 bricks (each with a number) placed on a square divided into 16 squares. You are allowed to move a brick into the empty square (assuming that the brick is on a neighbouring square). We can describe each configuration by a bipartite graph as indicated in the figure.



The left hand part are the 15 bricks. The right hand part are the 16 squares. The edges tell where the bricks are placed.

1. What happens with the number of crossings when a brick is moved?
   (a) horizontally?
   (b) vertically?

   Look at the various possibilities and observe how the parity of the number of crossings changes.
2. If we start and terminate with a configuration where the empty square is the lower right square (number 16), then show that the number of times that a brick has been moved up equals the number of times that a brick has been moved down. (Hint: concentrate on the empty square.)
3. Which of the above configurations can be obtained from the leftmost configuration?

E5.5 (BM 5.2.4) Let $A_1, A_2, \ldots, A_m$ be subsets of a set S. A *system of distinct representatives* for the family $(A_1, A_2, \ldots, A_m)$ is a subset $\{a_1, a_2, \ldots, a_m\}$ of S such that $a_i \in A_i$, $1 \leq i \leq m$, and $a_i \neq a_j$ for $i \neq j$. Show that $(A_1, A_2, \ldots, A_m)$ has a system of distinct representatives if and only if $\left|\bigcup_{i \in J} A_i\right| \geq |J|$ for all subsets J of $\{1, 2, \ldots, m\}$.

A.6 Week 6

E6.1 (4.8 FN) Solve the job assignment problem for Table A.2.

5	2	3	3
3	4	1	1
6	4	2	1
2	4	1	1

Table A.2: Table for E6.1

E6.2 (4.9 FN) Solve the job assignment problem for Table A.3.

8	6	7	10
9	8	8	6
4	2	1	10
8	6	6	10

Table A.3: Table for E6.2

E6.3 (FN Exam January 1993, Problem 3) The domino board in Figure A.4 contains 18 black and 18 white cells. Find the maximum number of dominoes you can cut out of the figure. Show in two different ways that the given number is maximum:

- By a marking process
- By Konigs Theorem

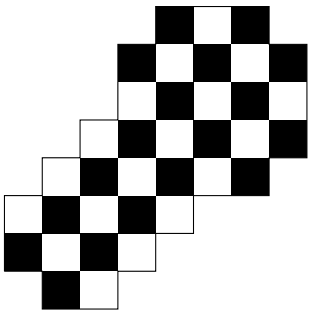


Figure A.4: The domino square.

E6.4 (FN Exam January 1994, Problem 2) Find the structure rank of the matrix A. Now find a minimum system of rows and columns in A which covers all 1's. Determine the complexity of the algorithm you use for an $n \times n$ matrix.

$$A = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

A.7 Week 7

E7.1 (FN 3.1) Find a maximum s-t flow in the network in Figure A.5. Then show a minimum s-t cut.

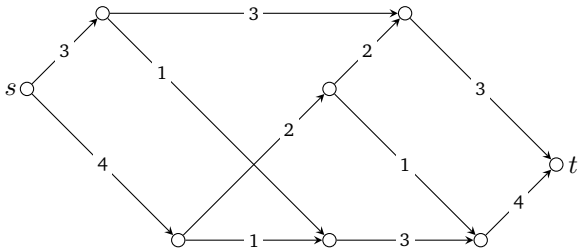


Figure A.5: A flow-network with capacity on each edge.

E7.2 (FN 3.2) Find a maximum s-t flow in the undirected network in Figure A.6. Then show a minimum s-t cut.

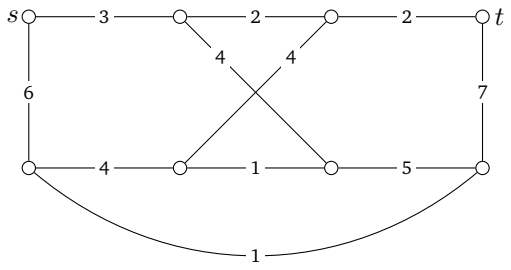


Figure A.6: An undirected flow-network with capacity on each edge.

E7.3 (BM 11.2.1) Consider the network shown in Figure A.7.

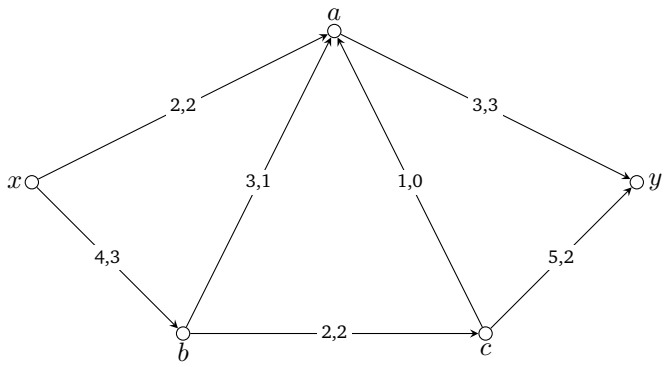


Figure A.7: A flow-network with flow and capacity on each edge.

(a) Find all x-y cuts.

(b) Find the capacity of a minimum x-y cut.

(c) Show that the flow indicated is a maximum x-y flow.

E7.4 (BM 11.2.3) If $(S, \overline{S})$ and $(T, \overline{T})$ are minimum cuts in N, show that $(S \cup T, \overline{S \cup T})$ and $(S \cap T, \overline{S \cap T})$ are also minimum cuts in N.

E7.5 (BM 11.2.2) Show that, if there exists no directed (x, y)-path in N, then the value of a maximum flow and the capacity of a minimum cut are both zero.

E7.6 Consider the following algorithm for finding a minimum covering in a bipartite graph: Take a vertex x of maximum degree and delete it. In the new graph, take a vertex y of maximum degree and delete it. (If there is a choice, take one on the same side as x if possible.) Continue like this. Stop when all vertices have degree 0. Prove that there is a tree on 7 (or less?) vertices which shows that the algorithm does not always find a minimum covering.

A.8 Week 8

E8.1 Find the critical edges and edges with optimum capacity in the graphs in Figure A.8.

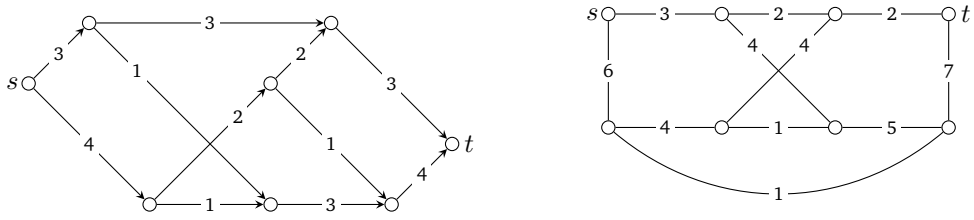
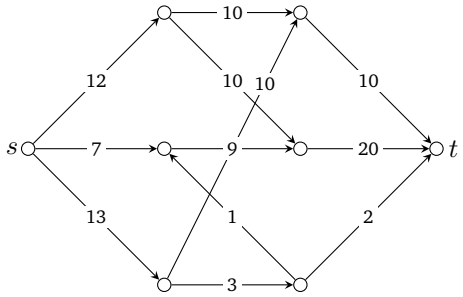


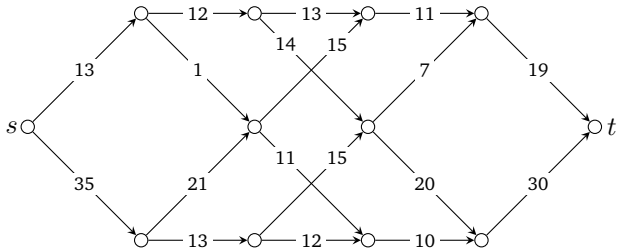
Figure A.8: Two networks with source and sink denoted s and t, respectively.

E8.2 (FN 3.8) Find the critical edges and edges with optimum capacity in the following network.



E8.3 (FN 3.4) Consider the network shown below. Find a maximum flow and find a minimum cut. Then find the number of maximum integer flows.

(Hint for the last question: Start by finding the critical edges and edges with optimum capacity.)



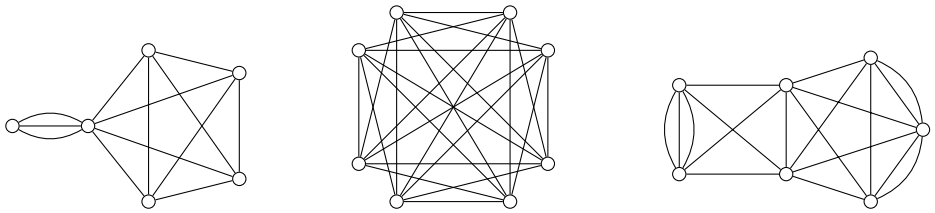
E8.4 If every edge has flow 0, then the value of the flow is clearly zero. Is the converse true? (In other words, is it possible that the value of a flow can be zero, although some of the edges have positive flow?)

E8.5 If s, t are vertices in a network N, then describe an algorithm for deciding if there is a flow of value 0 such that some of the edges have positive flow.

E8.6 Let f be a flow from s to t in a network N. Divide the vertices of N into two sets X, Y such that X contains both s and t. Prove that the sum of flow values from X to Y equals the sum of flow values from Y to X.

A.9 Week 9

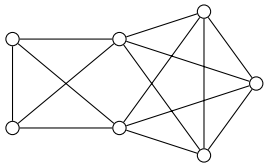
E9.1 (1.21 FN) Find the connectivity, edge-connectivity and minimum degree of the following graphs



(a) G_1 (b) G_2 (c) G_3

E9.2 Draw all the spanning trees of K_4

E9.3 Find the number of spanning trees in



E9.4 Find the number of spanning trees in $K_{4,3}$.

E9.5 Derive Theorem 6.1 from the following Lemma

> **Lemma A.1** *Let x and y be two non-adjacent vertices of a graph G. Then the maximum number of internally-disjoint (x, y)-paths in G is equal to the minimum number of vertices whose deletion destroys all (x, y)-paths.*

E9.6 Let G be a graph and let S and T be two disjoint subsets of $V(G)$. Show that the maximum number of vertex-disjoint paths with one end in S and one end in T is equal to the minimum number of vertices whose deletion separates S from T (that is, after deletion no component contains a vertex of S and a vertex of T).

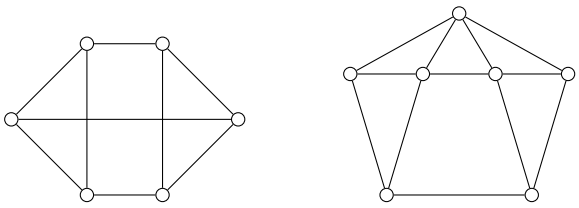
E9.7 Suppose that G is a k-connected graph and that S is a set of $k + 1$ vertices in G. Can you be sure that G has a cycle which contains all vertices in S? (The cycle is allowed to contain some other vertices as well.)

E9.8 Show that if G is k-connected with $k \geq 2$, then any k vertices of G are contained together in some cycle.

A.10 Week 10

E10.1 Read an understand the proof of Theorem 2.8 in BM page 33.

E10.2 Find the number of spanning trees in the graphs below. Try also the Petersen graph if you can. Use the method of Theorem 2 in BM page 33 except for the Petersen graph, for which you probably need a calculator.



E10.3 (BM 1.2.13) A simple graph G is vertex transitive if, any two vertices u and v, are equivalent under some element of its automorphism group. Informally speaking, a graph is vertex-transitive if every vertex has the same local environment, so that no vertex can be distinguished from any other based on the vertices and edges surrounding it. The same definition applies for edge transitivity. Find a graph, which is vertex-transitive but not edge-transitive. Then find a graph which is edge-transitive, but not vertex-transitive.

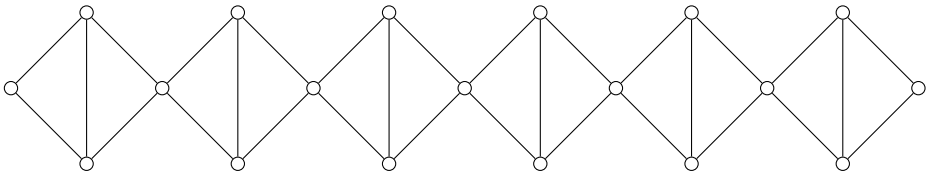
E10.4 Prove that if G is a connected edge-transitive graph with n vertices and m edges, and e is any edge of G, then the number of spanning trees containing e is $(n-1)\tau(G)/m$. Use this to show that if e is an edge of K_n, then $\tau(K_n - e) = (n-2)n^{n-3}$.

E10.5 Use Kirchoff's tree theorem, to prove Scoin's formula:

$$\tau(K_{n,m}) = n^{m-1}m^{n-1}$$

Here $K_{n,m}$ is the complete bipartite graph with n vertices in one part, and m vertices in the other part.

E10.6 Find the number of spanning trees in the following graph on 19 vertices.



E10.7 Find a closed formula for the number of spanning trees in the wheel graph.

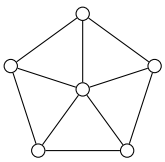
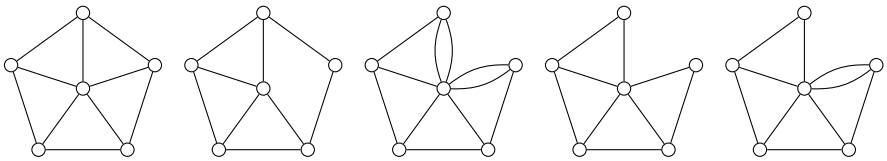


Figure A.9: The wheel graph with 5 spokes.

We consider five families of graphs, as indicated below. Each graph contains an edge e such that the contraction or deletion of e results in a graph in one of the other families. Now we can use the contraction-deletion formula to write a system of recurrence relations for the number of spanning trees in these graphs. We write w_n to denote the wheel with n nodes. Try to find a system of relations. Try to solve it.

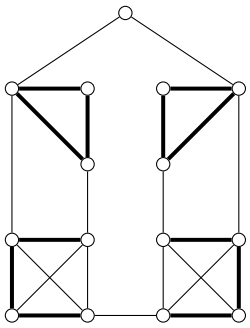


(a) w_6 (b) a_6 (c) b_6 (d) d_6 (e) e_6

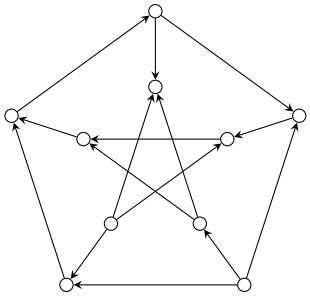
Figure A.10: The five families of graphs we use to find an explicit formula for the number of spanning trees in the wheel. We use the Contraction-Deletion Theorem on the emphasized edges, to find a system of recurrence relations.

A.11 Week 11

E11.1 (5.6 FN) Let A be the set of bold edges in the graph G below. Draw $G - A$ and G/A.



E11.2 (5.11 FN) Find the rank of the adjacency matrix A, the cycle matrix C and the cut matrix D for the graph below.

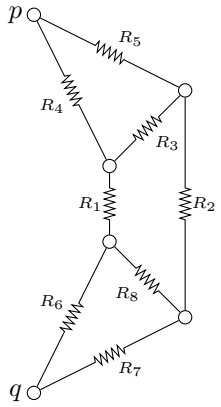


E11.3 (6.3 FN) Find, if possible, a resistance network with K edges, such that its determinant is

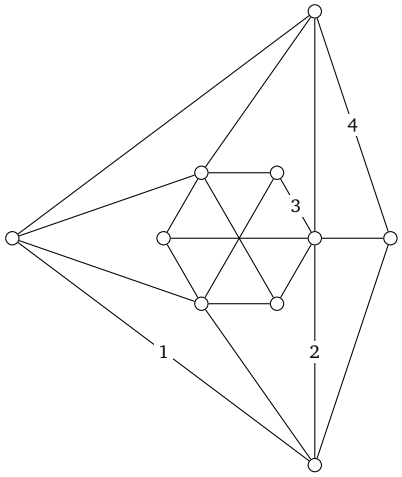
1. $R_1R_2 + R_1R_4 + R_1R_5 + R_2R_3 + R_3R_4 + R_3R_5$, $K = 5$

2. $R_1R_2R_5 + R_3R_5 + R_4R_5$, $K = 5$
3. $R_1R_2 + R_1R_3 + R_2R_3$, $K = 3$
4. $R_1R_2 + R_3R_4 + R_1R_3$, $K = 4$
5. $R_1R_2 + R_1R_3 + R_1R_4 + R_2R_3 + R_2R_4 + R_3R_4$, $K = 4$

E11.4 (6.5 FN) In the network below, find the driving point resistance R between p and q as a function of x and y, when $R_1 = x$,$R_2 = y$ and $R_i = 1$ for $i = 3, 4, 5, 6, 7, 8$.



E11.5 (6.7 FN) In an electrical network like the one below, the edge 1 is a voltage generator with voltage 1, and the remaining edges are resistors. For $j = 2, 3, 4$, answer the following: Is it possible to pick the resistors such that the current in edge j is 0?



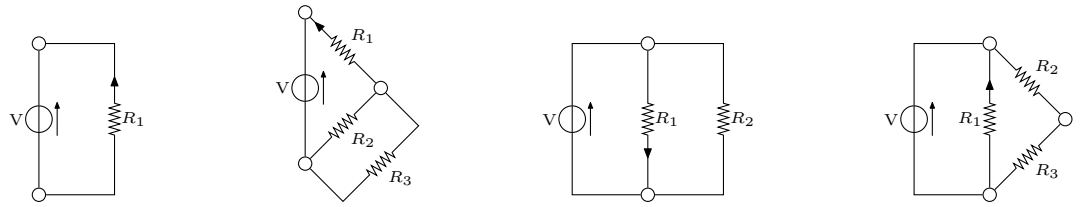
A.12 Week 12

E12.1 (a) Let a, b be diametrically opposite vertices of a cube and let c be a neighbor of a. Find the probability that a random walk starting at c gets to a before b.

(b) Let a, b be vertices of distance 2 in a cube and let c be a neighbor of a but not a neighbor of b. Find the probability that a random walk starting at c gets to a before b.

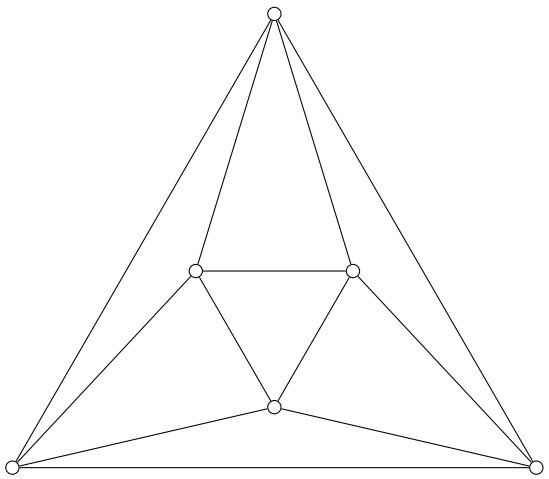
(c) Let a, b be neighboring vertices in a cube. For each third vertex c, find the probability that a random walk starting at c gets to a before b.

E12.2 For each of the following circuits, find the network determinant and then use Kirchhoff's rule to find the current through the resistor R_1 in the direction indicated.



(a) (b) (c) (d)

E12.3 For any three vertices a, b, c in the octahedron, find the probability that a random walk starting at c gets to a before b.



A.13 Week 13

E13.1 (BM 9.2.2) A plane graph is self-dual if it is isomorphic to its dual

1. Show that if G is self-dual, then $e(G) = 2v(G) - 2$.
2. For each $n \geq 4$, find a self-dual plane graph on n vertices.

E13.2 (BM 9.2.6) A plane triangulation is a plane graph in which each face has degree three. Show that every simple plane graph is a spanning subgraph of some simple plane triangulation ($n \geq 3$).

E13.3 (BM 9.2.1)

1. Show that a graph is planar if and only if each of its blocks is planar.
2. Deduce that a minimal nonplanar graph is a simple block.

E13.4 (BM 3.2.4) Show that a connected graph which is not a block has at least two blocks that each contain exactly one cut-vertex.

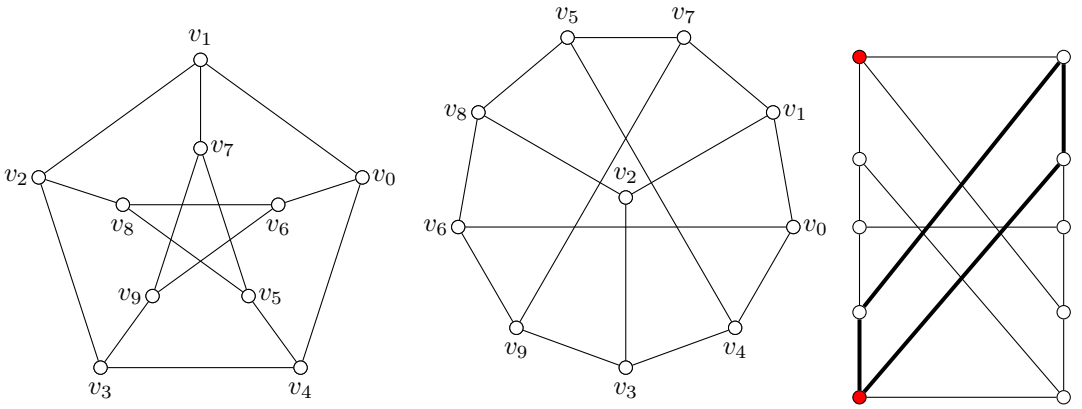
E13.5 (BM 3.2.3) Show that if G has no even cycles, then each block of G is either K_2 or an odd cycle.

B

SOLUTIONS FOR WEEKLY EXERCISES

B.1 Week 1

E1.1 The first two graphs are isomorphic. A bijection between them is shown below (there are several other bijections). The third graph is not isomorphic to the Petersen Graph. To prove this, one needs to find a property of this graph not shared by the Petersen Graph. Sometimes this is very easy. For example if the number of vertices or edges differ, or if the degree sequences are not concordant, then clearly the two graphs can not be isomorphic. Unfortunately in this case all these properties are the same. However, look at the bold cycle in the third graph. This cycle has length four, but careful inspection reveals that the Petersen Graph does not have a cycle of length four. Consequently the third graph is not isomorphic to the Petersen Graph. Another property that differs between the two graphs is the diameter. To find the diameter of a graph, consider the shortest distance between all pairs of vertices u and v in G. The diameter is then the longest of these distances. The Petersen Graph is vertex transitive (all vertices have the same properties), so to find the diameter you can take an arbitrary vertex and consider the shortest distances from that to every other vertex. None of these distances are greater than 2, so the diameter of the Petersen Graph is 2. But the shortest distance between the two red vertices in the third graph is 3, so the diameter of this graph must be at least 3 (in fact it is exactly 3).

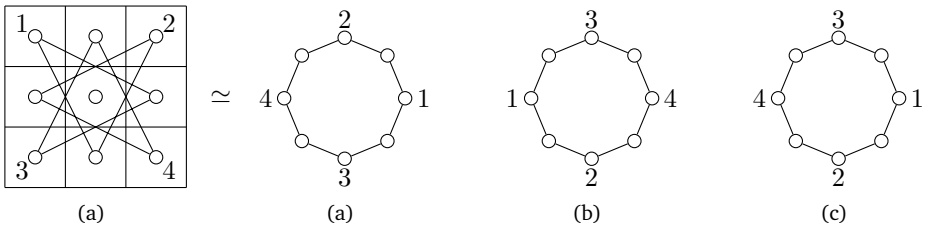


E1.2 No. If G contains a bridge xy, the vertex set of G can be partitioned into a left and

right part A and B such that xy is the only edge between A and B. Suppose that G is eularian and contains the euler tour C. Start in any vertex of C and follow the euler tour. C includes all edges in G exactly once, so at some point it will cross the edge xy. The tour can not return to the start vertex, since there are no other edges between A and B, contradicting that C is an euler tour.

An alternative definition of eulerian is the following: "A graph G is eulerian if all vertices have even degree". With this definition, we can prove the statement in another way. Consider a graph G with a bridge xy. In an eulerian graph, all vertices have even degree. Now remove the bridge from G to split G into two components A and B, this will change the degree of x and y to be odd. Now look at A, and lets assume x is in A, in A, all vertices have even degree, except for x which have odd degree, but this is a contradiction since the number of vertices of odd degree in any component is always even.

E1.3 The configuration (b) can be obtained from (a) (in 16 moves), but (c) can not. To see why, consider the graph of the permitted moves shown below. By rearranging the nodes in the movement graph, it becomes clear that the graph is a cycle. Moving according to the rules, the order of the knights on this cycle can not be changed (overtakings are impossible), showing that configuration (c) can not be obtained from (a). By moving each knight 4 times (a total of 16 moves), configuration (a) can be transformed into (b).



E1.5 Yes and no. In any tree, all edges are cut-edges. Every connected graph G contains a non-cut vertex. Consider a spanning tree T in G and take any leaf v (a vertex of degree 1) in that tree. Such a leaf can not be a cut-vertex in G, because $T - v$ is also a spanning tree of $G - v$, and hence the graph stays connected after the removal of v.

In case you are not that confident about spanning trees yet, you can find a non-cut vertex in G as follows: If G has no cycle then G is a tree. Take any leaf in that tree. Otherwise G contains one or more cycles. In that case consider a cycle and remove an edge in that cycle. This will not disconnect G. Continue removing edges from cycles until G contains no cycles. The final graph is connected and has no cycle, so it is a tree. Now consider a leaf in this tree. Removing this leaf from the original graph will not disconnect it, since all the other vertices stays connected (e.g. by a path in the tree).

E1.6 This is equivalent (by contraposition) to the following statement:

> If G is *not* connected then it is *not* the case that $d(v) \geq \frac{1}{2}(n-1)$ for all vertices.

Proof (by contradiction) Assume G is disconnected and $d(v) \geq \frac{1}{2}(n-1)$ for all n vertices. Since G is not connected it contains at least two components A and B. Consider a vertex $v \in A$. Since $d(v) \geq \frac{1}{2}(n-1)$, A must in addition to v contain at least $\frac{1}{2}(n-1)$ vertices. That is $|A| \geq \frac{1}{2}(n-1)+1$. By the same argument $|B| \geq \frac{1}{2}(n-1)+1$. So the number of vertices in G is at least $|A| + |B| = n + 1$, which is a contradiction.■

The bound is best possible in the sense that the following statement is false:

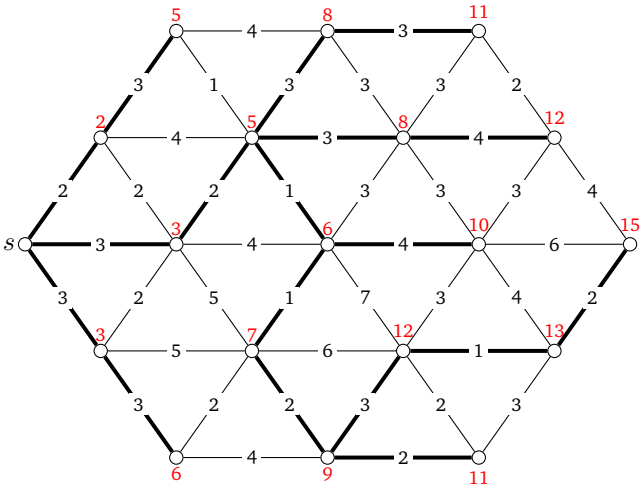
A simple graph G is connected if $d(v) \geq \frac{1}{2}(n-1) - \frac{1}{2}$ for all $v \in V(G)$.

A counterexample is the disconnected graph containing two disjoint copies of $K_{\frac{n}{2}}$ where $n > 2$ is even. In that graph every node has degree $\frac{n}{2} - 1 = \frac{1}{2}(n-1) - \frac{1}{2}$, but the graph is not connected.

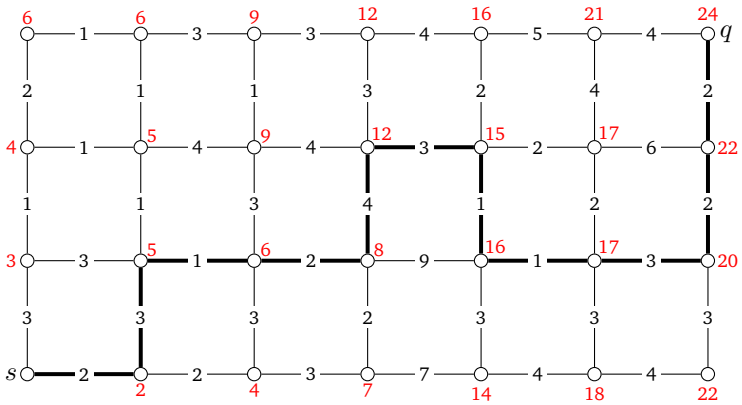
On a side note: The statement proven in this exercise is not very strong. We can say a lot more about a graph G with minimum degree $\frac{1}{2}(n-1)$ besides it being connected. In fact any two nodes in G will be connected by a path of length 1 or 2, and furthermore G will contain a Hamilton path! We leave the proofs of these two claims to the reader (the latter can be derived from Dirac's Theorem).

B.2 Week 2

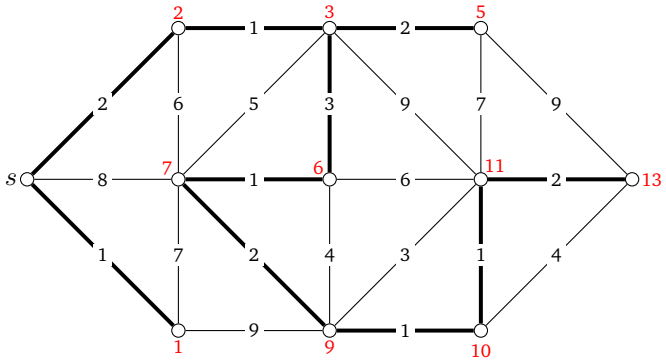
E2.1 Below all vertices are labeled with their distance from s (in red). The shortest path tree rooted at s is highlighted in bold. Note that the shortest path tree is not unique.



E2.2 Below all vertices are labeled with their distance from s (in red). The shortest path from s to q is highlighted in bold and has length 24.



E2.3 Below all vertices are labeled with their distance from s (in red). The shortest path tree rooted at s is highlighted in bold.



E2.4 We can extend the instructions in the algorithm in two different ways.

One way is to maintain which edges are part of shortest paths while the algorithm is running. When the label $l(v)$ of a node v is updated, we also note from which vertex $a(v)$ the vertex v is updated. We will use $a(v)$ to denote the vertex which updated v, namely the *ancestor* of v.

It is also possible to start by running Dijkstra's Algorithm and then use the algorithm in Section 3.1.1 to compute the actual shortest paths.

E2.5 To determine the components of a graph, we could use the following algorithm:

Algorithm 9 Finding the components of a graph G

1. Let $C = 1$
2. Pick an arbitrary vertex v in G and find the distance classes from v.
3. Label the vertices in every distance class with the number C (they are in the C^{th} component)
4. Remove all the vertices found. If G has more vertices, increase C by one and go to step 2.

When the algorithm stops, each vertex is labeled by the number of the component it is part of. The number C will be the total number of components in G. The algorithm considers each vertex and each edge a constant number of times, so the running time is $O(n + m)$, where n and m is the number of vertices and edges, respectively, in G.

To determine the girth of a graph G, the following algorithm could be used:

Algorithm 10 Finding the girth of a graph G

1. Let $L = \infty$.
2. For each edge $xy \in E(G)$ do:
   Find the shortest distance d from x to y in $G' = (V(G), E(G) \setminus xy)$.
   If $d + 1 < L$ then let $L = d + 1$.
3. If $L \neq \infty$ then
   Return L
   else Return `"No cycle in` G`"`

For each of the m edges in G, the algorithm computes the shortest distance between two vertices. Using Dijkstra's algorithm, computing this distance can be done in $O(n^2)$ time, so total running time of the algorithm becomes $O(mn^2)$. To get a better bound, Dijkstra's algorithm can be implemented to run in $O(m + n \log n)$ time, in which case the running time becomes $O(m^2 + mn \log n)$. As it suffices to use distance classes instead of Dijkstra's algorithm, the complexity can even be lowered to $O(m^2)$.

E2.6 The first step in solving this exercise is to understand the relationship between $\boldsymbol{d}$ and $\boldsymbol{d'}$. Let d_1 be the first number in the sequence $\boldsymbol{d}$. The formal definition $\boldsymbol{d'} = (d_2 - 1, d_3 - 1, \ldots, d_{d_1+1} - 1, d_{d_1+2}, \ldots, d_n)$ can be a little confusing, but it simply means: To construct $\boldsymbol{d'}$, take the d_1 numbers after d_1 in $\boldsymbol{d}$ and subtract 1 from each of them. These numbers are the first numbers in $\boldsymbol{d'}$ (or more precisely, in a permutation of $\boldsymbol{d'}$). The remaining numbers of $\boldsymbol{d'}$ are the same as in $\boldsymbol{d}$. For example let $\boldsymbol{d} = (4, \underline{4, 4, 3, 2}, 2, 2, 1)$, then $\boldsymbol{d'}$ consists of the 4 numbers succeeding the first element of $\boldsymbol{d}$ (the underlined numbers), but where 1 is subtracted from each of these numbers. Thus $\boldsymbol{d'}$ is $(3, 3, 2, 1, 2, 2, 1)$ and after the permutation $\boldsymbol{d'} = (3, 3, 2, 2, 2, 1, 1)$, since the sequence must be non-increasing. Now let us consider how to solve this exercise.

(a) First, we should prove that $\boldsymbol{d}$ is graphic if and only if $\boldsymbol{d'}$ is graphic. One direction is easy: Suppose that $\boldsymbol{d'} = (d'_1, d'_2, \ldots, d'_n)$ is graphic and consider a simple graph G having this degree sequence. Now add a vertex to G and connect it to the first $d'_1 + 1$ nodes of $\boldsymbol{d'}$. This new graph will then have the non-increasing degree sequence $\boldsymbol{d}$.

The other direction is more interesting. We must show that if there is a graph with degree sequence $\boldsymbol{d}$, then there is also one with degree sequence $\boldsymbol{d'}$. So let us assume that $\boldsymbol{d}$ is graphic, and investigate a simple graph G having this degree sequence. Consider the vertex v corresponding to d_1 (i.e. v has degree d_1). If this vertex is adjacent to the vertices corresponding to $d_2, d_3, \ldots, d_{d_1+1}$ as illustrated in Figure B.1 then we are done, since removing v from G will create a graph with degree sequence $\boldsymbol{d'}$.

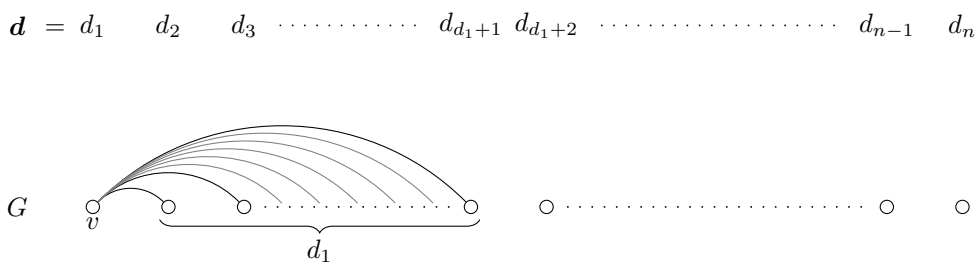


Figure B.1: If G has the above structure, we can simply remove v to obtain a new simple graph with the degree sequence $\boldsymbol{d'} = (d_2 - 1, d_3 - 1, \ldots, d_{d_1+1} - 1, d_{d_1+2}, \ldots, d_n)$.

So let us assume that v is not adjacent to at least one of the vertices corresponding to $d_2, d_3, \ldots, d_{d_1+1}$. Suppose that x is such a vertex, then v must also be adjacent to a vertex z corresponding to one of $d_{d_1+2}, d_{d_1+3}, \ldots, d_n$, since if this was not the case $d(v) < d_1$, which is a contradiction. Since $\boldsymbol{d}$ is non-increasing $d(x) \geq d(z)$ and from this fact follows a very important observation:

There must be a vertex y adjacent to x but not to z.

If this is not the case, the degree of z would be strictly greater than that of x, which is a contradiction. The situation can be illustrated as shown in Figure B.2.

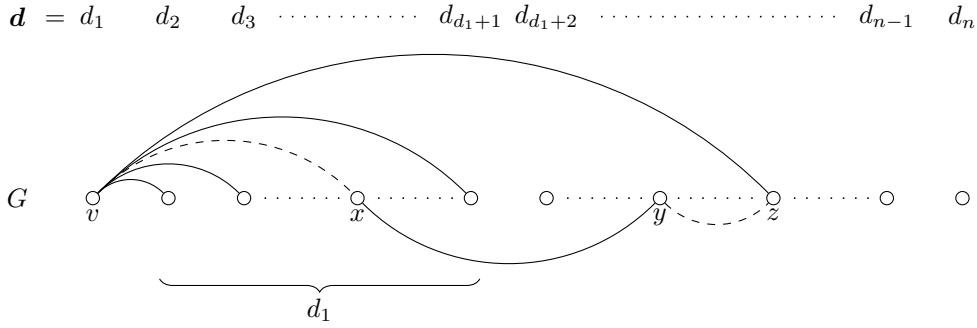


Figure B.2: The case when v is not adjacent to all the vertices corresponding to $d_2, d_3, \ldots, d_{d_1+1}$. The dashed edges vx and yz illustrates missing edges. Note that y can be anywhere in G.

We now remove the edges xy and vz from G and insert the edges vx and yz. This operation causes v and x to become adjacent without changing the degree sequence of G. Note that none of the edges from v to $d_2, d_3, \ldots, d_{d_1+1}$ are removed, so we can repeat this operation until v is adjacent to all of these vertices. Then we have the situation illustrated in Figure B.1, and we then remove v to obtain a graph having the degree sequence $\boldsymbol{d'}$.

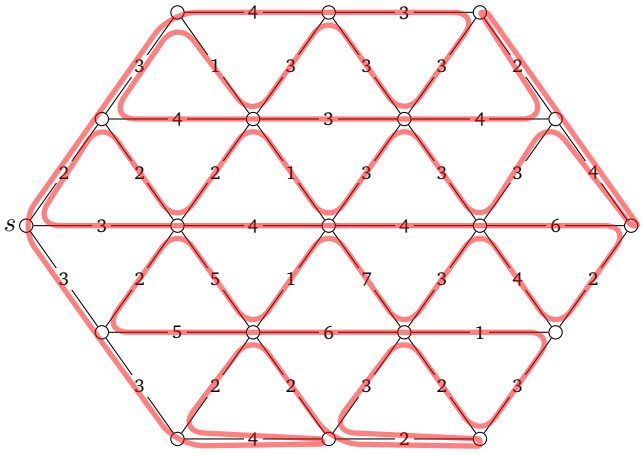
(b) Given a sequence $\boldsymbol{d}$, the construction given in the previous proof also provides an algorithm for finding a graph having this degree sequence, or proving that no such graph exists. The algorithm is as follows: From $\boldsymbol{d}$ construct $\boldsymbol{d'}$ and repeat until you have a sequence consisting of only 1's and 0's (or until you have a sequence that can not be formed, e.g. $(4, 1, 1, 0, 0, 0)$, in which case $\boldsymbol{d}$ is non-graphic). This sequence of 1's and 0's is graphic if and only if the number of 1's is even. In that case, construct a

graph G' (a matching) with this degree sequence by pairing vertices (If the number of 1's is odd, then by the theorem, $\boldsymbol{d}$ can not be graphic). Now use the construction from the proof to add vertices to G' until you get a graph with degree sequence $\boldsymbol{d}$.

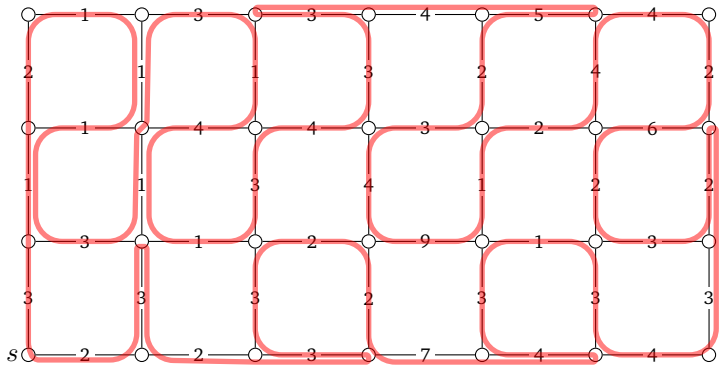
B.3 Week 3

E3.1 When the input graph does not have distinct edge weights, there might be a choice of which edge to add next in Kruskal's algorithm, and this choice can lead to different minimum spanning trees in G. In this exercise you simply have to verify that there is an execution of Kruskal's algorithm that results in the indicated spanning tree.

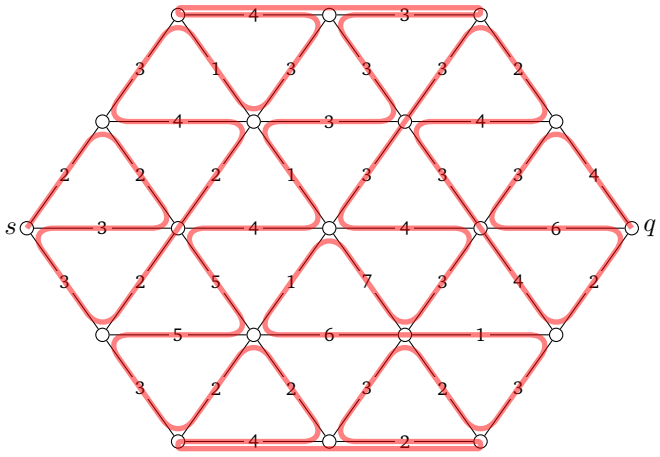
E3.2 A minimum tour in the graph is shown below (there are many others). The waste graph is K_6 and contains a minimum perfect matching of size $5 + 6 + 6 = 17$. The total edge weight is 130, so the tour has length $130 + 17 = 147$.



E3.3 A minimum trail is shown below (there are many others). The waste graph is K_{14} and contains a minimum perfect matching of size $1+5+3+3+5+4+2 = 23$. The waste graph is quite big, so you should spend a while convincing yourself that no smaller matching exists. The total edge weight is 130, so the tour has length $130 + 23 = 143$.



E3.4 A minimum trail is shown below (there are many others). The waste graph is K_4 and contains a minimum perfect matching of size $7 + 6 = 13$. The tour has length $130 + 13 = 133$.



In general when the postman starts and ends in distinct vertices s and q, the goal is to create a graph having an *euler trail* between s and q. This is the case if and only if the degrees of s and q are odd and the degrees of all other vertices are even. We normally add all vertices of odd degree to the waste graph W, since this causes the degree of these vertices in G to be increased by one, when the minimum matching in W is added to G. Hence, in this case we should only add s to W iff $d(s)$ is even (and the same goes for q).

Another and maybe more convenient way to achieve exactly the same thing is as follows. Start by adding an extra edge e between s and q having a very large weight say $w(w) = \infty$ (or if you don't like ∞, take a number greater than the sum of all weights), and denote this graph as G'. Then solve the Chinese Postman problem as usual on G' to obtain an euler tour. Finally remove the edge between s and q to get an euler trail between s and q (since in G', e will only be traversed once).

E3.5 If all edge weights are positive, a spanning tree which minimizes the sum of the edge weights will also minimize the product, so we can just use Kruskal's algorithm as usual.

Proof Let G be a connected graph with positive edge weights, and let T be a spanning tree returned by Kruskal's algorithm that minimizes the sum of the edge weights. We denote the edges in T as $e_1, e_2, \ldots, e_{n-1}$. Now suppose you replace all edge weights in G by their logarithmic value and then run Kruskal's algorithm on this modified graph. Since $\log$ is an increasing function, taking the logarithm does not change the order in which Kruskal's algorithm considers the edges, so Kruskal's algorithm will return the same spanning tree T as for the original graph. This means that T also is the spanning tree where

$$\log w(e_1) + \log w(e_2) + \ldots + \log w(e_{n-1}) = \log(e_1 \cdot e_2, \cdot \ldots \cdot e_{n-1})$$

is smallest possible. Again, due to the monotonicity of logarithms, this implies that T is also the spanning tree in which $e_1 \cdot e_2 \cdot \ldots \cdot e_{n-1}$ is smallest possible. ∎

E3.6 No and yes. Prim's algorithm is not the same as Kruskal's algorithm, since Kruskal's algorithm grows a forest and Prim's algorithm grows a tree.However, as Kruskal's algorithm, Prim's algorithm will always find a spanning tree of smallest total weight. The proof, stated below, is similar to that for Kruskal's algorithm.

Proof (by contradiction) Consider a connected weighted graph G. Let T_P be the spanning tree returned by Prim's algorithm, let and T_M be a minimum spanning tree in G. Assume for the sake of a contradiction that $w(T_M) < w(T_P)$. Among all the minimum spanning trees in G, we choose T_M such that it agrees with T_P for as long as possible. This means that T_M is the minimum spanning tree that maximizes the time before Prim's algorithm adds an edge to T_P, which is not in T_M. We now show how to construct a minimum spanning tree T' that agrees with T_P one step longer than T_M, thereby reaching a contradiction.

Consider the point where Prim's algorithm picks an edge xy (where $x \in T_P$ and $y \notin T_P$), which is not in T_M. At that point $T_P \subseteq T_M$. In T_M there must be a path from y to T_P in T_M (otherwise T_M is not a spanning tree). Let zw be the last edge on this path (i.e. $w \in T_P$). This edge is not in T_P, but at the time when Prim's algorithm chose xy it could have chosen zw instead. This implies that $w(xy) \leq w(zw)$. Now suppose you add the edge xy to T_M. This creates a fundamental cycle that also contains the edge zw. Now consider the tree $T' = T_M + xy - zw$. This is a spanning tree that agrees with T_P one step more than T_M, so we are done if we can show that this tree is also minimum. Since T_M is minimum, $w(T_M)$ is less than or equal to the weight of any other spanning tree. In particular we have that

$$w(T_M) \leq w(T') = w(T_M) + w(xy) - w(zw) \leq w(T_M) \ ,$$

where the last inequality comes from the fact that $w(xy) \leq w(zw)$. This implies that $w(T') = w(T_M)$, and thus T' is a minimum spanning tree. ∎

B.4 Week 4

E4.1 Consider the matching of size 30 shown below. The two unsaturated vertices are both white, implying that this is a maximum matching.

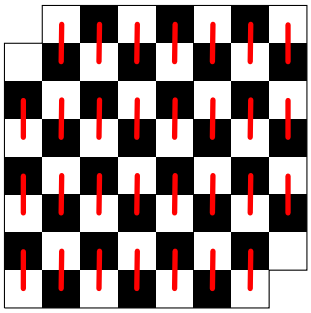


Figure B.3: A maximum matching.

E4.2 If k is even, just consider K_{k+1}. This graph is k-regular but contains no perfect matching, since it has an odd number of vertices. If k is odd, we can use the construction illustrated in Figure B.4.

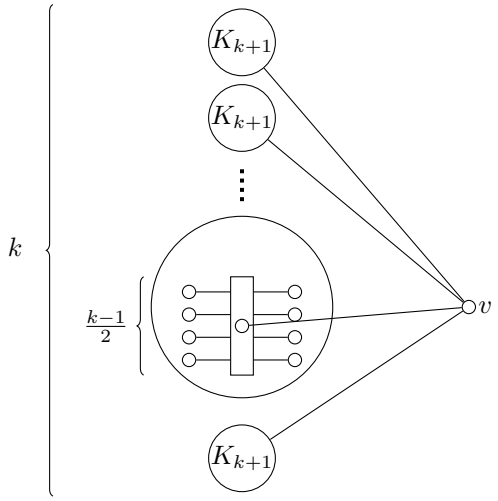


Figure B.4: For odd $k > 1$, a k-regular graph with no perfect matching can be constructed as follows. Create k copies of K_{k+1} and denote them $C_1, C_2, \ldots, C_k$. Let M_i be a matching of size $\frac{k-1}{2}$ in C_i. For each edge in M_i insert a vertex on the middle, and glue those vertices together. Finally create a single vertex v, and join it to the special vertex we created in each of the components $C_1, \ldots, C_k$.

Let G be the graph obtained by the above construction, we must argue that

- $\boxed{K_{k+1} \text{ contains a matching of size } \frac{k-1}{2}}$. In fact, since any two vertices are connected, K_{k+1} contains a matching of size $\frac{k+1}{2}$.
- $\boxed{G \text{ is } k\text{-regular}}$. The degree of v is k, since it is joined to the special vertex in each of the k components $C_1, C_2, \ldots, C_k$. The degree of the special vertex is $2 \cdot \frac{k-1}{2} + 1 = k$. The remaining vertices also have degree k, since they came from K_{k+1}.
- $\boxed{G \text{ does not contain a perfect matching}}$. Suppose G has a perfect matching and let C_i be the component to which v is matched. Now consider some C_j, $j \neq i$. This component has $k + 2$ vertices, which is odd, so the matching can not saturate all vertices in C_j.

E4.3 That a tree has at most one perfect matching can be proved by contradiction as follows.

Proof Consider a tree T and suppose T contains two perfect matchings M_1 and M_2. Now consider $H = M_1 \Delta M_2 = (M_1 \cup M_2) \setminus (M_1 \cap M_2)$. In the general case, the components of H are either:

1. A path with edges alternately in M_1 and M_2, or
2. An even cycle with edges alternately in M_1 and M_2,

but since T is a tree 2. is not possible, so all components of H must be of the first type. Now consider a longest path in H and let u denote one of the vertices of degree 1. Since u can not be both M_1- and M_2-saturated, M_1 and M_2 can not both be perfect matchings. ∎

E4.4 Let $G = (V, E)$ be a bipartite graph and let (A, B) denote the bipartition of V. We must prove that G has a perfect matching if and only if $|N(S)| \geq |S|$ for all $S \subseteq V$. Remember that $N(S)$ is the set of vertices directly joined to a vertex in S.

Proof The forward direction is easy. Suppose that G has a perfect matching M and consider any set $S \subseteq V$. For each $v \in S$, take the vertex that v is matched to under M. This gives a set of size $|S|$, so $|N(S)| \geq |S|$.

To prove the other direction assume that $|N(S)| \geq |S|$ for any $S \subseteq V$. By Hall's theorem (see Theorem 6.3) we then know that there is a matching M saturating all vertices in A. We now show that $|A| = |B|$ implying that M is a perfect matching. By our assumption $|A| \leq |N(A)|$ and $|B| \leq |N(B)|$. Also, since G is bipartite $|N(A)| \leq B$ and $|N(B)| \leq |A|$. Together this gives

$$|A| \leq |N(A)| \leq |B| \leq |N(B)| \leq |A| \,,$$

implying that $|A| = |B|$. ∎

If G is not bipartite, the above theorem is not true. A counterexample is K_3. Here $|N(S)| \geq |S|$ for any $S \subseteq V$, but the graph has no perfect matching.

E4.5 Let M denote the $(0, 1)$-matrix. Consider the bipartite graph $G = (A \cup B, E)$ where each row in M has a corresponding vertex in A and each column has a corresponding vertex in B. Two vertices $x \in A$ and $y \in B$ are joined by an edge iff $M[x, y] = 1$. Observe, that in this context

1. The minimum number of lines containing all the 1's in M is equivalent to the size of a minimum covering in G.
2. The maximum number of 1's, no two of which are in the same line is equivalent to the size of a maximum matching in G.

We know from the Kőnig-Egerváry theorem (see Theorem 6.1) that in any bipartite graph, the size of a maximum matching equals that of a minimum covering, thereby proving the claim about M.

E4.6 We must prove that the first player has a winning strategy if and only if G contains no perfect matching.

Proof Suppose G has a perfect matching, then the second player can win by using the following strategy

Strategy for the second player: Always pick the vertex matched to v_{i-1}.

Hence, in this case the first player has no winning strategy.

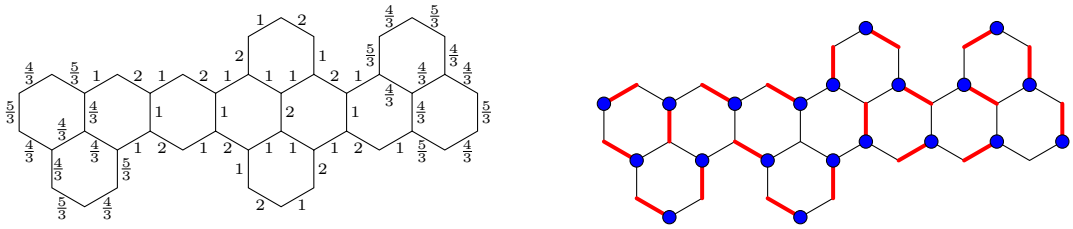
To prove the other direction, assume that G has no perfect matching and let M denote a maximum matching in G. Consider the following strategy for the first player:

Strategy for first player: Start by picking a vertex unsaturated by M, then always pick the vertex matched to v_{i-1} under M.

If the second player is able to pick the last vertex v_j, then this is because v_j is unsaturated by M, but then $v_1, v_2, \ldots, v_j$ is an M-augmenting path in G contradicting that M is maximum. Thus, the above strategy is a winning strategy for the first player, in case G has no perfect matching. ∎

B.5 Week 5

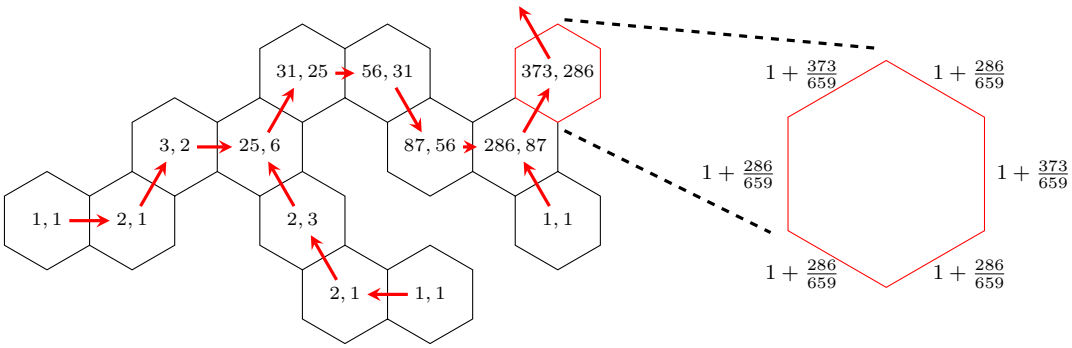
E5.1 (a) can be realized. It has 9 perfect matchings and the Pauling Bonds are as shown in Figure B.5(a). (b) does not have perfect matching and can not be realized. To see why, consider the matching M (shown in red) and the covering C (shown in blue) in Figure B.5(b). Since they have the same size, M is a maximum matching (and C is also a minimum covering). This is a consequence of Lemma 6.1.



(a) The Pauling Bonds. (b) A matching and a covering of size 20.

Figure B.5: Solution to **E5.1**.

E5.2 The solution is shown below.



(a) Finding the number of perfect matchings in the benzoid. (b) The Pauling bonds.

Figure B.6: Solution to exercise **E5.2**.

E5.3 We start by showing that if T has a perfect matching, then $o(T-v)=1$ for all $v \in V(T)$. Consider some vertex v in T.

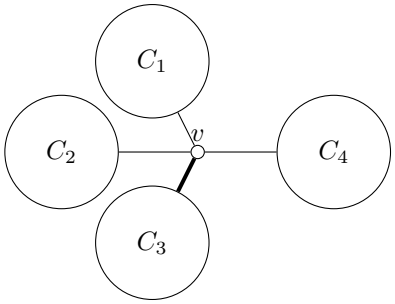


Figure B.7: T, where v is matched to a vertex in C_3. Here C_1, C_2, C_3 and C_4 are the connected components of $T - v$.

Let $C_1, C_2, \ldots, C_k$ be the connected components of $T - v$. Since T has a perfect matching, v is matched to a vertex in one of the components C_i. Inside all other components C_j such that $j \neq i$, there is a perfect matching, thus in C_j, the number of vertices must be even. Since $C_i + v$ has a perfect matching, C_i is the only component with an odd number of vertices, so $o(T - v) = 1$.

Now we show that if $o(T - v) = 1$ for all $v \in V(T)$, then T has a perfect matching. We construct a perfect matching as follows: Take an unmatched vertex a, then match a to the neighbouring vertex b in the odd component of $T - a$. b is unique due to our assumption. We must also show that b is matched to a, since otherwise the matching is not proper.

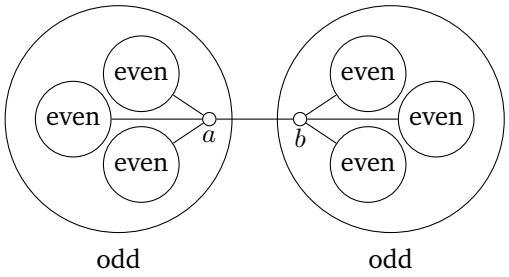
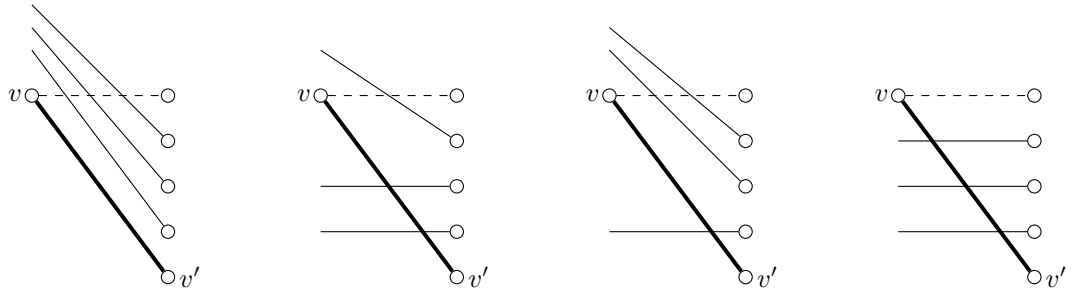


Figure B.8: When a is removed, the only odd component we get is the one containing b. When b is removed, the only odd component we get is the one containing a.

When removing the vertex b, the component X containing a contains a number of even components joined to a, so X is an odd component. Since $o(T - v) = 1$ for all $v \in T$, X is the unique odd component of $T - b$, so b will also be matched to a.

E5.4 Moving a brick horizontally, corresponds to moving the right endpoint of an edge e down one vertex in the bipartite graph. Thus, moving a brick horizontally, does not change the number of crossings. Moving a brick vertically can change the number of crossings in 4 different ways. By moving a brick vertically, the right endpoint of the edge e is moved 4 vertices in the bipartite graph. There are four cases to consider, and they are shown in Figure B.9.



(a) **Case 1:** 3 less crossings (b) **Case 2:** One more crossing (c) **Case 3:** One less crossing (d) **Case 4:** Three more crossings

Figure B.9: The four cases to consider when moving a brick vertically. In this Figure, the brick moved from the dashed line, to the bold line.

In Case 1, we reduce the number of crossings by 3. In Case 2, we increase the number of crossings by 1. In Case 3, we reduce the number of crossings by 1. In Case 4, we increase the number of crossings by 3. Thus the number of crossings can increase or decrease by 1 or 3. If an edge e had its left endpoint above v, and its right endpoint below v', or if it had its left endpoint below v and its right endpoint above u, then the number of edges crossing e does not change.

If we start in a situation where the empty square is in the lower right, and we end in a situation where the empty square is in the lower right corner, we know that the number of times the empty square has been moved up, is equal to the number of times the empty square is moved down. Thus, the number of vertical moves of the empty square is even. Since we make an even number of vertical moves with the empty square, the parity (even or odd) of crossings does not change.

In the last situation, the number of crossings is 1 in the start. Since we can never change the parity of the number of crossings, we can never solve the puzzle.

E5.5 We can look at the problem in the following way. Each set A has to pick a vertex $a \in A$ as its representative, such that no two sets share the same representative. We construct a bipartite graph $G = (X \cup Y, E)$ in the following way. For each of the sets A_i create a vertex v_i in X, and for each element a_j, create a vertex u_j in Y. Now we join a vertex $v_i \in X$ with a vertex $u_j \in Y$ if and only if the element a_j is in the set A_i.

Now a system of distinct representatives corresponds to a matching saturating all elements in X. Let S be a subset of $\{A_1, A_2, \ldots, A_m\}$. The union of the elements of S, $\bigcup_{A \in S} A$, corresponds to the neighbour set of the corresponding vertices in X. The condition that $|\bigcup_{i \in J} A_i| \geq |J|$ from the question, is equivalent to the condition that $|N(S)| \geq |S|$ for all subsets S of X. So the statement is equivalent to the one of Hall's Theorem and consequently true.

B.6 Week 6

E6.1 An optimal assignment of total sum 14 is shown in Table B.1.

5	2	③	3	3
3	④	1	1	1
⑥	4	2	1	4
2	4	1	①	1
2	3	0	0	

Table B.1: An optimal solution. The helping numbers are also shown in the matrix to verify that this is actually an optimal solution.

E6.2 An optimal assignment of total sum 33 is shown in Table B.2.

8	6	⑦	10	7
9	⑧	8	6	8
4	2	1	⑩	7
⑧	6	6	10	7
1	0	0	3	

Table B.2: An optimal solution. The helping numbers are also shown in the matrix to verify that this is actually an optimal solution.

E6.3 A maximum of 17 dominoes can be carved out of the board. This can be seen by considering the red matching and the blue covering shown in Figure B.10. Since they have the same size, the matching is maximum and the covering is minimum.

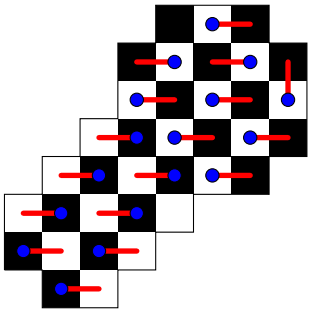
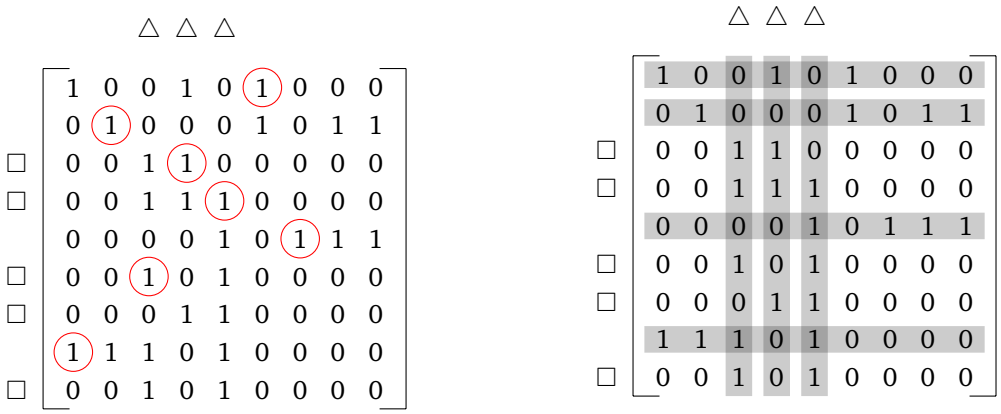


Figure B.10: A maximum matching in red and a minimum covering in blue.

E6.4 The structure rank of the matrix is 7 and corresponds to the size of a maximum matching between the rows and columns as shown in Figure B.11(a). The marking procedure stops as indicated with squares and triangles. The unmarked rows and the marked columns is a covering of size 7 as shown in Figure B.11(b). Consequently the matching is maximum and the covering is minimum.



(a) A maximum matching. (b) A minimum covering.

Figure B.11: Solution to **E6.4**.

B.7 Week 7

E7.1 Figure B.12 shows a maximum s-t flow F. The value of the flow is $|F| = 3 + 3 = 6$. The dashed line shows a s-t cut C of size $|C| = 1 + 1 + 1 + 3 = 6$. It follows from the Max-Flow-Min-Cut theorem that F is a maximum flow and C a minimum cut.

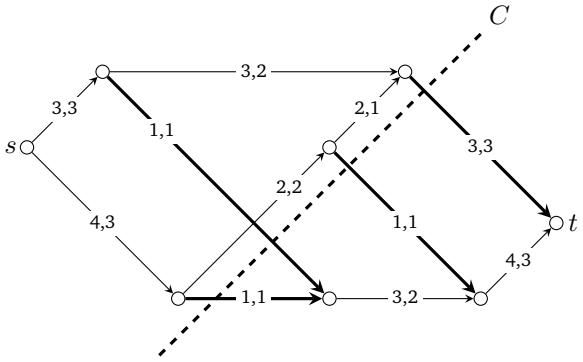


Figure B.12: Solution to **E7.1**.

E7.2 Figure B.13 shows a maximum s-t flow F (the arrows indicate the direction of the flow). The value of the flow is $|F| = 2 + 6 = 8$. The dashed line shows a s-t cut C of size $|C| = 3 + 4 + 1 = 8$. It follows from the Max-Flow-Min-Cut theorem that F is a maximum flow and C a minimum cut.

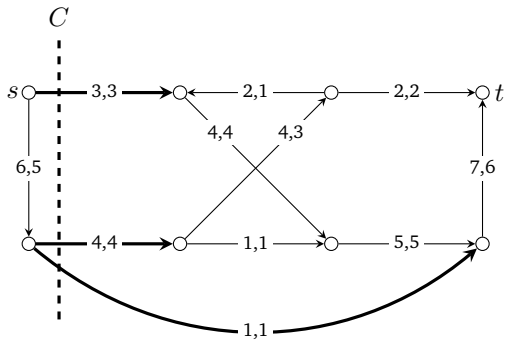


Figure B.13: Solution to **E7.2**.

E7.3 (a) There are 8 x-y cuts (A, B) such that $x \in A$ and $y \in B$. This can easily be seen by considering the three vertices a, b, c. For each of these we must make a binary choice (should it be in A or B). The following list shows the $2^3 = 8$ possible cuts and their capacities:

$$|\,(\{x\},\{a,b,c,y\})\,| = 6 \qquad |\,(\{x,a\},\{b,c,y\})\,| = 7$$
$$|\,(\{x,b\},\{a,c,y\})\,| = 7 \qquad |\,(\{x,c\},\{a,b,y\})\,| = 12$$
$$|\,(\{x,a,b\},\{c,y\})\,| = 5 \qquad |\,(\{x,b,c\},\{a,y\})\,| = 11$$
$$|\,(\{x,a,c\},\{b,y\})\,| = 12 \qquad |\,(\{x,a,b,c\},\{y\})\,| = 8$$

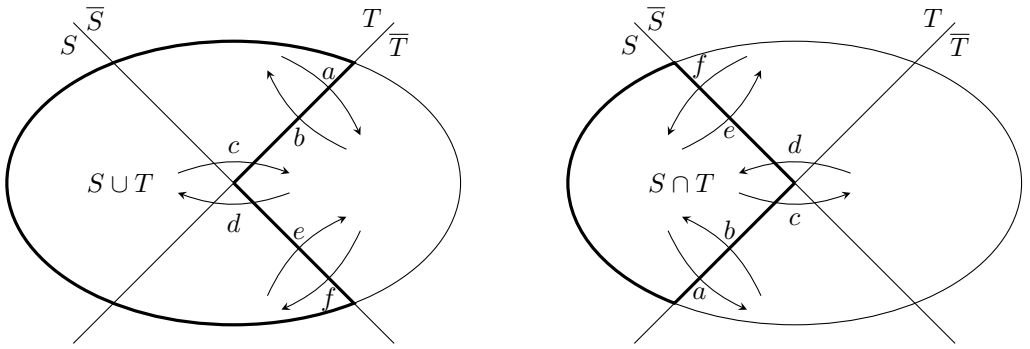
Not all of these cuts are easy to draw as a line.

(b) Inspecting the above cuts, we see that the minimum cut is unique and has capacity 5.

(c) The value of the indicated flow F is also 5, and since the value of any flow is less than the capacity of any cut, F is a maximum cut. Notice that we do not need the Max-Flow-Min-Cut theorem.

E7.4 Recall that having a maximum flow, a cut (A, B) is minimum if and only if all edges going from A to B are saturated and all edges from B to A has zero flow.

Suppose we have found a maximum flow in the graph and consider the cut $(S \cup T, \overline{S \cup T})$. The edges in this cut are of the six types a, b, c, d, e, f shown in Figure B.14(a). By considering each of these edges separately, we can show that a, c, d are saturated and b, d, f has zero flow, and hence that $(S \cup T, \overline{S \cup T})$ is a minimum cut. The edges a and c must be saturated since they are part of the minimum cut $(T, \overline{T})$, and e must be saturated because it is part of the minimum cut $(S, \overline{S})$. Likewise b and d has zero flow since they go from $\overline{T}$ to T, and f has zero flow because it goes from $\overline{S}$ to S. The exact same argument on the edges in Figure B.14(b) shows that $(S \cap T, \overline{S \cap T})$ is also a minimum cut.



(a) The edges in the cut $(S \cup T, \overline{S \cup T})$.

(b) The edges in the cut $(S \cap T, \overline{S \cap T})$.

Figure B.14: The edges in the two cuts $(S \cup T, \overline{S \cup T})$ and $(S \cap T, \overline{S \cap T})$.

E7.5 Since there is no (x, y)-path in N, there is some cut $C = (A, B)$ in N, where $x \in A$ and $y \in B$, and such that there are no edges from A to B. Hence the capacity of this cut is $|C| = 0$. Since the value of any flow is less than the capacity of any cut, the value of a maximum flow is also 0.

E7.6 Consider the tree on 7 vertices shown in Figure B.15.

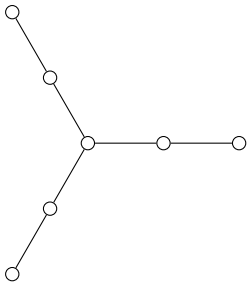


Figure B.15: A tree on 7 vertices on which our algorithm fails.

In the first step, the algorithm picks the vertex of degree 3, and then it is also forced to pick three of the six remaining vertices to get a covering. This gives a covering of size 4, but the graph has a covering of size 3 (take the three vertices of degree 2).

B.8 Week 8

E8.1 We start by finding a maximum flow in the graph, then we perform the marking process to find the red vertices and the green vertices. At last, we use the different rules to find the critical and optimal edges.

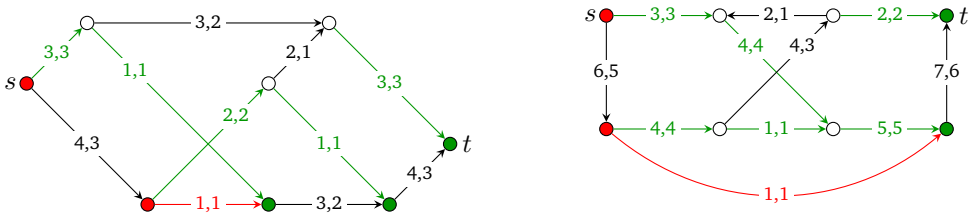


Figure B.16: The red edges are critical, and the green edges are optimal. The edges which are neither critical nor optimal are black.

E8.2 As in the last exercise, we find a maximum flow, mark the red and green vertices, and determine which edges are critical and optimal.

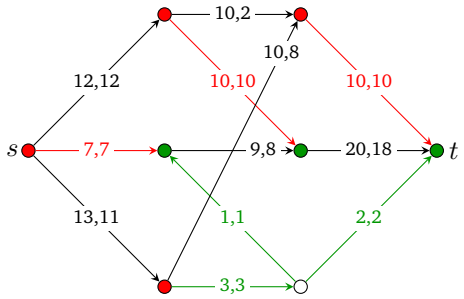
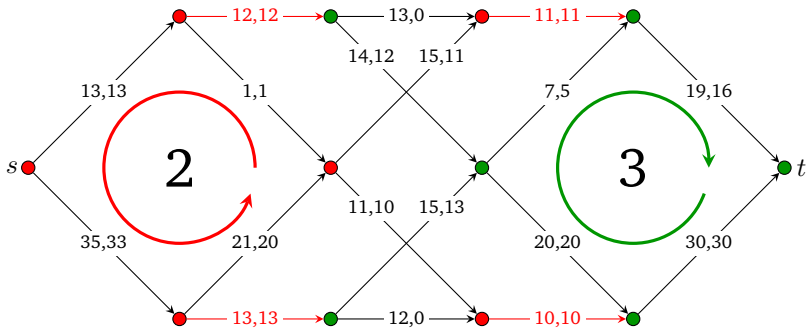


Figure B.17: The red edges are critical, and the green edges are optimal. The edges which are neither critical nor optimal are black.

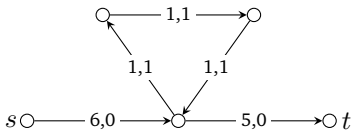
E8.3 A maximum flow of value 46 is indicated in the figure below. The critical edges are shown in red. There are no optimal edges. There are 6 maximum flows.



To count the number of maximum integer flows, we consider the ways we can change the flow, while maintaining the maximum flow value. This can only be achieved by sending flow around a cycle in the network. We only need to consider the cycles that lie completely within the red, white or green vertices, since the flow in the remaining cycles can not be changed. In this case, we can restrict our attention to the red and green cycle indicated in

the figure. In the red cycle, we can choose the flow in 2 ways, and in the green in 3 ways. Hence the number of maximum flows is 6.

E8.4 The converse is not true. It is indeed possible to have a flow of value zero such that some edges have positive flow. An example is given below.



E8.5 For a network N, we define a 0^+-flow to be a flow of value 0 where one or more edges in the network has positive flow. To determine if a network has a 0^+-flow, the following lemma is useful.

> **Lemma B.1** *Assuming positive edge capacities, a network N has 0^+-flow if and only if N contains a directed cycle.*
>
> **Proof** If N contains a directed cycle C, we can create a 0^+-flow by taking the zero flow and sending some flow around C (this does not change the value of the flow). Contrary, if N contains a 0^+-flow, consider an edge xy with positive flow. y must have an outgoing edge yz with positive flow, since any flow that goes into y must also exit y (even if y is s or t). Follow the edge to z and continue following an outgoing edge with positive flow. Since N is finite, we will eventually come back to a vertex, we have previously considered, thus having found a directed cycle in N. ∎

By the above lemma, it suffices to check if the network has a directed cycle. An intuitive algorithm for this is the following.

Algorithm 11 Determining if a network N has a directed cycle

1. For each edge $xy \in N$ do:
   Search for a directed path from y to x in $N - xy$ using distance classes.
   If such a path is found: Return `"There is a directed cycle in N"`
2. Return `"No directed cycle in N"`

For each of the m edges, we find the distance classes in N, thus the complexity of this algorithm is $O(m(m+n))$ where n is the number of vertices in N. There is a better algorithm using $O(m+n)$ time, and we leave it for the inquisitive student to find it.

E8.6 Consider a network $N = (V, E)$ and let (X, Y) be a partition of V such that $s \in X$ and $t \in X$. For each node $v \in Y$ we have that the flow into v equals the flow out of v, or put more formally

$$\sum_{\text{edge } e \text{ into } v} f(e) - \sum_{\text{edge } e \text{ out of } v} f(e) = 0 \quad .$$

Summing up these equations for every node $v \in Y$, the flow of the internal edges in Y cancels out, and only the flow into Y and the flow out of Y remain on the left-hand side. The right-hand side sums to zero. That is,

$$\sum_{\text{edge } e \text{ into } Y} f(e) - \sum_{\text{edge } e \text{ out of } Y} f(e) = 0 \quad ,$$

which is want we wanted to show.

B.9 Week 9

E9.1 G_1 has connectivity 1, edge-connectivity 3 and minimum degree 3. G_2, which is isomorphic to a complete graph on 8 vertices, minus a perfect matching, has connectivity 6, edge-connectivity 6 and minimum degree 6. G_3 has connectivity 2, edge-connectivity 4 and minimum degree 5.

E9.2 All the 16 spanning trees of K_4 are shown in Figure B.18.

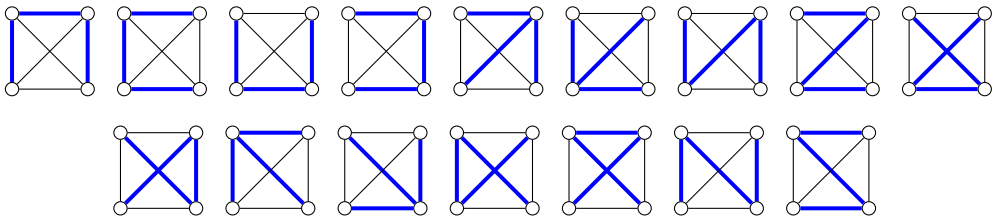


Figure B.18: The 16 spanning trees of K_4.

E9.3 Since the graph is K_4 and K_5 glued together, but missing an edge, we can use Theorem 5.5 to find the number of spanning trees as

$$\tau(K_4 - e)\tau(K_5/e) + \tau(K_5 - e)\tau(K_4/e) = 8 \cdot 50 + 75 \cdot 8 = 1000$$

E9.4 By Theorem 5.6 we find the number of spanning trees in $K_{4,3}$ to be $4^2 \cdot 3^3 = 432$. This can also be found by contraction-deletion (Theorem 5.1) or by the matrix-tree theorem (Theorem 5.7).

E9.5 We will show that in a bipartite graph G, the size of a maximum matching is equal to the size of a minimum covering. Let $G = (X, Y)$ be a bipartite graph, and let a denote the size of a maximum matching in G and a' denote the size of a minimum covering. Now we want to show $a = a'$ by using the following lemma

> **Lemma B.2** *Let x and y be two non-adjacent vertices of a graph G. Then the maximum number of internally-disjoint (x, y)-paths in G is equal to the minimum number of vertices whose deletion destroys all (x, y)-paths.*

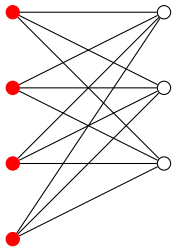
From G, we create a new graph N, by adding two new vertices s and t, where we connect s to every vertex in X and t to every vertex in Y. Now we let b denote the maximum number of internally distinct s-t paths and b' denote the minimum number of vertices different from s and t, such that their deletion will destroy all paths from s to t.

By our Lemma, we know that $b = b'$. We now show that $a = b$ and that $a' = b'$, implying that $a = a'$. If we have a matching of size a in G, we can use those edges in N to find at least a internally distinct s-t paths, thus $b \geq a$. If we have b internally distinct s-t paths in N, we can use the edges between X and Y on those paths to find a matching of size at least b in G, thus $a \geq b$. This gives us $a = b$.

If we have a' vertices in G which covers all the edges of G, then we can use these a' vertices to remove all edges between X and Y in N, and by that removing all paths from s to t, thus $b' \leq a'$. If we have b' vertices in N which deletion separates s from t, we can use these b' vertices in G to cover every edge, thus $a' \leq b'$. This gives us $a' = b'$ which implies $a = a'$.

E9.6 We have Menger's Theorem which states that the maximum number of completely disjoint s-t paths is equal to the minimum number of vertices which removal disconnects s from t. Now construct a new graph N from G by adding a vertex s, connecting it to every vertex in S and adding a vertex t connecting it to every vertex in T. Now Menger's Theorem gives us that the maximum number of completely disjoint paths from S to T is equal to the minimum number of vertices which removal disconnects S from T. This is a more general version of Menger's Theorem, which uses sets instead of vertices.

E9.7 No. Consider the following 3-connected graph where the red vertices are in S:



Now assume there is a simple cycle C using all 4 of the vertices in S. Since G is bipartite there are no edges between vertices in S, thus every other vertex on C must be in S, but since $|S|$ is even, this can not be the case.

E9.8 As shown in the previous exercise, the existence of a cycle containing an arbitrary $(k+1)$-subset S of the vertices of a k-connected graph G is not guaranteed for $k \geq 2$. In this exercise, we prove that if the size of S is lowered from $k+1$ to k, such a cycle will exist in G.

Theorem B.1 *Let $G = (V, E)$ be a k-connected graph, $k \geq 2$, and let $S \subseteq V$ such that $|S| = k$, then G has a cycle containing all vertices of S (and possibly some other vertices as well).*

Proof The proof is by induction on k. We start by proving the base case $k = 2$.
Base case: Since G is 2-connected any two vertices will be connected by at least two internally disjoint paths forming a cycle.
Inductive step: Now suppose G is k-connected and consider a set S of k vertices of G. Let S' be a $(k-1)$-subset of S and let x denote the vertex which is in S but not in S'. It follows from the induction hypothesis that G has a cycle $C = c_1 c_2 \dots c_\ell c_1$ containing the vertices of S'. We now show how to extend C to a cycle also containing x.

Case $\ell \geq k$: We add a vertex s to G and join it to k vertices of C. The resulting graph is still k-connected, so there must be k internally disjoint paths from x to s. All of these k paths must intersect C at some point, since s is only joined to vertices of C. Consider the $k-1$ vertices of S lying on C, it follows from the pidginhole principle that at least two of the paths from x to s must meet C in two vertices c_i and c_j such that none of the vertices $c_{i+1}, c_{i+2}, \dots, c_{j-1}$ are part of S. Hence the cycle $C' = x \dots c_i c_{i-1} \dots c_{j+1} c_j \dots x$ will contain all vertices of S.

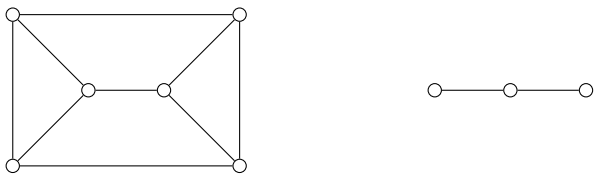
Case $\ell = k - 1$: In this case C contains just the vertices of S'. We add a vertex s to G and join it to the $k-1$ vertices of C. The resulting graph is at least $k-1$-connected, so there must be $k - 1$ internally disjoint paths from x to s. Hence every vertex of C is part of exactly one of these paths. Consider two of these paths $x \dots c_i \dots s$ and $x \dots c_{i+1} \dots s$ meeting C in two neighbouring vertices c_i and c_{i+1}. Then the cycle $C' = x \dots c_{i+1} c_{i+1} \dots c_{i-1} c_i \dots x$ will contain all vertices of S. ∎

B.10 Week 10

E10.2 The first graph has 75 spanning trees and the second has 368. The Petersen graph has 2000 spanning trees.

E10.3 One graph which is vertex transitive, but not edge transitive is the Prism graph shown in Figure B.19(a). By rotations and reflections every vertex of the Prism graph can be mapped to any other. The graph is clearly not edge transitive, since it has edges that are only contained in cycles of length four, and other edges which are contained in cycles of length three and four.

One graph which is edge transitive, but not vertex transitive is the graph shown in Figure B.19(b). By reflecting the graph at the center, one edge can be mapped into the other. The graph is clearly not vertex transitive, since it contains vertices of different degrees. This example can be generalized as follows. Take any complete bipartite graph $K_{p,q}$ where $p \neq q$. In this graph, every edge can be mapped into any other by a permutation of the vertices in one partition.



(a) (b)

E10.4 For a graph G containing n vertices and m edges, we let $\tau_e(G)$ denote the number of spanning trees of G containing a specific edge $e \in E(G)$. We start by proving the following lemma.

Lemma B.3 *For any edge e in an edge transitive graph G,*

$$\tau_e(G) = \tau(G/e) = \frac{(n-1)}{m}\tau(G)$$

Proof Let $e_1, e_2, \dots, e_m$ denote the edges of G and consider the sum

$$\tau_{e_1}(G) + \tau_{e_2}(G) + \ldots + \tau_{e_{m-1}}(G) + \tau_{e_m}(G)$$

Now focus on a specific spanning tree T of G. This spanning tree will contribute by one to exactly $n - 1$ of the terms in the above sum. Namely the terms corresponding to the edges in T. Since each of the $\tau(G)$ spanning trees are counted $n - 1$ times, we have that

$$\tau_{e_1}(G) + \tau_{e_2}(G) + \ldots + \tau_{e_{m-1}}(G) + \tau_{e_m}(G) = (n-1)\tau(G) \quad .$$

G is edge transitive so all edges must have the same properties. In particular they must be part of the same number of spanning trees, and hence $\tau_{e_1}(G) = \tau_{e_2}(G) = \ldots = \tau_{e_m}(G)$. If we denote this number by $\tau_e(G)$, we have established that

$$\tau_e(G) \cdot m \;=\; \tau_{e_1}(G) + \tau_{e_2}(G) + \ldots + \tau_{e_{m-1}}(G) + \tau_{e_m}(G) \;=\; (n-1)\tau(G) \quad ,$$

and this proves the lemma. ∎

Using this lemma, we can now prove Theorem 5.4.

Proof (Theorem 5.4) We start by proving the second part of the theorem. Since K_n is edge transitive, we know from the previous lemma that $\tau_e(K_n) = \frac{(n-1)}{m}\tau(K_n)$. Furthermore we have that the number of edges m in a complete graph is $m = \binom{m}{2} = n(n-1)/2$. Hence,

$$\tau_e(K_n) = \tau(K_n/e) = \frac{(n-1)\tau(K_n)}{m} = \frac{(n-1)n^{n-2}}{(n-1)n/2} = 2n^{n-3} \quad .$$

The first part of the theorem now follows from the Contraction-Deletion theorem (see Theorem 5.1).

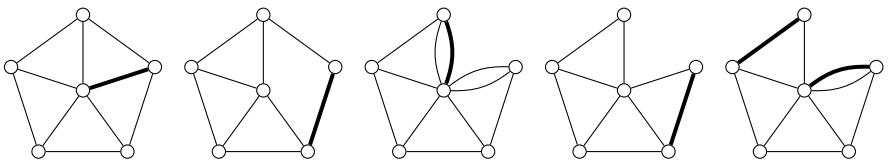
$$\tau(K_n - e) = \tau(K_n) - \tau(K_n/e) = n^{n-2} - 2n^{n-3} = (n-2)n^{n-3} \quad ,$$

which concludes the proof. ∎

E10.5 Scoin's formula can be proved using Kirchoff's Matrix-Tree Theorem. Start by removing a row and column from the Laplacian matrix for $K_{n,m}$. Then use row and column operations to transform the matrix into an upper- or lower triangular matrix for which the determinant can be calculated as the product of the diagonal.

E10.6 The graph has $8^6 = 262444$ spanning trees. This can be found by observing that the graph is composed of six copies of a much simpler graph having 8 spanning trees joined together by cut-vertices.

E10.7 We consider the 5 different graphs, denoted w_n, a_n, b_n, d_n, e_n where n denote the number of vertices, shown in Figure B.19



(a) w_6 (b) a_6 (c) b_6 (d) d_6 (e) e_6

Figure B.19: The five families of graphs we use to find an explicit formula for the number of spanning trees in the wheel. We use the Contraction-Deletion Theorem on the emphasized edges, to find a system of recurrence relations.

Now we find the following set of recurrence relations by using the Contraction-Deletion theorem on the bold edges, marked in the graphs.

$$\begin{aligned}
\tau(w_n) &= \tau(a_n) + \tau(b_{n-1}) \\
\tau(a_n) &= \tau(d_{n-1}) + \tau(w_{n-1}) \\
\tau(b_n) &= \tau(e_n) + \tau(b_{n-1}) \\
\tau(d_n) &= \tau(d_{n-1}) + \tau(e_{n-1}) \\
\tau(e_n) &= \tau(d_n) + \tau(e_{n-1}) = \tau(e_{n-1}) + \tau(b_{n-1})
\end{aligned}$$

Our idea is, to show that $\tau(a_n)$ and $\tau(d_n)$ follow the same recurrence relation, and then use that their sum $\tau(w_n)$ must also follow that recurrence relation. We start by considering the two last equations for $\tau(d_n)$ and $\tau(e_n)$.

$$\begin{aligned}
\tau(d_n) &= \tau(d_{n-1}) + \tau(e_{n-1}) \\
\tau(e_n) &= \tau(d_n) + \tau(e_{n-1})
\end{aligned}$$

Now we isolate $\tau(e_{n-1})$ in the first equation, and plug it into the second equation to obtain

$$(\tau(d_{n+1}) - \tau(d_n)) = \tau(d_n) + (\tau(d_n) - \tau(d_{n-1}))$$

which reduces to

$$\tau(d_{n+1}) = 3\tau(d_n) - \tau(d_{n-1})$$

When we have this, we change the index and get the following equation

$$\tau(d_n) = 3\tau(d_{n-1}) - \tau(d_{n-2}) \Rightarrow 0 = \tau(d_n) - 3\tau(d_{n-1}) + \tau(d_{n-2})$$

we change the index again, to get

$$0 = \tau(d_{n-1}) - 3\tau(d_{n-2}) + \tau(d_{n-3})$$

and then we subtract these two equations with different index from each other to get

$$\begin{aligned}
0 - 0 &= (\tau(d_n) - 3\tau(d_{n-1}) + \tau(d_{n-2})) - (\tau(d_{n-1}) - 3\tau(d_{n-2}) + \tau(d_{n-3})) \\
0 &= \tau(d_n) - 4\tau(d_{n-1}) + 4\tau(d_{n-2}) - \tau(d_{n-3})
\end{aligned}$$

which is our final recurrence relation for $\tau(d_n)$. From the last relation, we can see that $\tau(b_{n-1}) = \tau(d_n)$. Which we plug into the first relation $\tau(w_n) = \tau(a_n) + \tau(b_{n-1})$

$$\tau(w_{n-1}) = \tau(a_{n-1}) + \tau(d_{n-1})$$

Now, we plug this into the second relation $\tau(a_n) = \tau(d_{n-1}) + \tau(w_{n-1})$ to obtain

$$\tau(a_n) = \tau(d_{n-1}) + \tau(w_{n-1}) = \tau(d_{n-1}) + (\tau(a_{n-1}) + \tau(d_{n-1})) = \tau(a_{n-1}) + 2\tau(d_{n-1})$$

thus giving

$$\tau(a_n) - \tau(a_{n-1}) = 2\tau(d_{n-1})$$

We showed that $0 = \tau(d_{n-1}) - 3\tau(d_{n-2}) + \tau(d_{n-3})$, and from this we obtain

$$\begin{aligned} 2 \cdot 0 &= 2(\tau(d_{n-1}) - 3\tau(d_{n-2}) + \tau(d_{n-3})) \\ 0 &= 2\tau(d_{n-1}) - 2 \cdot 3\tau(d_{n-2}) + 2\tau(d_{n-3}) \\ 0 &= (\tau(a_n) - \tau(a_{n-1})) - 3(\tau(a_{n-1}) - \tau(a_{n-2})) + (\tau(a_{n-2}) - \tau(a_{n-3})) \\ 0 &= \tau(a_n) - 4\tau(a_{n-1}) + 4\tau(a_{n-2}) - \tau(a_{n-3}) \end{aligned}$$

which is the final recurrence relation for $\tau(a_n)$. Now we have that both $\tau(a_n)$ and $\tau(d_n)$ follow the recurrence relation $0 = x_n - 4x_{n-1} + 4x_{n-2} - x_{n-3}$, and therefore, their sum $\tau(w_n)$ must follow the same recurrence relation. Now we guess for a solution $\tau(w_n) = \alpha^n$.

$$0 = \alpha^n - 4\alpha^{n-1} + 4\alpha^{n-2} - \alpha^{n-3}$$

and divide by α^{n-3} to get

$$0 = \alpha^3 - 4\alpha^2 + 4\alpha - 1$$

Now, if $\alpha_1, \alpha_2, \alpha_3$ are the three roots of this equation, then

$$A\alpha_1^n + B\alpha_2^n + C\alpha_3^n$$

is the general solution to the recurrence relation. We see that $\alpha_1 = 1$ is a root in the equation, and isolate $(\alpha - 1)$ to get

$$0 = (\alpha - 1)(\alpha^2 - 3\alpha + 1)$$

where the last factor has roots

$$\alpha = \frac{3 \pm \sqrt{9-4}}{2} = \frac{3 \pm \sqrt{5}}{2}$$

So we have the general solution

$$A\left(\frac{3+\sqrt{5}}{2}\right)^n + B\left(\frac{3-\sqrt{5}}{2}\right)^n + C$$

We compute the initial conditions and get

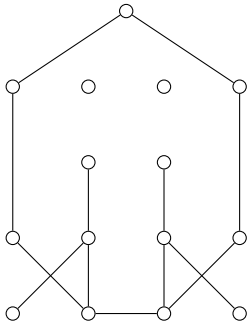
$$\tau(W_3) = 16 \quad \tau(W_4) = 45 \quad \tau(W_5) = 121$$

Now we solve with these initial conditions, to obtain the values $A = 1, B = 1, C = -2$, giving the final equation for the number of spanning trees in the wheel on n vertices

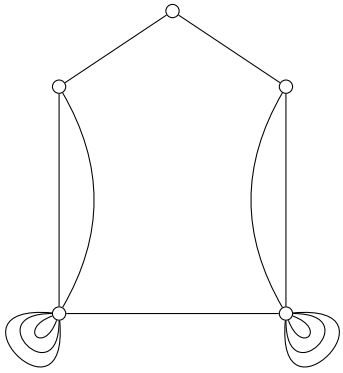
$$\tau(W_n) = \left(\frac{3+\sqrt{5}}{2}\right)^n + \left(\frac{3-\sqrt{5}}{2}\right)^n - 2$$

B.11 Week 11

E11.1 The solution to this exercise is indicated below.



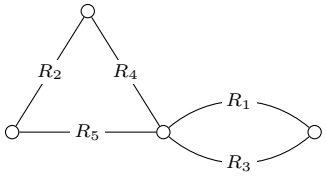
(a) $G - A$



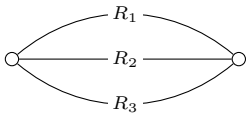
(b) G/A

E11.2 Let A denote the adjacency matrix, C the cycle matrix and D the cut matrix. By Theorem 39.3 in FN, we know that the rank of C can be found as $m-n+1$ which is $15-10+1=6$ in our graph, and that the rank of D is $n-1$ which is $10-1=9$ in our case. By Theorem 36.1 in FN, we know that the rank of A can be found in the same way as D, so the rank of A is also $10-1=9$.

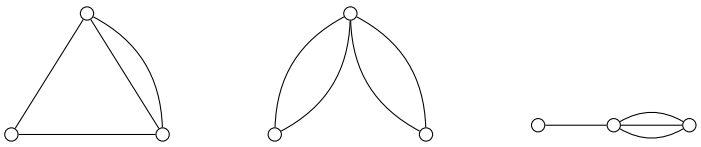
E11.3 1. Yes, the following graph



2. No. In a graph on n vertices, each spanning tree must have $n-1$ edges. Since we have both the term $R_1R_2R_5$ and the term R_3R_5, the graph must have a spanning tree with 2 edges, and a spanning tree with 3 edges, which is clearly not possible.
3. Yes, the following graph

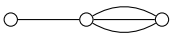


4. No. Any spanning tree leaves out 2 edges. Thus every spanning tree has 2 edges. Since the number of edges in a spanning tree is $n-1$, our graph has 3 vertices. There are three possible graphs with 3 vertices and 4 edges.



(a) $\tau = 5$ (b) $\tau = 4$ (c) $\tau = 3$

By the determinant, we can see that the graph we are considering must have exactly 3 spanning trees, which leaves us with



but since all four resistors appear in the determinant, this can not be the case either.

5. No. By the same argument as before. We still have 4 edges and 3 vertices, but now we need to find a graph with 6 spanning trees, which is not possible either.

E11.4 We can find the driving point resistance R between p and q by

$$R_{p,q} = \frac{\Delta(G/pq)}{\Delta(G)}$$

where $\Delta(G)$ is the network determinant of G. This gives us the driving point resistance

$$R_{p,q} = \frac{9xy + 12x + 12y + 12}{9x + 9y + 12}$$

We do not have an xy term in the denominator since no spanning tree in G can contain neither x nor y.

E11.5 Answer: No, Yes, No.

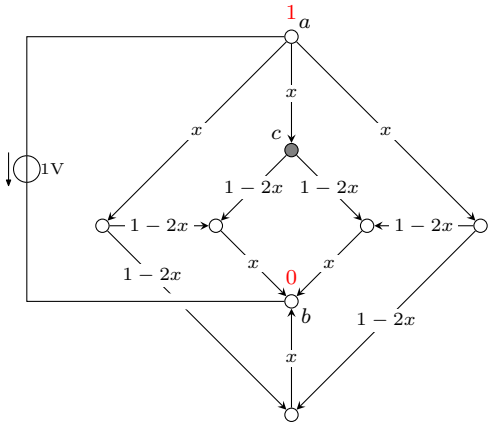
(a) For R_2 observe that that the cotree-product $\pi_{R_2}(\Gamma)$ of all R_2-trees Γ in N have the same sign. Hence there are no values of the resistors that can cause $\sum_\Gamma \pi_{R_2}(\Gamma)$ to be zero. Thus for the current to be zero, it follows from Kirchoff's rule that the network determinant $\Delta(N)$ must be infinitely large, which is impossible.

(b) For R_3 observe that this edge is part of fundamental cycles in N with opposite orientations. Consider one R_3-tree Γ of N in which the fundamental cycle containing R_3 has positive orientation. Now let the value of the resistors not in Γ be a very large number, and let the values of the resistors in Γ be 1. This will cause the cotree-product of Γ to dominate the sum of cotree-products, which will then be positive, so the current in R_3 will be positive as well. Now focus on a R_3-tree Γ' in which the fundamental cycle containing R_3 has negative orientation. By letting the resistors not in Γ' have a very large value and those in Γ' be 1, the sum of cotree-products will be dominated by a very large negative term, and hence the current in R_3 will be negative. Notice that the current in R_3 is a continuous function, which we have just argued attains both positive and negative values. This implies that the function has a root.

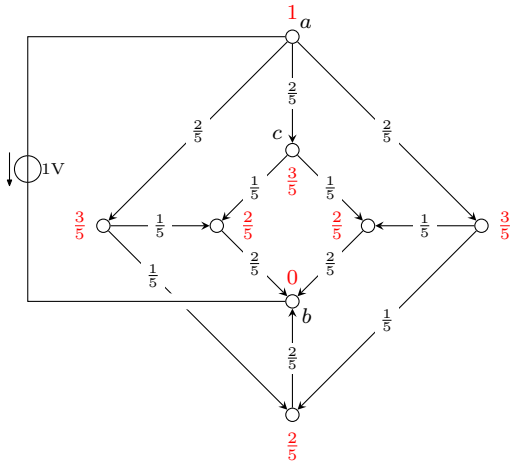
(c) For R_4, the argument is similar to that for R_2.

B.12 Week 12

E12.1 (a) The answer is $\frac{3}{5}$. The probability that a random walk starting in c gets to a before b is equal to the node voltage in c, when a voltage generator is connected between a and b such that $U(a) = 1$ and $U(b) = 0$. By rotation symmetry the current in the three edges leaving a must be the same (try to visualize the cube in 3D). The same is true for the three edges entering b. The total current leaving a must be equal to the total current entering b (notice that this value is *not* 1A!), since no current is generated in the circuit. Introducing an unknown x to denote the current in each of these six edges, we have the situation illustrated in Figure B.20(a).



(a) Finding an equation using Kirchhoff's Current Law on the node marked in gray.



(b) Solving the equation for x.

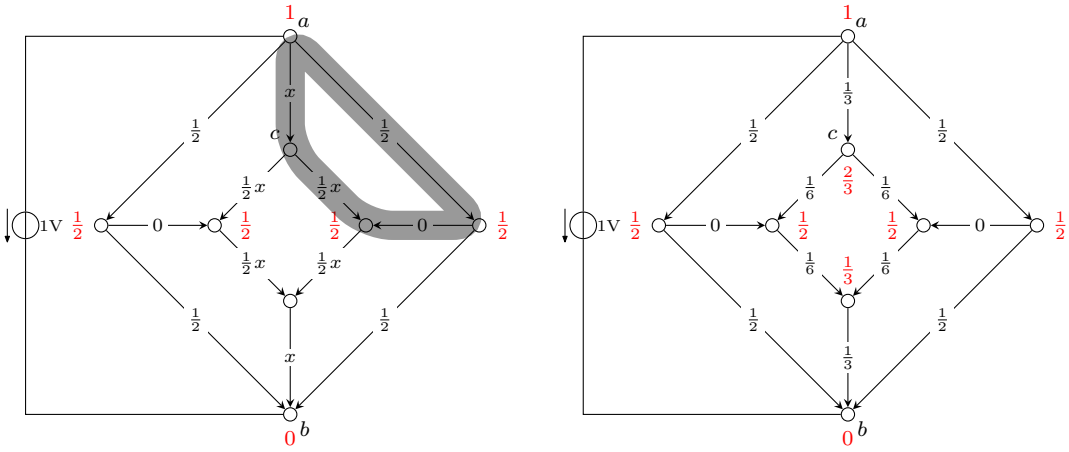
Figure B.20: Finding the node voltage $U(c)$. Node voltages are indicated in red.

Applying Kirchhoff's Current Law on the node c marked in gray yields the equation

$$x = 2(1 - 2x) \;\Leftrightarrow\; x = \frac{2}{5}\,.$$

Substituting this value for x we arrive at the solution shown in Figure B.20(b).

(b) The answer is $\frac{2}{3}$. Exploiting the reflection symmetry of the circuit, we immediately have that four of the node voltages are $\frac{1}{2}$. Introducing an unknown x to express the remaining currents, we have the situation illustrated in Figure B.21(a).



(a) Finding an equation using Kirchhoff's Voltage Law on the cycle indicated in gray.

(b) Solving the equation for x.

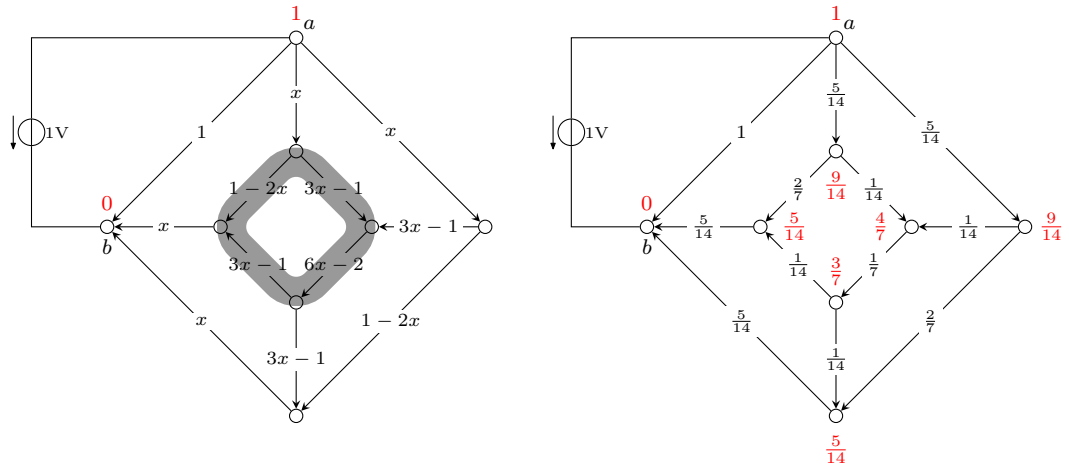
Figure B.21: Finding the node voltage $U(c)$. Node voltages are indicated in red.

Now consider the cycle indicated in gray. By Kirchhoff's Voltage Law the voltage drop along this cycle is equal to zero. That is,

$$\frac{1}{2} + 0 - \frac{1}{2}x - x = 0 \;\Leftrightarrow\; x = \frac{1}{3}\,.$$

Substituting this value for x we arrive at the solution shown in Figure B.21(b).

(c) The answer is shown in Figure B.22(b). First observe that the current in the edge going directly from a to b is 1. By symmetry, the two remaining edges leaving a and the two remaining edges entering b all have the same current (try to visualize the cube in 3D). If we denote the value of this current by x, the remaining currents can be expressed as shown in Figure B.22(a).



(a) Finding an equation using Kirchhoff's Voltage Law on the cycle indicated in gray.

(b) Solving the equation for x.

Figure B.22: Finding the node voltage $U(c)$. Node voltages are indicated in red.

Now consider the cycle indicated in gray. By Kirchhoff's Voltage Law the voltage drop along this cycle is equal to zero. That is,

$$(3x-1)+(6x-2)+(3x-1)-(1-2x)=0 \;\Leftrightarrow\; x=\frac{5}{14}\,.$$

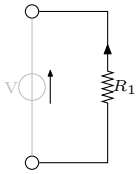
Substituting this value for x we arrive at the solution shown in Figure B.22(b).

E12.2 Recall that Kirchhoff's rule states that the current in R_1 is determined by the expression

$$i_{R_1}=\frac{-V}{\Delta(N)}\sum_{\mathcal{T}}\pi_{R_1}(\mathcal{T})\,,$$

where the sum is over all R_1-trees $\mathcal{T}$ in the circuit.

(a) The answer is: $i_{R_1}=\frac{V}{R_1}$. The network determinant is found to be $\Delta=R_1$ by contracting the voltage generator and considering the single spanning tree, which does not contain R_1. There is only a single R_1-tree:

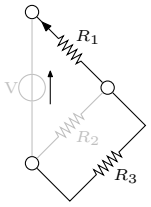


$\pi_{R_1}(\mathcal{T})=-1$

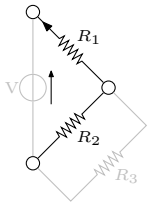
Thus the current in R_1 is

$$i_{R_1}=\frac{-V}{R_1}(-1)=\frac{V}{R_1}\,.$$

(b) The answer is: $i_{R_1}=\frac{R_2+R_3}{R_1R_2+R_2R_3+R_3R_1}V$. The network determinant can be calculated to be $\Delta=R_1R_2+R_2R_3+R_3R_1$. There are two R_1-trees:



$\pi_{R_1}(\mathcal{T})=-R_2$

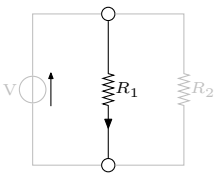


$\pi_{R_1}(\mathcal{T})=-R_3$

Thus the current in R_1 is

$$i_{R_1}=\frac{-V}{R_1R_2+R_2R_3+R_3R_1}(-R_2-R_3)=\frac{R_2+R_3}{R_1R_2+R_2R_3+R_3R_1}V\,.$$

(c) The answer is: $i_{R_1}=-\frac{V}{R_1}$. The network determinant is $\Delta=R_1R_2$. There is only a single R_1-tree depicted below

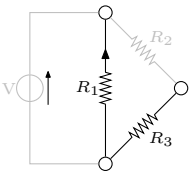


$\pi_{R_1}(\mathcal{T})=R_2$

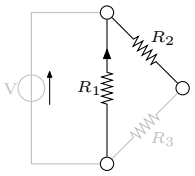
Thus the current in R_1 is

$$i_{R_1} = \frac{-V}{R_1 R_2}(R_2) = -\frac{V}{R_1} \ .$$

(d) The answer is: $i_{R_1} = \frac{V}{R_1}$. The network determinant is $\Delta = R_1(R_2 + R_3)$. The circuit has the two R_1-trees shown below



$\pi_{R_1}(\mathcal{T}) = -R_2$



$\pi_{R_1}(\mathcal{T}) = -R_3$

Thus the current in R_1 is

$$i_{R_1} = \frac{-V}{R_1(R_2 + R_3)}(-R_2 - R_3) = \frac{V}{R_1} \ .$$

E12.3 Due to the symmetry of the octahedron, there are only two cases to consider

(i) The distance between a and b is 1

(ii) The distance between a and b is 2

The second case is easy, since one can argue that the current in all edges leaving a and in all edges entering b must be the same. This gives the node voltages (probabilities) shown in Figure B.23.

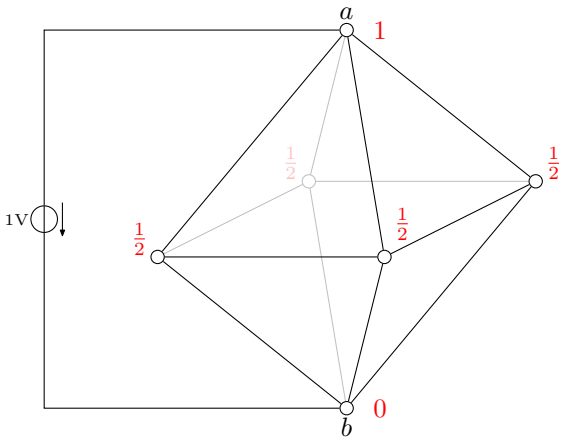
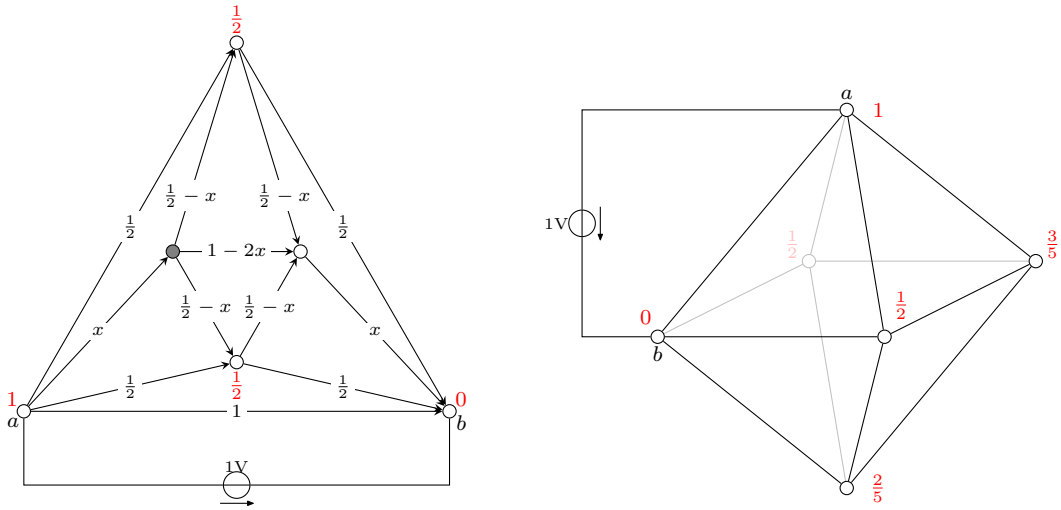


Figure B.23: The distance between a and b is 2.

We now consider the case in which the distance between a and b is 1. By reflection symmetry, we can quickly find two nodes having node voltage $\frac{1}{2}$. Introducing an unknown x to denote the current in the remaining edge leaving a, we can express all remaining currents by x as shown in Figure B.24(a).



(a) Finding an equation using Kirchhoff's Current Law on the node marked in gray.

(b) Solving the equation for x.

Figure B.24: The distance between a and b is 1.

Applying Kirchhoff's Current Law on the node marked in gray yields the equation

$$x = \left(\frac{1}{2} - x\right) + (1 - 2x) + \left(\frac{1}{2} - x\right) \;\Leftrightarrow\; x = \frac{2}{5}\,.$$

Substituting this value for x, we get the final solution depicted in Figure B.24(b).

B.13 Week 13

E13.1 For a planar graph G, we know that $v(G) - e(G) + f(G) = 2$. If a graph is self-dual the number of vertices must be equal to the number of faces, since the dual graph G' has a vertex for every face in G.

$$\begin{aligned} v(G) - e(G) + f(G) &= 2 \\ e(G) &= v(G) + f(G) - 2 \\ e(G) &= 2v(G) - 2 \end{aligned}$$

For $n \geq 4$ we give a self-dual graph on n vertices. One such graph-family, is the wheel-graphs.

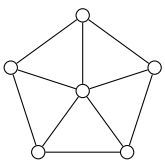


Figure B.25: The wheel graph with 6 vertices.

E13.2 We want to show that every simple plane graph is a spanning subgraph of some simple plane triangulation. To show this, we will show that any simple plane graph, can be extended to a plane triangulation by adding edges.

Consider a plane graph G which is not a triangulation. G must have some face f with degree greater than 3. Take two vertices x and y on the boundary of f, such that $xy \notin E(G)$. Add xy inside f. This will divide f into two smaller faces. We need to show that it is always possible to choose two vertices x and y in a face of degree ≥ 4 such that $xy \notin E(G)$.

In a plane graph G, consider a face f of degree k, and let $v_1, v_2, \ldots, v_k$ denote the vertices on the boundary of f. If v_1v_3 is not in $E(G)$, add v_1v_3 inside f, finishing the proof. If v_1v_3 is in $E(G)$, consider v_2v_4. Since v_1v_3 was in $E(G)$, it is not possible for v_2v_4 to also be in $E(G)$, since it must lie outside of f, contradicting the planarity of G.

E13.3 We start by showing that, if a graph is planar, all of its blocks are planar. This is easy to see, since a planar graph with a non planar block is a contradiction. Now we will show that if all the blocks of a graph G are planar, then G is planar itself. We do it in the following way: We consider a graph, and assume its blocks are planar. Then we draw each planar block in such a way, that we can connect them without loosing planarity.

Consider a graph G, where every block is planar and look at a cut vertex x connecting two blocks A and B. We start by finding a planar embedding of the two blocks A and B connected by x. Let y be a neighbour of x in A, and let z be a neighbour of x in B. Now we know, that the edge xy is adjacent to a face f_A in A. Now we redraw A in such a way, that xy lies on the outer face. That can be done in the following way: Draw A on a sphere, and create a small hole in f_A. Now unwrap the sphere, forming a disc, where xy is on the outer face.

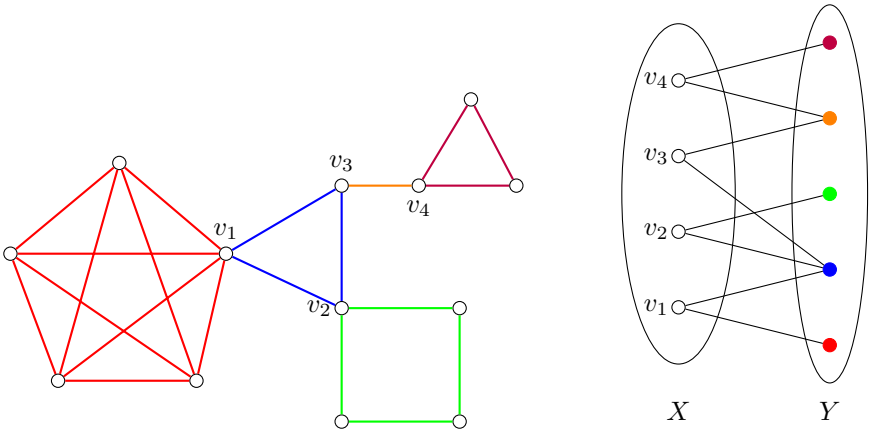
If we do this for both A and B, we get x on the outer boundary of both A and B, and we can now connect A to B by x without loosing planarity.

Now we will show, that a minimal nonplanar graph is a simple block. We assume different, namely that we have a minimal nonplanar graph G which is not a simple block. Here minimal means that removing anything from the graph, will result in planarity. If we consider one of its blocks B, it must be planar, since removing the rest of G to obtain B will result in planarity. In this way, we can argue that every block of G is planar. Now we know, from the previous result, that G itself must be planar.

E13.4 Let G be a connected graph which is not a block. We must prove that G contains at least two blocks that each contain exactly one cut-vertex. To do this, we will use an auxiliary bipartite graph $H = (X, Y)$ obtained from G as follows:

> **Construction of $H = (X, Y)$:** X consists of all cut-vertices of G and Y contains a vertex for each block in G. There is an edge from $v \in X$ to $b \in Y$ if and only if v is contained in the corresponding block in G.

Figure B.26 shows an example of this construction.



(a) The original graph G, where each block has its own color. The cut vertices are denoted v_1, v_2, v_3, v_4.

(b) The constructed bipartite graph H.

Figure B.26: Illustrating the construction of H from G.

It is easy to show that H always will be a tree: If H contained a cycle $v_1 b_1 v_2 b_2 \dots b_n v_1$ it would contradict the maximality of the blocks contained in that cycle, since each of these could be made bigger.

If we can find two block-vertices each of degree exactly one in H, we are done. But this is easy: Consider a longest path in H. Since H is a tree, the end-vertices of this path must have degree 1, since otherwise we could extend the path. Both of these end-vertices must be block-vertices, since any cut-vertex in H has degree at least two.

E13.5 Consider a block B of G. If B has no cycles it must be K_2. If B has a cycle it must by our assumption be an odd cycle C. If $B \simeq C$ we are done. Otherwise there must be a third path connecting two vertices x and y on C as illustrated in Figure B.27.

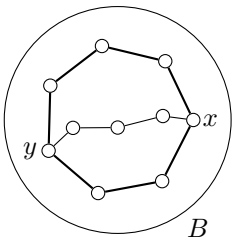


Figure B.27: An impossible block structure. The cycle C is marked with bold edges. Since C is an odd cycle (by assumption), one of the two smaller cycles has to be of even length.

Consider the three paths from x to y. Two of these must have the same parity (odd or even), and therefore B contains an even cycle, but this is a contradiction. So unless B is K_2 it will be just a single odd cycle.

C

COMPLEXITY OF ALGORITHMS

We often look at algorithms for different problems, and generally, it is interesting whether or not the algorithm is *efficient*. We need a way to talk about efficiency of an algorithm, and difficulty of a problem in a formal way. Sometimes, complexity is also used to denote efficiency.

C.1 Efficiency of an Algorithm

To denote the efficiency of an algorithm, we will introduce the so-called **Big O-notation**. Big O-notation says something about the time consumption of an algorithm on some input. We will say that an algorithm uses time $g(n)$ on input of size n. Now we can define O-notation as follows

Definition C.1 *We say that $g(n) \in O(f(n))$ if and only if there exists two constants c and n_0 such that $g(n) \leq cf(n)$ for all $n \geq n_0$.*

For example $2n \log_2 n \in O(n^2)$ because if we choose $c = 1$, then for all $n \geq 4$: $2n \log_2 n \leq n^2$. When we talk about time complexity and space complexity of algorithms, we use this notation to refer to some upper-bound for the running time or space requirements for the algorithm.

Example C.1 *Consider this simple algorithm for connecting n vertices into a complete graph:*

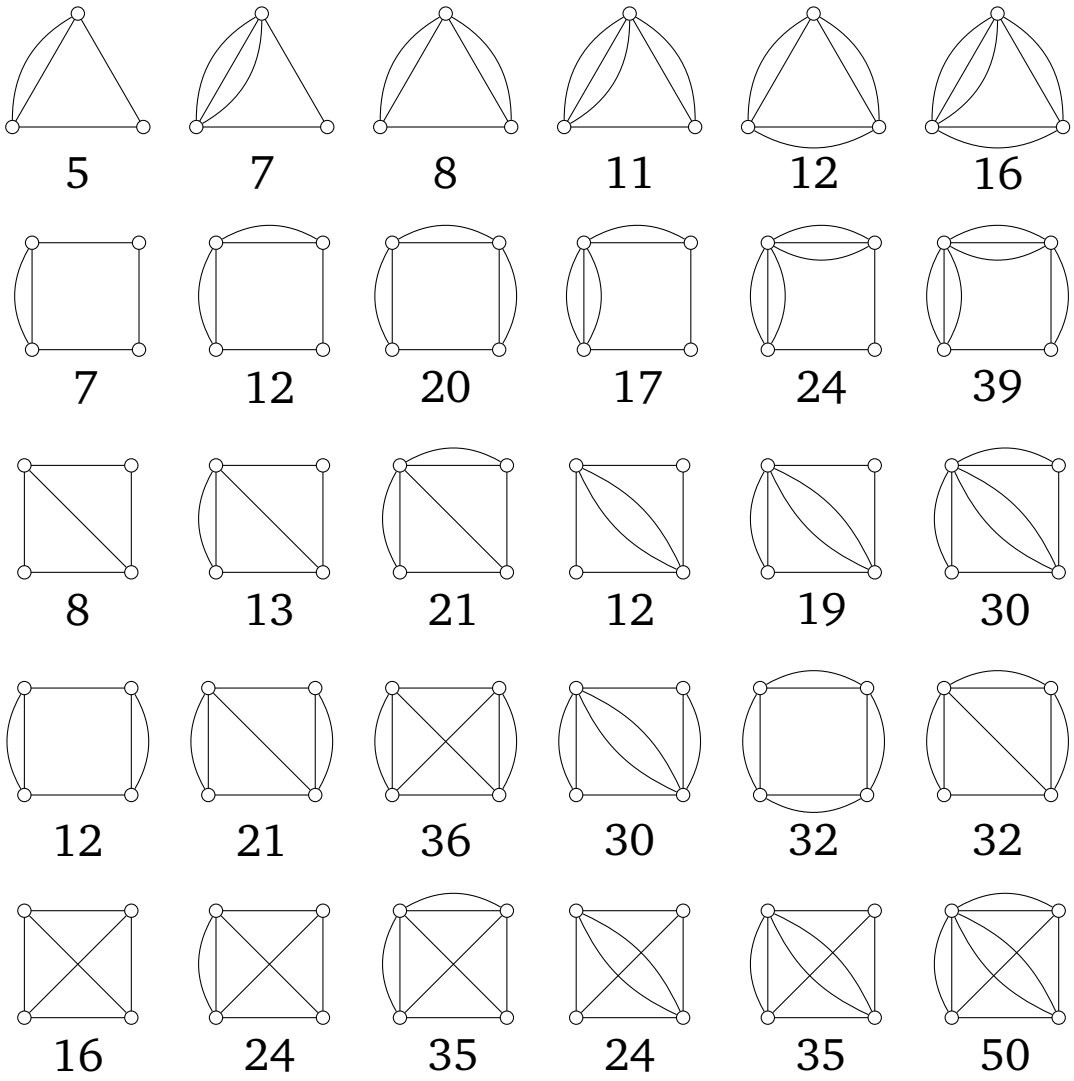
1. *For all n vertices.*

   Look at all other $n - 1$ vertices, and create an edge to them

Now we want to determine the efficiency of this algorithm. It looks at n vertices, and for each vertex, it looks at $n - 1$ other vertices, so the total number of pairs considered is $n(n - 1)$. For each pair, we create an edge, this takes constant time c. So the total time spend by the algorithm is $n(n - 1)c$. Now this can be reduced to $O(cn^2 - cn)$, and we can then ignore the constant factor to get $O(n^2 - n)$ where n^2 dominates. So we end up with an efficiency of $O(n^2)$.

D

Number of Spanning Trees for Selected Graphs



E

RECURRENCE RELATIONS

Sometimes, it is possible to establish recurrence relations for properties in a graph or in a family of graphs. In this case, one might try to solve the recurrence relation to get a closed formula. In this appendix, we will provide a very simple example on how to solve homogeneous linear recurrence relations. In general, solving a recurrence relation can be much more difficult.

Homogeneous Linear Recurrence Relations

If we have a homogeneous linear recurrence relation of the form

$$a_n = \sum_{i=1}^{k} c_i a_{n-i}$$

involving k terms, and we want to find a closed form for a_n - that is, a function - we can solve the recurrence relation. We guess that a_n is on the form α^n, and substitute this into the recurrence relation to obtain

$$\alpha^n = \sum_{i=1}^{k} c_i \alpha^{n-i}$$

and we divide by α^{n-k}, giving us

$$\alpha^k = \sum_{i=1}^{k} c_i \alpha^{k-i}$$

This is a polynomial of degree k with distinct roots $\{\alpha_1, \ldots, \alpha_k\} \in \mathbb{R}^k$, and now a_n can be written as

$$a_n = \sum_{i=1}^{k} \beta_i \alpha_i^n$$

where β_1 to β_k is constants which can be found by using k different initial conditions for the recurrence.

We have only described homogeneous linear recurrence relations, and we assumed that the roots $\{\alpha_1, \ldots, \alpha_k\}$ were distinct real numbers - but this is all which will be needed for this course.

F

LIST OF SYMBOLS

General Theory

$V(G)$	The vertex set of G
$E(G)$	The edge set of G
$v(G)$	The number of vertices in G
$e(G)$	The number of edges in G
$f(G)$	The number of faces in G
$d(v)$	The degree of vertex v
$d_{in}(v)$	The in-degree of vertex v
$d_{out}(v)$	The out-degree of vertex v
$\delta(G)$	The minimum degree in G
$\Delta(G)$	The maximum degree in G
$\lambda(G)$	The edge-connectivity of G
$\kappa(G)$	The vertex-connectivity of G
$G - e$	Removal of e from G
G/e	Contraction of e in G
K_n	The complete graph on n vertices
$K_{i,j}$	The complete bipartite graph with i vertices in one part, and j in the other
C_k	Cycle of length k
$N(v)$	Neighbours of vertex v
$G \simeq H$	G is isomorphic to H
$w(v, u)$	Weight of edge between v and u

Shortests Paths

$l(v)$	Distance to v from s
$N(v)$	Number of shortest paths from s to v
s	Starting vertex
D	Distance class

Euler Tours and The Chinese Postman Problem

W	The Waste Graph
H	The Auxiliary Graph

Spanning Trees

$\tau(G)$	Number of spanning trees in G
$G \ominus H$	Operation described on page 27
$G \oplus H$	Operation described on page 27
D	Degree matrix
A	Adjacency matrix
Q	Laplacian matrix

Matchings

M	A matching
C	A covering
L	Equality graph
ε	Bottleneck for change in helping numbers

Benzoids

B	A benzenoid
$p(t)$	Total number of Perfect Matchings
$p(x, y)$	The number of Perfect Matchings containing the edge xy

Network Flow

N	Flot network
$f(e)$	Flow over edge e
$c(e)$	Capacity of edge e
$c(S)$	Capacity of cut S
S_{min}	Minimum cut
$F(G)$	All possible flows over G
$\|f\|$	The value of flow f
$f_{max}(G)$	The maximum flow in G
δ	Bottleneck of augmenting path

Electrical Networks

N	Electrical Network
$\pi(\mathcal{T})$	Co-Tree Product of $\mathcal{T}$
$\mathcal{R}$	The Set of Resistors
$\mathcal{I}$	The Set of Current Generators
$\mathcal{V}$	The Set of Voltage Generators
ΔN	The Network Determinant
i_{R_k}	Current trough Resistor R_k
$\mathcal{T}_{R_k}$	R_k-tree
$R_{p,q}$	Driving Point Resistance between vertices p and q
$U(v)$	Node Voltage of v

INDEX