FRANK NIELSEN

GRAPH THEORY

- ALGORITHMS AND NETWORKS.

Department of Mathematics
Technical University of Denmark
1996

PREFACE

This book is written as a textbook for the course in graph theory at the Technical University of Denmark.

Chapter 1 contains a general introduction to graph theory, but the contents of the rest of the book is directed towards applications in technical sciences. Specifically in chapter 2 computer science, in chapters 3 and 4 operations research and, in chapters 5 and 6, the theory of electrical networks.

The first 25 pages of chapter 3 are guite detailed and assume only knowledge of few concepts from chapter 1. Since these pages of chapter 3 also contain the strongest theorem in the book, it may be profitable to begin the course with chapter 3.

In chapters 2, 3 and 4 a basic knowledge of computer science will be valuable, and some knowledge of electrical networks is a useful, but not necessary prerequisite for chapters 5 and 6.

While writing the book, I had very often support from Carsten Thomassen. It has been a great help.

The present translation into english is partly made by english speaking students, partly by the author. The reader is asked to consider the translation as "better than nothing". The author is well aware, that the translation does not reach professional standards.

January 1996 Frank Nielsen

TABLE OF CONTENTS.

CHAPTER 1. GRAPHS.

CHAPTER 2. GRAPHS ON A COMPUTER.

CHAPTER 3. TRANSPORT NETWORKS.

CHAPTER 4. GRAPH THEORETIC ALGORITHMS.

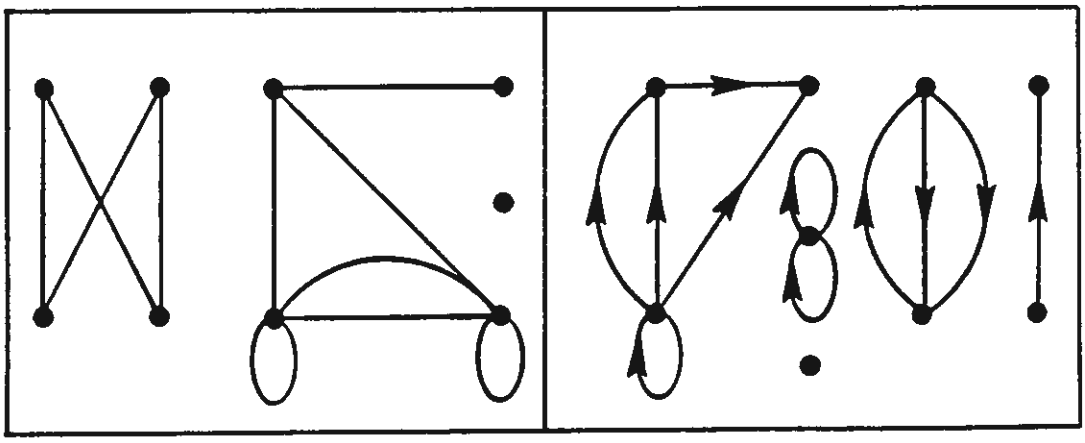
CHAPTER 5. CIRCUITS, CUTSETS AND TREES.

CHAPTER 6. ELECTRICAL NETWORKS.

CHAPTER 1

GRAPHS

1. THE CONCEPT OF A GRAPH. ISOMORPHISM A *graph* can be described by means of a figure which contains a finite set of *vertices* (•) and a finite set of *edges* (line segments or curves), where each edge connects two of the vertices; these are called the *end vertices* of the edge. If two end vertices of an edge are actually the same vertex, the edge is called a *loop*. An edge can be *directed*; this is shown by putting an arrow on the edge, and the end vertices are then also called the *startvertex* and the *finalvertex* of the edge respectively. If all edges in a graph are directed we shall call the graph itself *directed*. If there are no directed edges the graph is called *not–directed*. In the figure below, to the left we show a not–directed graph, to the right a directed graph; both graphs have 9 vertices and 11 edges.

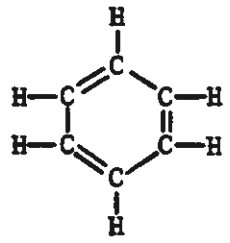
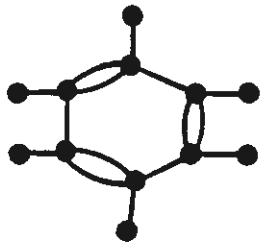
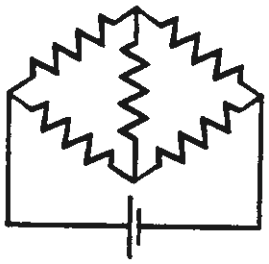
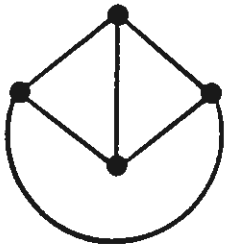
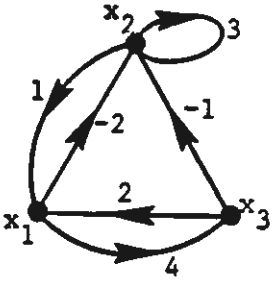
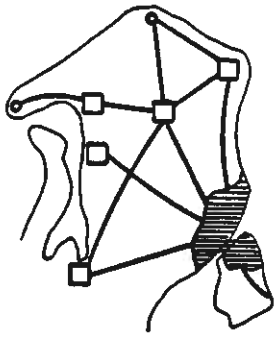
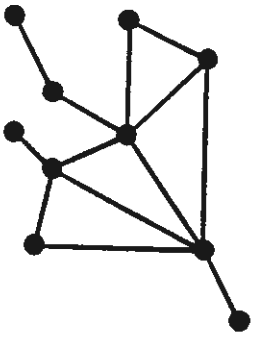


A *mixed graph* is a graph which may contain directed as well as not–directed edges.

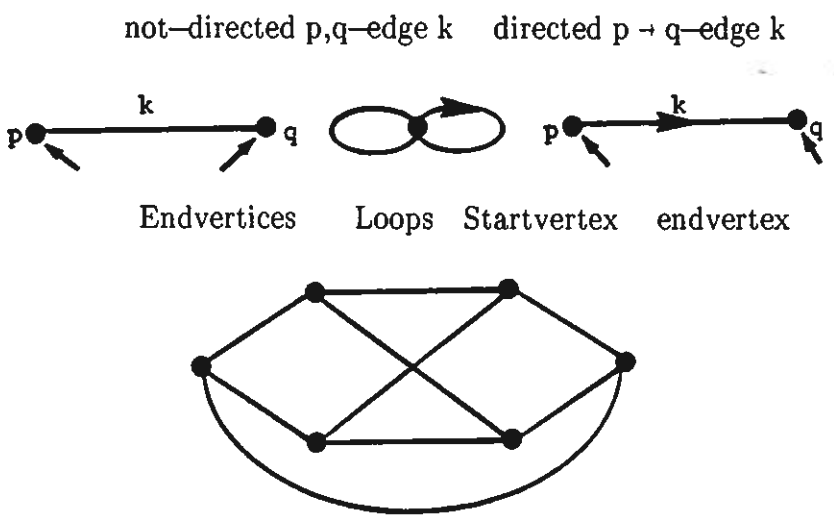
To explain how graphs naturally occur in engineering applications, we show on the next page in schematic form some examples in which, for a certain chemical, physical or mathematical system, there corresponds (in a natural way) a graph. In each of these cases –and in many others– it turns out that important properties of the system can be found by investigation of the corresponding graph. Note that even though the graph in each single case can be symbolised by means of a figure, the vertices and edges in the graphs in the different cases correspond to different types of objects. Therefore, it is not convenient to define a graph as a figure consisting of points and line segments (or curves) in the plane. Instead, we will allow the vertices and the edges of the graph to be arbitrary sets.

So even though in the following we shall always describe the graphs we consider by means of figures, then the formal definition we shall use is the following:

A graph Γ consists of a finite set $\mathscr{P}(\Gamma)$, whose elements are called *vertices of* Γ, and a finite set $\mathscr{K}(\Gamma)$, whose elements are called *edges in* Γ, together with a function which, for each edge $k \in \mathscr{K}(\Gamma)$ gives either a set $\{p,q\}$, where p and q are vertices in Γ, or an ordered set (p,q), where p and q are vertices in Γ. In the first case, k is called a *not–directed edge*, in the second a *directed edge*. In both cases, p and q are called *endvertices* of k, and k is said to be *incident* with p and q and to *connect* p and q. p and q are called *neighbours* and k is called a p,q–*edge*. If $p = q$ then k is called a *loop*. If k is directed we shall call p the *startvertex* of k, and q the *endvertex* of k. k is called a $p \to q$–edge.

SYSTEM	EKSEMPEL	GRAF	PUNKTER	KANTER
ORGANISK MOLEKYLE			ATOMER	BINDINGER
ELEKTRISK NETVÆRK				MODSTANDE, GENERATORER M.V.
HOMOGENT LINEÆRT LIGNINGS-SYSTEM	$x_1 = x_2+2x_3$ $x_2 = -2x_1+3x_2-x_3$ $x_3 = 4x_1$		UBEKENDTE	KOEFFICIENTER
VEJNET			BYER, VEJKRYDS	VEJE

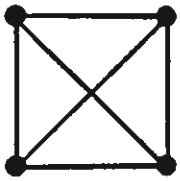
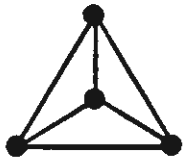
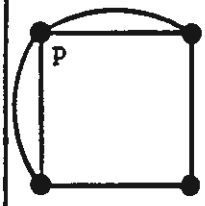
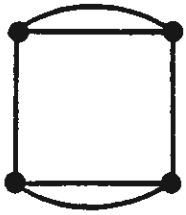
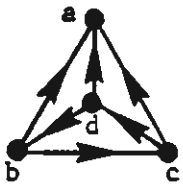
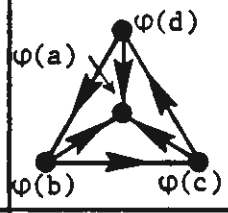
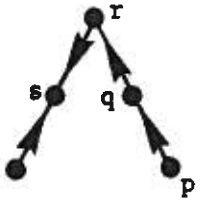
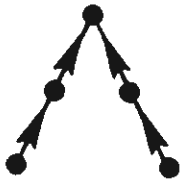
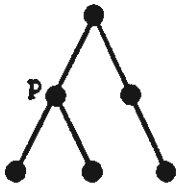
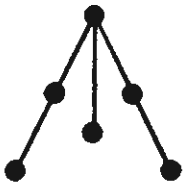
In figures, we shall show vertices and edges like this:



Remark: When we show a graph Γ in a figure then it may happen that two edges share a point in the plan which is not a vertex in $\mathscr{P}(\Gamma)$. The figure above shows an example.

Those properties of a graph, which we deal with in Graph Theory, are those which only are dependent upon the combinatorial structure of the graph, i.e. of which vertices are connected by edges and which are not. Let us make this description precise:

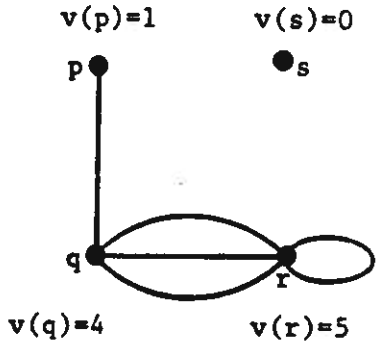
Two graphs Γ and Δ are called *isomorphic* if there exists a one–to–one mapping φ of $\mathscr{P}(\Gamma)$ on $\mathscr{P}(\Delta)$ such that for each pair (p,q) of vertices in Γ, the number of (p,q) edges in Γ is equal to the number of $\varphi(p)$, $\varphi(q)$–edges in Δ, and such that the number of $p \to q$–edges in Γ is equal to the number of $\varphi(p) \to \varphi(q)$–edges in Δ. φ is called an *isomorphic mapping* of Γ on Δ or an *isomorphism*. (If Γ and Δ are the same graph, then φ is called an *automorphism* of Γ.) Clearly, if Γ and Δ are isomorphic graphs then $|\mathscr{P}(\Gamma)| = |\mathscr{P}(\Delta)|$.($|\mathscr{M}|$ denotes the cardinality of the set $\mathscr{M}$, i.e. the number of elements in $\mathscr{M}$.) It is easy to see that if Γ and Δ are isomorphic, then $|\mathscr{K}(\Gamma)| = |\mathscr{K}(\Delta)|$ and for every $p \in \mathscr{P}(\Gamma)$ we also have that the number of edges in Γ, which is incident with p, is equal to the number of edges in Δ, which is incident with $\varphi(p)$. When φ is an isomorphic mapping of Γ on Δ, then φ can be extended to a one–to–one mapping ψ of $\mathscr{P}(\Gamma) \cup \mathscr{K}(\Gamma)$ on $\mathscr{P}(\Delta) \cup \mathscr{K}(\Delta)$ for which k is a p,q–edge ($p \to q$–edge) in Γ if and only if $\varphi(k)$ is $\varphi(p)$, $\varphi(q)$–edge ($\varphi(p) \to \varphi(q)$–edge) in Δ. Also, φ is called an *isomorphic mapping* of Γ on Δ. – In the figure on the previous page, the concept of isomorphism is demonstrated with some examples.

Γ	Δ	ISO-MORPHIC?	WHY?
		YES	In both graphs two arbitraty vertics are connected by exactly one edge.
		NO	In Γ , the vertex p is incident with 4 edges, but Δ does not contain a vertex with this property.
		YES	φ is an isomorphism.
		NO	In Γ there exist a p → q , a q → r and an r → s – edge, but 4 vertics with this property do not exist in Δ .
		NO	In Γ p is neighbour to two vertices, each at which has no other neighbour than p . Δ contains no vertex with this property.

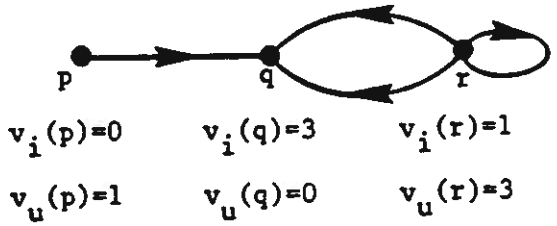
Graph theory deals with those properties of graphs which are invariant according to isomorphism, i.e. which is conserved by isomorphic mappings. For example, it is a graph theoretic property that a graph Γ contains two vertices which are connected by three edges. But it is not a graph thoeretic property of the graph Γ in the first example above that Γ contains 2 edges which share a common point in the plane; because this property has Δ not, and Δ is isomorphic to Γ .

When we have to investigate whether two given graphs Γ and Δ are isomorphic, it may happen that we can find a graph–theoretical property which one of the graphs has, and which the other graph does not have. Then the graphs cannot be isomorphic. But if we are not lucky enough to find such a property, we have to investigate the P! ($P = |\mathcal{P}(\Gamma)| = |\mathcal{P}(\Delta)|$) one–to–one mappings of $\mathcal{P}(\Gamma)$ onto $\mathcal{P}(\Delta)$ one at a time. If one of these mappings is an isomorphism, the graphs are isomorphic; otherwise, they are not. No algorithm is known which in the worst case is faster to determine isomorphism than the one we described here.

2. VALENCY. MULTIPLE EDGES. SUBGRAPHS. In this section we shall define a series of very easy but fundamental graph–theoretical concepts. Let Γ be a graph, and let p be a vertex in Γ. We define the *valency* $v(p) = v(p,\Gamma)$ of p as the number of edges in Γ which are incident with p. A loop incident with p contributes 2 to the valency of p. The *invalency* $v_i(p) = v_i(p,\Gamma)$ of p is defined as the number of directed edges which have p as end vertex. The *outvalency* $v_u(p) = v_u(p,\Gamma)$ of p is defined as the number of directed edges which have p as start vertex. For a directed graph Γ, we have that $v(p) = v_i(p) + v_u(p)$ for all $p \in \mathcal{P}(\Gamma)$. The figure illustrates the concept of valency.



A vertex such as s which has valency $v(s) = 0$ is called an *isolated vertex.* If we write down the valencies for the vertices in a graph Γ, ordered by size, we obtain a vector of numbers, which is called the *valency spectrum* for Γ. It consists of $|\mathcal{P}(\Gamma)|$ numbers. The valency spectrum for the graph in the figure above is (0, 1, 4, 5). The concepts of invalency and outvalency are also illustrated in the figure.



A graph is called *regular* or v–*regular* when all vertices have the same valency v. The graph is called an *Eulerian graph* when all vertices have even valency.

As our first example of a graph–theoretic theorem, we prove

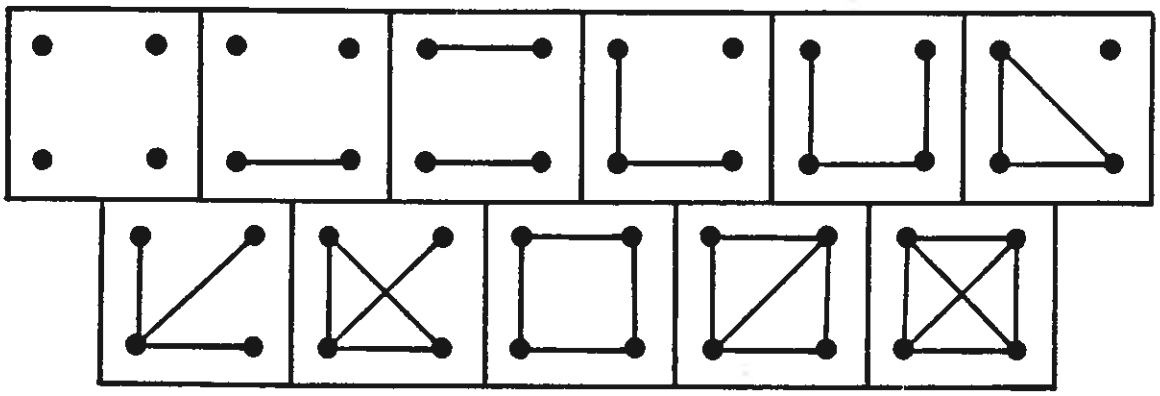
Theorem 2.1. In an arbitrary graph Γ, the number of vertices with odd valency is even.

Proof. Since each edge in Γ contributes with the value 1 to the valency of each of the two end vertices, we have that

$$2|\mathscr{K}(\Gamma)| = \sum_{p \in \mathscr{P}(\Gamma)} v(p) .$$

Since the left–hand side is an even number, the right–hand side must contain an even number of odd terms. Hence the number of vertices in Γ which have odd valency, is even. □

If two vertices p and q in a graph Γ are connected by t edges, where $t > 1$, we say that p and q are connected by a t–*double edge* or a *multiple edge*. Two or more directed edges are called *parallel* if they have the same start vertex and the same end vertex. A non–directed graph without loops or multiple edges is called a *simple graph*. In many applications, it is only the simple graphs which have interest. In the figure below, we have drawn all simple graphs with 4 vertices. "All" is to be understood in the following way: Any other simple graph with 4 vertices will be isomorphic to one of the graphs in the figure.



Observe that there are 11 simple graphs with 4 vertices. From the table below, you see that the number of simple graphs with P vertices grows very fast when P becomes larger.

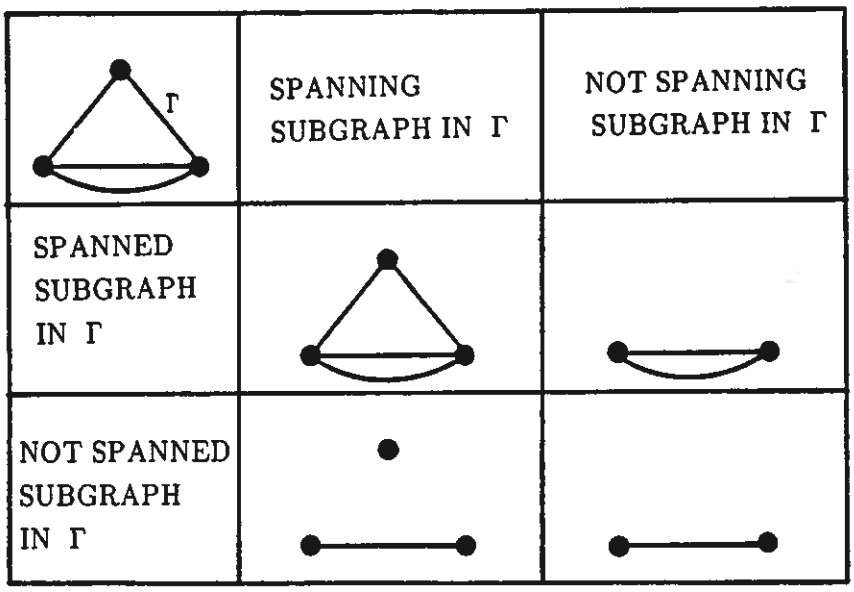
P	1	2	3	4	5	6	7	8	9
The number of simple graphs with P vertices.	1	2	4	11	34	156	1044	12346	274668

Even for small values of P, it is therefore very time–consuming to investigate all graphs with P vertices. If two vertices p and q in a simple graph are connected with an edge k, we often identify k with the pair (p, q) and write $k = (p, q)$, or just $k = pq$. Since the graph does not contain multiple edges, this cannot be misunderstood.

Let Γ be a graph. A graph Δ is called a *subgraph* in Γ if $\mathscr{P}(\Delta) \subseteq \mathscr{P}(\Gamma)$, $\mathscr{K}(\Delta) \subseteq \mathscr{K}(\Gamma)$, and if for every edge k in Δ, we have that its end vertices in Δ and Γ are the same. If k is a directed edge, then k must have the same start vertex in Δ and Γ, and also the same end vertex in Δ and Γ. If Γ is given by a figure, then a subgraph Δ in Γ is a graph which can be obtained by choosing a subset of the vertex set of Γ, and then take some of those edges which are incident to two of these vertices. We shall say that Γ *contains* Δ. For every graph Γ, Γ itself is a subgraph. All other subgraphs are called *proper*. The empty graph (without vertices and edges) is a subgraph in any graph. The heavy lined edges and vertices in the figure show a subgraph in the graph.

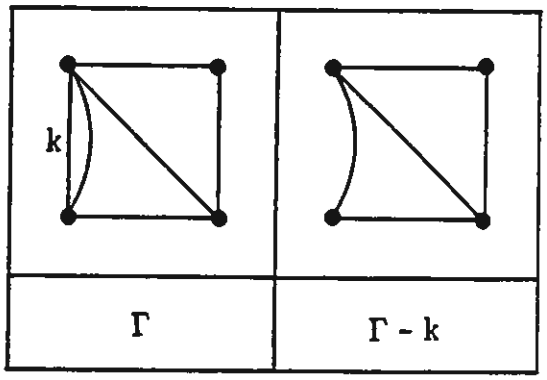
A subgraph Δ in a graph Γ is called *spanning* if $\mathcal{P}(\Delta) = \mathcal{P}(\Gamma)$. A subgraph Δ in a graph Γ is called *spanned* if $\mathcal{K}(\Delta)$ consists of all those edges in Γ which connect two vertices belonging to $\mathcal{P}(\Delta)$. A spanned subgraph Δ in Γ is therefore uniquely determined by Γ and $\mathcal{M} = \mathcal{P}(\Delta)$. Therefore we write $\Delta = \Gamma(\mathcal{M})$.

The concepts are illustrated in the figure below.

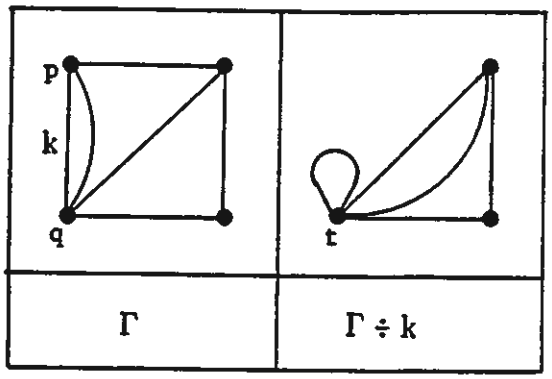


3. CONTRACTION AND DELETION. Let k be an edge in a graph Γ. We can remove k from Γ in two different ways, which are called deletion and contraction.

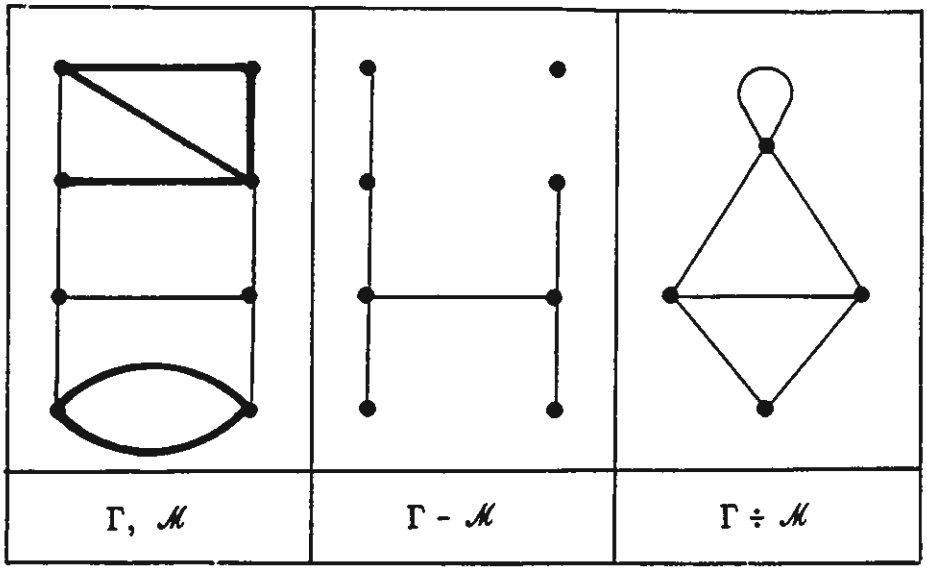
When we *delete* an edge k from Γ, we obtain a graph which is denoted $\Gamma - k$. It has the same vertex set as Γ and the edge set is $\mathcal{K}(\Gamma) - \{k\}$. An edge x in $\Gamma - k$ is incident with the same two vertices that it was in Γ. In the figure, $\Gamma - k$ is obtained from Γ by removing the edge k.



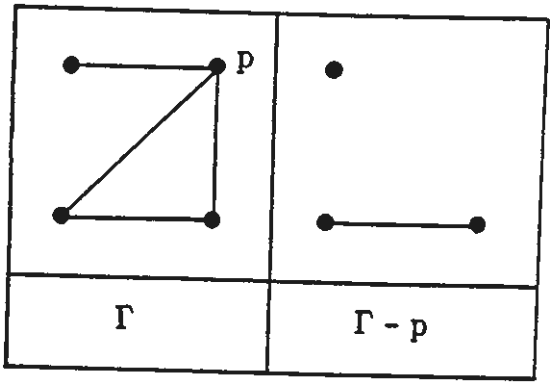
By *contraction* of k in Γ, we obtain a graph which is denoted $\Gamma \div k$. It is obtained from Γ by first deleting k and then replacing the end vertices p and q of k with a single vertex t, which in $\Gamma \div k$ is incident with all those edges which in $\Gamma - k$ are incident with p or q. In particular, we replace every p,q–edge k with a loop incident to t. $\Gamma \div k$ therefore has one edge and one vertex less than Γ.



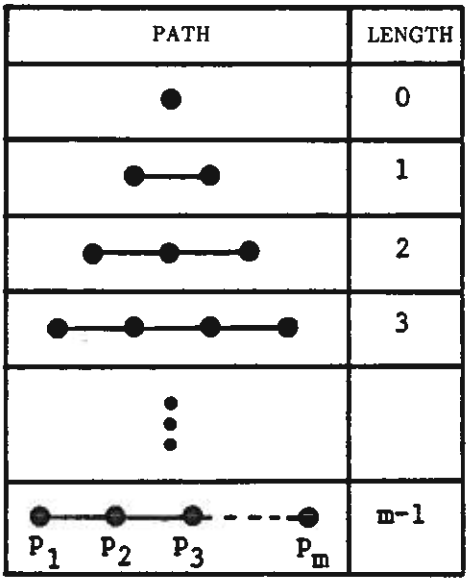
If $\mathscr{M} \subseteq \mathscr{K}(\Gamma)$ is a set of edges in Γ, we can also delete or contract the whole set $\mathscr{M}$. We obtain graphs which are denoted $\Gamma - \mathscr{M}$ (respectively $\Gamma \div \mathscr{M}$), and they are obtained from Γ by deleting – respectively contracting – the edges in $\mathscr{M}$ one at a time. The result is independent of the order in which the edges of $\mathscr{M}$ are chosen.



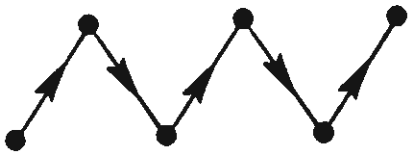
It is also possible to *delete* a vertex from a graph Γ. This is done by removing p from the vertex set, and removing all edges incident to p from the edge set. We obtain a graph which is denoted $\Gamma - p$. If Δ is a subgraph in Γ, we put $\Gamma - \Delta = \Gamma - \mathscr{P}(\Delta)$.



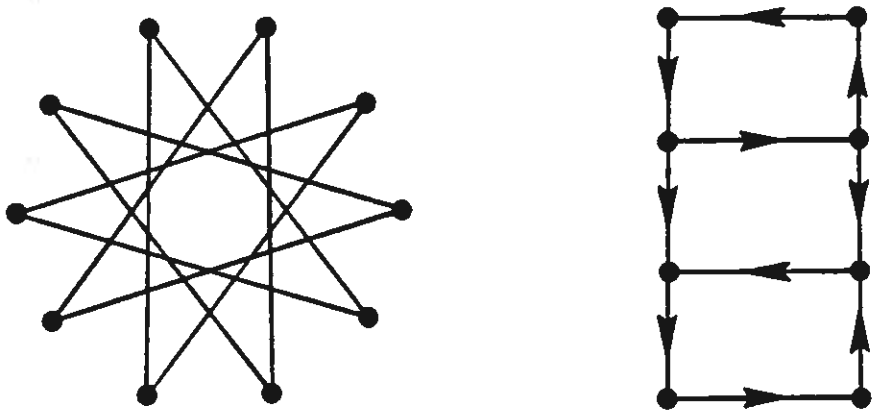
4. SOME IMPORTANT CLASSES OF GRAPHS. A graph V is called a *path* if its vertices can be named $p_1, p_2, \ldots, p_m$ such that p_i and p_j are connected by an edge if and only if $|i-j| = 1$; a path contains no multiple edges. We shall write $V = (p_1, p_2, \ldots, p_m)$ and call V a p_1, p_m–path. p_1 and p_m are called the *end–vertices* of the path, and the path is said to *connect* p_1 and p_m . The *length* n of V is defined as the number of edges belonging to V , $n = m - 1$. All non–directed paths of length n are isomorphic. In the figure, the non–directed paths are drawn.



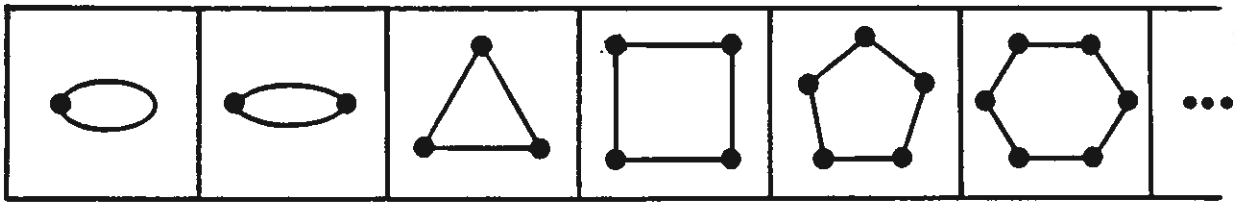
Directed paths are obtained from these by directing the edges in an arbitrary way. A path $V = (p_1, p_2, \ldots, p_m)$ is called a *one–way path* or a $p_1 \to p_m$–path when there exists a $p_i \to p_j$–edge precisely when $j - i = 1$. p_1 is called the *start–vertex* and p_m the *end–vertex* of V . The figure shows a one–way path of length 5.



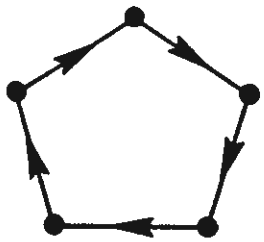
A graph is called *connected* when two arbitrary vertices in Γ are connected by a path in Γ. A directed graph Γ is called *strongly connected* when, for two arbitrary vertices p and q in Γ, there exists a $p \to q$–path aswell as a $q \to p$–path in Γ. The graph to the left in the figure below is not connected, while the graph to the right is connected but not strongly connected.



A connected graph Γ whose vertices all have valency 2 is called a *circuit*. The *length* of the circuit C is defined as the number of edges in C. A non–directed circuit of length n is denoted C_n. In the figure, the non–directed circuits are drawn.

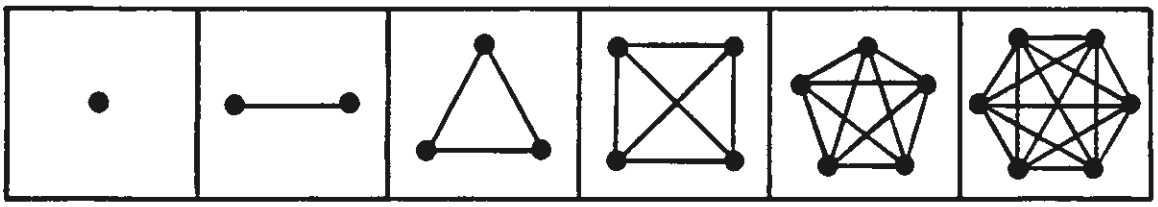


Directed circuits are obtained from these by directing the edges in an arbitrary way. The directed circuit C is called a *one–way circuit* when $v_i(p) = v_u(p) = 1$ for all $p \in \mathcal{P}(C)$. The figure shows a one–way circuit of length 5.



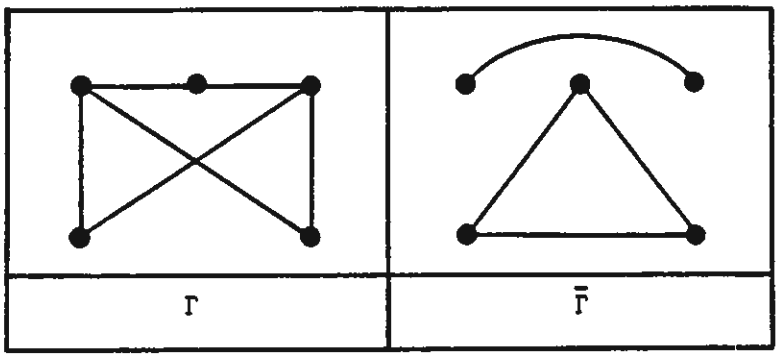
A circuit of length n is called an n–edge. A *Hamiltonian circuit* in a graph Γ is defined as a circuit in Γ containing all the vertices in Γ.

A graph K is called a *complete* graph if K is a simple graph in which two arbitrary vertices are connected by an edge. The complete graph with m vertices is denoted K_m. The figure shows the complete graphs with ≤ 6 vertices. K_m has $\binom{m}{2}$ edges.



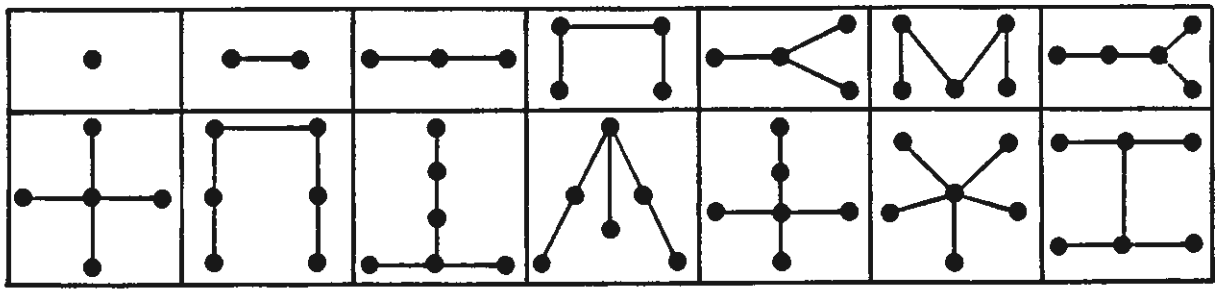
If you give each edge in K_m a direction, you obtain a graph which is called a *tournament*. A tournament with m vertices is denoted T_m. When $m \geq 3$, there exist several non–isomorhpic tournaments with m vertices.

Let Γ be a simple graph. The simple graph $\overline{\Gamma}$, which has the same vertex set as Γ, and in which two vertices are neighbours precisely when they are not neighbours in Γ, is called the *complement*, or the *complementary graph*, to Γ. The figure below shows the graph Γ with 5 vertices and the complementary graph $\overline{\Gamma}$. Clearly, $\overline{\overline{\Gamma}}$ is isomorphic to Γ.



One of the classes of graphs which plays the largest role for different applications of graph theory is trees. In classical investigations of Kirchhoff concerning electrical networks [4] from 1842, trees are the decisive tool. But also recent applications of graph theory in computer science use trees in different ways.

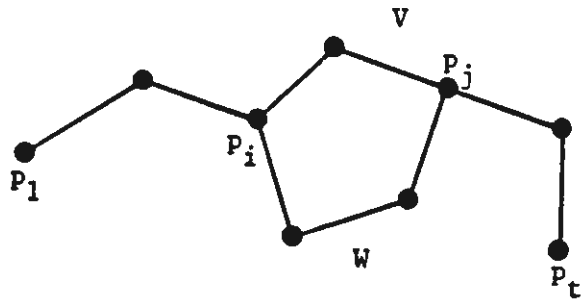
A *tree* is a connected graph which does not contain any circuits. In the figure, all non–directed trees with ≤ 6 vertices are drawn.



Theorem 4.1. Let Γ be a graph without looops with P vertices and K edges. Then the following statements are equivalent:

(a) Γ is a tree.
(b) Two arbitrary vertices in Γ are connected by one and only one path.
(c) Γ is connected and $P = K + 1$.
(d) Γ contains no circuits and $P = K + 1$.

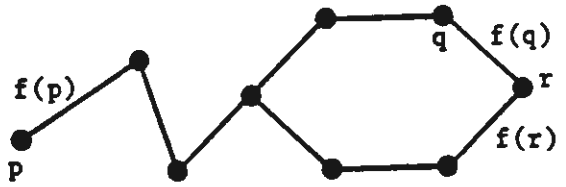
Proof. We shall prove that (a) $\Rightarrow$ (b) $\Rightarrow$ (c) $\Rightarrow$ (d) $\Rightarrow$ (a) . (a) $\Rightarrow$ (b) . Since Γ is connected, two arbitrary vertices of Γ are connected by a path. Suppose there were two vertices p_1 and p_t in Γ , which were connected by two different paths V and W . Put $V = (p_1, p_2,..., p_t)$. Let i be the largest index such that W contains the subpath $V_1 = (p_1, p_2,..., p_i)$, and let j be the smallest index greater than i such that both V and W contain p_j . Those subpaths of V and W which are between p_i and p_j will then constitute a circuit in Γ , which contradicts the fact that Γ is a tree.



(b) $\Rightarrow$ (c) . Since two arbitrary vertices in Γ are connected by a path, Γ is connected. The equation $P = K + 1$ is shown by induction after P . From (b) follows that for $P = 1$, K must be zero, so that the equation is satisfied. Suppose next that the equation is satisfied for graphs with fewer than P vertices satisfying (b) , and consider a graph Γ with P vertices which satisfies (b) . Let $k = (p,q)$ be an edge

in Γ . According to (b) , the only p,q–path in Γ is the edge k . Therefore Γ–k consists of two connected subgraphs Γ_1 and Γ_2 ; since these satisfy (b) according to the induction assumption, the equations $|\mathcal{P}(\Gamma_1)| = |\mathcal{K}(\Gamma_1)| + 1$ and $|\mathcal{P}(\Gamma_2)| = |\mathcal{K}(\Gamma_2)| + 1$. Since $K = |\mathcal{K}(\Gamma)| = |\mathcal{K}(\Gamma_1)| + |\mathcal{K}(\Gamma_2)| + 1$, we obtain the equation $P = K + 1$ by addition.

(c) ⇒ (d) . We shall show that Γ contains no circuits. Suppose, on the contrary, that Γ contains a circuit C . We shall define a one–one mapping f of $\mathcal{P}(\Gamma)$ into $\mathcal{K}(\Gamma)$ and by this show that $K \geq P$, which is a contradiction. f is defined as follows: C contains the same number of vertices and edges, and we let these correspond to each other by the mapping f . Next take any $p \in \mathcal{P}(\Gamma) \setminus \mathcal{P}(C)$. Let V be a shortest path connecting p with a vertex of C , and let k be the edge of V which is incident to p . We then put $f(p) = k$. A little argument shows that f is one–one.



(d) ⇒ (a) . We shall show that Γ is connected. If Γ is not connected, Γ will consist of $S \geq 2$ connected supgraphs, each of which is a tree. For such a subgraph Δ , we have $|\mathcal{P}(\Delta)| = |\mathcal{K}(\Delta)| + 1$. Adding these equations, we obtain $P = K + S$, and since $S \geq 2$, this is a contradiction. □

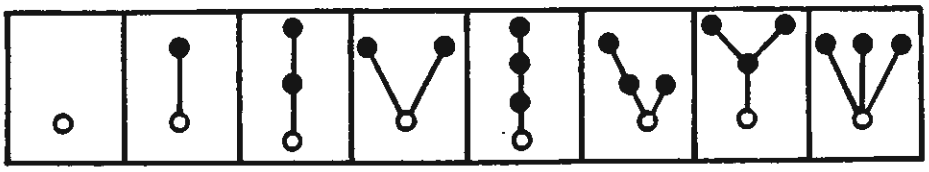
As a corollary of the theorem, we note that *all trees with P vertices have the same number of edges, namely P – 1 .*

Furthermore, we remark that since

$$\sum_{p \in \mathcal{P}(\Gamma)}{}' v(p) = 2K = 2(P-1) ,$$

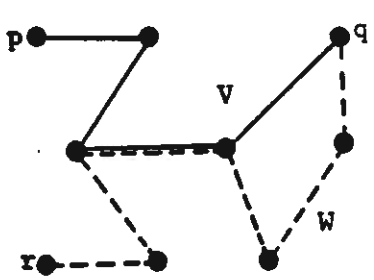
for $P \geq 2$, at least two of the P terms on the left hand side must be 1 . Therefore *a tree with at least two vertices contains at least two vertices with valency 1 .*

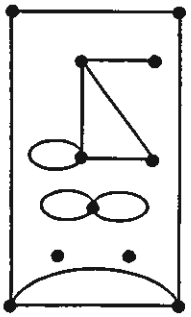
A *tree with root* is a tree in which a vertex is specified to be called the *root.* Two trees with root are called *isomorhpic* if there exists an isomorphic mapping taking the root in one tree to the root in the other tree. The figures below show all trees with root (o) with up to four vertices. There exist two natural ways to direct a tree with root: You can direct all edges away from the root, or all edges towards the root.



5. CONNECTIVITY IN GRAPHS. CUT VERTICES. BRIDGES. As we have mentioned before, the graph Γ is called *connected* if two arbitrary vertices in Γ *are* connected by a path in Γ. Consider now an arbitrary, not necessarily connected graph Γ, and let us define a relation $\sim$ on $\mathcal{P}(\Gamma)$ by the following definition:

$p \sim q$ means there exists a p, q–path in Γ.





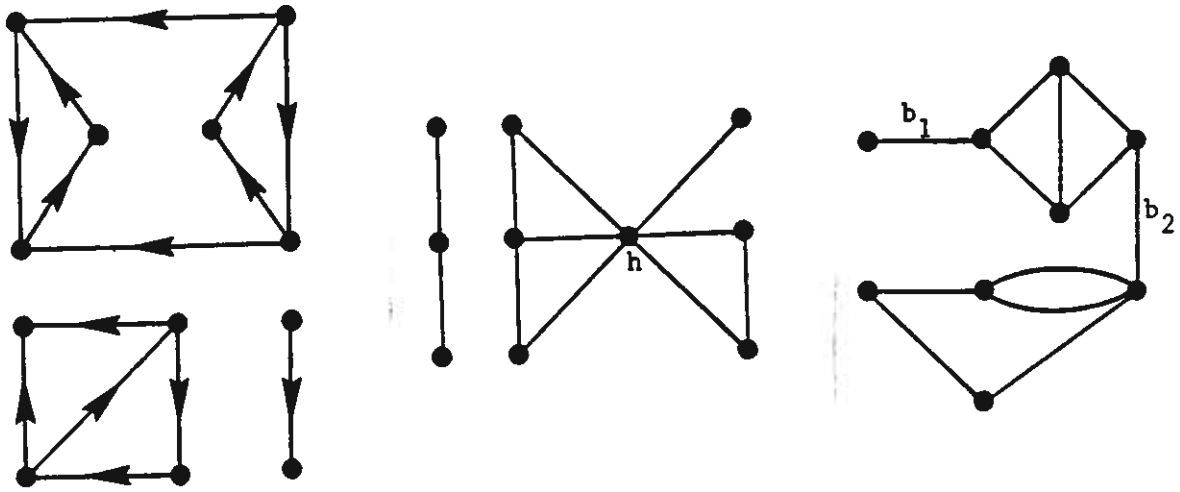
Then

1) $p \sim q$.
2) If $p \sim q$ then $q \sim p$.
3) If $p \sim q$ and $q \sim r$ then $p \sim r$.

Clearly, 1) and 2) are true. It is easy to prove 3); compare the figure. A relation satisfying 1), 2) and 3) is called an *equivalence relation.* Corresponding to this equivalence relation, there exists a partitioning of $\mathcal{P}(\Gamma)$ into equivalence classes such that two vertices in Γ belong to the same equivalence class if and only if they are connected by a path in Γ. A subgraph Γ whose vertex–set is an equivalence class, and whose edge–set is all the edges of Γ connecting two of these vertices, is called a connected *component* in Γ. In this way, every not–connected graph consists of connected subgraphs. The figure shows a graph with 5 connected components.

A connected component could also be defined as a "maximal" connected subgraph in Γ, that is a connected subgraph in Γ which is not a proper subgraph of any connected subgraph of the graph. A forest is a graph without circuits. Every connected component in a forest is a tree. If there are P vertices and S connected components in a forest, then there are $K = P - S$ edges.

If Γ is a directed graph, it is possible to define *strong components* in Γ in a similar way. Two vertices p and q in Γ belong to the same strong connected component in Γ if and only if there exist a $p \to q$–path in Γ and a $q \to p$–path in Γ. The edge–set for a strong component consists of all the edges which connect two vertices in the component. The figure shows a graph with 6 strong components. You see that 5 of the edges do not belong to any of these. Γ is called *strongly connected* when it has only one connected component.

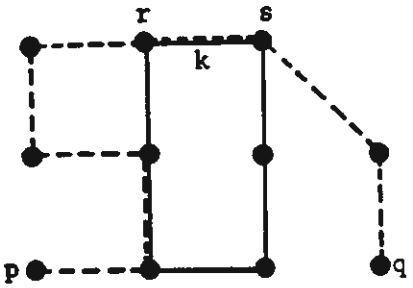


A vertex h in a graph Γ is called a cut–vertex in Γ when $\Gamma - h$ has more connected components than Γ. In the figure above, in the middle, h is a cut–vertex since Γ has two connected components, while $\Gamma - h$ has 4. A graph Γ is called a *block* if Γ is connected and does not contain any cut–vertices. When Γ is a block and $|\mathcal{P}(\Gamma)| \geq 3$, Γ is called a *2–connected graph.*

An edge b in a graph Γ is called a *bridge* in Γ if $\Gamma - b$ contains more connected components than Γ. The graph in the figure above right contains two bridges b_1 and b_2. When b is a bridge, then $\Gamma - b$ contains exactly one more connected component than Γ. An end–vertex of a bridge is a cut–vertex unless it has valency 1 (or is only incident to loops).

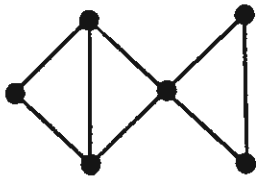
Theorem 5.1. An edge b is a bridge in a graph Γ if and only if b does not belong to any circuit in Γ.

Proof. Let k be an edge in Γ which is contained in a circuit C in Γ, and let Δ be the connected component in Γ which contains k. In Δ, two arbitrary vertices p and q are connected by a path V. If V does not contain k, then p and q are also connected with a path, namely V, in Δ–k. If V (as in the figure) contains the r,s–edge k, then there exists in Δ–k a p,r–path, an r,s–path, namely C–k, and an s,q–path. Therefore there exists also a p,q–path in Δ–k. Since Δ–k is connected, k is not a bridge in Γ.



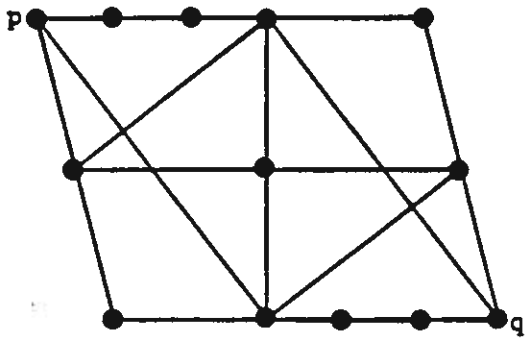
Assume conversely that $k = (p,q)$ is not a bridge in Γ . Then there exists a p,q–path V in $\Gamma - k$, and together with k , this path constitutes a circuit containing k .

A connected graph Γ without bridges with $|\mathscr{P}(\Gamma)| \geq 2$ is called *2–edge connected.* The figure shows a graph which is 2–edge connected, but not 2–connected. Every graph which is 2–connected, that is without cut–vertices, will also be 2–edge connected; for if there was a bridge, at least one of the end vertices would be a cut–vertex.



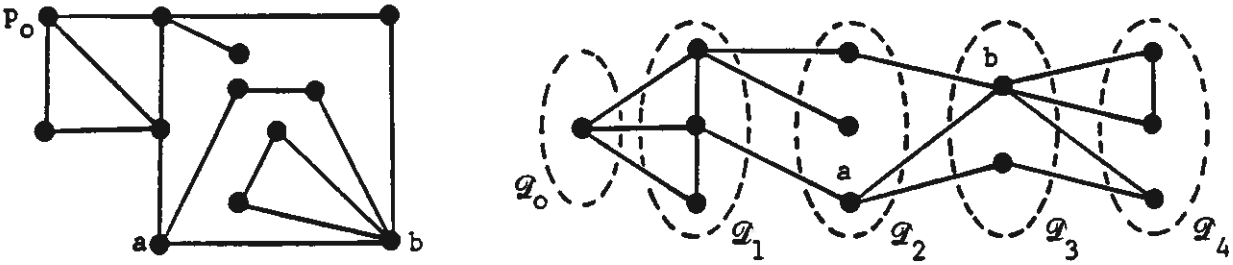
6. DISTANCE. DISTANCE CLASSES. BIPARTITE GRAPHS. COLOURING. In this section, we shall define the concept of distance between two vertices in a graph. As we should expect, it is only possible to define a natural distance measure in a graph Γ if Γ is connected.

So let p and q denote two vertices in Γ (they may be the same vertex). Then there exists a p,q–path in Γ . Among all the p,q–paths in Γ , there exist one or several, whose lengths are smaller than all the other p,q–paths in Γ . The length of such a shortest p,q–path in Γ is called the *distance* between p and q in Γ ; it is denoted $d(p,q)$ or $d_\Gamma(p,q)$. In the figure, $d(p,q) = 3$, and there exist 2 shortest p,q–paths.



We shall now define the concept of a distance class in a connected graph. The description of an arbitrary connected graph, which can be given by means of this concept, makes it easy in many cases to formulate clear arguments, when we are going to prove properties of graphs.

Let p_0 be a fixed vertex in a connected graph Γ. Let $\mathscr{D}_i$ denote the set of those vertices q in Γ for which $d_\Gamma(p_0,q) = i$, $i = 0, 1, 2, \ldots$. $\mathscr{D}_i$ is called the i'th *distance class* in Γ from p_0. In the figure below to the left, we have drawn a graph containing a vertex p_0. To the right, the same graph is drawn showing the distance classes $\mathscr{D}_i$.



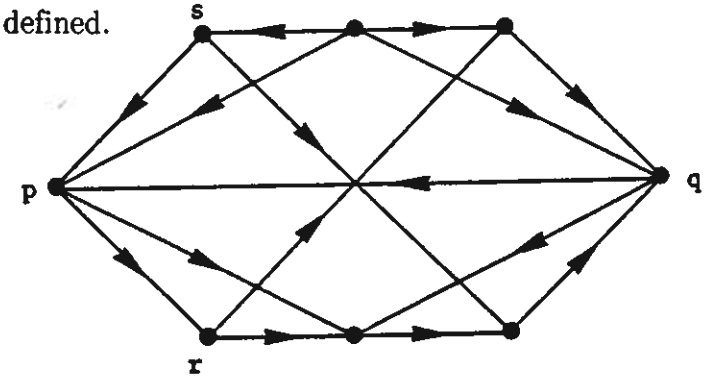
Theorem 6.1. Let p_0 be a vertex in a connected graph Γ, and let $\mathscr{D}_0 = \{p_0\}$, $\mathscr{D}_1, \mathscr{D}_2, \ldots$ denote the distance classes in Γ from p_0. Then:

1. The sets $\mathscr{D}_0, \mathscr{D}_1, \mathscr{D}_2, \ldots$ are pairwise disjoint, and every vertex in Γ belongs to one of these sets.
2. For $i = 1, 2, 3, \ldots$ every vertex in $\mathscr{D}_i$ will be connected with at least one vertex in $\mathscr{D}_{i-1}$ by an edge.
3. When $|i-j| > 1$, there exists no edge which connects a vertex in $\mathscr{D}_i$ with a vertex in $\mathscr{D}_j$.

Proof. Statement 1 is obvious. Let p_i denote a vertex in $\mathscr{D}_i$, and let $V = (p_0, p_1, \ldots, p_{i-1})$ be a shortest p_0, p_i–path in Γ. p_{i-1} is then a vertex in $\mathscr{D}_{i-1}$. From this, 2 follows. The proof of 3 is indirect. Suppose that there exist two numbers i and j such that $|i-j| > 1$, and such that there exists a $\mathscr{D}_i, \mathscr{D}_j$–edge k. Let p_i and p_j denote the end vertices of k, where $p_i \in \mathscr{D}_i$, $p_j \in \mathscr{D}_j$. We may assume that $j > i+1$. Then there exists a p_0, p_i–path of length i, and from this it follows that there exists a p_0, p_j–path of length $i+1$. It follows that $d_\Gamma(p_0, p_j) = j \leq i+1$. But since $p_j \in \mathscr{D}_j$, we have that $d_\Gamma(p_0, p_j) = j > i+1$, and we have a contradiction.

□

The concepts distance and distance class can also be defined for directed graphs. If p and q are two vertices in a directed graph Γ, and if there exists a $p \to q$–path in Γ, we define the distance $d_\Gamma(p \to q)$ *from* p *to* q as the length of a shortest $p \to q$–path in Γ. In the figure, $d(p \to q) = 3$ and $d(q \to p) = 1$. The distance $d(r \to s)$ is not defined.

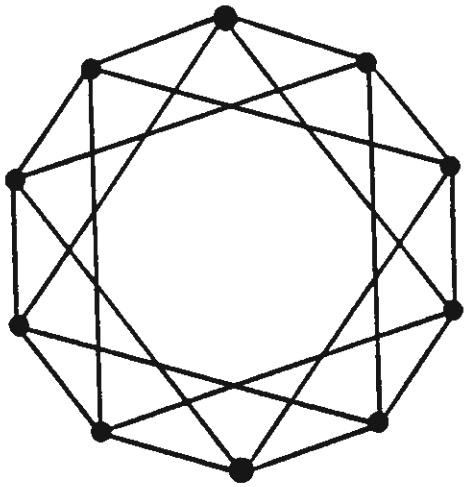


When p is a fixed vertex in a directed graph, we define a *(directed) distance class* $\mathscr{D}_i$ *from* p_0 as the set of those vertices q for which $d(p_0, q) = i$, $i = 0, 1, 2, \ldots$. If for every vertex q in Γ there exists a $p_0 \to q$–path in Γ, there is a theorem corresponding to 1.4 with the following two changes: In 2, every vertex in $\mathscr{D}_i$ will be connected to at least one vertex in $\mathscr{D}_{i-1}$ with a $\mathscr{D}_{i-1} \to \mathscr{D}_i$–edge. And in 3 there exists no $\mathscr{D}_i \to \mathscr{D}_j$–edge when $j - i > 1$.

The *diameter* $\delta = \delta(\Gamma)$ of a connected graph Γ is defined as the maximal distance between two vertices of Γ, that is

$$\delta(\Gamma) = \max_{p,\, q \,\in\, \mathscr{P}(\Gamma)} d_\Gamma(p,q) .$$

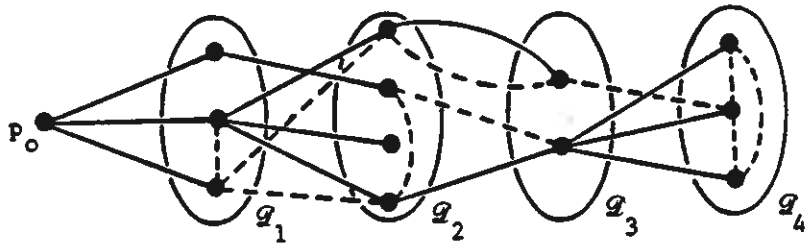
The graph in the figure has diameter 3. The two vertices drawn with large circles have distance 3, and there do not exist vertices with larger distance.



The next theorem is an example of a theorem which can be proved in a simple way using the concept of distance class.

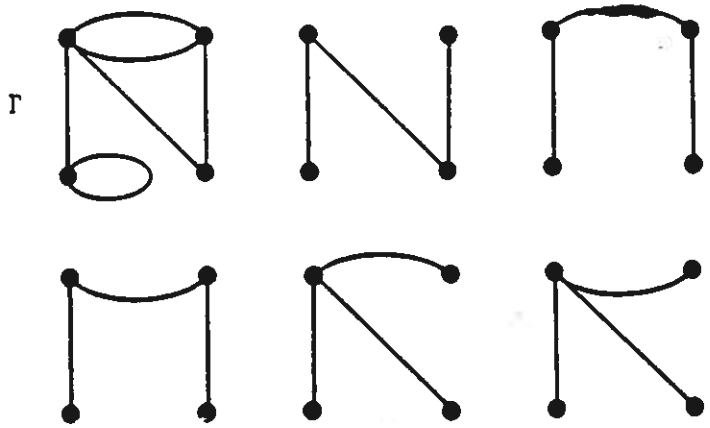
Theorem 6.2. Every connected graph Γ has a spanning tree.

Proof. Let p_0 be a fixed vertex in Γ and let $\mathscr{D}_0$, $\mathscr{D}_1$, $\mathscr{D}_2$, ... denote the distance classes from p_0. Let T be a subgraph of Γ obtained in the following way: Delete from Γ every edge which connects two vertices in the same distance class. For every vertex $p \neq p_0$ in $\mathscr{D}_i$ delete all p, $\mathscr{D}_{i-1}$–edges except one. According to theorem 6.1, 2, every vertex in p in Γ is connected to p_0 by a path in T. Therefore T is a connected and spanning graph. Assume that T contains a circuit C, and let q be a vertex in C with maximal distance from p_0. The two edges of C incident to q both connect q with $\mathscr{D}_{j-1}$, if q is in $\mathscr{D}_j$, contradicting the definition of T.

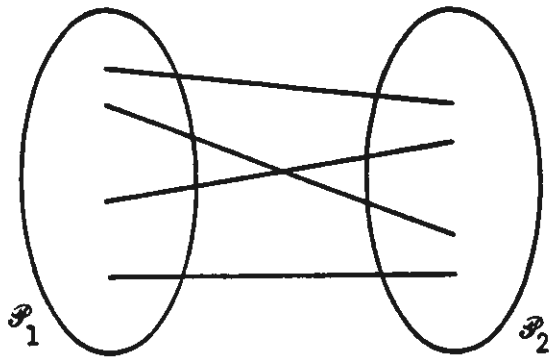


Therefore T is a spanning tree in Γ. Remark that in fact we have shown the stronger theorem, that if p_0 is a vertex in Γ, then Γ has a spanning tree T, for which $d_\Gamma(p_0, p) = d_T(p_0, p)$ for all $p \in \mathscr{P}(\Gamma)$.

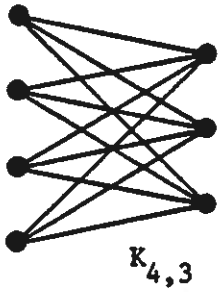
When we have chosen a specific spanning tree T in Γ, then the edges of T are called *branches*, and those edges in Γ which do not belong to T are called *chords*. The figure below shows a graph Γ and all the spanning trees in Γ.



The graph Γ is called *bipartite* when $\mathscr{P}(\Gamma)$ can be partitioned into two disjoint sets $\mathscr{P}_1$ and $\mathscr{P}_2$ such that every edge in Γ is a $\mathscr{P}_1$, $\mathscr{P}_2$–edge (that is, it has one end vertex in $\mathscr{P}_1$ and the other in $\mathscr{P}_2$). $\mathscr{P}_1$ and $\mathscr{P}_2$ are called *colour classes* of Γ (compare next page). A bipartite graph is shown in the figure.



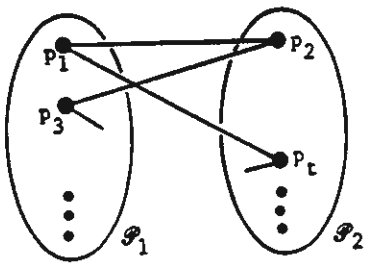
A simple bipartite graph with colour classes $\mathscr{P}_1$ and $\mathscr{P}_2$ is called a *complete bipartite graph* and denoted K_{m_1,m_2} if $m_1 = |\mathscr{P}_1|$, $m_2 = |\mathscr{P}_2|$ and if an arbitrary vertex in $\mathscr{P}_1$ is connected to an arbitrary vertex in $\mathscr{P}_2$ with exactly one edge. The figure shows the graph $K_{4,3}$.



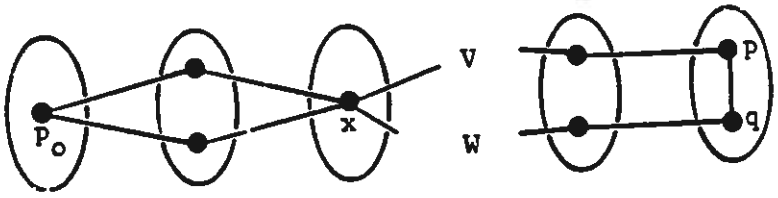
The graph $K_{m,1}$ is called a *star* with $m+1$ vertices. If $\mathscr{P}_1$ and $\mathscr{P}_2$ are colour classes for a bipartite graph Γ , then $(\mathscr{P}_1, \mathscr{P}_2)$ is called a *bipartition* of Γ .

Theorem 6.3. A graph Γ is bipartite if and only if every circuit in Γ has even length.

Proof. Let Γ be a bipartite graph with colour classes $\mathscr{P}_1$ and $\mathscr{P}_2$, and let C be a circuit in Γ consisting of the path $V = (p_1, p_2, \ldots , p_t)$ together with a p_t, p_1–edge. If for instance $p_1 \in \mathscr{P}_1$, then $p_2 \in \mathscr{P}_2$, $p_3 \in \mathscr{P}_1$, $p_4 \in \mathscr{P}_2$ and so on. Therefore $p_i \in \mathscr{P}_2$ if and only if i is even. Since p_t is connected to p_1 , p_t must be in $\mathscr{P}_2$; t is therefore even.

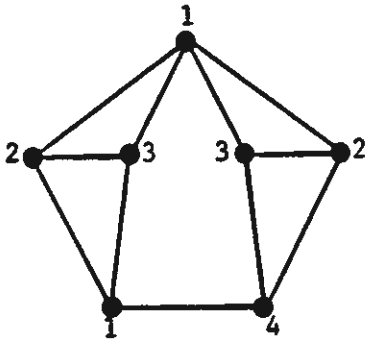


Assume conversely that Γ is a graph in which all circuits are even, and let $\mathscr{D}_0$, $\mathscr{D}_1$, $\mathscr{D}_2$, ... denote the distance classes from a vertex p_0 in Γ. Put $\mathscr{P}_1 = \mathscr{D}_0 \cup \mathscr{D}_2 \cup \mathscr{D}_4 \cup \ldots$ and $\mathscr{P}_2 = \mathscr{D}_1 \cup \mathscr{D}_3 \cup \mathscr{D}_5 \cup \ldots$. Assume that two vertices p and q in the same one of these two sets are connected by an edge in Γ. According to theorem 6.1, 3, p and q must belong to the same distance class $\mathscr{D}_i$. Let V be a shortest p_0, p–path and W be a shortest p_0, q–path, and let x be a vertex with maximal distance from p_0 which is common to V and W.



Then the x, p–part of V and the x, q–part of W together with the p, q–edge will be an odd circuit in Γ. This contradiction shows that all edges in Γ are $\mathscr{P}_1$, $\mathscr{P}_2$–edges, that is, Γ is bipartite. □

A k–*colouring* of a graph Γ is defined as a mapping of $\mathscr{P}(\Gamma)$ into a set with k elements, called "colours", such that when two vertices are connected by an edge then they have different colours. The figure shows a 4–colouring of a graph with the "colours" 1, 2, 3 and 4.



That a graph has a 2–colouring, is exactly the same as to say that the graph is bipartite. This follows from the fact that we can give the vertices of $\mathscr{P}_1$ on colour, and the vertices of $\mathscr{P}_2$ the other colour. This is the background for calling $\mathscr{P}_1$ and $\mathscr{P}_2$ colour classes.

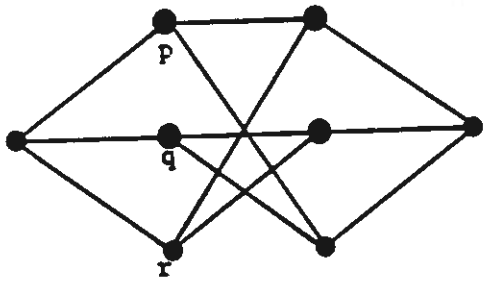
The smallest number $\chi = \chi(\Gamma)$ for which Γ has a χ–colouring, is called the *chromatic number* for Γ, and Γ is called χ–*chromatic*. Theorem 6.3 gives a simple characteristic of the two chromatic graphs. It is not possible in a similar way to characterise graphs with $\chi > 2$.

7. CONNECTIVITY. The definitions of the concepts connected graph and cut–vertex in Section 5 imply that there are two natural ways to measure the degree of connectivity of a graph. You can either investigate *how well it sticks together*, that is, how many paths there are connecting two arbitrary vertices; but you can also investigate *how easy it is to pull it apart*, that is, how few vertices it is sufficient to take away if you want to form another connected graph. Let us make precise the two ways to measure connectivity.

We first have to define the concept of a cut in a graph. A set $\mathscr{S} \subseteq \mathscr{P}(\Gamma) \cup \mathscr{K}(\Gamma)$ is called a *cut* in the graph Γ if there exists a connected component Δ in Γ such that $\Delta - (\mathscr{S} \cap (\mathscr{P}(\Delta) \cup \mathscr{K}(\Delta)))$ is not connected. A cut which consists of vertices in Γ is called a *vertex–cut*. A cut consisting of edges in Γ is called an *edge–cut*. A cut–vertex is a vertex–cut consisting of one vertex, and a bridge is an edge–cut consisting of one edge.

We shall now describe the two different ways to measure the "degree of connectivity" of a graph.

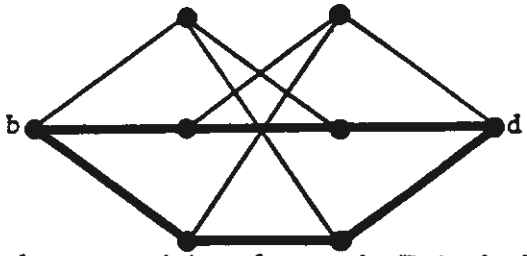
Let us start by considering the graph in the figure. The 4 large vertices constitute a vertex–set. This does not contain any proper subset which is also a vertex–set. Nevertheless, the graph has a vertex–cut consisting of only 3 vertices, for instance $\{p, q, r\}$ is a vertex–cut.



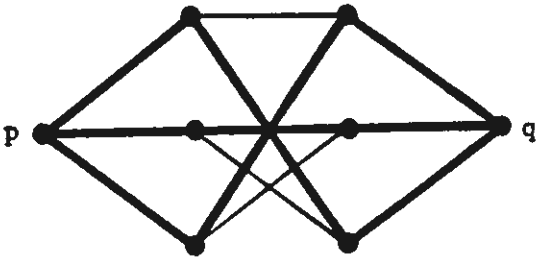
A natural measure for the degree of connectivity, or just the connectivity, of a graph Γ, is *the smallest number* $\sigma = \sigma(\Gamma)$ *for which there exists a vertex–cut consisting of* σ *vertices.* (If two arbitrary vertices in Γ are connected by an edge, there exists no vertex–cut in Γ. In this case we put $\sigma(\Gamma) = |\mathscr{P}(\Gamma)| - 1$.) For the graph in the figure above, it is possible by trying a row of possiblities to prove that $\sigma(\Gamma) = 3$.

The other way by means of which we can measure connectivity is by use of disjoint paths connecting the different vertices in a graph. So let Γ be a graph and let p and q be two vertices in Γ. For simplicity, in this context we shall call two p,q–paths *disjoint* when the only vertices they have in common are p and q. For instance, the

two heavy–lined paths in the figure are disjoint p,q–paths. These paths cannot be supplemented with a third one which is disjoint to the two given; nevertheless, there exist – as the next figure shows – 3 pairwise disjoint p,q–paths.



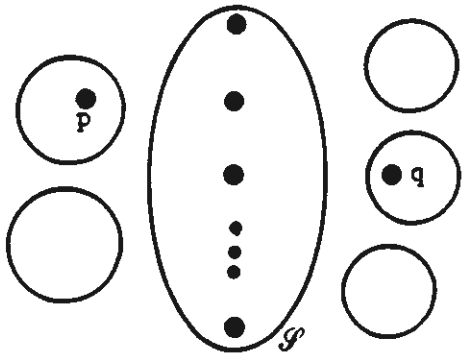
A natural measure for the connectivity of a graph Γ is the largest number $\kappa = \kappa(\Gamma)$ for which for every pair (p, q) of vertices in Γ, there exist at least κ pairwise disjoint p,q–paths in Γ. By trying some possibilities, it is possible to prove that the graph in the figure has $\kappa = 3$.



For complete graphs, we have that $\kappa(K_m) = m-1$ and this value of κ holds for any graph with m vertices in which two arbitrary vertices are connected by at least one edge.

Now let Γ be an arbitrary graph with a vertex–cut $\mathscr{S}$ consisting of $\sigma(\Gamma)$ vertices, and let p and q be in two different connected components in $\Gamma - \mathscr{S}$. In Γ, there exist $\kappa(\Gamma)$ pairwise disjoint p,q–paths. Each of these must contain at least one vertex of $\mathscr{S}$. Therefore we have

$$\sigma(\Gamma) = |\mathscr{S}| \geq \kappa(\Gamma) .$$



A famous theorem, which is called Menger's theorem, states that in fact in this inequality we have an equality sign:

Menger's theorem. For every graph Γ, we have that $\kappa(\Gamma) = \sigma(\Gamma)$.

The proof of Menger's Theorem is complicated; In section 27, we return to the probelm. The common value of $\sigma(\Gamma)$ and $\kappa(\Gamma)$ is called the *connectivity* of Γ; normally it is denoted $\kappa(\Gamma)$. For every $n \leq \kappa(\Gamma)$ we call Γ an *n–connected graph.* Menger's theorem can then be formulated as follows:

Theorem 7.1. For every graph Γ the following three statements are equivalent:

(a) Γ is n–connected.

(b) $|\mathcal{P}(\Gamma)| \geq n + 1$ and two arbitraty vertices in Γ are connected by at least n pair–wise disjoint paths.

(c) $|\mathcal{P}(\Gamma)| \geq n + 1$ and Γ does not have a vertex cut with cardinality $< n$.

We obtain a related way to measure connectivity by operating with edges instead of with vertices. Corresponding to Menger's Theorem, we have that for an arbitrary graph with at least two vertices, the smallest number λ for which Γ has an edge cut consisting of λ edges is equal to the largest number μ for which two arbitrary vertices in Γ are connected with at least μ pair–wise edge disjoint paths. The common value of λ and μ is called the *edge–connectivity* of Γ; normally it is denoted $\lambda(\Gamma)$. For every $n \leq \lambda(\Gamma)$, we call Γ *n–edgeconnected.* Corresponding to Theorem 7.1, we have the following edge version of Menger's Theorem:

Theorem 7.2. For every graph Γ the following three statements are equivalent:

(a) Γ is n–edgeconnected.

(b) 2 arbitrary vertices in Γ are connected with at least n pair–wise edge disjoint paths.

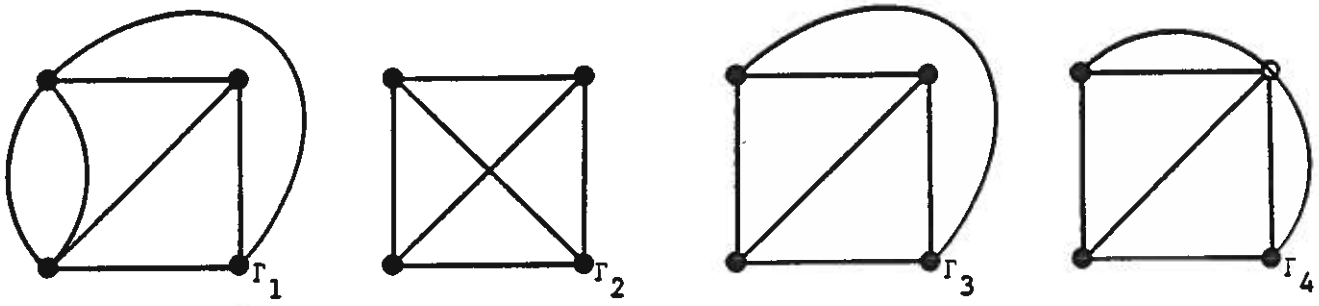
(c) Γ does not have any edge cut with cardinality $< n$.

It is easy to show that for every graph Γ we have that

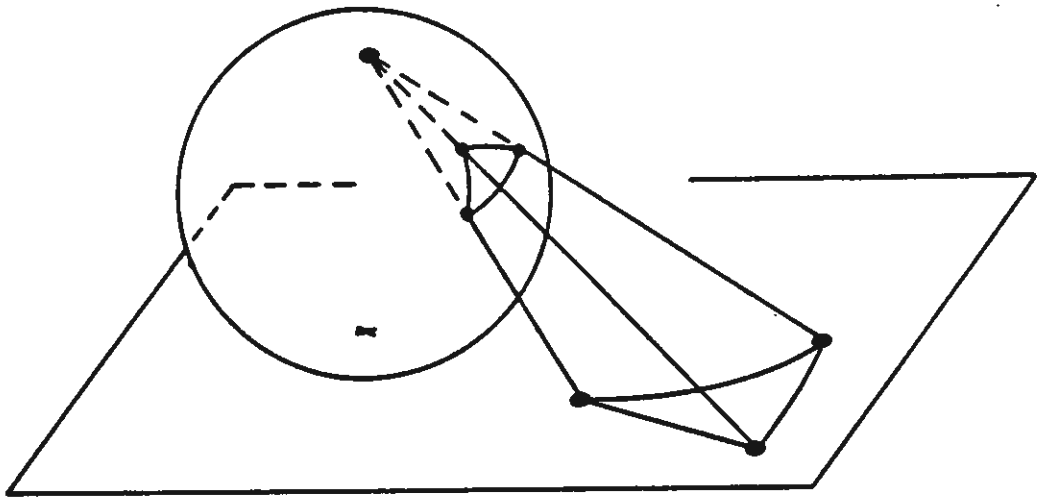
$$\kappa \leq \lambda \leq \nu ,$$

where ν is the valency of a vertex in Γ with minimal valence.

8. PLANAR GRAPHS. A graph is called *plane* if its vertex set is a set of points in the plane, and when an arbitrary p,q–edge is a continuous curve without double points in the plane, connecting the points p and q , and when two arbitrary edges do not have other points in common than, maybe, their common points. A graph is called *planar* when it is isomorphic to a plane graph. In the figure on the next page, Γ_1 is a plane graph, while Γ_2 is not plane. Γ_2 is planar, since Γ_3 is a plane graph isomorphic to Γ_2 . The graph Γ_4 which has 6 edges, of which one is drawn through a point which it is not incident to, is not plane. The edges of a plane graph sub–divide the plane into (maximal) regions, the interiors of which do not contain vertices or edges of the graph. These regions are called *meshes*. The set of meshes of a plane graph Γ is denoted $\mathcal{M}(\Gamma)$. One of the meshes is the infinte mesh surrounding the whole graph. To say that a graph is plane, is the same as to say that it can be drawn on a sphere without two edges having other points in common than, maybe, common end points.

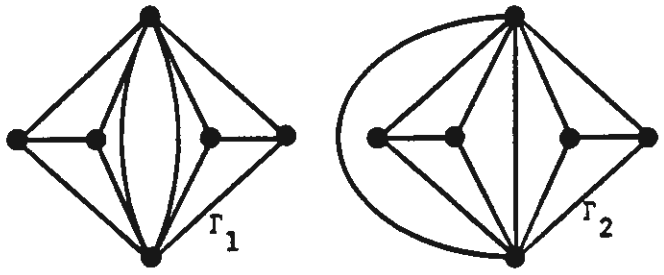


This is seen by mapping the plane onto a sphere, with the plane as a tangent plane, by projection from the diametrically opposite point to the point of contact. By projecting on another plane to the sphere, we see that when M is an arbitrary mesh in a plane graph Γ., then Γ is isomorphic to another graph in which M corresponds to the infinite mesh. Thorough descriptions of the theory on planar graphs can be found in [10] and in [18].



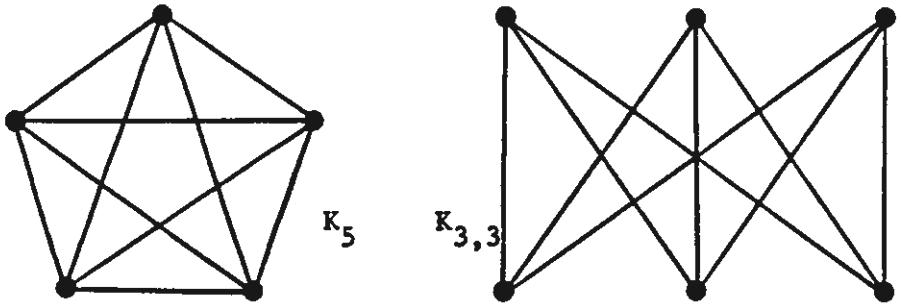
If k is an edge in a plane graph Γ, then k belongs to the edge of two meshes M_1 and M_2. If k is a bridge in Γ, then $M_1 = M_2$, otherwise $M_1 \neq M_2$. Normally, the boundary of a mesh is a circuit in Γ, but if Γ contains cut vertices, this is wrong for certain meshes on the boundary of which there is a cut vertex; it is also wrong for certain meshes in not–connected graphs.

In the figure, we have shown two isomorphic plane graphs Γ_1 and Γ_2. Γ_1 contains a mesh whose boundary is a circuit of length 2 ; this is not the case with Γ_2. It is therefore not possible to find an isomorphic mapping of Γ_1 on Γ_2, which can be extended to a one–to–one mapping $\mathscr{P}(\Gamma_1) \cup \mathscr{K}(\Gamma_1)$ on $\mathscr{P}(\Gamma_2) \cup \mathscr{K}(\Gamma_2)$, which preserves incidences and which takes the boundary of the mesh in Γ_1 to the boundary of the mesh in Γ_2.

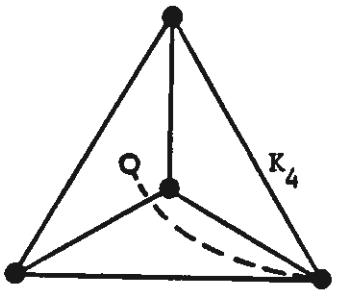


This phenomenon can only occur for graphs which are not 3–connected. It is possible to show that a planar, 3–connected graph Γ can only be drawn in one way as a plane graph. "only" in the sense that we can arbitrarily choose which mesh we want to be the infinite one, but when this is done the graph theoretic structure of Γ uniquely determines which circuits in the graph are boundaries of a mesh.

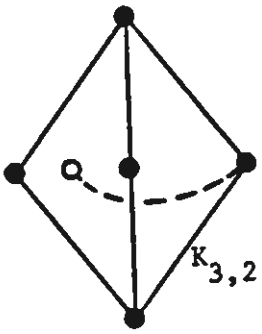
One of the most important problems in the theory of plane graphs is to characterise those graphs which are planar. As an introduction to this problem, we shall show that K_5 and $K_{3,3}$ *are not planar.*



If we delete a vertex from K_5, we obtain the graph K_4. When K_4 is drawn as a plane graph, we obtain a partitioning of the plane in 4 meshes. The boundary of each mesh is a triangle. The fifth vertex of K_5 must belong to one of these meshes, but when you connect it with the vertices in K_4 which do not belong to the boundary of this mesh, we obtain a non–planar graph. Therefore, K_5 is not planar.



In a corresponding way, it can be shown that $K_{3,3}$ is not planar: If we delete a vertex from $K_{3,3}$, we obtain the graph $K_{3,2}$, consisting of 3 paths of length 2 with common end vertices. When this graph is drawn in the plane, we obtain 3 meshes. When the vertex we deleted is put into one of these meshes and when we connect it to the vertex of $K_{3,2}$, not belonging to the boundary of this mesh, we obtain a non—plane graph.



When we (maybe serveral times) replace a p,q–edge in a graph Γ by a path connecting p and q (and whose vertices, different from p and q, have valency 2), we obtain a new graph which is called a *sub–division* of Γ. Clearly a sub–division of K_5 and $K_{3,3}$ is not planar. It is the contents of a deep theorem by Kuratowski, [6], [2], that the converse statement is also true:

Kuratowski's Theorem. A graph is planar if and only if it does not contain a sub–graph which is a sub–division of K_5 or $K_{3,3}$.

It is complicated to prove Kuratowski's theorem. The simplest proof is due to Carsten Thomassen, [20]. From a practical point of view, Kuratowski's theorem is not very useful because it is not algorithmically easy to check whether a given graph contains a subdivision of K_5 or $K_{3,3}$. From a theoretical point of view, Kuratowski's theorem is interesting for the following reason, among others: In the definition of planarity, elements of the geometry of the plane occur in an essential way. Nevertheless, Kuratowski's theorem gives a purely graph–theoretical condition for a graph to be planar.

In 1752, Euler published, in [4], a formula which gives a relation between the number of vertices, edges and faces in a polyhedron. This formula is not only valid for polyhedrons, but for arbitrary plane graphs:

Euler's formula. For an arbitrary plane graph Γ with S connected components, we have that

$$|\mathscr{P}(\Gamma)| + |\mathscr{M}(\Gamma)| - |\mathscr{K}(\Gamma)| = S + 1 .$$

The formula can be proved by induction on the number of edges. If Γ contains only one edge k, the formula is easy to prove, both when k is a bridge and when k is a loop. Let us after this assume that the formula is correct for all graphs with less than K edges, and let Γ be an arbitrary graph with K edges. Let k be an edge in Γ, and put $\Gamma_1 = \Gamma - k$. It follows from the induction assumption that

$$|\mathscr{P}(\Gamma_1)| + |\mathscr{M}(\Gamma_1)| - |\mathscr{K}(\Gamma_1)| = S_1 + 1 \tag{1}$$

where S_1 denotes the number of connected components in Γ_1. Now we have:

If k is not a bridge in Γ:

$$|\mathscr{P}(\Gamma)| = |\mathscr{P}(\Gamma_1)|\ ,\ |\mathscr{M}(\Gamma_1)| = |\mathscr{M}(\Gamma)| - 1\ ,\ |\mathscr{K}(\Gamma_1)| = |\mathscr{K}(\Gamma)| - 1\ ,\ S = S_1\ .$$

If k is a bridge in Γ:

$$|\mathscr{P}(\Gamma)| = |\mathscr{P}(\Gamma_1)|\ ,\ |\mathscr{M}(\Gamma_1)| = |\mathscr{M}(\Gamma)|\ ,\ |\mathscr{K}(\Gamma_1)| = |\mathscr{K}(\Gamma)| - 1\ ,\ S_1 = S + 1\ .$$

In both cases, we obtain Euler's formula for Γ by inserting in (1). □

As an example of the application of Euler's formula, we shall show that for a simple plane graph Γ with at least 3 vertices, we have that:

$$|\mathscr{K}(\Gamma)| \leq 3|\mathscr{P}(\Gamma)| - 6\ . \tag{2}$$

By adding "diagonals" in a suitable way in the meshes of Γ, you can obtain a connected plane graph Γ_1 in which the boundary of each mesh is a triangle. From this it follows that $3|\mathscr{M}(\Gamma_1)| = 2|\mathscr{K}(\Gamma_1)|$. From Eulers formula, we obtain

$$|\mathscr{K}(\Gamma_1)| = |\mathscr{P}(\Gamma_1)| + |\mathscr{M}(\Gamma_1)| - 2 = |\mathscr{P}(\Gamma_1)| + \tfrac{2}{3}|\mathscr{K}(\Gamma_1)| - 2\ .$$

From this we obtain

$$|\mathscr{K}(\Gamma_1)| \leq 3|\mathscr{P}(\Gamma_1)| - 6\ .$$

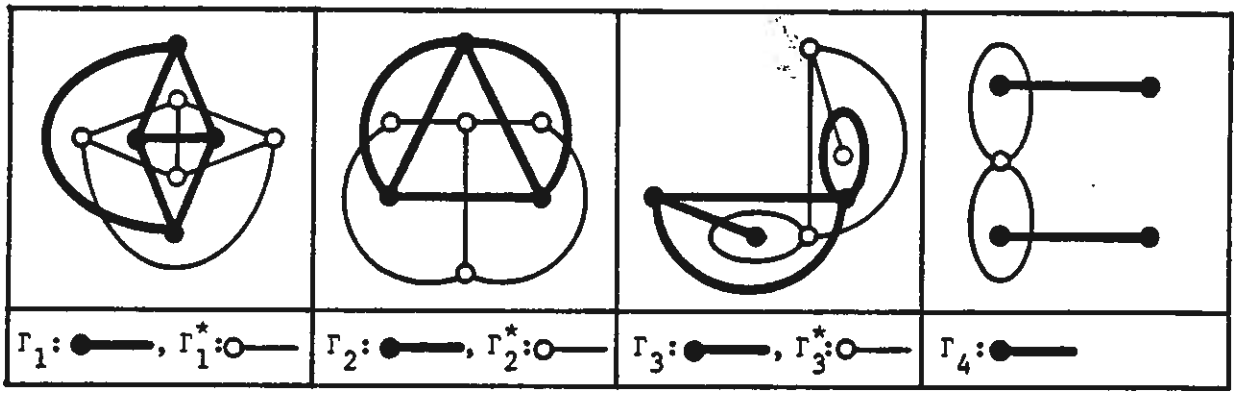
and from this again, that

$$|\mathscr{K}(\Gamma)| \leq |\mathscr{K}(\Gamma_1)| = 3|\mathscr{P}(\Gamma)| - 6\ .$$

The formula (2) expresses that plane graphs are "thin"; cf. Section 11.

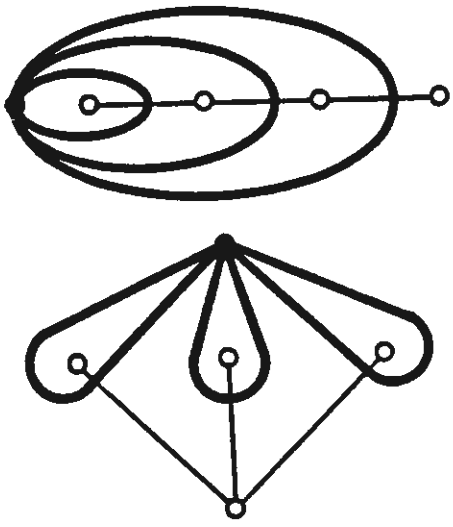
We conclude this section by mentioning the concept of a dual graph of a plane graph. Let Γ be a connected plane graph with $\mathscr{K}(\Gamma) \neq \emptyset$. *The dual graph* Γ^* *of* Γ is obtained in the following way : $\mathscr{P}(\Gamma^*)$ consists of one vertex in the interior of each mesh in Γ, and for each edge k in Γ, there is an edge k' in Γ^* which connects the vertices of the two meshes of which k belongs to the boundary. If k is a bridge in Γ, k' becomes a loop in Γ^*. k' is drawn such that it has exactly one point in common with k and no points in common with other edges in Γ, and furthermore,

the edges of Γ^* are drawn such that Γ^* becomes a plane graph. Naturally, we can choose the position of the points in Γ^* in different ways, and the same is valid for the shape of the edges (straight or curved lines, for instance), but Γ^* is uniquely determined up to isomorphism. This makes it allowable to use the word *the* dual graph Γ^* of Γ. In the figures below, we have shown 3 graphs Γ_1, Γ_2 and Γ_3, drawn with black vertices and heavy lines. Their dual graphs are shown with circles and thin lines. Since Γ_1 is isomorphic to Γ_1^*, the first figure shows that K_4 is self–dual. Γ_2 is not isomorphic to Γ_2^*. The third figure illustrates that bridge (respectively loop) in Γ corresponds to loop (respectively bridge) in Γ^*.

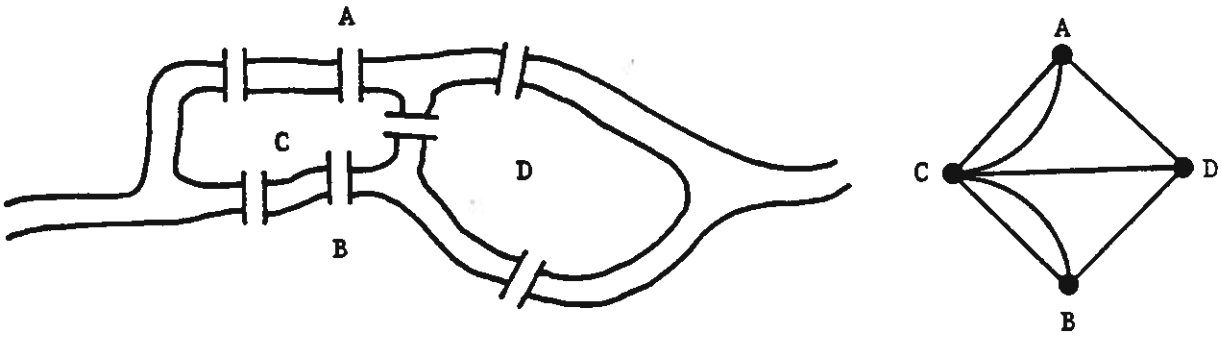


It is possible to show that if Γ^* is dual graph to Γ, then Γ is dual to Γ^*. If we had extended the definition of a dual graph to cover not–connected graphs, this statement would have been wrong, as the graph Γ_4 in the figure above shows. It is not difficult to see that when Γ is connected, then Γ^* is also connected.

We mentioned earlier that a (at most 2–connected) graph Γ can sometimes be drawn in the plane in several ways. In the figure, we show how a graph with one vertex and three loops can be drawn in two plane versions. Note that the corresponding dual graphs are not isomorphic!



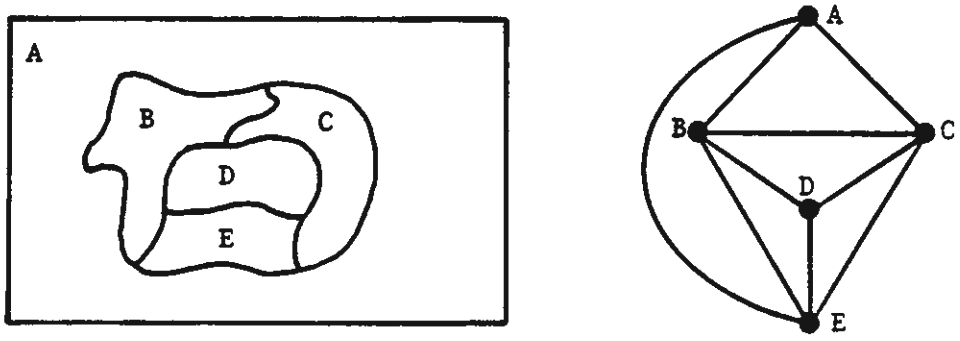
9. ABOUT HISTORY AND LITERATURE. The first time the concept graph occurs in mathematical literature is in 1736, when the Swiss mathematician Leonhard Euler wrote a paper called "The solution of a problem in connection with positions", [7]. The subject was the following: In the city Koenigsberg (Kaliningrad), there were 7 bridges across the river Pregel, which runs through the town. Euler was asked whether it was possible to go for a walk in Koenigsberg such that you passed over each bridge once. Euler considered the graph shown in the figure, which has an edge corresponding to each bridge in Koenigsberg.



The problem is then whether it is possible to order the 7 edges in the graph in such a way that for 3 arbitrary consecutive edges, we have that the edge in the middle has one end vertex in common with the preceding edge, and the other end vertex in common with the third edge. Such a sequence of edges is called an *Eulerian tour* in the graph, and Euler showed that the graph corresponding to the bridges of Koenigsberg does not have an Eulerian tour. It is not hard to prove; try it yourself. Euler showed that a necessary and sufficient condition for the graph Γ to have an Eulerian tour is that Γ contains two or no vertices with odd valency.

Many times in the history of mathematics, it has happened that a new mathematical discipline has been created because other sciences came up with problems of a mathematical nature. In this way, the first large contribution to graph theory occurred. It is due to G. Kirchhoff. Kirchoff found the laws for electrical networks, which now carry his name. These laws imply systems of linear equations for the determination of unknown currents and voltages in electrical networks. Among other things, Kirchhoff found out how you can find maximal subsistence of linearly independent equations among the network equations, and how it is possible to solve the equations by means of graph theory, [1]. We return to this and other similar problems in the chapter on electrical networks.

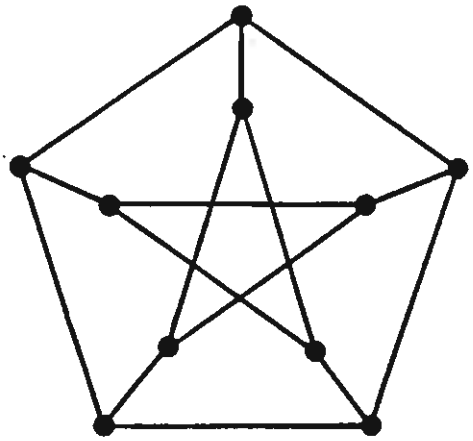
The so–called 4–colour problem was first formulated in the middle of the 19th century. The problem is, whether it is always possible to use only four colours if we want to colour a map such that two arbitrary countries who share a piece of boundary obtain different colours. It is rather easy to see that it is possible to formulate the problem as follows: Consider a graph which can be drawn in the plane such that no edges intersect. Is it then possible, using at most four colours, to give each vertex in the graph a colour, such that two vertices which are connected by an edge always get different colours? In the figure we have shown a map where it is necessary to use four colours, and a corresponding graph which has a vertex corresponding to each country, and in which two vertices are connected by an edge if and only if the corresponding countries have a piece of boundary in common.



The four colour problem probably has no practical interest, but nevertheless there has been an incredible amount written about it. The problem is probably solved, since in 1977, Appel and Haken gave an affirmative answer. However, in the original proof there were several errors, but it turns out that every time somebody finds a new error, Appel and Haken are able to repair the proof. However, the proof is very complicated, consisting of several hundred sides of text supplemented by a lot of computer programmes. It is difficult to know what to do with such a proof.

Around 1895, Julius Petersen, who was a professor at the Technical University of Denmark, published some papers on graph theory, which contain some of the deepest known results on graphs. One of the problems which Petersen worked with was the following [17]: Let Γ be a 2–edge–connected graph in which all vertices have valency 3. Is it then possible to colour the edges of Γ with 3 colours, such that two edges incident with the same vertex always get different colours?

Petersen found that the answer is no, and gave as a counterexample the graph in the figure, which still occurs very often in graph–theoretical literature, and is known over the whole world as *Petersen's graph.*

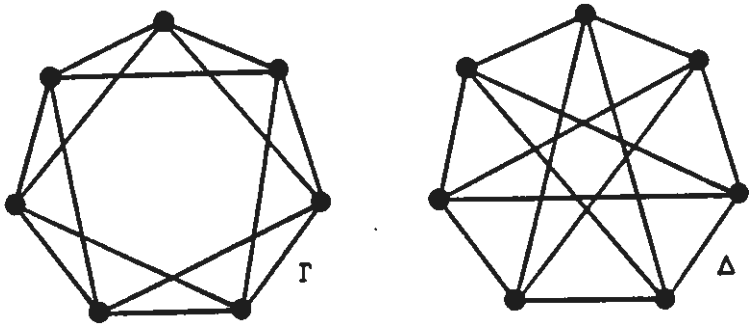


It is remarkable that if we assume that Γ can be drawn in the plane without two edges intersecting, then the above problem will be equivalent to the 4–colour problem.

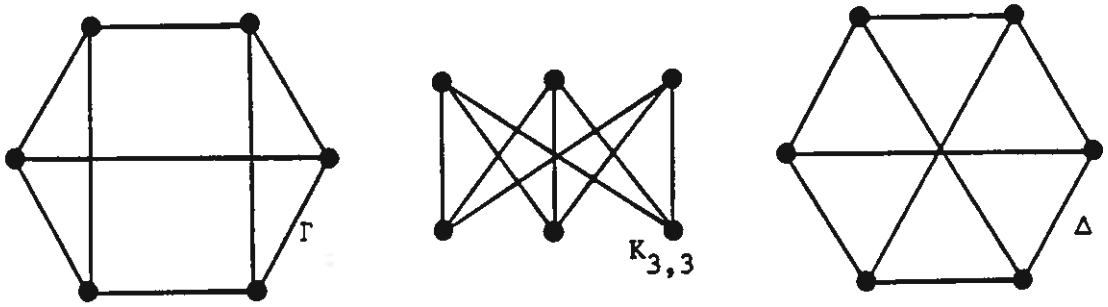
As a mathematical discipline, graph theory has developed in particular since the end of the 19th century. The first book of graph theory appeared in 1936. It was Denes Koenigs "Theorie der Endlichen und Unendlichen Graphen". It was later reprinted. Koenig's book is not directed towards the applications; it is written for mathematicians. In 1962, Ford and Fulkerson's "Flows in Networks" came out. It deals with transport networks. In "Linear Graphs and Electrical Networks" from 1961, Seshu and Read gave a thorough treatment of the connection between graphs and the theory of electrical networks. Later books on the same subject include Mayedas "Graph Theory" from 1972. A standard reference is Harary's "Graph Theory" from 1969. It is written in a rather entertaining style. A readable, rather easy but good introduction to graph theory is Robin Wilson's "Introduction to Graph Theory" from 1972. Bondy and Murty's "Graph Theory with Applications" gives short descriptions of many applications of graph theory. Later books dealing with discrete optimization include Christofides: "Graph Theory. An Algorithmic Approach", 1975, and Lawler: "Combinatorial Optimization" from 1972. In "The Design and Analysis of Computer Algorithms" from 1974, Aho, Hopcroft and Ullman deal with, among other things, representations of graphs on computers, and graph–theoretical algorithms. These are the main subject of Shimon Evans' "Graph Algorithms" from 1979. Lovasz: "Combinatorial Problems and Excursions" and Bela Bollobas: "Extremal Graph Theory" are two large and very good books on graph theory. Finally, it should be mentioned that the first book on the history of graph theory appeared in 1976. It was Biggs, Lloyd and Wilson "Graph Theory 1736 – 1936". Reference [23] contains descriptions of many different applications of graph theory.

PROBLEMS FOR CHAPTER 1 .

1.1. Are the graphs in the figure isomorphic?



1.2. Which of the graphs shown in the figure are isomorphic?



1.3. Draw all not–directed trees with 7 vertices.

1.4. Draw all tournaments with 4 vertices.

1.5. Draw all directed circuits of length 6.

1.6. Draw all directed paths of length 4.

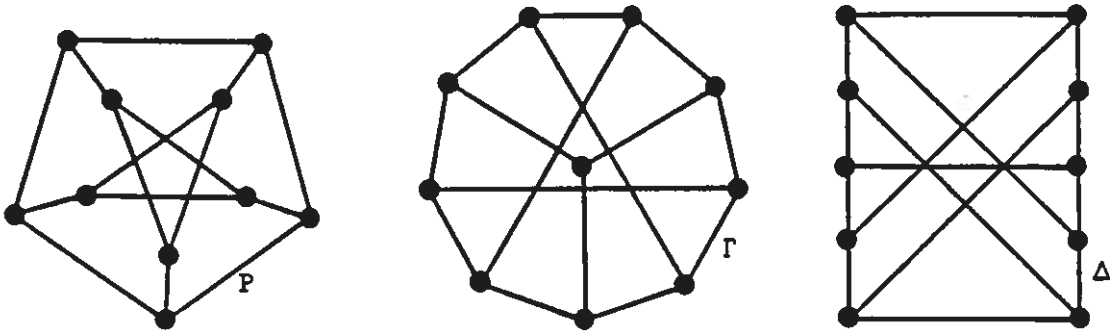
1.7. Draw all simple graphs with 5 vertices and 4 edges.

1.8. Determine all simple graphs with at most 7 vertices which are isomorphic to their compliment.

1.9. Draw all directed graphs without loops with 3 vertices and 3 edges.

1.10. Find all non–directed graphs Γ for which $\Gamma - x$ is a circuit for all $x \in \mathscr{P}(\Gamma)$.

1.11. Are the graphs in the figure isomorphic?



1.12. Let Γ be a connected graph and let t denote the length of a longest path in Γ. Show that two arbitrary paths V_1 and V_2 in Γ, both having lenght t, have a vertex in common.

1.13. Show that a simple graph Γ with P vertices is connected if $v(p) \geq \frac{1}{2}(P-1)$ for all $p \in \mathscr{P}(\Gamma)$. Is the bound we have given best possible ?

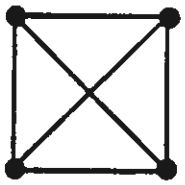
1.14. Determine all simple graphs with a valency spectrum (2, 3, 3, 4, 4, 4).

1.15. Determine all simple graphs with a valency spectrum (2, 2, 3, 3, 4, 4).

1.16. Do there exist graphs without loops with the following valency spectras,
a) (1, 2, 2, 3, 5) b) (2, 2, 3,, 3, 4) c) (1, 1, 2, 6)
d) (1, 2, 3, 4, 4) e) (1, 2, 2, 2, 3, 5, 6, 7) ?

Do there exist graphs without multiple edges with these valency spectras ?

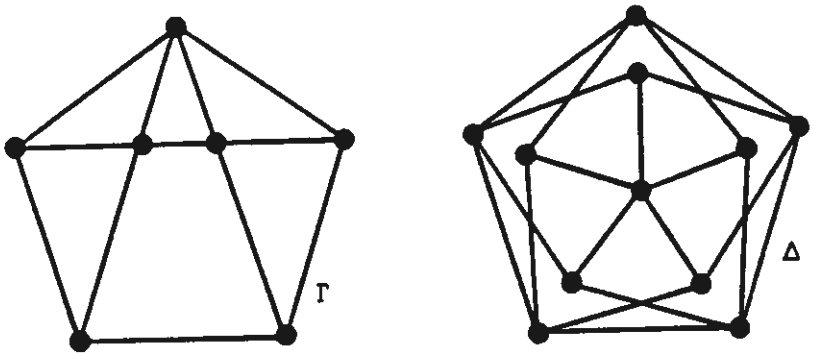
1.17 Draw all spanning trees in the graph in the figure. Two trees are considered different if they do not contain precisely the same edges.



1.18. Can a Eulerian graph contain a bridge?

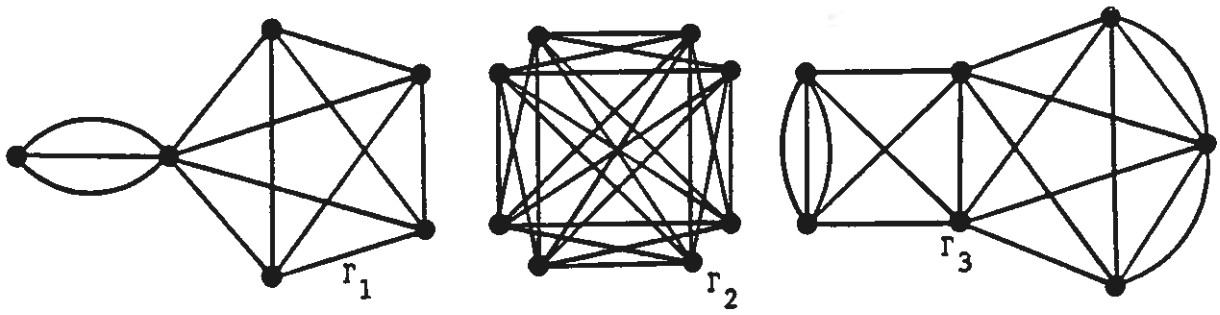
1.19. Must a simple graph Γ with at least 2 vertices necessarily contain two vertices with the same valency?

1.20. Determine the chromatic number for each of the graphs in the figure.

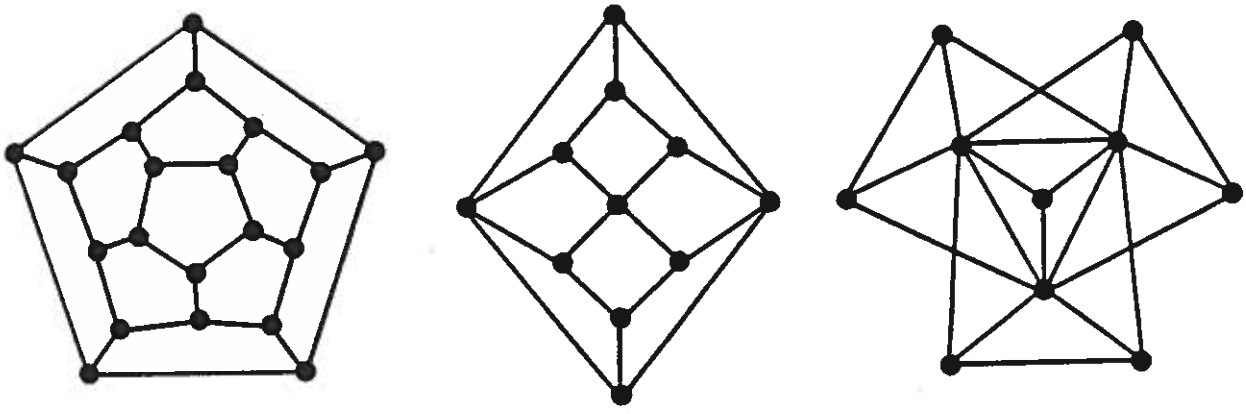


Is it valid for every edge k that the chromatic number goes down when you delete k from the graph (Γ or Δ) ?

1.21. Find the connectivity κ, the edge connectivity λ, and the minimumvalency ν for each of the graphs in the figure.

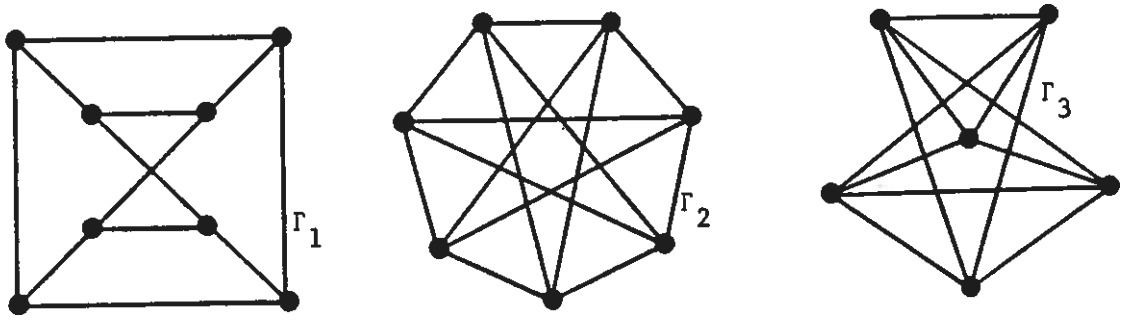


1.22. Show that when Γ has a Hamiltonian circuit and when $\mathcal{M}$ is a vertexcut in Γ, then the number of connected components in $\Gamma - \mathcal{M}$ is at most $|\mathcal{M}|$. Which of the following graphs has a Hamiltonian circuit?



1.23. Is it possible to colour the edges of Γ_6 with two colours without creating a triangle in which all three edges have the same colour?

1.24. Which of the graphs in the figure are planar and which are not planar ?



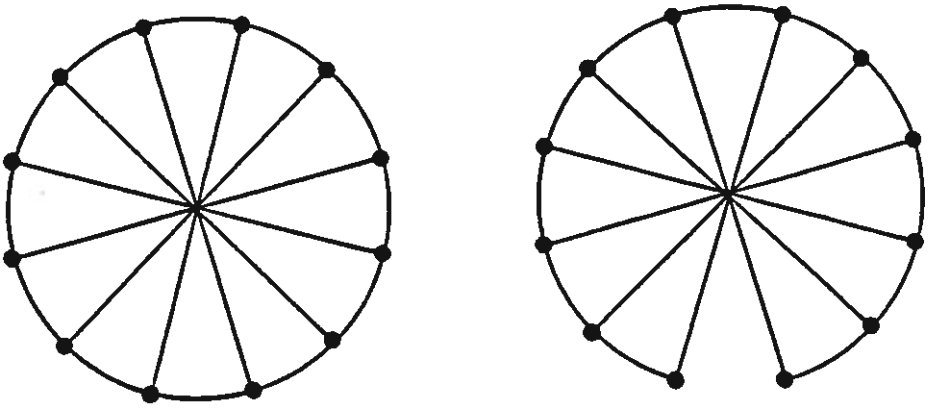
1.25. Find all simple plane graphs with at most 3 vertices.

1.26. Show that a simple planar graph contains a vertex with valency ≤ 5 . Is this result best possible?

1.27. Characterise those connected simple plane graphs Γ for which

$$| \mathscr{K}(\Gamma) | = 3 | \mathscr{P}(\Gamma) | - 7$$

1.28. Are the graphs in the figure plane ? If so, draw them in the plane, if not, draw them in the plane with a minimum number of edge intersections. (Three or more edges are not allowed to have a common intersection.)



1.29. Show that if in a graph Γ there exists a circuit containing the edges k_1 and k_2 and a circuit containing the edges k_2 and k_3 then there exists in Γ also a circuit containing k_1 and k_3 .

LITERATUR FOR CHAPTER 1.

[1] Aho, Alfred V., Hopcroft, John E., Ullman, Jeffrey D.: The Design and Complexity of Computer Algorithms. Addison-Wesley 1975.

[2] Biggs, N.L., E.K. Lloyd, R.J. Wilson: Graph Theory 1736-1936. Claredon Press 1976.

[3] Bollobás, Béla: Extremal Graph Theory. Academic Press 1978.

[4] Bondy, J.A., U.S.R. Murty: Graph Theory with applications. The MacMillan Press Ltd. 1976.

[5] Cayley, A.: On the theory of the analytical forms called trees. Phil. Mag. (4) 13 (1857), 172-176.

[6] Christofides, Nicos: Graph Theory. An Algorithmic Approach. Academic Press 1975.

[7] Euler, L.: Solutio problematis ad geometriam situs pertinentis. Comm. Acad. Sci. Imp. Petropol. 8 (1736), 128-148.

[8] Even, Shimon: Graph Algorithms. Computer Science Press, Inc. 1979.

[9] Ford, L.K. Jr., D.K. Fulkerson: Flows in Networks. Princeton University Press 1962.

[10] Harary, Frank: Graph Theory. Addison-Wesley Publishing Company 1969.

[11] Kirchhoff, G.R.: Über die Auflösung der Gleichungen auf welche man bei der Untersuchung der linearen Vertheilung galvanischer Ströme geführt wird. Ann. Phys. chem. 72 (1842), 497-508.

[12] König, Dénes: Theorie der endlichen und uendlichen Graphen. Chelsea Publishing Company 1935.

[13] Kuratowski, C.: Sur le problème des courbes gauches en topologie. Fund. Math. 15 (1930), 271-283.

[14] Lovász, L.: Combinatorial Problems and Exercises. North Holland Publishing Company 1979.

[15] Lawler, E.L.: Combinatorial Optimization: Networks and Matroids. Holt Rinehart and Winston, 1976.

[16] Mayeda, W.: Graph Theory. John Wiley & Sons, inc. 1972.

[17] Petersen, J.: Sur le théorème de Tait. Interméd. Math. 5(1898), 225-227.

[18] Sachs, H.: Einführung in die Theorie der endlichen Graphen. Teil II. B.G. Teubner, 1972.

[19] Seschu, S., M.B. Reed: Linear Graphs and Electrical Networks. Addison-Wesley, 1961.

[20] Thomassen, C.: Kuratowski's Theorem. J. Graph Theory, 5 (1981), 225-241.

[21] Whitney, H.: Congruent graphs and the connectivity of graphs. Amer. J. Math. 54 (1932), 150-168.

[22] Wilson, R.J.: Introduction to Graph Theory. Longman Group 1972.

[23] Wilson, R.J., Beineke, L.: Applications of Graph Theory. Academic Press, 1979.

CHAPTER 2

GRAPHS.ON A COMPUTER

10. REPRESENTATIONS OF A GRAPH ON A COMPUTER. In Chapter 1, we normally described a graph with the help of a figure. For the human imagination, a figure is that representation of a graph, which most easily gives access to information about the graph. When you are going describe a graph to a computer, you must of course use other data structures.

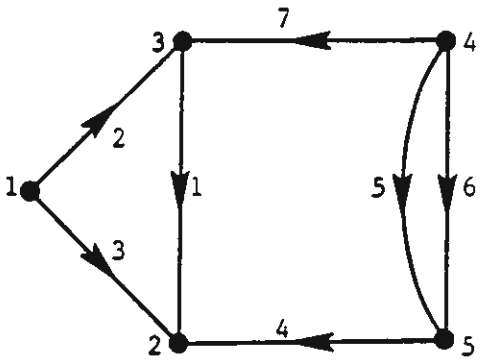
In the following, we shall describe several different ways, which can be used when you have to represent a graph on a computer. Later in the book, we shall describe a selection of graph–theoretical algorithms. When these algorithms have to be executed on a computer, we shall assume that the graph we have to investigate is described by means of one of those representations. This makes it possible to calculate an upper bound for the number of steps which the computer must execute when a certain algorithm is applied to a graph of a given size which is represented in a certain way. This upper bound is called the *complexity* of the algorithm. A calculation of the complexity requires that we specify what we want to consider as a single step. In complexity theory in general, we talk about one step when a computer executes an order – or a sequence of orders – which are such that the time needed for the computer to execute the order does not depend upon the size of the input. In the following, when we talk about "a step", we shall assume that this condition is satisfied. Then it is assumed that the graph is not allowed to be too large, so that the working memory of the computer has room for both the representation of the graph, and the information needed to carry out the order.

Let Γ be a directed graph without loops, with P vertices, denoted $p_1, p_2, \ldots, p_P$ and K edges, denoted $k_1, k_2, \ldots, k_k$. The *incidence matrix* A for Γ is then defined as the $P \times K$ matrix $A = (a_{rs})$, which is defined to be

$$(1) \qquad a_{rs} = \begin{cases} 1 & \text{when } k_s \text{ begins in } p_r, \\ -1 & \text{when } k_s \text{ ends in } p_r, \\ 0 & \text{otherwise.} \end{cases}$$

In the figure below, we show a graph and its incidence matrix.

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



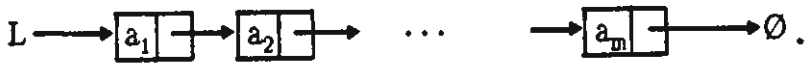
In the figure, the names p_r and k_s are replaced by r and s respectively.

Each column in the matrix A contains precisely 2 elements $\neq 0$, one which is 1, and one which is -1. Conversely, each $P \times K$ matrix A with this property is the incidence matrix for one and only one graph Γ. Γ has P vertices and K edges. If the vertices are denoted $p_1, p_2, \ldots, p_P$, and the edges $k_1, k_2, \ldots, k_K$, then the definition equation (1) describes which vertices are the start vertex and end vertex respectively of k_s, $s = 1, 2, \ldots, K$. Therefore it is possible to use A as a representation of the graph Γ. When we say that we represent the graph Γ by means of the matrix A, we mean that, for given r and s, in one step we can read off the element a_{rs}. Therefore, in one step we can answer the following two questions: "Is p_r the start vertex of k_s?", "Is p_r the end vertex of k_s?". We say that we have *direct access* to a_{rs}. The order "Find the start vertex of k_s" for given s, can only be executed by answering the questions, "Is $a_{1s} = 1$?", "Is $a_{2s} = 1$?", ..., until the answer is yes. Therefore this order in the worst case will require $P - 1$ steps. Similarly, for a given r it can require up to K steps to execute the order, "Find an edge which begins in p_r." In other representations, these questions can be answered in one step. This is one of the reasons why the incidence matrix is actually rarely used when we have to represent a graph Γ on a computer. Another reason is that we have to store $P \cdot K$ numbers if we are going to store A in a computer. And this of course is not very effective; when we know where we have the ± 1's then the zeroes do not contain any new information.

When we want to represent a graph, we can use many different data structures. Here we shall start by using vectors (or matrices) and *(linked) lists*. We shall stress the difference between a vector

$$v = (v_1, v_2, \ldots, v_n)$$

and a list



In a representation, the vector v can only be used in the following way: When i is given, then in one step it is possible to read and/or change v_i .

The list L consists of some "elements" (the boxes); each contains some information (here the a_i's) and furthermore the address of the next element in the list (indicated by means of the arrow). The list L can be used in the following way: When the name L is known, it is possible in one step to follow the arrow to the first element in L . From here it is possible to follow the arrow to the second element in L , etc. Therefore it requires i steps to reach the i–th element of L . When this is done, then in one step it is possible to read a_i , change a_i , delete the element from the list by putting the arrow from the $(i-1)$th element to point to the $(i+1)$th element, or in the same way to insert a new element after the i–th . Sometimes *doubly–linked* lists are used, which can be approached from both ends. When you use doubly–linked lists, then in one step it is possible to link a list L_2 after a list L_1 .

Before it is possible to make a motivated choice of representation of a graph on a computer, you must clarify which demands for storage you will be willing to accept, and which basic orders you expect to carry out so often, that it should be possible to carry those orders out in few steps.

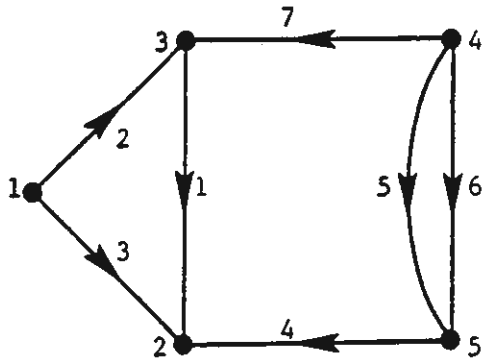
W.r.t. storage, an application of the incidence matrix would require storage of $P \cdot K$ numbers. But, of course, this is not the best possible way. A graph can be represented by $2K$ numbers, namely by giving the names of each end vertex for each edge in the graph. The 3 representations, which we describe below, all demand a storage of this size.

To be more concrete, we shall consider the following 6 simple orders, where p_i is a given vertex and k_j is a given edge:

1. Find an edge which starts in p_i .
2. Find those edges which start in p_i .
3. Find the start vertex of k_j .
4. Find the end vertex of k_j .
5. Delete k_j .
6. Contract k_j .

We shall now describe three different representations of directed graphs, each of which are well suited when certain orders have to be executed in few steps. All the three representations can be considered as compressed versions of the incicdence matrix. Here we may accept graphs which contain loops.

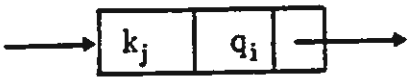
1. Edge lists. This representation consists of K lists, one belonging to each edge in the graph. The list corresponding to k_t contains first the start vertex p_i of k_t , next the end vertex p_j of k_t . This is equivalent to saying, that for each given edge k_t we have direct access to the vector (p_i, p_j) . The next figure shows an example.



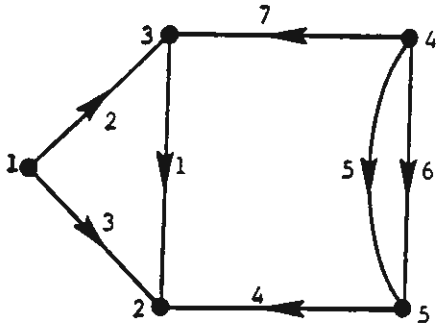
EDGE NAME	LIST	
1	3	2
2	1	3
3	1	2
4	5	2
5	4	5
6	4	5
7	4	3

Clearly, in an edge–list–representation, the orders 3, 4 and 5 (page xxx) can be executed in one step. The orders 1 and 2 can demand up to K steps, since it may be necessary to read the first element in each edge list. Finally, we consider contraction: Assume that in the above example we wish to contract edge 6 . We shall then read the end vertices 4 and 5 in the sixth edge list, and next delete this list. This can be considered as one step, because it is just as fast in a large graph as in a small graph. Next we must choose a name for the new vertex which occurs during the contraction. If we choose 4 as a name for the new vertex, we obtain the edge list for $\Gamma \div 6$, by searching through the lists (except the 6th list) for appearances of 5 . Each time 5 occurs, change it to 4 . Thus it requires $2(K - 1)$ steps to contract an edge. O stands for Order of size, and we shall return to this symbolism later.

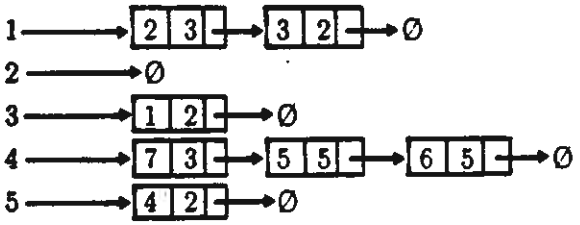
2. Neighbourlists. This representation consists of P lists, one list belonging to each vertex in the graph Γ. The list $L(p_i)$ corresponding to the vertex p_i contains $v_u(p_i)$ elements, one corresponding to each edge which starts in p_i.



The element in the list of p_i which corresponds to the edge k_j (with start vertex p_i) has 3 coordinates, namely k_j itself, its end vertex q_i and the address of the next element in the list. Taken together the neighbourlists contain precisely K elements. Below we show an example.



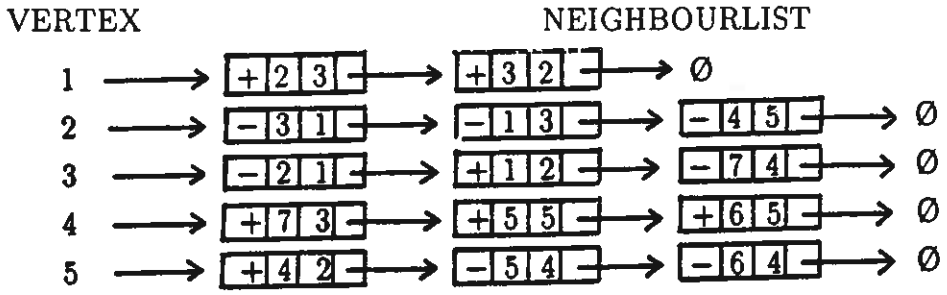
VERTEX NEIGHBOUR LISTS



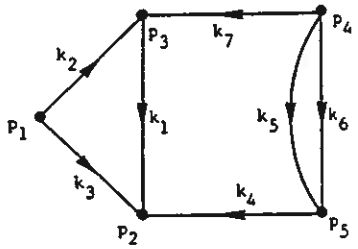
It is clear that in a neighbourlist representation, the orders 1 and 2 (page xxx) can be carried out in 1 and $v_u(p_i)$ steps respectively, which is the best possible. Each of the orders 3 and 4 demand in the worst case one passage, through all the neighbourlists, that is K steps. To delete k_j requires for the same reason K steps, but if in advance we have found p_i in a neighbourlist, it is possible to delete it in one step, such as is illustrated in the figure, where the edge 5 in the example above is deleted.

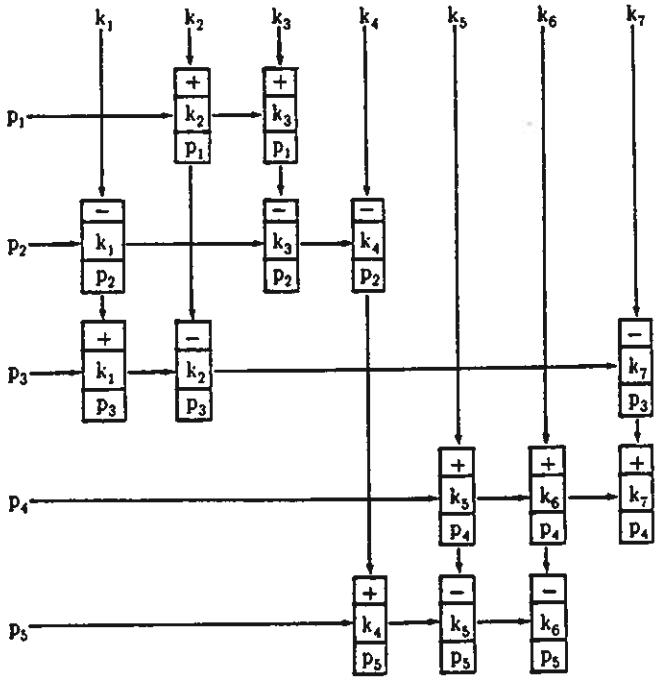
Finally a contraction of a $p_i \to p_j$ edge k_t : First we have to find the element belonging to k_t in a neighbourlist, and then we have to delete it as above. This requires as mentioned 1 or K steps, depending upon the position of k_t being known in advance or not. If the new vertex, which occurs by the contraction, is denoted p_i, we have to link p_j's lists to p_i's. This can be done in one step (if we work by doubly linked lists). Finally, all occurrences of p_j in the list s have to be changed to p_i. This requires up to K steps. Therefore contraction can be done in $O(K)$ steps in a neighbourlist representation.

In some conection it is convenient to use another type of neighbourlists where each $p \to q$ edge k is reprecented twice: Once (labelled $+$) in the neighbourlist of p and once a (labelled $-$) in the neighbourlist of q . For the graph in the figure above we obtain the following neighbourlists:



3. Doublelists This representation is obtained from the incidence matrix A , by changing rows and columns to lists. This is done in the following way: each vertical or horizontal sequence of zeroes between two elements a and b (both $\neq 0$) is exchanged with an arrow from a to b (when a comes before b in the row or column). Furthermore we exchange each 1 with the name of the vertex and the edge which the 1 represents. The two figures below will make it clear what is going on.



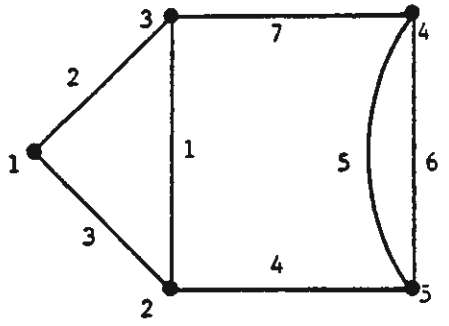


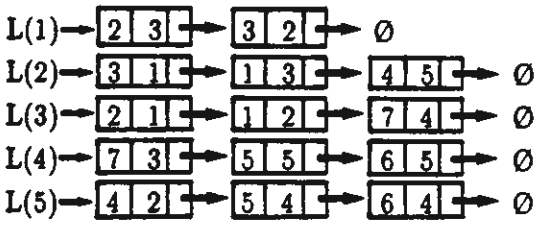
This representation is a combination of the two earlier described representations, and it unites the best of their properties. In this representation, the orders 1 and 2 (page 40) can be done in 1 or $v_u(p_i)$ steps respectively, the orders 3, 4 and 5 can each be carried out in one step. When we have to carry out contraction of a $p_i \rightarrow p_j$—edge k_t, the most demanding work (as in the neighbourlist) is to exchange the name p_j with p_i. This can be done in $v(p_j)$ steps in p_j's neighbourlist. When this is done, we link the list after p_i's neighbourlist, and the k_t's list is deleted. At the start of the next page we summarise all of the results we have obtained until now.

	ANTAL SKRIDT	REPRÆSENTATION		
		Kantlister	**Nabolister**	**Dobbeltliste**
ORDRE	Find en kant, der starter i p	K	1	1
	Find de kanter, der starter i p	K	$v_u(p)$	$v_u(p)$
	Find begyndelses-punktet af k	1	K	1
	Find slutpunk-tet af k	1	K	1
	Slet k	1	K(eller 1)	1
	Sammentræk p → q-kanten k	K	K	v(p)

Until now in this section, we have only dealt with directed graphs. Non–directed graphs can also – with small, natural changes – be represented by means of either edgelists, neighbourlists or doublelists. Note that there is one edgelist for each edge, but that the neighbourlists (as in the doublelists) contain two elements corresponding to a p,q–edge k : one in the neighbourlist of p and one in the neighbourlist of q. It can be useful to add arrows between these two elements, such that (say if you want to delete an edge) it is possible in one step to find one appearance of the edge by means of the other, just as we can in a doublelist.

As an example which shows how a non–directed graph can be represented, we show in the figure a non–directed graph Γ, together with its neighbourlists $L(p)$, $p \in \mathcal{P}(\Gamma)$.





11. THE COMPLEXITY OF AN ALGORITHM. An algorithm is, of course, a systematic way to obtain a certain "output" from a certain "input". Of course, by construction of an algorithm we try to solve a certain type of problem, but in the formulation of the algorithm, or in the name of the algorithm, there is no hidden statement that the algorithm does, in fact, solve the problem. Therefore, belonging to an algorithm, there should be a theorem which expresses the condition for the algorithm to stop and the conditions for the output to be a solution of the problem.

When we are going to carry out a theoretical assessment of the quality of an algorithm, and in particular when we are going to compare two algorithms which solve the same problem type, it is necessary to measure how "the number of steps" which are going to be executed when the algorithm is used once, grows with "the size" of the problem which is used as input.

We shall shortly explain how the two exdpressions in the quotation marks are defined.

The Size of a graph theoretical problem is measured often by the number P of vertices in the graph used as input. In other cases, it is the number K of edges which is the natural measure of the size of the graph. Also, expressions like $P \cdot K$ or $P + K$ can sometimes be useful to use as a measure of the size of a graph.

The definition of *a step* in an algorithm will obviously depend upon how detailed an investigation of the algorithm we are going to perform. In this book we shall, as earlier mentioned, consider an operation whose duration does not depend on the size of the problem we are going to solve, as one step in the execution of an algorithm.

The number of steps which are going to be executed when an algorithm is used once, depends not only on (the mathematical idea in) the algorithm. The number of steps also depends on the details in the program and upon the representation used. This, of course, is clearly seen in the figure at the top of page xxx. Very often it is possible to talk about the *complexity of an algorithm* (– and not only about the complexity of the corresponding program). This is the case in situations where the choice of the algo-

rithm (that is, the way of solving the problem), in a natural way, points out the choice of the data representation. In the following chapters, we will often get the opportunity to comment upon this topic.

When we have determined what we will call "a step" in a certain algorithm, A, which solves a certain type of theoretical problem, then in principal it is possible to calculate the maximum number of steps, denoted $f_A(n)$, which have to be performed when the algorithm is used to solve a problem for a graph of size n. Here the word maximum denotes maximum taken over all graphs of size n, and also over all possible ways the algorithm can behave for such graphs. But different types of "single steps" may not take the same time, we are not interested in the detailed behaviour of the function $f_A(n)$, even though it is the function in its details which determines how long a time it will take to use the algorithm in a concrete case. But what will interest us here is how $f_A(n)$ behaves for large values of n. In general $f_A(n) \to \infty$ as $n \to \infty$, but this can happen more or less violently. In general, $f_A(n)$ will be determined by a very complicated expression. Therefore in general we compare $f_A(n)$ with a simpler function $g(n)$, a function which it is easier to describe. If $\frac{f_A(n)}{g(n)} \to c$ as $n \to \infty$ we shall say that the algorithm is an $O(g(n))$–*algorithm* (c denotes a constant which is independent of n).

An algorithm A is called *polynomial* (or *good*) if there exists c and q, such that $\frac{f_A(n)}{cn^q} \to 1$ as $n \to \infty$. A is also called an $O(n^q)$–*algorithm* (O denotes order of size) and the algorithm is said to have *complexity* n^q. (Since we have only taken the use of time into consideration, not the storage, and since we are only interested in large values of n we should have talked about the *asymptotic time complexity.*) For example, we have seen on page xxx that an edge in a graph can be deleted in $O(K)$ steps (– we say in "*linear time*" –) when we use neighbourlists, while it can be done in $O(1)$ steps (– we say in "*constant time*" –) when the graph is represented by an edge-list.

In the definition of complexity, $f_A(n)$ is denoted "the maximal number of steps which in the worst case is going to be executed" when the algorithm A is used with a graph of size n as input. Here in practical applications, it would be more relevant if we exchanged the words in quotation marks with "the number of steps which, *on average*, is going to be performed." Such a change would have 2 disadvantages : First, it would in every concrete case be much more difficult to calculate the size of $f_A(n)$, and secondly we would miss a very central observation in the mathematical theory on complexity, which we shall describe on page xxx.

Algorithms in which the number of steps grow faster than any polynomium in n are often called *bad algorithms* (but this expression should not be considered condescending). Such algorithms will typically be exponential or factorial.

An algorithm A is called *exponential* if there exist constants c and a, such that $\frac{f_A(n)}{ca^n} \to 1$ as $n \to \infty$. In this case we talk about an O(exp n)–algorithm. An exponential algorithm can, for example, occur if we want to investigate all the 2^n subsets of a given set with n elements. A is called *factorial*, if there exists a natural number k and a constant c, such that $\frac{f_A(n)}{c(kn)!} \to 1$ as $n \to \infty$. A is called an O(n!)–algorithm. A factorial algorithm can, for instance, occur when we want to investigate all the n! permutations of elements in a set with n elements.

In most practical applications of graph theory, one has to deal with sparse or "thin" graphs, that is, graphs where only a small fraction of the vertex pairs are connected by an edge. More formally, a set of graphs is called *sparse* when it includes graphs with arbitrarily many vertices, and there exists a constant c (independent of Γ) such that $|\mathcal{K}(\Gamma)| \leq c|\mathcal{P}(\Gamma)|$ for all graphs Γ in the considered set. A set of graphs is called *dense* when it contains graphs with arbitrarily many vertices for which $|\mathcal{K}(\Gamma)| \geq c|\mathcal{P}(\Gamma)|^2$ for some constant c. The complete graphs are a dense set, graphs with maximum valency ≤ 7 are sparse. A more important set of sparse graphs are the simple plane graphs. We have shown that if Γ is a simple plane graph, then $|\mathcal{K}(\Gamma)| \leq 3|\mathcal{P}(\Gamma)| - 6$ (see page xxx).

As is well known, there is currently a very rapid development of the effectiveness of computers. Therefore, it could be thought that the necessity for using good algorithms is becoming less and less. In fact, the opposite is true. To illustrate this suppose we have to decide if a given graph Γ of size n has a certain property. We assume that we have two algorithms at our disposal. One demands 2^n steps to investigate a problem of size n, the other n^2 steps. If our computer can perform 100 steps per minute (s/m), then in one minute we can treat problems up to size $n = 7$ with our 2^n–algorithm, while with the n^2–algorithm we can treat problems up to size $n = 10$. The figure below shows the amount of profit w.r.t. problem size one will have with an increase in the calculation:

Number of steps per minute

	$100^{s}/m$	$10^{3\,s}/m$	$10^{4\,s}/m$	$10^{5\,s}/m$	$10^{6\,s}/m$
2^n-algoritme	n = 7	n = 10	n = 13	n = 17	n = 20
n^2-algoritme	n = 10	n = 32	n = 100	n = 316	n = 1000

Problem size which can be handled in one minute

One sees that the necessity that one has a good algorithm grows with the calculating speed! – Large problems can only be handled on computers with the aid of good algorithms.

In practice, the storage space required by an algorithm will play as large a role as the time used. But the problems concerning storage have not been investigated theoretically to the same extent as the problems concerning use of time. In this book, we shall therefore essentially only mention the use of time by different algorithms. In simple cases though, we shall give an upper bound for the storage needed by an algorithm.

12. "BREADTH FIRST SEARCH" In this, and the following section, we shall describe two algorithms on a graph Γ . They are called BREADTH FIRST SEARCH and DEPTH FIRST SEARCH , and search through Γ following two different principles. The most important application of BREADTH FIRST SEARCH is distance determination in a graph. DEPTH FIRST SEARCH can be used for several purposes. For example, finding bridges and strongly connected components.

"Breadth first search" can be applied on a graph Γ , both when Γ is directed and when Γ is undirected. The description of the algorithm is the same in both cases. Γ shall be represented by giving a neighbour–list L(p) for each $p \in \mathcal{P}(\Gamma)$ as described in section 10. Essentially, BREADTH FIRST SEARCH consists of repeated applications of a procedure called INVESTIGATE(p) , in which L(p) is scanned once. In the procedure, we use the following variables: A labelling function m: $\mathcal{P}(\Gamma) \to \mathbb{N}_0 \cup \varnothing$, $\mathbb{N}_0 = \{0, 1, 2, 3,\dots\}$. An edge set $\mathcal{T}$. A queue Q (– whose first element is p). (In a queue we write the first elements to the right, the last to the left.) The procedure INVESTIGATE(p) is defined as follows.

procedure INVESTIGATE(p)

1. Search through L(p) once. For each element (k,q) in L(p) ,
2. if m(q) = 0 ,
3. put m(q) = m(p)+1 , $\mathcal{T} = \mathcal{T} \cup \{ k \}$ and put q last in Q .
4. Remove the first element from Q .

The algorithm itself calls INVESTIGATE(p) once for each $p \in \mathcal{P}(\Gamma)$ starting with a specified vertex s , called the "source":

algorithm: BREADTH FIRST SEARCH

Input: A directed or undirected graph Γ represented by a set of neighbour–lists L(p) , $p \in \mathcal{P}(\Gamma)$, $s \in \mathcal{P}(\Gamma)$.

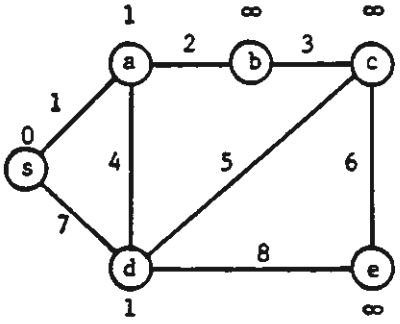
Output: An edge set $\mathcal{T}$. A labelling function $m : \mathcal{P}(\Gamma) \to \mathbb{N}_0 \cup \{ \infty \}$.

Variables: p is a vertex in Γ . Q is a queue of points .

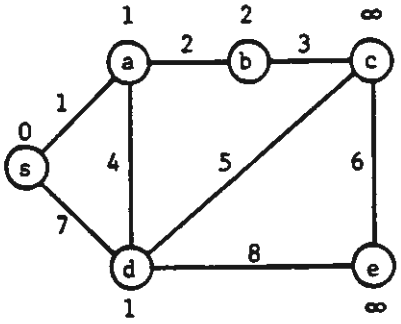
Initialising: $\mathcal{T} = \emptyset$, Q = (s) , m(s) = 0 , $m(p) = \infty$ for $p \in \mathcal{P}(\Gamma)$, $p \neq s$.

1. **While** Q is not empty, let p denote the first point in Q .
2. INVESTIGATE(p) .
3. STOP when Q is empty.

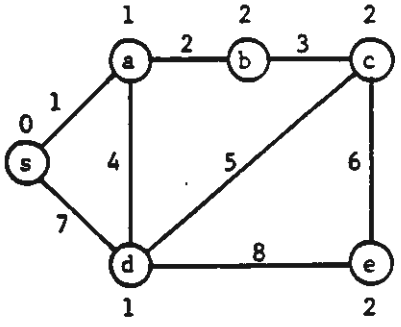
The figures on the next page show an example of how the algorithm can look perform when applied to an undirected graph. The numbers by the vertices are labellings. The situation is shown after the first, second and third call of INVESTIGATE. When the algorithm subsequently discovers that neither b , e or c have unlabelled neighbours then it stops with the result shown in the figure on the far right.



$\mathcal{T} = \{1,7\}$, Q = (d,a)



$\mathcal{T} = \{1,7,2\}$, Q = (b,d)



$\mathcal{T} = \{1,7,2,5,8\}$, Q = (e,c,b)

Theorem 12.1. When the graph Γ is undirected and connected, and BREADTH FIRST SEARCH has stopped, then $\mathcal{T}$ is the edge–set for a spanning tree T in Γ, and for $p \in \mathcal{P}(\Gamma)$, $m(p)$ is the length of a shortest s,p—path in Γ. The algorithm is an $O(K)$ algorithm.

Proof. It is easily seen that the algorithm stops, since each vertex appears at most once in Q, and every time INVESTIGATE is called an element is removed from Q. Therefore INVESTIGATE can be called at most P times.

What happens when the algorithm is used is the following:

1a. First s is investigated and every neighbour $q(\neq s)$ of s is labelled $m(q) = 1$ (that is $m(q) = d(s, q)$).

1b. When this is done, Q therefore contains all vertices q with $d(s, q) = 1$.

2a. Next, those vertices are investigated one at a time and every unlabelled neighbour q is labelled $m(q) = 2$ (that is $m(q) = d(p_0, q)$).

2b. When this is done Q contains all vertices q with $d(s,q) = 2$.

3a. Next, those vertices are investigated ...

In this way, one can continue (by induction) at most P times and therefore $m(p) = d(s, q)$ for all $p \in \mathcal{P}(\Gamma)$.

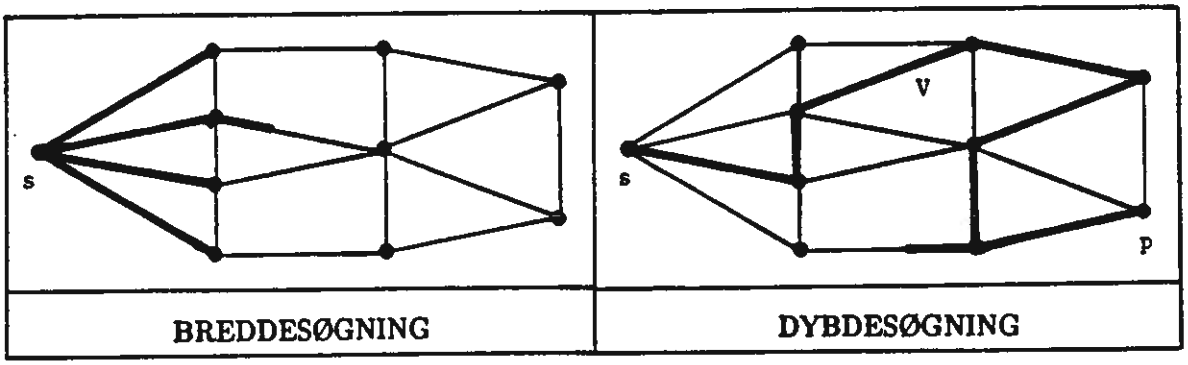
We shall now show that $\mathcal{T}$ is an edge–set for a spanning tree in Γ. We now consider at an arbitrary time (whilst the algorithm is running) the graph T, whose vertex–set is the labelled vertices and whose edge–set is $\mathcal{T}$ at that time. T is initialised as a tree and every extension of T is carried out in such a way that k is an edge which connects a vertex of T with a vertex q outside T. Thus a new tree is formed, and when all the vertices in Γ are labelled, one stops with a spanning tree in Γ. T is called a breadth first search tree.

And finally, the complexity. First we must describe how the labels and $\mathcal{T}$ are represented. We assume that the vertices are called $s = p_1, p_2, \ldots p_P$, and the edges are called $k_1, k_2, \ldots k_K$. For the labelllings, we use a vector $m = (m_1, m_2, \ldots m_P)$, where $m_i = m(p_i)$. A labelling can therefore be read and changed in one step. To represent $\mathcal{T}$ we use a vector $t = (t_1, t_2, \ldots t_K)$, where t_i is 0 if $k_i \in \mathcal{T}$, or 1 if not. Every updating of $\mathcal{T}$ then can be done in one step. Now the basis for deciding the complexity of the algorithm is that what happens when the algorithm deals with Γ is that each

vertex's neighbour–list is searched through. The work involved in dealing with one element of a neighbour–list can be done in constant time, and since there are $2K$ neighbour–list elements, this gives $O(K)$ steps. To this is added the work done in taking points out of the queue. Since this gives $P \leq K + 1$ steps, the complete complexity of the algorithm is $O(K)$. With repeated use of the algorithm in an unconnected, undirected graph Γ, *one can find the connected components* of Γ in $O(K)$ steps, that is *linear time.*

If Γ is a directed graph, one can prove a theorem which differs from theorem 12.1 in certain details: A vertex q becomes labelled if and only if there exists an $s \to q$–path in Γ. $\mathcal{T}$ becomes the edge–set for a tree T, whose vertex set is the labelled vertices. From an arbitrary point in T, one can find a path to s by going against the arrows. The complexity will remain $O(K)$.

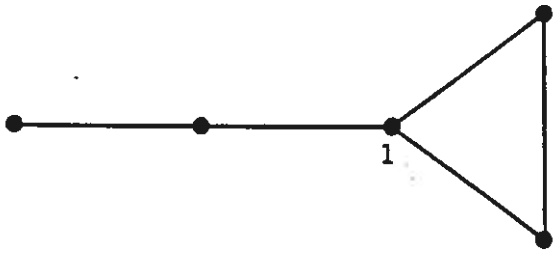
13. DEPTH FIRST SEARCH. In this section we shall describe a new method to search through a connected graph. Like breadth first search, depth first search starts in a vertex s. But the principle of the searching is new: When you carry out breadth first search, you consider first all edges starting in s. When you perform depth first search, you start in s, by using a path V until you reach a vertex p, from which you cannot proceed without going to a vertex already visited. The words "breadth first search" and "depth first search" describe very well what is going on.



Depth first search can be executed on a graph Γ, both when Γ is not–directed and when Γ is directed. The description of the algorithm is the same in the two cases. We assume that Γ is represented by, for each $p \in \mathcal{P}(\Gamma)$, giving a neighbourlist $L(p)$, which is defined in section 10. The essential contents of depth first search is application of a recursive procedure called SEARCH(p), in which $L(p)$ is searched once, but where a lot of things happen before you are finished. The procedure calls namely itself and this repeats until the condition for the execution of the recursive call is not satisfied. In this way the procedure will be finished from the "inside". In

2.7. Describe a variation of CUT VERTICES such that we obtain a new algorithm called BRIDGES which determines the bridges in Γ .

2.8.

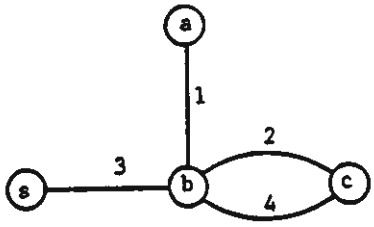


Describe, referring to page 2.28, LOWPOINT(1) in the graph in the figure. Choose the labelling yourself.

2.9. DEPTH FIRST SEARCH is used on a directed graph Γ . How can the algorithm be modified such that it decides whether there exists a one way circuit in Γ .

2.10. Let T be a DEPTH FIRST SEARCH tree in a graph Γ . Can 2 vertices, with valency 1, in T , be connected by an edge in Γ ?

2.11. Describe exactly as on page 2.16 how DEPTH FIRST SEARCH can be carried out in the graph in the figure.



2.12. Sort the coordinates in the vector

$$v = (5,4,2,3,7,1,6),$$

first by BUBBLESORT and then by MERGESORT. Using MERGESORT

the procedure we use the following variables: A labelling function $m: \mathcal{P}(\Gamma) \to \{old, new\}$. A counter function $N : \mathcal{P}(\Gamma) \to \mathbb{N} = \{1, 2, 3, \dots\}$, which gives each $p \in \mathcal{P}(\Gamma)$ a number $N(p) = n$. A edge set $\mathcal{T}$. A function $f : \mathcal{P}(\Gamma) - \{s\} \to \mathcal{P}(\Gamma)$ (where $f(p)$ is called the father of p). The procedure SEARCH(p) can now be defined as follows:

procedure: SEARCH(p).

1. Put $m(p) = old$, $N(p) = n$ and then $n := n + 1$.
2. Search through $L(p)$ once. For each element (k,q) in $L(p)$,
3. **if** $m(q) = new$
4. put $\mathcal{T} := \mathcal{T} \cup \{k\}$, put $f(q) = p$ and SEARCH(q) .
5. STOP when $L(p)$ is searched.

In the algorithm DEPTH FIRST SEARCH, there is specified a vertex s , *the source*, where the search starts. The algorithm can be described as follows:

algorithm: DEPTH FIRST SEARCH.

Input:	A directed or non–directed graph Γ represented by a neighbourlist $L(p)$ for each $p \in \mathcal{P}(\Gamma)$. A vertex $s \in \mathcal{P}(\Gamma)$.
Output:	An edge set $\mathcal{T}$. A function $f : \mathcal{P}(\Gamma) - \{s\} \to \mathcal{P}(\Gamma)$. A function $N : \mathcal{P}(\Gamma) \to \mathbb{N}$.
Variables:	A labelling function $m : \mathcal{P}(\Gamma) \to \{new, old\}$. A number $n \in \mathbb{N}$.
Initialising:	$\mathcal{T} = \emptyset$, $m(p) = new$ for all $p \in \mathcal{P}(\Gamma)$. $n = 1$.

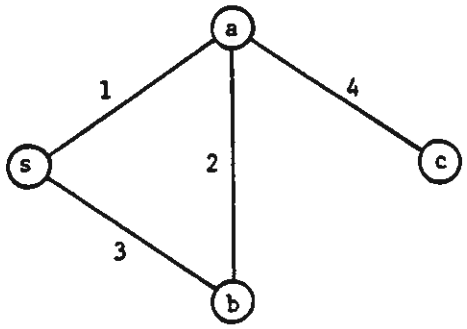
1. SEARCH(s).

Maybe it is easier to understand what is going on if we explain it in words:

Start in s and ↳walk along a path through new vertices until you arrive to a vertex p from where you cannot go on to a new vertex. If it is s you stop and the algorithm stops. Otherwise you go one step back along the path you first arrived at p along and then ——

The connection between the two descriptions of the algorithm requires some careful consideration: When you stand in the end vertex p of the first path V , you have started but not yet ended SEARCH(q) for all vertices q on V . You are now working on the execution of SEARCH(p) and discover that the condition in line 3 in the procedure is not satisied for any element in the neighbourlist. Therefore SEARCH(p) stops without any new call of the procedure and the algorithm proceeds to continue the work on SEARCH(q) for $q = f(p)$.

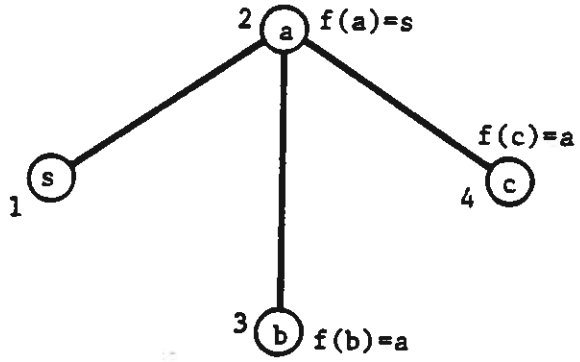
Below we give an example which shows how depth first search can proceed.



$L(s) \rightarrow (1, a) \rightarrow (3, b) \rightarrow \emptyset$
$L(a) \rightarrow (1, s) \rightarrow (2, b) \rightarrow (4, c) \rightarrow \emptyset$
$L(b) \rightarrow (3, s) \rightarrow (2, a) \rightarrow \emptyset$
$L(c) \rightarrow (4, a) \rightarrow \emptyset$

SEARCH(s).
m(s) = old. N(s) = 1 . n = 2 .
Start searching through L(s) .
Consider the element (1, a) : $\mathcal{T} = \{1\}$, f(a) = s .
SEARCH(a) .
m(a) = old. N(a) = 2 . n = 3 .
Start searching through L(a) .
Consider the element (1, s).
Consider the element (2, b) : $\mathcal{T} = \{1, 2\}$, f(b) = a .
SEARCH(b) .
m(b) = old. N(b) = 3 . n = 4 .
Start searching through L(b) .
Consider the element (3, s) .
Consider the element (2, a) .
L(b) search finished.

STOP SEARCH(b) .
Consider element (4, c) : $\mathcal{T} = \{1, 2, 4\}$, f(c) = a .
SEARCH(c) .
m(c) = old. N(c) = 4 . n = 5 .
Start searching through L(c) .
Consider element (4, a) .
L(c) search finished.
STOP SEARCH(c) .
L(a) search finished.
STOP SEARCH(a) .
Consider element (3, b) .
L(s) search finished.
STOP SEARCH(s) .

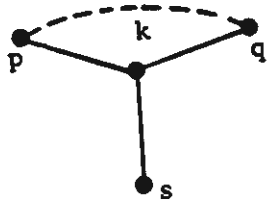


In the figure, we show the result consisting of the set $\mathcal{T}$ containing the 3 edges, and the functions f and N . The spanning tree T is called an DEPTH FIRST SEARCH tree. When p = f(q) , we call p a *father* to q and q a *son* of p . When p = f(f(···f(q))···) , we call p an *ancestor* of q and q a *descendent* of p . p is not an ancestor to himself but it is practical to consider p among its own descendants. When p is an ancestor of q then N(p) < N(q) ; the converse is not necessarily true. The number n = N(p) is used as a name of p (– instead of p itself).

Theorem 13.1. When Γ is a connected not–directed graph then DEPTH FIRST SEARCH is an O(K)–algorithm. When the algorithm has stopped, t is the edge set of a spanning tree T in Γ and for each $p \in \mathcal{P}(\Gamma)$, the vertices on the paths in T from p to s are as follows: p, f(p), f(f(p)), ··· , s , and N(p) > N(f(p) > N(f(f(p))) > ··· > N(s) = 1 .

Proof. For $p \in \mathcal{P}(\Gamma)$ the procedure SEARCH(p) will be called at most once. Because when SEARCH(p) is called, m(p) is changed from new to old. Therefore the algorithm stops. The work by carrying out SEARCH(p) , not considering the recursive calls, consists of changing m(p) from new to old, defining n(p) , updating n and furthermore, in running through L(P) once. For each neighbourlist element, the work can be carried out in one step, assuming that we describe the functions f , m and n using 3 P–dimensional vectors. Since there are 2K neighbourlist elements there will be at most $P + 2K$ steps, since $P \leq K + 1$, the complexity of the algorithm, is therefore $O(K)$. To show the statement on T we first remark that for every $p \in P(\Gamma)$, SEARCH(p) is called. This is seen as follows: If there are vertices which are not visited, there will, since Γ is connected, exist a p,q–edge k , such that SEARCH(p) is called and SEARCH(q) is not called. During the execution of SEARCH(p) , you reach the neighbourlist element (k, q) , and since m(q) is new, the SEARCH(q) will be called. This contradiction shows that all vertices will be searched. By updating $\mathcal{T}$ in line 4, we form, in each step, an edge set for a tree, and since all vertices will be searched, we end up with a spanning tree. When the p, q—edge k in line 4 is added to the tree, $f(q) = p$ is the first vertex belonging to the path in T from q to s and $N(q) > N(p)$. □

Lemma 13.2. When DEPTH FIRST SEARCH is carried out on an undirected graph and when k is a p, q–edge , then either p is a descendent of, or an ancestor to q .

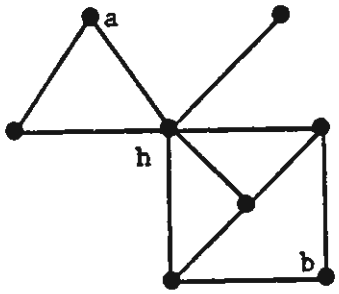


In the figure, the full lines constitute a DEPTH FIRST SEARCH tree. Then the lemma states that an edge such as k cannot exist in Γ since p is neither a descendent nor an ancestor to q .

Proof of the lemma. If $k \in \mathcal{T}$, then either $p = f(q)$ or $q = f(p)$. Suppose that $k \notin \mathcal{T}$. We may assume that SEARCH(p) was called before SEARCH(q) . This means that m(q) is new at the time when SEARCH(p) was started. During the execution of SEARCH(p) , (recursive calls included) all new vertices we meet will be made descendents of p . But one of the elements in L(p) is (k, q) , so q belongs to the new vertices we reached during the execution of SEARCH(p) . □

When DEPTH FIRST SEARCH is executed, we often assume that as soon as N(p) is calculated we use, instead of p , the number n = N(p) as the name for p .

14. BLOCKS. In the same way as a non–directed graph can be split up into connected components, a connected graph can be split up into cut vertex–free parts called *blocks*. In this section we shall give a description of this spliting and after that we shall show that the cut vertices and the blocks in a graph can be determined in linear time using DEPTH FIRST SEARCH . The description in this section is taken from references [1] and [3].



A vertex h in a connected non–directed graph Γ is called a cut vertex in Γ if $\Gamma - h$ is not connected. If Γ itself is not connected, we call h a cut vertex in Γ when h is a cut vertex in one of the connected components of Γ . This definition can be transformed to the following 2 statements:

1°. If h is a cut vertex in Γ , then $\Gamma - h$ contains 2 vertices a and b , such that every a, b–path in Γ contains h .

2°. If, conversely, there exist 2 vertices a and b such that every a, b–path in Γ contains h different from a and b , then h is a cut vertex in Γ .

The graph Γ is called a block when Γ is connected and when Γ does not contain any cut vertex. We shall now give a description of blocks.

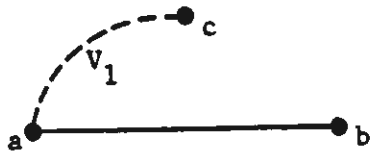
It is convenient, in this section, to call 2 a, b–paths, V_1 and V_2 , in a graph Γ , *disjoint*, if a and b are the only vertices which belong to both V_1 and V_2 . We first show

Theorem 14.1. A graph with at least 2 edges is a block if and only if 2 arbitrary vertices in Γ are connected with at least 2 disjoint paths in Γ .

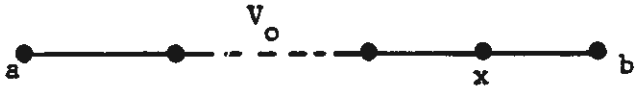
Remark: This theorem is a special case of Mengers theorem (page 1.25).

Proof. "if". It is given that 2 arbitrary vertices in Γ are connected by at least 2 disjoint paths. We are going to show that Γ does not contain a cut vertex, but this is clear. For if Γ had a cut vertex h then (according to 1°) Γ would contain 2 vertices a and b such that all a, b–edges contain k .

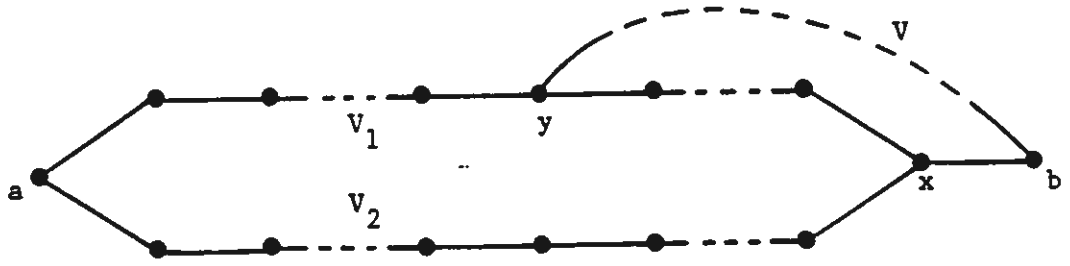
"only if". It is given that Γ is a block containing at least 2 edges. Therefore Γ contains no cut vertices. We shall show that 2 arbitrary vertices a and b are connected by at least 2 disjoint paths. The proof is by induction after the distance $n = d(a, b)$.



For $n = 1$, there is an a, b–edge k . If there are 2 a, b–edges , there is nothing to prove. Otherwise Γ contains a third vertex c . Then $\Gamma - k$ must contain both, an a, c–path V_1 and a c, b–path V_2 , for otherwise a or b would be a cut vertex. The paths V_1 and V_2 contain together an a, b–path in $\Gamma - k$ and therefore there are 2 disjoint a, b–paths in Γ . Assume now that when $d(p, q) < n$ then there exists 2 disjoint p, q–paths in Γ , and consider 2 vertices a and b with $d(a, b) = n \geq 2$, and we shall show that there exists 2 disjoint a, b–paths in Γ .



Let V_0 be an a, b–path of length n and let x be the neighbour to b on V . Then $d(a, x) = n - 1$ and there exists 2 disjoint a, x–paths V_1 and V_2 .



$\Gamma - x$ is connected and contains therefore an a, b–path V . When we walk through V from a towards b , there must be a last vertex y belonging to either V_1 or V_2 , assume V_1 . It may happen that $y = a$. Then the a, y–path of V_1 , together with the y, b–path of V is one a, b–path and V_2 is together with the x, b–edge , another a, b–path and these 2 paths are disjoint.

From theorem 14.1 we have immediately

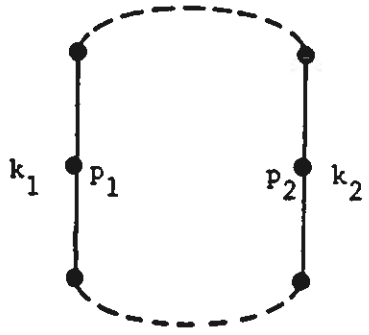
Corollary 14.2. A graph Γ containing at least 2 edges is a block if and only if 2 arbitrary vertices belong to the same circuit in Γ.

Using the corollary it is not difficult to prove

Theorem 14.3. A graph Γ without isolated vertices is a block if and only if two arbitrary different edges belong to the same circuit in Γ.

Proof. "If" part: It is given that 2 arbitrary different edges belong to the same circuit in Γ. If Γ has only 2 vertices it is a block, and if Γ has more than 2 vertices we shall show that 2 arbitrary vertices p and q in Γ belong to the same circuit. We can choose 2 different edges incident to p and q respectively. A circuit containing these edges will contain p aswell as q and according to the corollary then Γ is a block.

"only if" part: It is given that Γ is a block and we have to show that 2 arbitrary different edges k_1 and k_2 belong to the same circuit. We introduce 2 new vertices p_1 and p_2 with valency 2 on, respectively, k_1 and k_2. Consequently, we cannot create a new cut vertex and therefore, according to the corollary, the circuit in the new graph Γ, corresponding to C, contains both k_1 and k_2. □



Having characterised blocks in different ways, we can now describe how a graph can be uniquely split up into blocks.

Let Γ be a given graph. Define an equvalence relation $\sim$ on $\mathscr{K}(\Gamma)$ in the following way:

When $\mathscr{K}_1$ and $\mathscr{K}_2$ are edges in Γ then $\mathscr{K}_1 \sim \mathscr{K}_2$ means that either there exists a circuit in Γ containing both $\mathscr{K}_1$ and $\mathscr{K}_2$ or $\mathscr{K}_1$ is equal to $\mathscr{K}_2$.

It is easy to see that $\sim$ is an equivalence relation.

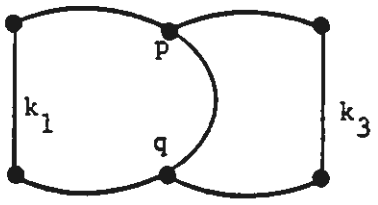
1) $k \sim k$ for all $k \in \mathscr{K}(\Gamma)$. ($\sim$ is reflexive.)

2) when $k_1 \sim k_2$, $k_2 \sim k_1$. ($\sim$ is symmetric.)

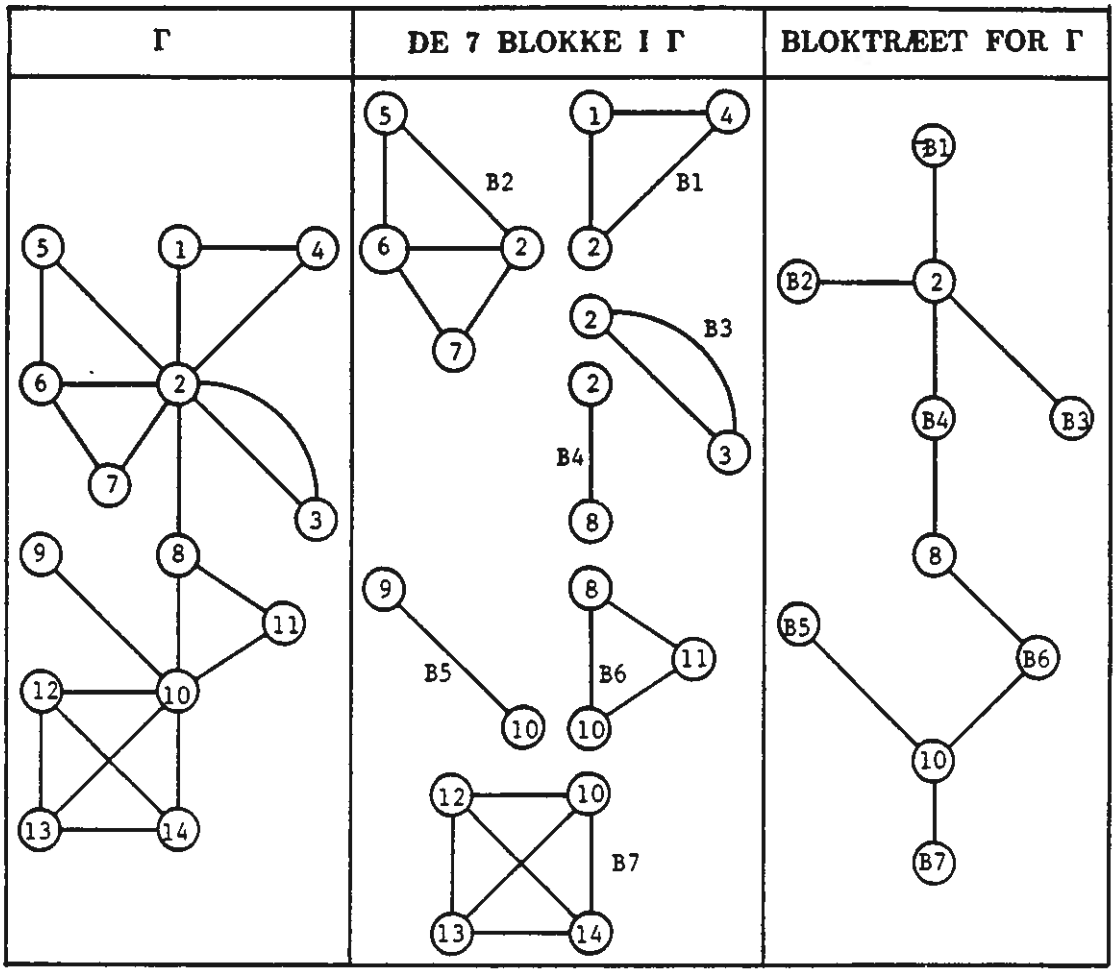
3) When $k_1 \sim k_2$ and $k_2 \sim k_3$ then $k_1 \sim k_3$. ($\sim$ is transitive.)

The first 2 conditions are obvious, the 3rd can be proved as follows:

Let K_1 be a circuit containing both k_1 and k_2. If k_3 belongs to K_1 we are finished, otherwise we consider a circuit K_2 contains k_2 aswell as k_3. If we walk on K_2 from each of the 2 end vertices of K_3, away from k_3, until we appear in a vertex of K_1 for the first time, then we have reached 2 different vetices p and q on K_1. If after that we remove that p, q–path on K_1, which do not contain k_1, we have a circuit containing both k_1 and k_3.



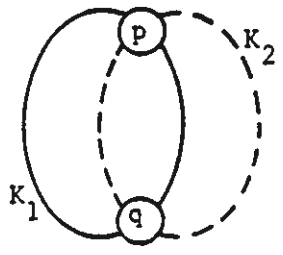
Since $\sim$ is therefore an equivalence relation we can partition $\mathscr{K}(\Gamma)$ into disjoint subsets such that 2 edges k_1 and k_2 belong to the same subset if and only if k_1 is equivalent to k_2. In particular, every bridge in Γ will constitute a subset in itself. A subgraph B in Γ, the edge set of which is one of these subsets and whose vertex set is the end vertices of these edges, is called a block in Γ. According to theorem 14.3, B is a block. The figure below shows a graph Γ and the blocks in Γ. We return later to the block tree.



According to the definition of a block in Γ, 2 blocks have no edge in common. In the figure, we have at most one vertex in common and such a common vertex is a cut vertex. This is always the case.

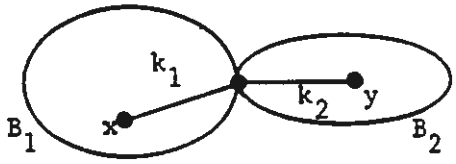
Theorem 14.4. 2 blocks in a graph Γ have at most one vertex in common. When h is a common vertex for 2 blocks in a graph Γ, then h is a cut vertex in Γ. Every cut vertex in Γ is a common vertex to 2 or more blocks in Γ.

Proof. First let us assume that 2 blocks B_1 and B_2 in Γ have 2 vertices p and q in common. First we consider the case where B_1 and B_2 both have at least 2 edges.

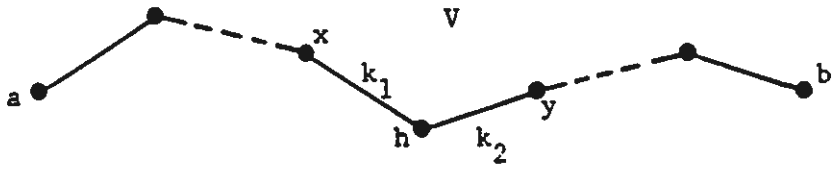


According to corollary 14.2, there exists a circuit K_1 in B_1 containing both p and q , and a circuit K_2 in B_2 containing both p and q . But then a p, q–path in K_1 together with a p, q path in K_2 will constitute a circuit in Γ which contains an edge from B_1 aswell as an edge from B_2 , contradicting the definition of a block. A corresponding but simpler argument holds if B_1 and/or B_2 only consists of 1 edge.

Next, let h be a common vertex for two blocks B_1 and B_2 in Γ . Then there exists an edge k_1 in B_1 and an edge k_2 in B_2 such that k_1 and k_2 are both incident to h . Let k_1 be an x, h–edge, and k_2 a y, h–edge. If there was an x, y–path in $\Gamma - h$, then there would be a circuit in Γ containing both k_1 and k_2 . This contradiction shows (cf. 2° p. 2.19) that h is a cut–vertex.



And finally: Let h be a cut–vertex in Γ , and let a and b (cf. 2° p. 2.19) be two vertices such that every a, b–path V in Γ contains h . Let the two edges of V which are incident to h be the x, h–edge k_1 and the h, y–edge k_2 .



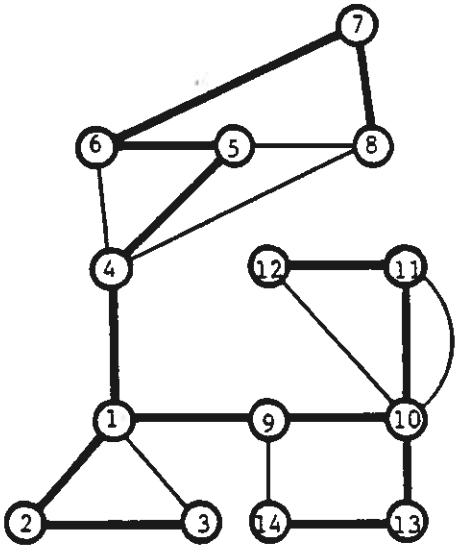
If k_1 and k_2 were contained in a circuit in Γ , there would be an x, y–path – and therefore an a, b–path – in Γ–h . This contradiction shows that k_1 and k_2 belong to separate blocks in Γ .

The blocks in a graph are linked together by the cut–vertices of the graph. By this observation, we obtain a tree–like structure, which we shall now describe in detail.

Corresponding to a given graph Γ , we define a new graph $B(\Gamma)$, which is called the *block–graph* of Γ . The vertex set of $B(\Gamma)$ consists of the cut–vertices in Γ and one vertex (called a *block–vertex*) corresponding to each block in Γ . The edges in $B(\Gamma)$ are obtained in the following way: When B is a block in Γ , then the corresponding block–vertex is connected by means of one edge to each of those cut–vertices in Γ which belongs to B . Through this construction, we obtain a bipartite graph with the

cut–vertices in Γ as one colour class and the block–vertices as the other one. One example is shown in the figure on page 2.23. It is easy to see that $B(\Gamma)$ is circuit–free, for if $B(\Gamma)$ contained a circuit, then Γ would contain a circuit with edges belonging to different blocks in Γ. If Γ is connected, therefore $B(\Gamma)$ is a tree. It is called the *block–tree* of Γ. According to the remark, p. 1.14, at the bottom of the page, $B(\Gamma)$ has at least two vertices with valency 1, when Γ has more than one block. Γ therefore has at least two blocks which each contain only one cut–vertex.

We shall now describe how, by using depth–first–search, we can determine the blocks in a given connected graph in linear time. We consider a connected, not–directed graph Γ, and carry out DEPTH–FIRST–SEARCH on Γ with start in a vertex s. Hereby, we construct a depth–first–search tree T, and the vertices are labelled and named with the function N in the order they are visited.



Then we have

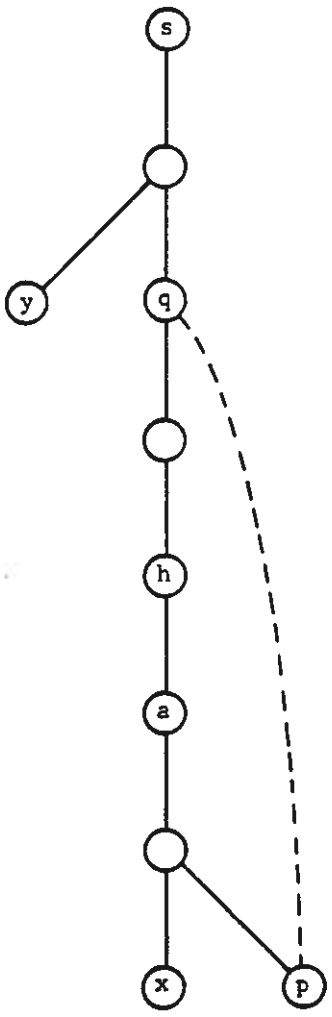
Theorem 14.5. A vertex h is a cut–vertex in Γ if and only if
either h = s and h has at least two sons a and b,
or h ≠ s and h has a son a with the property that there exists no cord from a descendant of a to an ancestor of h.

Remember that a itself is among its own descendants. It is instructive to find the different situations mentioned in the theorem in the figure above. For instance, 10 has sons 11 and 13. There exists a cord from a descendant of 13 to an ancestor of 10, but there exists no cord from a descendant of 11 to an ancestor of 10. Therefore 10 is a cut–vertex.

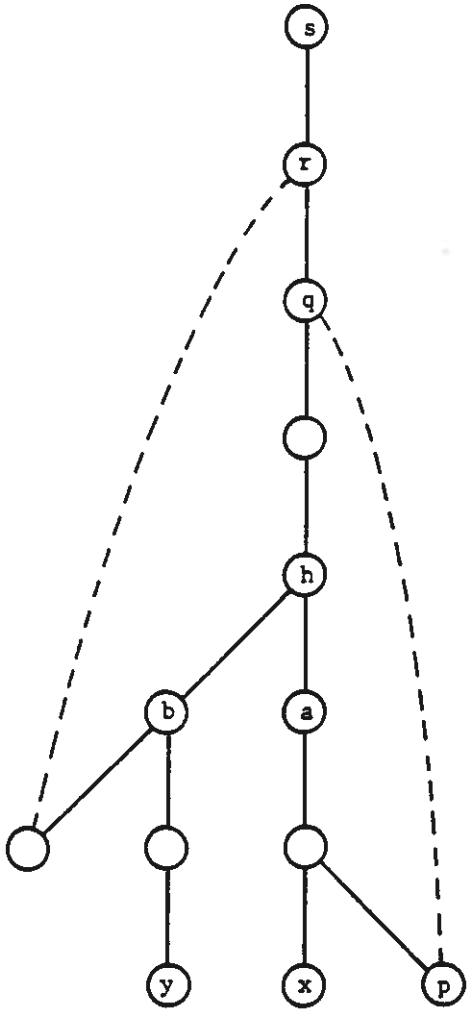
Proof. "if" part: The two cases are proved at the same time. Let Δ denote the subgraph in Γ whose vertex set is a and the descendants of a and whose edge–set is those edges of Γ which connect two of these vertices. According to Lemma 13.2, in $\Gamma - h$ there are no edges from a vertex in Δ to a vertex outside Δ (including b or s). Therefore Γ–h is not connected.

"only if" part: Let h be a cut–vertex in Γ . If h = s , DEPTH–FIRST–SEARCH searches through one connected component in Γ before it starts on the next component. Therefore s must have as many sons as there are connected components in $\Gamma - s$. Next, let $h \neq s$, and let x and y be two vertices in different connected components in $\Gamma - h$. At least one of the vertices x and y must then be a descendant of h , because otherwise neither the s, x–path in T nor the x, y–path in T would contain h , and x and y would then be in the same connected component in $\Gamma - h$. We may assume that x is a descendant of h . Let a be the son of h of which x is a descendant. We now carry out the proof indirectly, and assume that for every son c of h , there is a cord from a descendant p of c to an ancestor q of h .

CASE 1



CASE 2



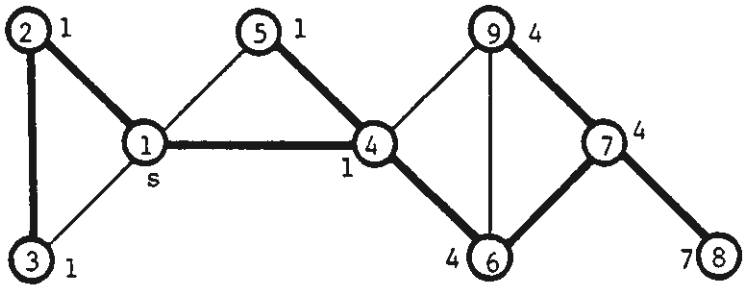
Case 1. y is not a descendant of h . In this case, you can go from x via p and q , and reach y by a path which does not contain h , which is a contradiction.

Case 2. y is a descendant of h . y cannot be a descendant of a . Let b be the son of h which y is a descendant of, $b \neq a$. For $c = b$, the assumption implies that there exists a cord from a descendant of b to an ancestor r of h . Now it is possible to reach y from x via q, r and p without passing through h , which again is a contradiction. □

When we want to find out by means of depth–first–search whether a vertex is a cut–vertex or not, we can control whether the conditions of theorem 14.5 are satisfied. With this purpose, we introduce – when DEPTH–FIRST–SEARCH has been executed – a function LM on $\mathcal{P}(\Gamma)$. LM(a) is called the *low–point* of a (Danish: Lavmål), and it is defined by LM(s) = 1 and for $p \neq s$

(1) LM(p) = MIN{q| there exists an edge in Γ from a descendant of p to q} .

Remember that the names of the vertices are those numbers which the vertices get in DEPTH–FIRST–SEARCH.



On the figure, the numbers in the circles are these numbers, while the numbers at the different vertices are the low–points of the vertices. Every son of s will always have the low–point 1. Every son a of p will have a low–point

$$LM(a) \leq p .$$

Theorem 14.5 can now be reformulated to a criterion containing the low–point function:

Corollary 14.6. A vertex $h \neq s$ is a cut–vertex if and only if h has a son a with LM(a) = h .

Proof. If h is a cut–vertex, h has a son a such that the smallest vertex which a descendant of a is connected to is $\geq$ h . Therefore LM(a) $\geq$ h , but since a is connected to h , LM(a) = h . If, conversely, h has a son a with LM(a) = h , no descendant of a can be connected to an ancestor of h . According to theorem 14.5 , h is therefore a cut–vertex. □

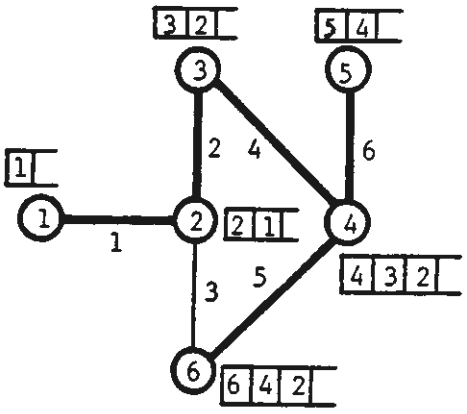
We can now turn to describe how procedure SEARCH(p) can be extended, such that it also calculates LM(p) . First remark that for all $p \in \mathcal{P}(\Gamma)$ the lowpoint of p can be determined by the formula

(1) LM(p) = MIN({LM(a)|a is a son of p} ∪ {q| there exists a p,q–edge in Γ}).

The formula gives correct results when p = s , and if $p \neq s$ the edge mentioned in the definition which goes to q either starts in a descendant of a son of p or from p itself. The formula can be used in the following way: First we calculate LM(x) , for a vertex $x \neq s$, which has valency 1 in the depth first search tree T . After this, we can calculate LM(y) for a vertex y , which has valency 1 in $T - x$, and so on. A computation of this kind can be built in to SEARCH(p) , such that we obtain a procedure for calculation for LM(p) :

```
procedure: LOWPOINT (p).
1.  Put m(p) = old, p:= n and LM(p) = p and n := n + 1 .
2.  Search through L(p) once. For each element (k, q) in L(p) ,
3.     if m(q) = new, put 𝒯:= 𝒯∪ {k} , and f(q) = p ,
4.        LAVMÅL (p)
5.        if LM(q) = p and p ≠ s , p is cut vertex.
6.        LM(p) := MIN{LM(p), LM(q)} .
7.     otherwise, that is, if m(q) = old ,
8.        LM(p) := MIN{LM(p), q} .
9.  STOP when L(p) is completely searched.
```

We shall illustrate how LOWPOINT(1) is carried out on the graph in the figure overleaf. On the figure we have only given those vertex numbers which are determined in line 1. In the boxes we first give the initial values of the lowpoint function and after that those values the function is changed to when the procedure is executed. We do not show the updating of $\mathcal{T}$, n and f in the description.



L(1) : (1, 2) .
L(2) : (1, 1), (2, 3), (3, 6) .
L(3) : (4, 4), (2, 2) .
L(4) : (6, 5), (4, 3), (5, 6) .
L(5) : (6, 4) .
L(6) : (5, 4), (3, 2) .

LOWPOINT(1). m(1) = old. Search L(1).
Consider (1, 2).
LOWPOINT(2). m(2) = old. Search L(2).
Consider (1, 1) : LM(2) = MIN(2, 1) = 1 .
Consider (2, 3) :
LOWPOINT(3). m(3) = old. Search L(3).
Consider (4, 4) :
LOWPOINT(4). m(4) = old. Search L(4).
Consider (6,5) :
LOWPOINT(5). m(5) = old. Search L(5).
Consider (6, 4) : L(5) = MIN(5, 4) = 4.
Then LM(5) = 4 is 4 cut vertex.
LM(4) = MIN(4, 4) = 4 .
Consider (4, 3) : LM(4) = MIN(4, 3) = 3.
Consider (5, 6).
LOWPOINT(6). m(6) = old. Search L(6).
Consider (5, 4) : LM(6) = MIN(6, 4) = 4.
Consider (3, 2) : LM(6) = MIN(4, 2) = 2.
LM(4) = MIN(3, 2) = 2.
Consider (2, 2) : LM(3) = MIN(3, 2) = 2.
LM(3) = MIN(2, 2) = 2.
Then LM(3) = 2 is 2 cut vertex.
LM(2) = MIN(1, 2) = 1.
Consider (3, 6) = MIN(1, 2) = 1.
LM(1) = MIN(1, 1) = 1.

algorithm: CUTVERTICES.

Input:	A connected, non–directed graph Γ given by a set of neighbourlists $L(p)$, $p \in \mathcal{P}(\Gamma)$. A vertex $s = 1$ in Γ .
Output:	The cutvertices in Γ . The lowpoint function on Γ .
Variables:	m , $\mathcal{T}$, f , N , n as mentioned previously.
Initialising:	$m(p) =$ new and $f(p) = \phi$ for all p . $\mathcal{T} = \phi$. $n = 1$.

1. LOWPOINT(1)

2. **if** 1 has at least 2 sons, 1 is a cutvertex.

Theorem 14.7. Algorithm CUTVERTICES finds the lowpoint function and the cutvertices in Γ correctly. (Observe that a cutvertex can be found more than once.) The complexity is O(K).

Proof. LM(p) is determined by means of the formula (1), p. 2.27 in line 6 and line 8 of the procedure. Therefore, *if* LM(q) is correct for all values of q , for which LOWPOINT(p) directly calls LOWPOINT(q) , then LM(p) is also correct. This *if* is the general step in proof by induction. The start of this proof is the case where LOWPOINT(p) does not call LOWPOINT itself. In this case, LM(p) is determined in line 8 only, corresponding to the last term in formula 1. This is correct since p has no sons. Remark that when lines 5 and 8 are carried out, then the names p and q are updated in line 1. Since the lowpoint function therefore is determined correctly, the cutvertices will also be determined correctly in line 5, since the calculation of LM(q) is finished and the name p is updated. If a cutvertex has more than one son satisfying the condition of corollary 14.6, p will be determined several times in line 5 of the procedure. It is clear that the complexity is the same as for DEPTH FIRST SEARCH, i.e. O(K) .

When we have determined the set $\mathcal{H}$ of cutvertices in Γ , in principle it is not difficult to find the blocks of Γ in linear time in the number of edges: First you can find the connected components in $\Gamma - \mathcal{H}$, and after that, for each of these connected components, you can add those edges which connect a connected component to a cutvertex. Finally, you distribute those edges which are incident to two cutvertices. This

process however can be carried out much more elegantly (but with the same complexity) by changing LOWPOINT(p) to

procedure: LOWPOINTSTACK (p) .

1. Put m(p) = old, p := n , LM(p) = p and n : n + 1 .
2. Search through L(p) . For each element (k, q) in L(p) ,
3. **if** k is not in STACK, put it there.
4. **if** m(q) = new, put $\mathcal{S}$:= $\mathcal{S}$∪ {k} and f(q) = p ,
5. LOWPOINTSTACK (q) .
6. **if** LM(q) = p ,
7. remove elements from STACK until k is removed. The removed edges is the edge set of a block.
8. LM(p) := MIN{LM(p), LM(q)} .
9. **otherwise,** that is if m(q) = old
10. LM(p) := MIN{LM(p), q} .
11. STOP when L(p) is searched.

By means of the procedure, the blocks in Γ can be found by use of the following algorithm:

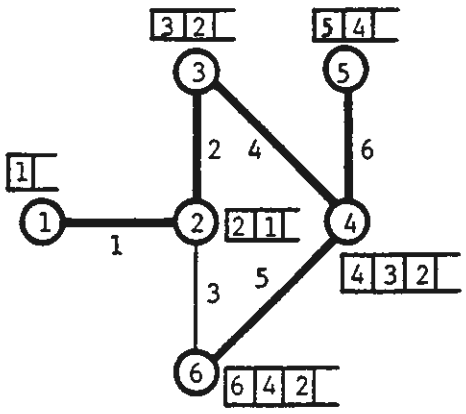
algorithm: BLOCKS.

Input:	A connected, non–directed graph Γ given by a set of neighbourlists L(p) , p ∈ $\mathcal{P}$(Γ) . A vertex s = 1 in Γ .
Output:	The blocks in Γ .
Variables:	m , $\mathcal{S}$, f , N , n as previously described.
Initialising:	m(p) = new and f(p) = Ø for all p . n = 1 . $\mathcal{S}$= Ø .

1. LOWPOINTSTACK(1)

Before we proceed to prove that the algorithm works correctly, we take an example. Below the figure on the next page, we show pictures of STACK (in the boxes), just before we remove elements from STACK. The elements which are removed from STACK are shown with { , and the numbers to the right of STACK are those edges which are considered in line 2 of LOWPOINTSTACK, but which are not going to be removed because of the condition of line 3.

We shall now explain that line 3 in LOWPOINTSTACK does not, as you may believe, take K steps, but can be carried out in one step. In line 3, we have to decide whether the p,q–edge k is already in the stack. k appears in L(p) as well as L(q) . If m(q) = new then it is not in the stack, for in this case we have not yet started searching through L(q) . If m(q) = old there are two possibilities: If q is father to p , we put k into STACK when we met (k, p) on L(q) . If q is not father to p then q must be an ancestor of p according to lemma 13.2. In this case we have not met (k, p) on L(q) yet and therefore k is not in the stack. So therefore line 3 can be carried out in one step.



L(1) : (1, 2) .

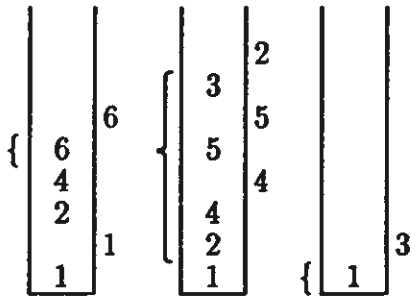
L(2) : (1, 1), (2, 3), (3, 6) .

L(3) : (4, 4), (2, 2) .

L(4) : (6, 5), (4, 3), (5, 6) .

L(5) : (6, 4) .

L(6) : (5, 4), (3, 2) .



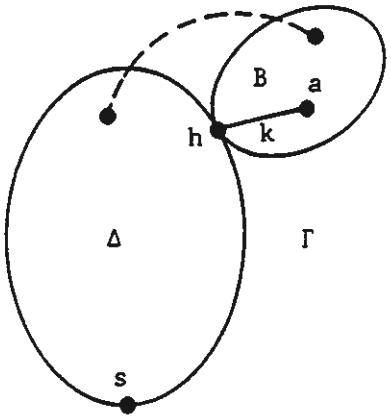
Theorem 14.8. BLOCKS finds the blocks in Γ correctly and has complexity O(K) .

Proof. That the complexity is O(K) follows from the descriptions above, and from the fact that all the executions of line 7 taken together demand K steps.

We shall now show the statement that when LM(q) = p in line 5 , then the STACK–edges down to and including k is the edge set of a block in Γ . The proof of this statement is by induction on the number n of blocks in Γ .

For n = 1 the equation LM(q) = p in line 5 is only satisfied for p = s = 1 , and hence when you are back in s you have gone through all edge lists and therefore put all edges in the stack (compare with theorem 17.1).

Assume after this that the statement is true for all graphs with $n-1$ blocks and let Γ be a graph with n blocks. Let B be a block in Γ which has valency 1 in the blocktree of Γ and which is not the block the search starts in. B then contains one and only one cutvertex h in Γ. In the depth first search tree, there exists an edge k from h to a son of h in B. That part of BLOCKS which is executed from the time where (k, a) is chosen for the first time in line 2, until we continue the search of $L(h)$, behaves exactly like BLOCKS on B, starting in h. This follows from the recursive nature of the algorithm and the fact that an edge, like the one shown as a dotted line in the figure, does not exist. Therefore the edges in B are removed from the stack correctly (compare with the case $n = 1$). The remaining part of the algorithm behaves exactly like BLOCKS on $\Delta = \Gamma - (B - h)$ and therefore, according to the induction assumption, gives correctly found blocks.



15. SORTING. In itself, sorting is not a graph theoretic subject. The reason why, nevertheless, we shall describe sorting algorithms here is that later in sections 29, 30 and 31 we shall need a good sorting algorithm.

Sorting can be ordering of numbers according to size, but it can also be ordering of words in alphabetical order. To get a reasonably general concept, we consider a set $\mathcal{M}$, which may contain repetitions. Assume that on $\mathcal{M}$ we have defined a relation " $\leq$ ", such that for every pair of elements a and b in $\mathcal{M}$ it can be decided whether $a \leq b$ or not. Then $(\mathcal{M}, \leq)$ is called a (*completely*) *ordered* set if

1. $a \leq a$ for all $a \in \mathcal{M}$.
2. $a \leq b$ and $b \leq c$ implies that $a \leq c$.
3. $a \leq b$ and $b \leq a$ implies that $a = b$.
4. For two arbitrary elements a and b in $\mathcal{M}$, we have that either $a \leq b$ or $b \leq a$ (or both).

Let us consider three examples:

1. If $\mathcal{M}$ is a set of real numbers, and $\leq$ is the usual inequality relation, then $(\mathcal{M},\leq)$ is an ordered set.
2. If $\mathcal{M}$ is a set of Danish words (for instance, {fy, fyr, fyr, fyr, fyr, fyren, fyret, fyret, fyrren}) and if $\leq$ means alphabetical order, then $(\mathcal{M}, \leq)$ is an ordered set.
3. If E is a given set and $\mathcal{M}$ is the set of subsets of E and $\leq$ means $\subseteq$, that is "subset of", then $(\mathcal{M}, \subseteq)$ is *not* an ordered set. The condition 4 is not satisfied.

The conditions 1–4 make sure that when $\mathcal{M}$ is a finite set, then the elements in $\mathcal{M}$ can be named $a_1, a_2, \ldots, a_m$ such that

$$a_1 \leq a_2 \leq \ldots \leq a_m \,,\; m = |\, \mathcal{M} \,| \,.$$

When such an order has been found, we say that $\mathcal{M}$ is *sorted.*

In this section, we shall describe four sorting algorithms.

INTEGER SORTING. We want to sort a finite set $\mathcal{M}$ (which may contain repetitions) of integers, which all belong to the interval $[1;K]$. Here K is an integer which is known by the programmer and which is not too large (for example, not larger than the working memory of the machine). The numbers in $\mathcal{M}$ are given as coordinates in the vector $v = (v_1, v_2,\ldots, v_m)$, $m = |\, \mathcal{M} \,|$. The sorting is now carried out in the following way: we define K empty,double–linked lists L(1), L(2),..., L(K). After that, we put v_1 in $L(v_1)$, v_2 in $L(v_2)$ and so on. When this is done, we form a new list by linking the lists L(1), L(2),..., L(K) after each other in this order. Then L will contain the elements of $\mathcal{M}$, sorted according to size. Since it demands K steps to define the lists, m steps to put the elements of $\mathcal{M}$ into the list and again K steps to link the lists up to each other, the algorithm has complexity $O(\max\{m, K\})$. With a slightly optimistic attitude, you can say that the sorting takes place in linear time.

Remark: INTEGERSORTING can not be generalised to, for example, alphabetic sorting, because you can't know which word the user wants to sort and therefore not know which list you should put the word into.

BUBBLESORT. To make the use of the word "bubble" natural, we assume that the elements of the ordered set we want to sort are given as a vertical vector $\mathbf{v}$. Let $\begin{bmatrix} v_i \\ v_{i+1} \end{bmatrix}$ be a vector consisting of two consecutive elements in $\mathbf{v}$. The order BYT $\begin{bmatrix} v_i \\ v_{i+1} \end{bmatrix}$ is defined in the following way: When $v_i \leq v_{i+1}$, nothing is changed, but if not we interchange in $\mathbf{v}$ the elements v_i and v_{i+1}. So, when we have executed BYT $\begin{bmatrix} v_i \\ v_{i+1} \end{bmatrix}$, then $v_i \leq v_{i+1}$. Then BUBBLESORT goes on in the following way:

$$\mathbf{v} = \begin{bmatrix} v_1 \\ v_2 \\ \cdot \\ \cdot \\ \cdot \\ v_m \end{bmatrix}$$

1. BYT $\begin{bmatrix} v_{m-1} \\ v_m \end{bmatrix}$.

2. BYT $\begin{bmatrix} v_{m-2} \\ v_{m-1} \end{bmatrix}$. BYT $\begin{bmatrix} v_{m-1} \\ v_m \end{bmatrix}$.

3. BYT $\begin{bmatrix} v_{m-3} \\ v_{m-2} \end{bmatrix}$. BYT $\begin{bmatrix} v_{m-2} \\ v_{m-1} \end{bmatrix}$. BYT $\begin{bmatrix} v_{m-1} \\ v_m \end{bmatrix}$.

.....

m–1. BYT $\begin{bmatrix} v_1 \\ v_2 \end{bmatrix}$. BYT $\begin{bmatrix} v_2 \\ v_3 \end{bmatrix}$. ⋯ BYT $\begin{bmatrix} v_{m-1} \\ v_m \end{bmatrix}$.

The number of comparisons which have to be performed is

$$1 + 2 + 3 + \ldots + m-1 = \tfrac{1}{2}\, m(m-1)$$

BUBBLESORT is therefore an $O(m^2)$ – algorithm. Below we show first a vector $\mathbf{v}$ with 7 coordinates, and after that the vectors which $\mathbf{v}$ will be changed to in the 1st, 2nd,..., 6th step

	1.	2.	3.	4.	5.	6.
7	7	7	7	7	→7	1
9	9	9	9	→9	1	2
3	3	3	→3	1	2	3
1	1	→1	1	2	3	3
3	→3	2	2	3	3	7
→8	2	3	3	3	8	8
2	8	8	8	8	9	9

Remark: The small elements go up as bubbles, i.e. the higher, the smaller they are! That the algorithm ends up with a correct sorting is seen as follows:

In step 1, we put v_{n-1} and v_n in the correct order.
In step 2, we replace v_{n-2} on the correct position in relation to v_n and v_{n-1}.
In step 3, we put v_{n-3} on the correct position in relation to v_{n-2}, v_{n-1} and v_n.
....
In step n–1, we put v_1 in the correct position. □

For small values of n, up to approx. n = 15, bubblesort is good enough, but for larger values of n, you can do better by means of MERGESORT. Again, let the coordinates in $\mathbf{v} = (v_1, v_2,..., v_m)$ be the set which is going to be sorted. The idea here is to use a "divide and rule" technique, in the following way: We cut the vector **v** in two halves and sort both of these in the same way. After that we merge the two halves.

Let $\mathbf{x} = (x_1, x_2,..., x_m)$ and $\mathbf{y} = (y_1, y_2,..., y_n)$ be the two vectors whose coordinates are elements in $\mathcal{M}$, and which are already ordered,

$$x_1 \le x_2 \le ... \le x_m \ , \ y_1 \le y_2 \le ... \le y_n \ .$$

The order $\mathbf{z} = \text{MERGE}(\mathbf{x},\mathbf{y})$, $\mathbf{z} = (z_1, z_2 ,..., z_{m+n})$ is then carried out by first comparing x_1 and y_1. If $x_1 \le y_1$, we put $z_1 = x_1$ and x_1 is deleted from **x**. Otherwise, we put $z_1 = y_1$ and y_1 is deleted from **y**. In a corresponding way, we determine z_2 by comparing, in the new vectors **x** and **y**, x_1 and y_1, and so on. If one of the vectors has become empty, we link the remaining coordinates of the other vector after **z**. MERGE(**x**,**y**) demands at most m + n – 1 comparisons.

MERGESORT uses a recursive procedure which is called SORT(x). It changes the order of the coordinates in the vector $\mathbf{x} = x_1, x_2,..., x_{2p}$, until they are sorted according to size.

procedure: SORT(**x**)
1. **if** dim **x** = 1, STOP .
2. **otherwise** x = MERGE(SORT($x_1, x_2,..., x_p$) , SORT($x_{p+1}, x_{p+2},..., x_{2p}$)).

In the algorithm which follows below, log denotes the logarithm with base 2 . $k = \log n$ means therefore that $n = 2^k$. $\lceil a \rceil$ denotes the smallest integer which is $\geq a$. For instance, $\lceil \log 7 \rceil = 3$, $\lceil \log 8 \rceil = 3$, and $\lceil \log 9 \rceil = 4$. When $2^{k-1} < a < 2^k$ then $\lceil \log a \rceil = k$.

algorithm: MERGESORT.

Input: A vector $\mathbf{v}$, the coordinates of which constitute an ordered set $\mathcal{M}$, which may contains repetitions, $\mathbf{v} = (v_1, v_2, \ldots, v_m)$.

Output: The vector $\mathbf{v}$, with coordinates sorted according to size.

Variables: k and n are integers ≥ 0 .

1. Change – if necessary – $\mathbf{v}$ by adding $(\infty, \infty, \ldots, \infty)$ until $n = \dim \mathbf{v} = 2^k$, $k = \lceil \log m \rceil$.
2. SORT($\mathbf{v}$).
3. Delete the last $n - m$ coordinates in $\mathbf{v}$.

As an example, we consider

$$\mathbf{v} = (3, 2, 3) .$$

Line 1 gives $\mathbf{v} = (3, 2, 3, \infty)$.

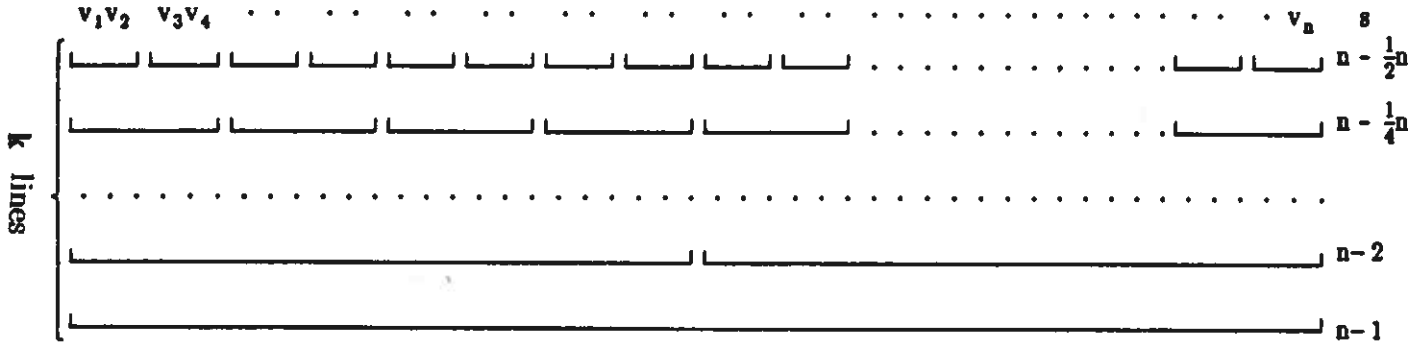
Line 2 is executed as follows:

SORT(3, 2, 3, ∞)
  SORT(3, 2)
    SORT(3) = (3)
    SORT(2) = (2)
    MERGE((3),(2)) = (2, 3)
  SORT(3, ∞)
    SORT(3) = (3)
    SORT(∞) = (∞)
    MERGE((3),(∞)) = (3, ∞)
  MERGE((2, 3),(3, ∞)) = (2, 3, 3, ∞)

Line 3 gives the result $\mathbf{v} = (2, 3, 3)$.

Theorem 15.1. MERGESORT sorts $\mathbf{v} = (v_1, v_2, \ldots, v_m)$ by means of at most $2m\lceil \log m \rceil - m+1$ comparisons. If m is a power of 2, we obtain at most $m \log m - m + 1$ comparisons. The complexity of the algorithm is $O(m \log m)$.

It is clear that $\mathbf{v}$ eventually will become sorted correctly. Hence we shall only count the number of comparisons. We consider the vector $\mathbf{v}$ which is constructed in line 1 and which has the dimension $n = 2^k$. The second line in the figure shows those pairs of elements which are going to be compared in those calls of SORT(x) , where $\mathbf{x}$ has dimension 2. Any of the lines below shows how the sorted vectors in the line above is merged.



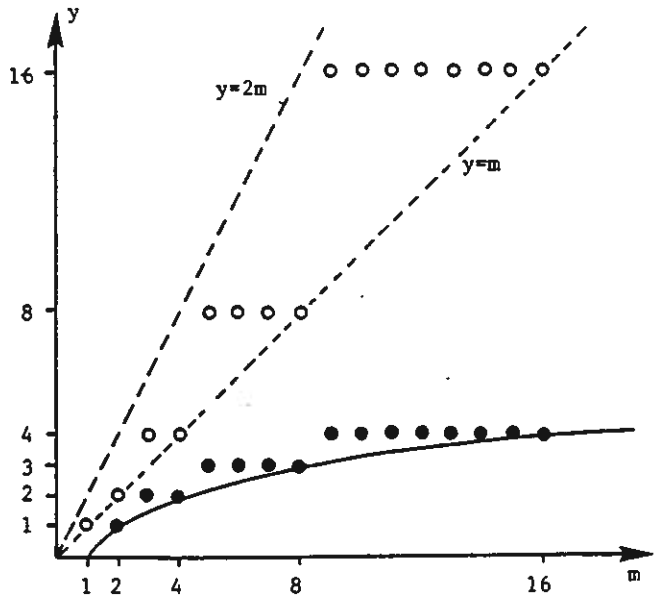
The number s to the right of each line is an upper bound for the number of comparisons which are necessary to get the numbers in the brackets in that line sorted. An upper bound for the total number of comparisons is

$$S = n \cdot k - (1 + 2 + 4 + \ldots + \tfrac{1}{2} n) = n \log n - (n - 1) \ .$$

If $m = n = 2^k$ we have proved the result of the theorem in this case. If m is not a power of 2 then the number we have found, which is denoted by S , can be expressed by m as follows: Since $n = 2^k$ and $k = \lceil \log m \rceil$ we have that

$$S = n \log n - n + 1 = 2^{(\log m)}\, k - 2^{(\log m)} + 1 \ .$$

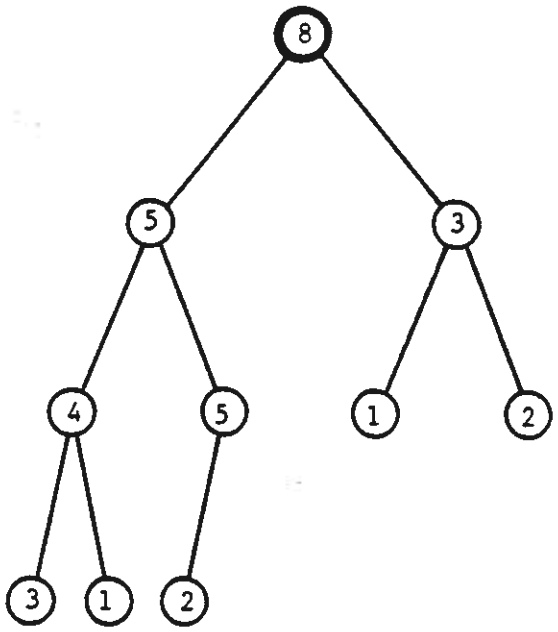
In the figure, the curve is $\log x$, $x \geq 1$; the black circles show $\log m$ and the open circles show $2^{(\log m)}$. Observe that $m \leq 2^{\log m} < 2m$, and therefore we have that $S < 2m\lceil \log m \rceil - m+1$, which we should show. It is now clear that the complexity is $m \log m$, that is, better than bubblesort.



It is natural to ask, whether there exists an algorithm which sorts an arbitrary ordered set in linear time, like INTEGER SORT. It is a surprising and interesting fact that by means of a simple argument, we can show that there does not, if we assume an algorithm which consists of a sequence of comparisons. The argument goes as follows: Let us consider a possibly not yet found sorting algorithm, which can sort every ordered set $\mathcal{M}$ without repetitions with m elements by means of a sequence of at most s comparisons. In the 1st comparison, we compare two elements x and y and there are two possible results: Either $x \geq y$ or x is not $\geq y$. *For both of these two*, the second comparison can have two different results. Therefore, there are 4 possible results of the two first comparisons. In this way we can continue and prove that there are at most 2^s possible results as outputs of the algorithm. Since there are $m!$ possible sortings of $\mathcal{M}$, the number of possible outputs from the algorithm must be $m!$, and therefore $2^s \geq m!$ from which $s \geq \log(m!)$. According to a formula called Stirlings Formula, [3], we have that $\log m! > m \log m - \frac{m}{\ln 2} > m \log m - 1.45m$. Therefore we have

Theorem 15.2. Consider a sorting algorithm which can sort any ordered set by means of a sequence of at most S_m comparisons of 2 elements, where $m = 2^k$ is the number of elements in the set. Then $m \log m - 1.45\, m < S_m \leq m \log m - m + 1$.

The last sorting algorithm which we will describe uses a data structure which is called a *heap*. The algorithm can sort a set with n elements in $O(n \log n)$ steps. It has the same complexity as mergesort.



The figure shows how the set

$$\mathcal{A} = \{5, 1, 2, 5,, 1, 8, 2, 4, 3, 3\}$$

Can be organised in a heap. Observe that a heap is a binary tree with a root. "binary" because each vertex p ("the father") has at most 2 "sons", that is neighbours which are further away from the root than p itself. The vertices of the tree are the elements in $\mathcal{A}$. The root is the largest element in $\mathcal{A}$ and along each path from the root to a "leaf", i.e. a vertex (different from the root) with valency 1), the numbers are ordered according to size. When b_1 and b_2 are leaves, the difference between the distances from the root to b_1 and b_2 must not be > 1. The distance classes measured from the root are called *levels*. The lowest level must be filled up from the left. The *depth* d of a heap is the largest distance from the root to a leaf. If n is the number of elements in $\mathcal{A}$, that is the number of vertices in the tree, then

$$1 + 2 + 4 + \cdots + 2^{d-1} < n \leq 1 + 2 + 4 + \cdots + 2^d \ .$$

We put $S = 1 + 2 + 4 + \cdots + 2^d$, than $2s = 2 + 4 + \cdots + 2^d + 2^{d+1}$, and by subtraction we get $S = 2^{d+1} - 1$. Therefore we have that $2^d - 1 < n \leq 2^{d+1} - 1$ or $2^d \leq n < 2^{d+1}$, from which $d \leq \log n < d + 1$, that is $d = \lfloor \log n \rfloor$. The depth is therefore the largest integer $\leq \log n$. In the figure on the previous page, we have $d = 3$ and $\lfloor \log n \rfloor = \lfloor \log 10 \rfloor = \lfloor 3{,}32 \rfloor = 3$.

In a computer a heap can be represented by a vector (– sufficiently large –) in which the root is mentioned first and after that in due order the levels beneath that, each level in order taken from left to right. We shall however, in this book, assume that it is possible directly to work with a heap as it is drawn on the previous page.

Adding a new element a **to a heap**. This is done in the following way: First you add a in the lowest level as far to the left as possible (– that is leftmost in the new level if the lowest level was completely filled up –). After that, you exchange(– normally several times –) a with its father until the numbers on the path V from the root to the new leaf stand in correct order. Since every number of V has, in this procedure, been replaced by a new number $\geq$ the old number, we still have the numbers in the paths $\neq$ V in correct order. *Therefore, the adding of a new element to a heap can be executed in* $d = \lfloor \log n \rfloor$ *steps.*

By applying this process several times, you observe that a set with n elements can be organised in a heap in O(n log n) steps.

Deletion of the root from a heap. This is done in the following way: First you take away the number from the root, leaving an empty place. After that, you take the contents of the last leaf to the root and delete the last leaf.Finally, you exchange (–normally several times–) a with the largest son, until order is restored. This process demands also O(n log n) steps.

You see that a set $\mathcal{A}$ with n elements can be organised in a heap in n log n steps and after that, the elements in $\mathcal{A}$ can be taken from the heap in order of size in n log n steps. This sorting algorithm is called HEAPSORT. It has complexity O(n log n).

Problems for Chapter 2.

2.1. Refer to the diagram on page 44, but for a given vertex p with orders

1. Delete p.
2. Find $v_u(p)$.
3. Find $v_i(p)$.

2.2. Let Γ be a non–directed graph which is represented by a set of edge lists. How many steps does in take to construct

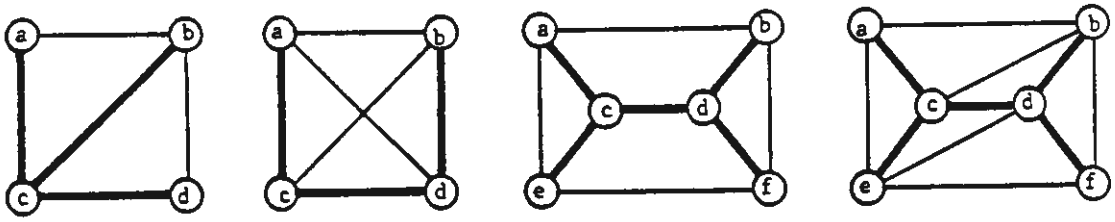
a) The neighbourlist of Γ .

b) The doublelist of Γ .

2.3. Let Γ be a connected graph and let s be a vertex in Γ . Describe a variation of BREADTH FIRST SEARCH such that you obtain an algorithm which, for each vertex p in Γ , finds the number of shortest s, p–paths in Γ . What will the complexity be, assuming that it is one step to add 2 numbers.

2.4. Let Γ be a connected graph without loops. Describe a variation of BREADTH FIRST SEARCH such that you obtain an algorithm which determines whether Γ contains an odd circuit or not. Find the complexity.

2.5.



In each of the above graphs, there exists a tree, T , shown by heavy lines. In each case decide whether

a) T is a BREADTH FIRST SEARCH tree.

b) T is a DEPTH FIRST SEARCH tree.

2.6. Let Γ be a connected graph which contains a subgraph K , which is a complete graph. Let T be a DEPTH FIRST SEARCH tree in Γ . Show that T contains a path containing all vertices of K .

LITERATURE FOR CHAPTER 2.

[1] Aho, Alfred V., Hopcroft, John E., Ullman, Jeffrey D.: The Design and Complexity of Computer Algorithms. Addison-Wesley 1975.

[2] Cohen, D.I.A., Basic Technics of Combinatorial Theory. John Wiley & Sons 1978.

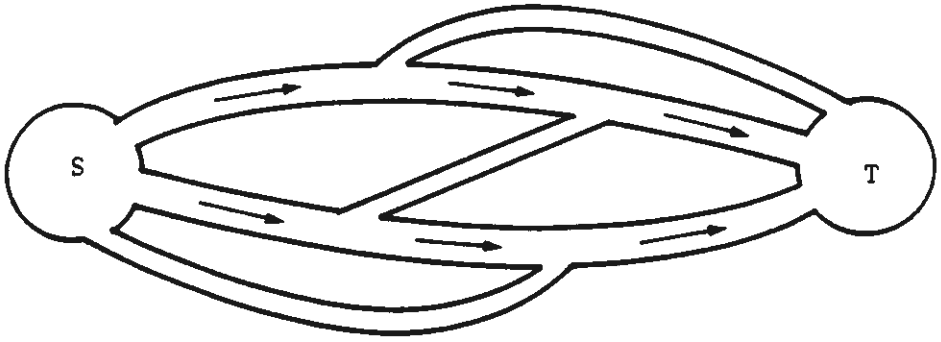
[3] Even, Shimon: Graph Algorithms. Computer Science Press, Inc. 1979.

CHAPTER 3

TRANSPORT NETWORKS

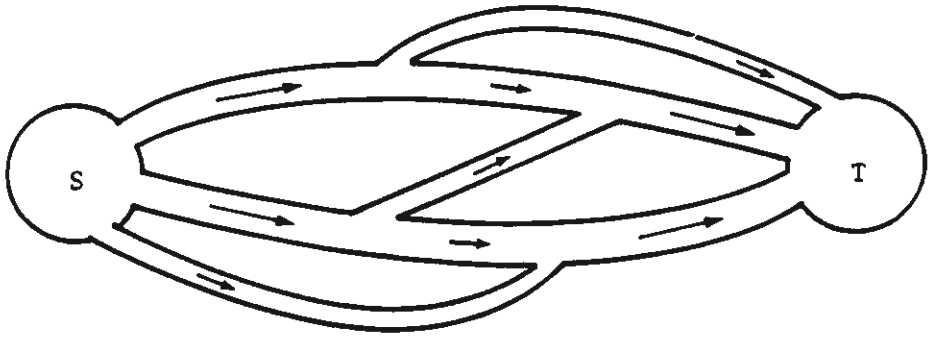
16. INTRODUCTION. When you have to solve a physical or technical problem by means of mathematical methods, you usually have to reformulate the problem first to make it simpler and more precise. In many cases, it is necessary to simplify the problem so much that it becomes uninteresting both from a technician's and a mathematician's point of view. In this chapter we are going to look at the theory of transport networks which is illustrated in the following example: For a certain type of transportation network, we shall give an interesting mathematical theorem, the result of which is a good algorithm for determining a maximal flow in the network.

Let us illustrate the kind of problems we are going to deal with, with a simple example.



The diagram above shows the roads from the town S , where people live, to the city centre T , where the people work. Assume that the two main roads can take 2000 cars per hour (in the direction from S to T) and that each of the three smaller roads can take 1000 cars per hour. We want to find the maximum number of cars that can drive from S to T per hour. Assume that at a given time, the traffic is distributed with 2000 cars per hour on each of the two large roads while nobody uses the three smaller roads. If the traffic flow from S is going to be greater than 4000 cars per hour, there is only room on the small roads, but if you use them, there will be too many cars where the small roads join the larger ones. However, this does not mean that the traffic flow we have, is maximal. It only shows that the distribution chosen is not suitable. In fact, you can obtain a flow from S to T with 5000 cars per hour by

distributing the cars as shown in the figure below, where the small arrow means 1000 cars per hour and the large arrow means 2000 cars per hour.

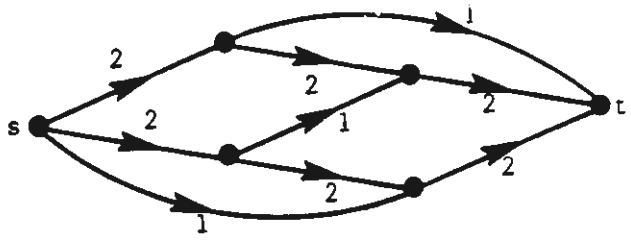


The example indicates that an algorithm to determine a maximal flow in a transport network is not so simple. We shall describe such an algorithm later in this chapter. Also, we will characterise what it is in the structure of a transportation network which puts the upper bound on how much traffic can go through the network. The theory of transport networks can be applied to many things other than road networks, for example, electrical distribution nets, tube networks where there are liquid flows and transportation of information in a telephone network can all be treated by means of the theory.

First we consider the case where the transport network is directed corresponding to the situation where we have only one–way roads. We assume that all traffic starts at a certain place S, the *source*, and that the traffic is going to the same place T , the *terminal.* Later, we shall see that it will be easy to deal with the non–directed networks and directed networks with more than one source and terminal.

The basic theory of transport networks (in particular the max flow min cut theory) was developed in the 1950's by two American mathematicians, Ford and Fulkerson, in connection with military transportation problems. Both the basic theory and it's later development were inspired by the development of computer. We have here an example which shows that the mathematics develops, not only according to it's own interior laws, but also as new mathematical theories are derived from applications in the real world. The newest result in this chapter is Dinics theorem which was discovered in 1970.

17. DIRECTED TRANSPORT NETWORKS A *(directed)* transport network N, is a directed graph, which for each edge k there is a real number $c_k \geq 0$, the *capacity* of k , and in which two different vertices s and t are selected. They are called the source and terminal in N , respectively.



In the figure we have shown an example of a transport network. For every edge, the number is the capacity of the edge. This network has the same structure as the example in the introduction. The contents of the definition is that we have a transport network where something can flow in the direction of the arrows and where we want as much as possible to flow from s to t, subject to the condition that the capacity of each edge is the upper bound for the amount of flow in that edge. A flow f in a transport network is defined, that for each edge k in the network, there is a real number f(k) or f_k (called the flow in the edge k) which satisfies the following two conditions:

S1. ***(The Equilibrium Condition).*** For each vertex p in N, different form s to t, we have

$$\sum_{k \text{ begins in } p} f(k) = \sum_{k \text{ ends in } p} f(k)$$

ie. flow into p = flow out of p

S2. ***(Limit Condition).*** For each edge k, we have:

$$0 \leq f(k) \leq c(k)$$

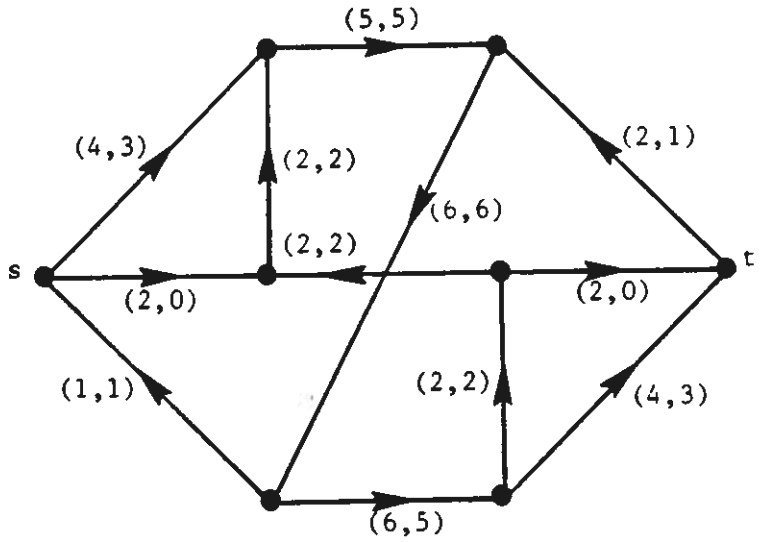
In other words the flow in an edge is a non negative real number not greater than the capacity of the edge.

The value F (or F(f)) of a flow f is defined as the net flow out of s. That is:

$$F = F(f) = \sum_{k \text{ begins in } s} f(k) - \sum_{k \text{ ends in } s} f(k)$$

A flow is *maximal* if its value is greater than or equal to the value of any other flow. Zero flow is when the flow in each edge has flow zero. The zero flow has value 0.

Normally we describe the flow in a transport network by writing a pair of numbers $(c_k, f(k))$ near each edge ie. first the capacity, then the flow value. The figure below shows an example where the value of the flow is 2. The flow in this case is not maximal.



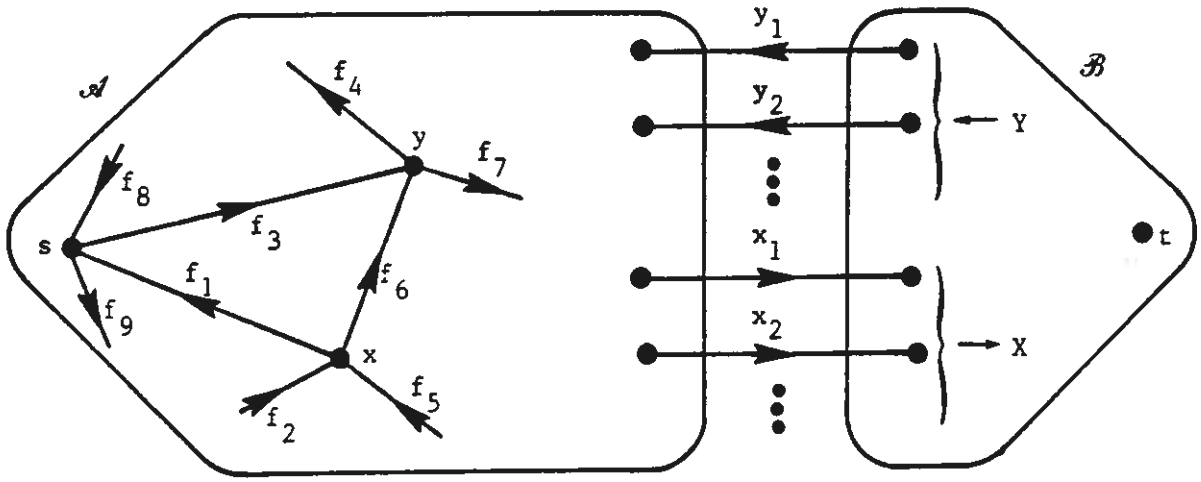
The example is chosen as an illustration of the fact that flow may go from t to s and that flow is allowed to circulate in a one–way circuit.

We conclude this introduction to transport networks by showing a lemma which uses only the equilibrium condition and therefore has nothing to do with the capacities. The lemma looks rather trivial but is the basis for the more interesting considerations in the next section.

Lemma 17.1 When we partition the vertices in a transport network N into 2 sets $\mathcal{A}$ and $\mathcal{B}$ such that $s \in \mathcal{A}$, $t \in \mathcal{B}$ then the value F of a flow f will be determined by :

$$F = X - Y$$

where $X = x_1 + x_2 + \ldots$ is the total flow in the $\mathcal{A} \rightarrow \mathcal{B}$ edges, and $Y = y_1 + y_2 + \ldots$ is the total flow in the $\mathcal{B} \rightarrow \mathcal{A}$ edges.



Proof. For every vertex $(\neq s)$ in $\mathcal{A}$, we write the equation which expresses the total flow to the vertex is equal to the total flow away from the vertex.

$$\begin{aligned} x: &\quad f_5 + f_2 + \ldots = f_6 + f_1 + \ldots \\ y: &\quad f_6 + f_3 + \ldots = f_4 + f_7 + \ldots \\ \hline s: &\quad F + f_1 + f_8 + \ldots = f_9 + f_3 + \ldots \end{aligned}$$

The last equation we have written expresses the definition of the value F of the flow. Now let us add all these equations. The flow f_k in an edge k which connects two vertices in $\mathcal{A}$, will appear precisely once on each side. When we add the equation, such flows (like f_1, f_3 and f_6) will appear once on each side of the "equals" sign and therefore we obtain an equation which can be reduced to

$$F + y_1 + y_2 + \ldots = x_1 + x_2 + \ldots$$

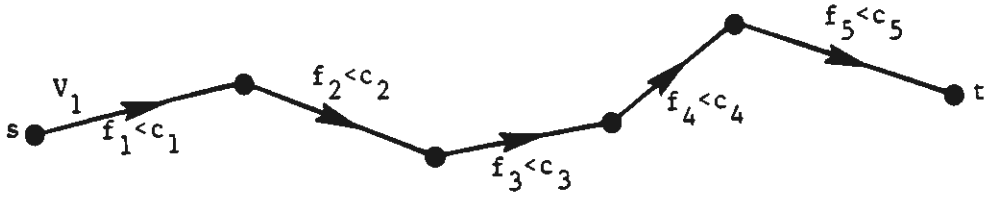
In particular, we put $\mathcal{B} = \{t\}$ and $\mathcal{A} = \mathcal{P}(N) - \{t\}$ we obtain from the lemma that the value of the flow is equal to the net flow into t. This is not surprising, since the equilibrium condition says that the no flow disappears or appears between s and t.

18. ON MAXIMAL FLOWS

As mentioned in the introduction, it is not very easy to find a simple method to determine a maximal flow in a transport network. In this section, we shall describe two observations, which are useful in the solution of the problem. First, we will describe a method which can be used to change a flow to another with a larger value. Then we will explain how, by looking at a transport network, it is easy to give an upper bound for the value of a flow.

Let us consider a transport network with flow f with value F. If there exists an $s \to t$ path V_1 in which all edges are unsaturated, then it is easy to construct a flow with a larger value.

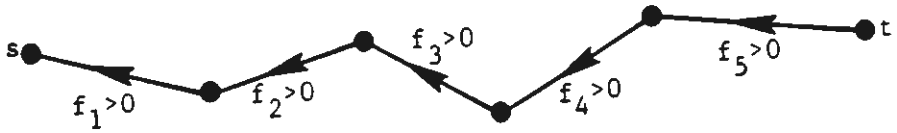
Since no edge in V_1 is saturated, we can add a constant flow ΔF to the flow in each edge of V_1 without exceeding the capacities. The new flow value will be $F + \Delta F$ (and the equilibrium condition is of course still satisfied). The largest ΔF that can be used is



$$\Delta = \min \{ c_k - f_k \mid k \in \mathcal{K}(V_1) \},$$

i.e. the minimum difference between the capacity and the flow in the edges on V_1.

But it can also happen that the flow f is constructed so awkwardly that there is a path V_2 from t to s, in which all edges have flow > 0 (>!).



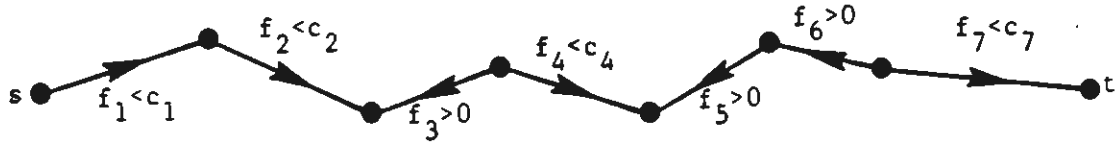
In this case, it is easy to construct a new flow with a larger value. Since no edge on V_2 has flow zero, we can subtract a constant flow ΔF from the flow in every edge on V_2 without making any flow negative, the flow still being smaller than or equal to the capacity.

The new value becomes $F + \Delta F$ (+!) (and the equilibrium condition is of course satisfied). The largest ΔF we can use is

$$\Delta F = \min \{ f_k \mid k \in \mathcal{K}(V_2) \},$$

i.e. min. flow in an edge on V_2. And now we have a very useful observation:

If we can find an $s \to t$ path V, in which the edges which are directed from s to t are unsaturated, while those edges which are directed from t to s, all have flow greater than zero, then we can construct a new flow with larger value. Therefore we



call this path an *augmenting path* w.r.t. the flow f.

Note that the paths V_1 and V_2 considered above are special types of augmenting paths. An augmenting p,q path is defined in a similar way. It can, of course, only be used to increase the flow value when $p = s$ and $q = t$.

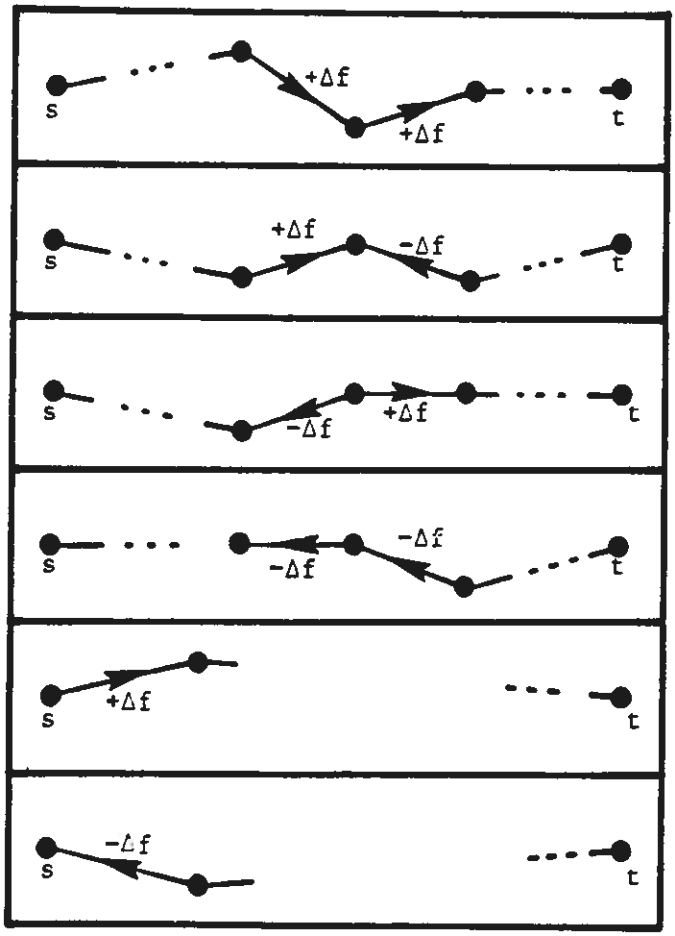
If we want to *use* the augmenting s,t path for the construction of a flow with larger value than F, we proceed as follows: We choose a value $\Delta V > 0$, and then we *add* ΔV to the flows in the edges on V which are directed from s towards t, but we *subtract* ΔV from the flows in those edges of V directed from t to s. If we choose ΔV to be the smallest of the two numbers

$$\min \{ c_k - f_k \mid k \in \mathcal{K}(V) \text{ and k directed from s to t} \}$$

and

$$\min \{ f_k \mid k \in \mathcal{K}(V) \text{ and k directed from t to s} \},$$

then it is clear that no flow exceeds the capacity and no flow becomes negative. This ΔV is called the *value* of the path. That the equilibrium condition is satisfied, is seen from the figures, where the changes in the flow are given in the different cases.



Finally, it is easy to see that the value of the new flow will be $F + \Delta V$. For if the edge of V which is incident with s , begins in s , then ΔV will be added to the flow, and if it ends in s , ΔV is subtracted from the flow. In both cases, the net flow out of s is ΔV larger than before.

The first observation can be formulated as shown:

Lemma 18.1. If there is an augmenting s,t path w.r.t. the flow f , then f is not maximal.

As we shall later see, the concept of the augmenting path will play an important role in determining the maximal flow in the network. Therefore, we shall describe a labelling procedure called MARKER (f,s) , which can be used either to determine an augmenting path, or to prove that there is no augmenting path.

The procedure requires a network N and a flow f as input. The procedure uses a vertex s in N , which normally, but not always, is the source. The result is that certain vertices p will be labelled $m(p) = (q,\Delta p)$, where $q \in \mathcal{P}(N)$ and $\Delta p > 0$.

---

procedure: MARKER (f,s) .

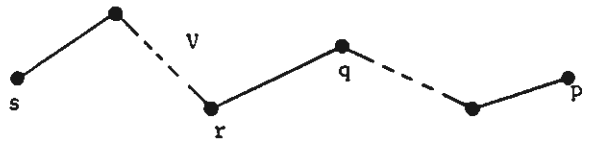
1. Put $m(s) = (s,\infty)$ and $m(p) =$ "unlabelled" for $P \in \mathcal{P}(N)$, $p \neq s$
2. **while** there exists
3. **either** an unsaturated $p \rightarrow q$ edge k , where p is labelled and q unlabelled,
4. put $m(q) = (p,\Delta q)$, where $\Delta q = \min (\Delta p, c_k - f_k)$,
5. **or** a $q \rightarrow p$ edge k with $f_k > 0$, where p is labelled and q unlabelled,
6. put $m(q) = (p,\Delta q)$, where $\Delta q = \min (\Delta p, f_k)$.
7. When such edges no longer exist, let $\mathcal{A}$ be the set of labelled vertices and $\mathcal{B}$ be the set of unlabelled vertices . STOP.

We have not specified any data representation here. Therefore we cannot decide the complexity of the procedure, but we shall prove the following theorem:

Theorem 18.2. When MARKER (f,s) is stopped, the following is valid for every $p \in \mathcal{P}(N)$:

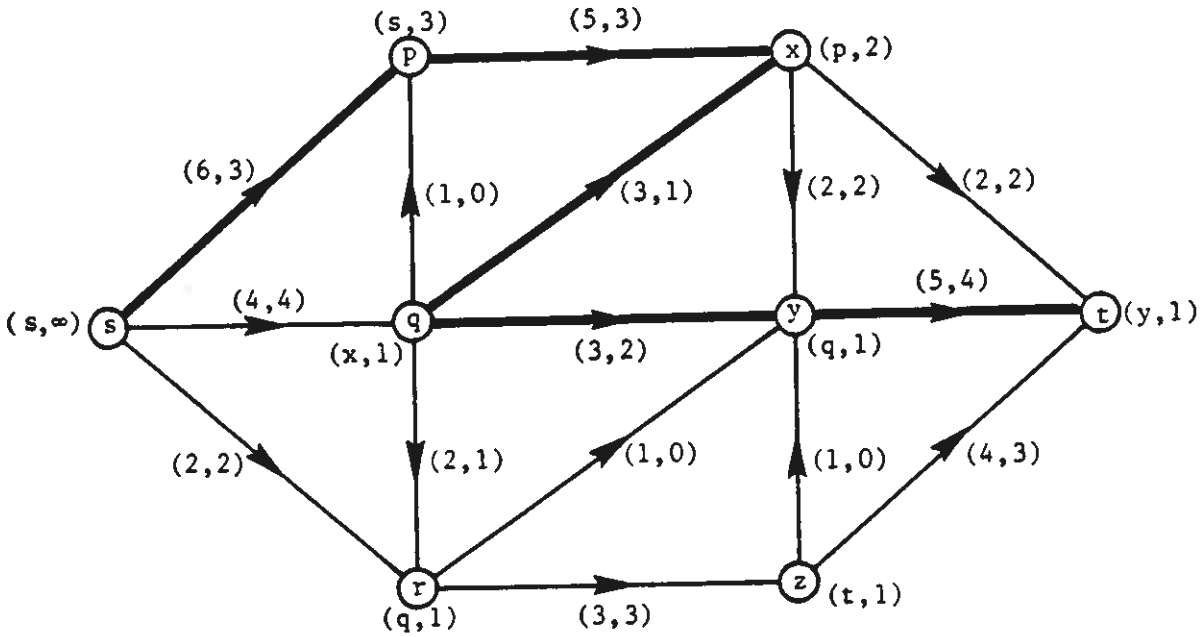
1. If p is labelled, there exists an augmenting s,p path V w.r.t f in N with value $\Delta V = \Delta p$.
2. If p is not labelled, there does not exist an augmenting s,p path w.r.t. f in N .

Proof. Suppose p is labelled $(q,\Delta p)$, then q must be labelled, let us say, $(r,\Delta r)$. Also, r must be labelled, and so on. Then it is clear that s,...,r,q,p is an augmenting path V with value $\Delta V = \Delta p$. V is called the augmenting s,p–path *determined by the labelling procedure.* Now suppose, conversely, that p is not labelled. We shall show that there is no augmenting s,p–path . The proof is indirect. So we assume that there is an augmenting s,p–path V . Since s is labelled and p is not labelled there must – when we walk along V from s towards p – be a last labelled vertex r on V . But since V is augmenting, and r is labelled, then we will be able to label the next vertex q on V , in contradiction with the fact that MARKER (f,s) is already stopped. □

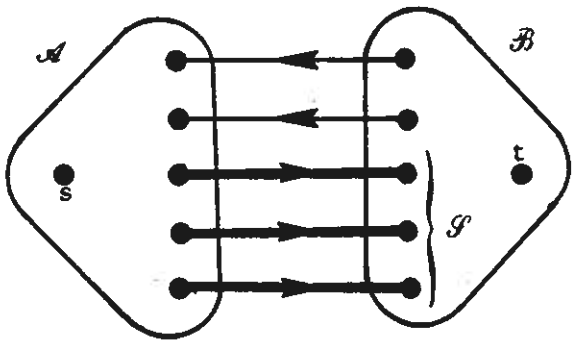


It follows from theorem 18.2, that the set of labelled vertices is independent of the order in which the edges are considered when we use the procedure.

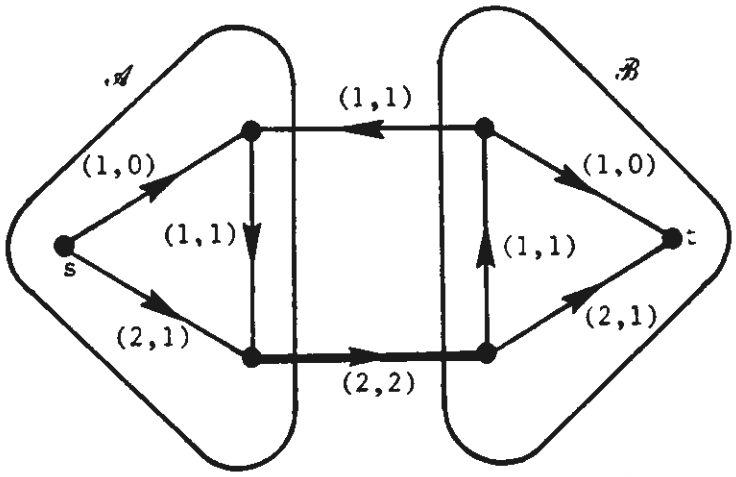
The figure shows an example where the labelling process has stopped, and where t has been labelled. The path shown in bold is the augmenting s,t–path, determined by the labellings. In general, we find that if t is labelled with $(q,\Delta t)$, then Δt is the value of an augmenting s,t–path, which can be determined by going back step by step, first from t to q , then to the point q is labelled from, and so on, until s is reached. This path is said to be determined by the labelling process.



Having found a method by means of which we can make small flows larger, we will search now for a form of "bottleneck" which gives an upper bound on the value of the flow. It is possible to find an upper limit by considering a *cut* in a transport network N . A cut is defined in the following way: Partition $\mathcal{P}(N)$ into two sets $\mathcal{A}$ and $\mathcal{B}$ such that $s \in \mathcal{A}$ and $t \in \mathcal{B}$. The set $\mathcal{S}$ of edges starting in $\mathcal{A}$ and ending in $\mathcal{B}$ is called a *cut* in N , and we write $\mathcal{S} = (\mathcal{A}, \mathcal{B})$. The number $c(\mathcal{S}) = \sum_{k \in \mathcal{S}} c_k$ is called the *capacity of the cut.* A cut is called *minimal* if it has capacity less than or equal to the capacity of any other cut.



Now consider a flow f in N such that all edges in a cut $\mathcal{S}$ are saturated. Since all the flow from s to t has to pass one of the edges in $\mathcal{S}$ and since these are all saturated, we cannot have more flows going from s to t . We should therefore expect that f is a maximal flow.



However, this is not always the case, as can be seen from the example in the figure, where $\mathscr{S}$ consists of one edge. This is an example which shows that you should make your arguments clear in discrete maths. If you use arguments which are not absolutely precise, you can come to wrong conclusions. Below we have a second useful observation where a cut is used in the correct way:

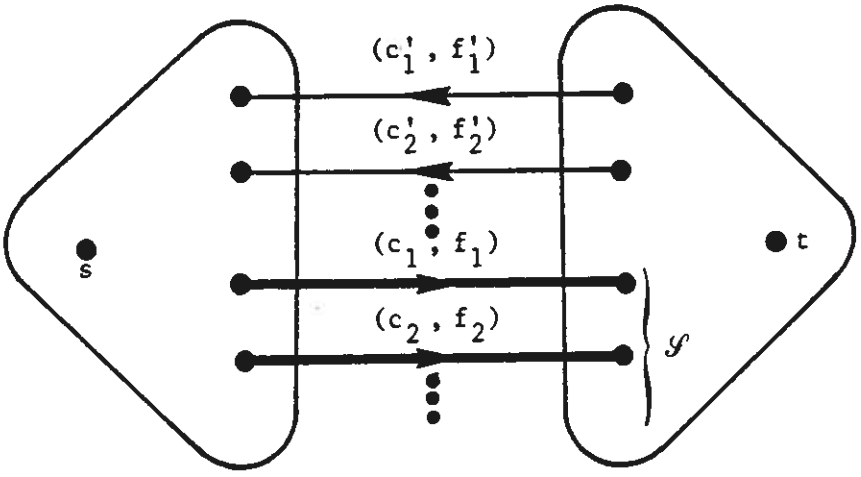
Lemma 18.3 If f is an arbitrary flow in a transport network N, and if $\mathscr{S}$ is an arbitrary cut in N, then

$$F \leq c(\mathscr{S}),$$

where F is the value of f and $c(\mathscr{S})$ is the capacity of $\mathscr{S}$.

Proof. The Lemma is easy to show by means of Lemma 18.1, which, with the notation shown in the figure, gives

$$F = f_1 + f_2 + \ldots - (f_1' + f_2' + \ldots),$$



from which we can obtain

$$F \leq f_1 + f_2 + \ldots \leq c_1 + c_2 + \ldots = c(\mathscr{S}).$$ □

From Lemma 18.3 follows immediately (think about it) the

Corollary 18.4. If f is a flow with value F and if $\mathscr{S}$ is a cut with capacity $c(\mathscr{S})$, and if

$$F = c(\mathscr{S}),$$

then f is a maximal flow and $\mathscr{S}$ is a minimal cut.

Lemma 18.3 says that there is an upper bound for the value of a flow from s to t in a transport network. This, however, does not imply that there exists a maximal flow; you could imagine that there exist flows with values arbitrarily near to a number F_0, but now flow with value F_0 or more. However, we have

Theorem 18.5. Every transport network N has a maximal flow.

We shall later see that Theorem 18.5 follows from the Edmonds–Karps Theorem (p. 103). But the proof of Theorem 18.5 which we obtain in this way is complicated, because the proof of the Edmonds–Karps Theorem is complicated. Below, we shall give a short direct proof of the Theorem. In this proof, we use the fact that it is a fundamental property of the real numbers, that every bounded sequence

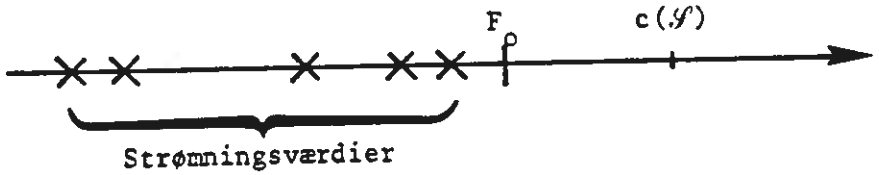
$$a_1, a_2, a_3, \ldots, a_n, \ldots$$

has a convergent subsequence

$$a_{i_1}, a_{i_2}, a_{i_3}, \ldots, a_{i_n}, \ldots,$$

where the i_j's are natural numbers, and where $i_1 < i_2 < i_3 < \ldots < i_n < \ldots$. The fact that eh sequence (a_n) is bounded means that there exist constants A and B (independent of n) such that $A < a_n < B$ for all n. That the sequence (a_{i_n}) is convergent, means that the limit $\lim_{n \to \infty} a_{i_n}$ exists.

Proof of Theorem 18.5. According to Lemma 18.3, the set of flow values is bounded above by the capacity $c(\mathcal{S})$ of an arbitrary cut $\mathcal{S}$. Let F_0 be the smallest number which is greater than or equal to the flow values. We will show that there exists a flow with value F_0. Since F_0 is an upper bound for the set of flow values, there exists a sequence $f_1, f_2, \ldots$ of flows such that $F_i \to F_0$ for $i \to \infty$, where F_i denotes the value of the flow f_i. Let us number the edges in N by $1,2,\ldots,n$ and let c_j denote the capacity of edge j and let f_{ij} denote the flow in edge j in the flow f_i. Since $0 \leq f_{i1} \leq c_1$ the sequence $f_{11}, f_{21}, f_{31}, \ldots$ is bounded. Therefore it has a convergent subsequence. One is denoted f_{i1}, $i \in I_1$; its limit is denoted f_1^*. Since $0 \leq f_{i2} \leq c_2$ for all $i \in I_1$ the sequence f_{i2}, $i \in I_1$ is bounded; therefore it has a convergent subsequence f_{i2}, $i \in I_2 \subseteq I_1$; its limit is denoted by f_2^*.



Of course, the sequence f_{i1}, $i \in I_2$ is convergent with limit f_1^*. In this way the process can be repeated n times and we still have a convergent subsequence. The result is a subsequence f_i, $i \in I_n$ of flows such that the sequence F_i, $i \in I_n$ has limit F_0 and such that the sequence f_{ij} has the limit f_j^* $(j = 1,2,\ldots,n)$. Then it is clear that, if for $j = 1,2,3,\ldots,n$ we send the flow f_j^* through edge j, then we have a flow in N whose value is F_0. □

The proof is maybe more complicated than we would expect is necessary. We shall later give a completely combinatorial proof for the theorem but this proof cannot be given until larger parts of the theory of transport networks are developed.

19. A FLOW ALGORITHM. THE MAX–FLOW–MIN–CUT THEOREM.

Lemma 18.1 makes it natural to consider the following algorithm, which calls the labelling procedure (p. xxx):

algorithm: MAXIMAL FLOW

Input:	A transport network N with source s, terminal t and a capacity $c_k \geq 0$ for each edge k .
Output:	A flow f . A cut $\mathcal{S} = (\mathcal{A}, \mathcal{B})$.
Variables:	F is the value of f . V is an augmenting path w.r.t. f . ΔV is the value of V . m(q) = (p,Δq) is a labelling of the vertex p .
Initialising:	f is the zero flow (or any flow with value f)

1. MARKER (f,s) .
2. **If** $t \notin \mathcal{A}$, STOP.
3. **Otherwise** (that is if t is labelled)
4. Use the augmenting s,t–path V , which is determined by the labelling procedure, such that you obtain a new flow f with larger value.
5. **Go to** line 1.

Remark: If $\mathcal{A}_i$ is the set of those vertices which become labelled in the i^{th} application of line 1 , then $\mathcal{A}_1 \supseteq \mathcal{A}_2 \supseteq \mathcal{A}_3 \supseteq \ldots$

It is easy to see that if all capacities are rational numbers (in particular integers) which we shall assume are written with a common denominator N , then the algorithm will stop if we start with a zero flow. This follows from the fact, that all flows and capacities in that case are multiples of 1/N , and therefore the value of an augmenting path will also be a multiple of 1/N . If $\mathcal{A}$ is a cut, the algorithm , according to lemma 18.3, will stop, after having used at most $N \cdot c(\mathcal{S})$ augmenting paths.

Remark – that it is in no way clear that when MAXIMAL FLOW has stopped, then the obtained flow is maximal. Of course, there are no more augmenting paths, but there could be other ways to make the flow larger. However, we shall show the very important

Theorem 19.1 If the maximal flow algorithm is stopped, then the output is a maximal flow.

Lemma 18.3 says that every cut capacity is $\geq$ every flow value. It is not clear that for every network it is possible to obtain the equal sign here, but we shall show that this is actually the case:

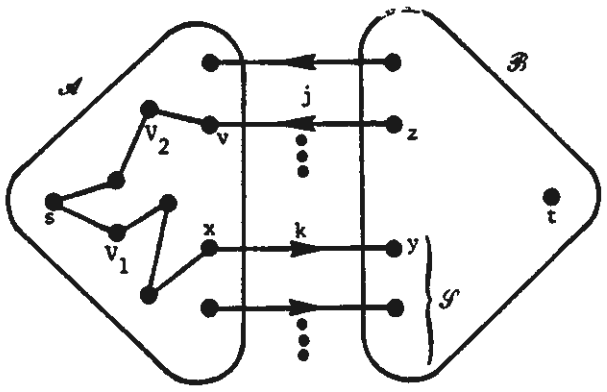
The max–flow–min–cut Theorem For every transport network, the value of a maximal flow is equal to the capacity of a minimal cut

$$\max_{f \text{ flow}} F(f) = \min_{\mathscr{S} \text{ cut}} c(\mathscr{S}) .$$

It is a point (which in particular has been stressed by Jack Edmonds), that there exists an argument which shows simultaneously that the output of the maximal flow algorithm is a maximal flow and that the max–flow–min–cut theorem is correct. Both theorems follow immediately from the following lemma:

Lemma 19.2. When f is a flow in the transport network N and when there does not exist any augmenting path w.r.t. f, then f is maximal and the value of f is equal to the capacity of a minimal cut in N.

Proof of the Lemma. Let $\mathscr{A}$ denote the set of vertices p in N for which there exists an augmenting s,p–path. Then $s \in \mathscr{A}$ and $t \in \mathscr{B} = \mathscr{P}(N) - \mathscr{A}$.



1. Every $\mathscr{A} \to \mathscr{B}$ edge k is saturated. Let x denote the start vertex of k and y the end vertex, $x \in \mathscr{A}$, $y \in \mathscr{B}$. Then there exists an augmenting s,x–path V_1, and if k was not saturated, then k could be added to V_1 and we would have an augmenting s,y–path, and then $y \in \mathscr{A}$, a contradiction.

2. Every $\mathcal{B} \to \mathcal{A}$ edge is empty. Let z denote the start vertex for j and v the end vertex, $z \in \mathcal{B}$, $v \in \mathcal{A}$. Then there exists an augmenting s,v–path V_2, and if j was non–empty, j could be added to V_2 and we would have an augmenting s,z–path, and then $z \in \mathcal{A}$, a contradiction.

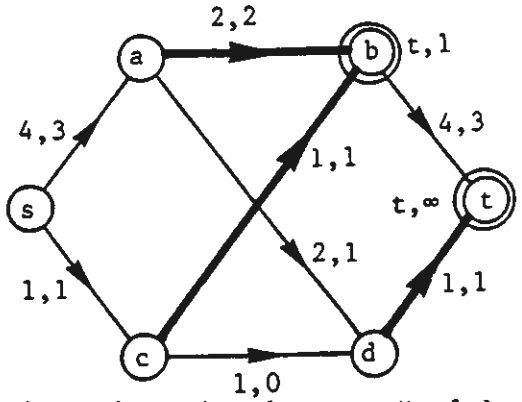
From 2 and lemma 17.1, it follows that the value F of f is the sum of the flows in the $\mathcal{A} \to \mathcal{B}$ – edges. From 1 it then follows that F is the sum of the capacities in the $\mathcal{A} \to \mathcal{B}$ – edges, i.e.

$$F = c(\mathcal{S}) ,$$

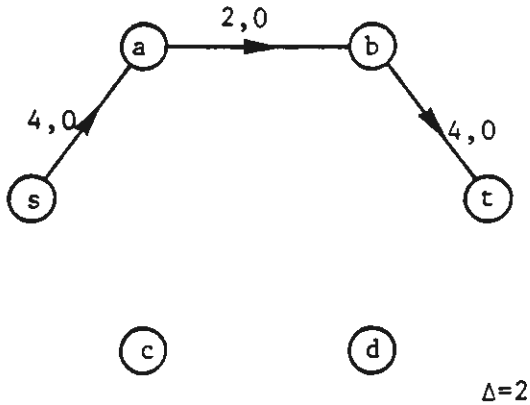
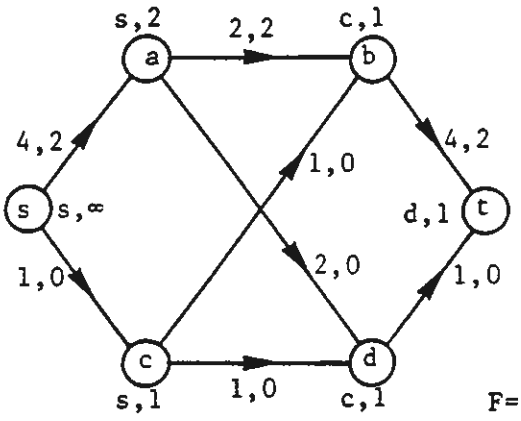
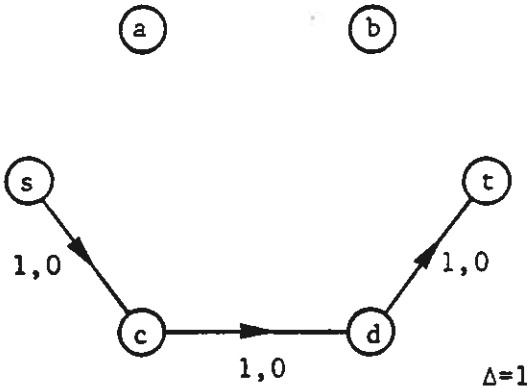
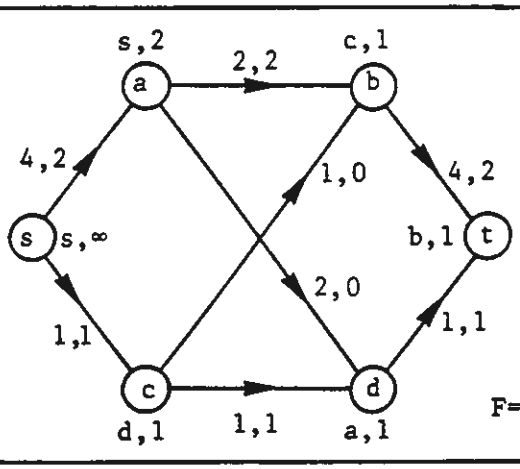
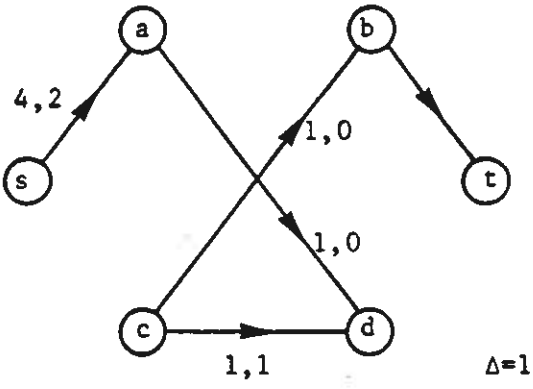
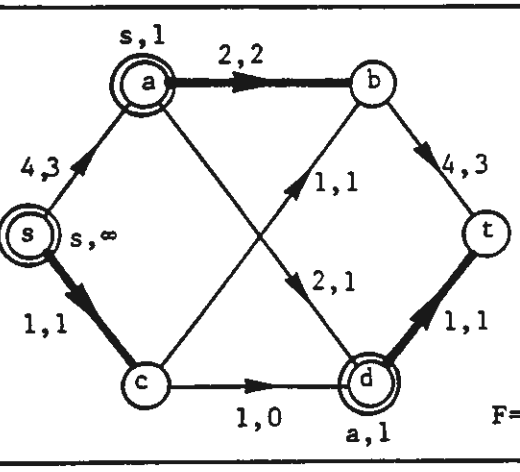
where $\mathcal{S}$ denotes the cut $\mathcal{S} = (\mathcal{A}, \mathcal{B})$. From corollary 18.4, it now follows, that f is a maximal flow and that $\mathcal{S}$ is a minimal cut. Thus the theorems are proved. □

The next page illustates the way the algorithm works, with an example. The way of thinking in the above proof is typical for many proofs in discrete optimisation. It is therefore recommended that the reader make himself/herself completely familiar with the proof.

When the algorithm has stopped, the set $\mathcal{A}$ is constructed, and therefore the minimal cut is $\mathcal{S} = (\mathcal{A}, \mathcal{B})$. Therefore, the algorithm could also be called "algorithm minimal cut". In the last of the figures on page xxx, with a double circle we show the set $\mathcal{A}$ of those vertices p for which there exists an augmenting s,p–path. The edges in the corresponding minimal cut are drawn in heavy lines in the figure.



If, instead of the set $\mathcal{A}$, we determine the set $\mathcal{M}$ of those vertices q for which there exists an augmenting q,t–path, then the edge set $\mathcal{S} = (\mathcal{P}(N)\setminus\mathcal{M}, \mathcal{M})$, consisting of those edges beginning outside $\mathcal{M}$ and ending inside $\mathcal{M}$, is also a minimal cut.This can be seen by changing all arrow directions in N and changing the names s and t. In the figure, we show this "labelling out from t", carried out for the flow we have determined in the example on page xxx. Observe that we havce determined two different minimal cuts.

FRUGTBARE VEJE	STRØMNINGER MARKERINGER
	
	
	

We shall now investigate whether the maximal flow algorithm stops. The problem is whether it can happen that the algorithm determines an infinite sequence V_1, V_2, V_3, ... of augmenting paths with the values ΔV_1, ΔV_2, ΔV_3, ... Since the set of flow values are bounded above, the infinite series $\Delta V_1 + \Delta V_2 + \Delta V_3 + ...$ must be convergent. That a thing like this can actually happen, we shall illustrate with a – necessarily somewhat complicated – example.

We shall use the number

$$r = \tfrac{1}{2}\,(\sqrt{5} - 1) = 0.618... \ .$$

We put $a_n = r^n$, $n = 0, 1, 2, 3, ...$. The sequence $a_0, a_1, a_2, a_3, a_4, ...$ will be

$$1,\ 0.618...\ ,\ 0.382...\ ,\ 0.236...\ ,\ 0.146...\ ,\ ...$$

A little calculation shows that

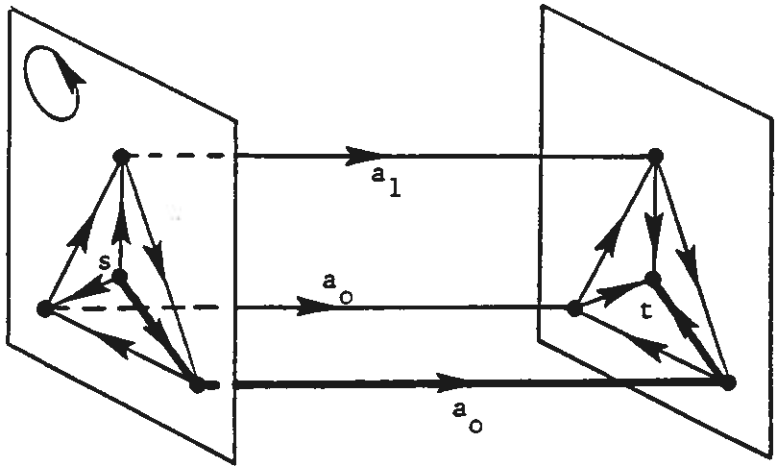
$$a_n = a_{n+1} + a_{n+2} \ .$$

The infinite series $a_0 + a_1 + a_2 + ...$ is a quotient series with the sum

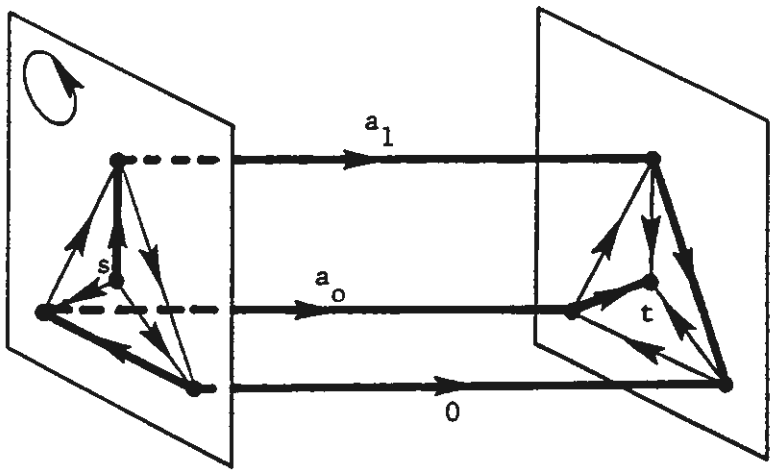
$$S = a_0 + a_1 + a_2 + ... = \frac{1}{(1-\frac{1}{2}(\sqrt{5}-1))} = \tfrac{1}{2}(3 + \sqrt{5}) = 2.618...$$

Consider now the network in the figure below. It contains 3 edges with capacities a_0 , a_0 and a_1 . Beside these edges, there are 12 edges all with the capacity $S = 2.618$ ...

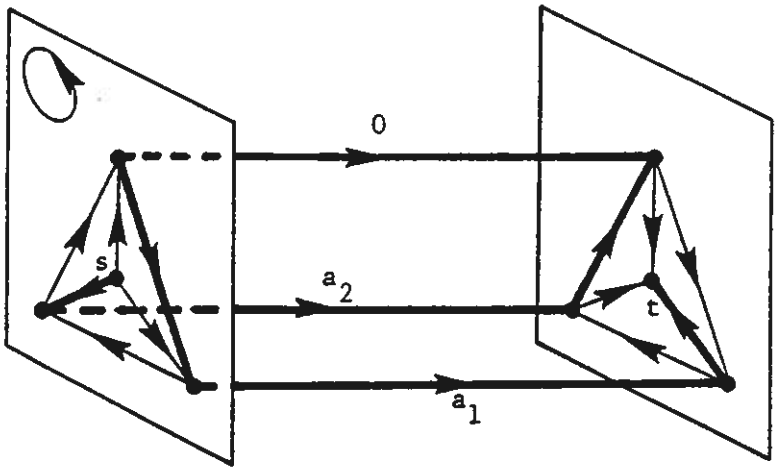
We now use the augmenting path with a value a_0 , shown with heavy lines, and obtain the rest – capacities shown in the figure at the top of page xxx (the rest – capacity is the capacity minus the flow).



Here, we can use the augmenting path, shown with heavy lines, with value a_1 , and we obtain the rest – capacities shown in the figure below, and the new augmenting path with value a_2 .

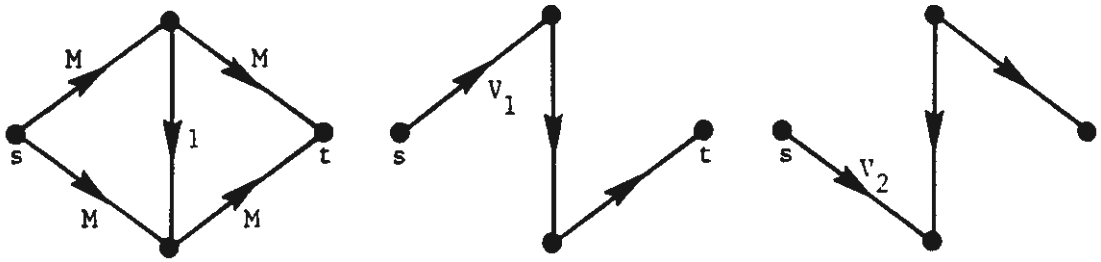


In this way we can continue infinitely. In each step, the figure will be turned throough 120° in the direction of the arrow and 1 is added to the index of the two rest – capacities, and to the index of the value of the paths.



We can conclude that the algorithm MAXIMAL FLOW *does not always stop*. When the algorithm does not stop, there is no reason to believe that the value of the constructed flow will converge towards to the value of the maximal flow. This is because there can easily be some undetected augmenting paths.

In general, when we use a computer, we use rational approximations for real numbers. Therefore, it is not very important that the algorithm does not stop when the capacities are irrational numbers. A more serious fault of the algorithm is that in some cases it is very slow. The figure below shows an example of this type.



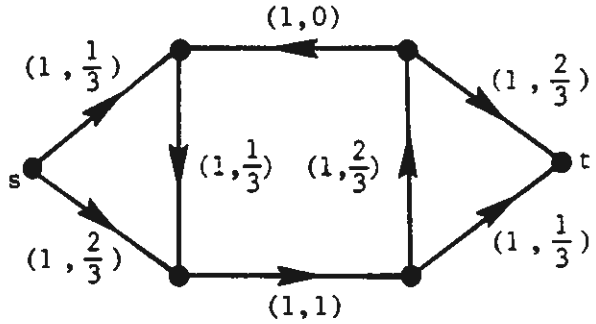
If we use the algorithm such that we use V_1 and V_2 alternately as augmenting paths, then the flow value will increase by 1 in each step, and therefore we need 2M steps before the algorithm stops. But a reasonable algorithm should fill this network up in 2 steps. In this case, the number of steps in the algorithm is dependent upon the values of the capacities. We would like a version of the algorithm such that there is an upper bound for the number of steps, which is alone determined by the graph theoretical structure of the network, and which is independent of the capacities. In section 21 we shall describe such a version of the algorithm.

20. THE INTEGER VALUE THEOREM AND LINEAR PROGRAMMING The theorem we shall show in this section can seem to be trivial and not very useful. Of course, since the theorem is described here, it is not surprising that this negative attitude will be denied and that we shall try to describe why the integer value theorem is interesting both from the theoretical and the practical point of view.

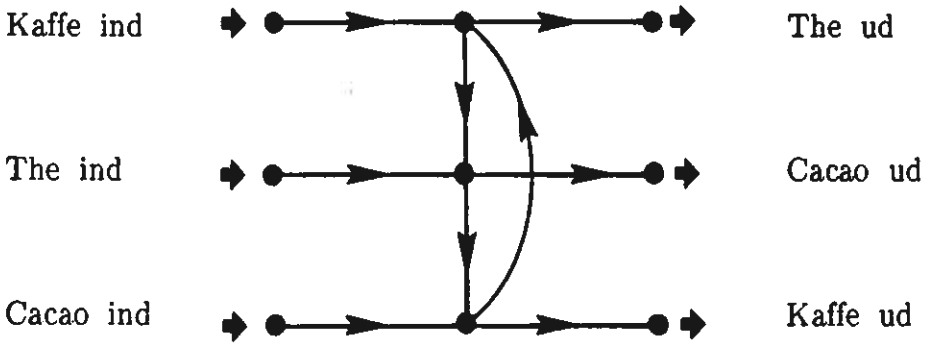
When we use the maximal flow algorithm in the network with integer valued capacities and when we start with zero flow, then in each step we change the flow in each edge by an integer. The resulting maximal flow therefore has integer valued flow in each edge. Therefore we have shown:

The Integer Value Theorem When all capacities in a transport network are integers then there exists a maximal flow, such that, the flow in each edge is an integer.

From a mathematical point of view, this theorem is interesting and has a large content. That the proof is so simple is only due to the fact that we have prepared the proof when proving the fact that the algorithm works. It is easy to give an example of a network with integer valued capacities which has a maximal flow containing non–integer valued flows. An example is shown in the diagram.



But for non–mathematicians it might be easier to understand why the integer value theorem in fact has a content by working with the example below. All capacities are 1 , and the 3 commodities (coffee, tea and cocoa) are going to flow in the edges such that the total flow is as large as possible without any capacity being exceeded.



This last example is taken from the theory on multiple–commodity flow. This theory is not developed as completely as the theory on the usual transport network. Only the theory about non–directed 2–commodity networks is well established, see section 28. Furthermore,a simple theory for integer valued multi–commodity flows probably doesn't exist.

The central problem in this chapter is to find a maximal flow in a transport network. We conclude this section by giving a new formulation of this problem, a formulation which tells how another proof of the integer value theorem could be given.

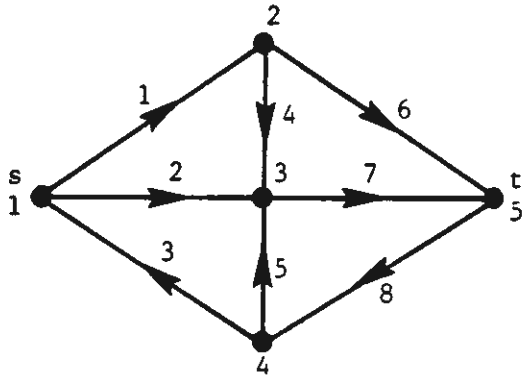
Consider a transport network, N , the vertices of which are labelled 1(s), 2, 3,..., P(t), and whose edges are labelled 1, 2,..., K . J c f_k denotes the flow in edge k , the maximum flow problem can be formulated as follows: Find the maximum value for the variable F under the assumption that

$$(1) \qquad \mathbf{A}\begin{bmatrix} f_1 \\ f_2 \\ \cdot \\ \cdot \\ \cdot \\ f_K \end{bmatrix} = \begin{bmatrix} F \\ 0 \\ 0 \\ \cdot \\ \cdot \\ -F \end{bmatrix}$$

$$(2) \qquad 0 \le f_k \le c_k \quad \text{for} \quad k = 1, 2, \ldots, K .$$

Here, $\mathbf{A}$ denotes the incidence matrix of the network that is $P \times K$ – Matrix $\mathbf{A} = (a_{rs})$ which is defined by

$$a_{rs} = \begin{cases} +1 & \text{when the edge number } s \text{ begins in vertex number } r . \\ -1 & \text{when the edge number } s \text{ ends in vertex number } r . \\ 0 & \text{when the edge number } s \text{ is not incident with vertex number } r . \end{cases}$$



As an example we consider the network N in the figure, where the numbers describe the labelling of the vertices and edges and where the capacity of edge number k is c_k . The flow maximum problem for N can be formulated as follows.

Find the maximum for F under the assumption that

$$(1) \qquad \begin{bmatrix} 1 & 1 & -1 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & -1 & 0 & -1 & -1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 & 0 & -1 & -1 & 1 \end{bmatrix} \begin{bmatrix} f_1 \\ f_2 \\ f_3 \\ f_4 \\ f_5 \\ f_6 \\ f_7 \\ f_8 \end{bmatrix} = \begin{bmatrix} F \\ 0 \\ 0 \\ 0 \\ -F \end{bmatrix} , \text{ and}$$

$$(2) \qquad 0 \le f_k \le c_k \quad \text{for} \quad k = 1, 2, \ldots, 8 .$$

If you think about it for a moment, you will understand that if we add all the equations in (1), then we obtain the equation $0 = 0$. Every one of the equations in (1) is therefore a consequence of the other equations. We have used this earlier since the

last equation in (1) does not occur in the definition of a flow; this equation was proved on page 3.5.

The formulation which we have given of the maximum flow problem above shows that this problem is a special case of linear programming (LP) . In general, the optimum for an LP problem will not be integer valued even though all coefficients (a_{ij}'s and c_i's) are integers. In the case of maximum flows in the network, the optimum will be attained in an integer valued point. The reason for this is, firstly, there occurs only one variable in each of the relations (2), and secondly the matrix **A** , as we shall see later on page XXX is totally unimodular, that is every sub–determinant in **A** has the value 0, +1 or −1 . – We shall not proceed any further with this topic. There are some very interesting relations between linear programming, integer programming and transport networks. Interested readers can consult reference [8].

21. EDMONDS–KARPS ALGORITHM. In the book, by Ford and Fulkerson, on transport networks [7], which is the first large publication on this subject, the authors stress the fact that the algorithm certain cases, does not stop and that in other cases it runs very slowly. The version of the algorithm, in which the above mentioned difficulties are overcome, was given by Edmonds and Karp. In 1968 they proved the following theorem, which wasn't published until 1972, [3].

Edmonds– Karps theorem Let N be a transport network. If algorithm MAXIMAL FLOW is used in a way such that each time we used line 4 we used the shortest augmenting path with respect to f , then the algorithm will stop after having used at most, $|\mathcal{P}(N)| \cdot |\mathcal{K}(N)|$ augmenting paths.

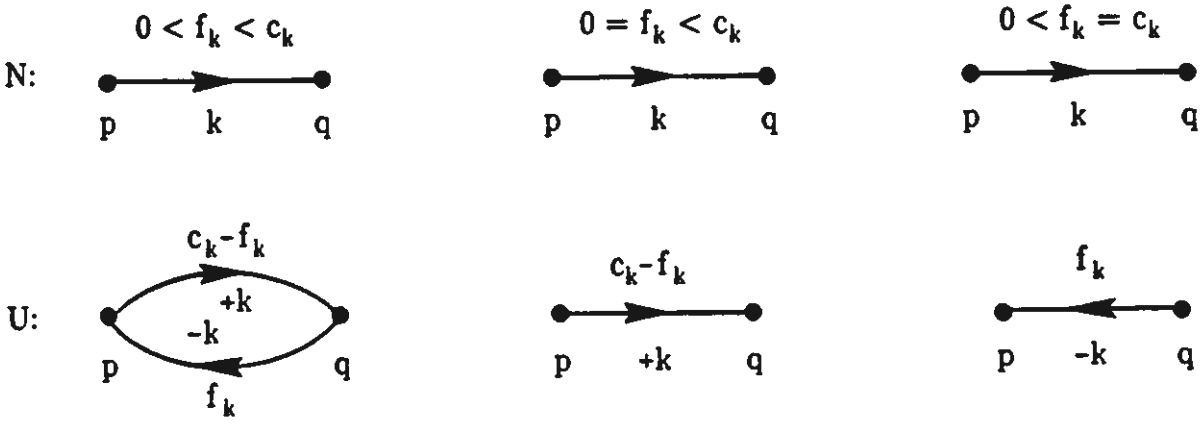
Remark: This nice theorem states that the algorithm always (for arbitrary real capacities) stops and that the number of augmenting paths which are used are bounded above by a number which does not depend on the capacities.

Proof of Edmonds– Karps theorem Let V_1 be an augmenting path (with respect to f_1), which is used in line 4 and let V_2 be the augmenting path (with respect to f_2), which is used next time we go execute line 4. Then we will show that

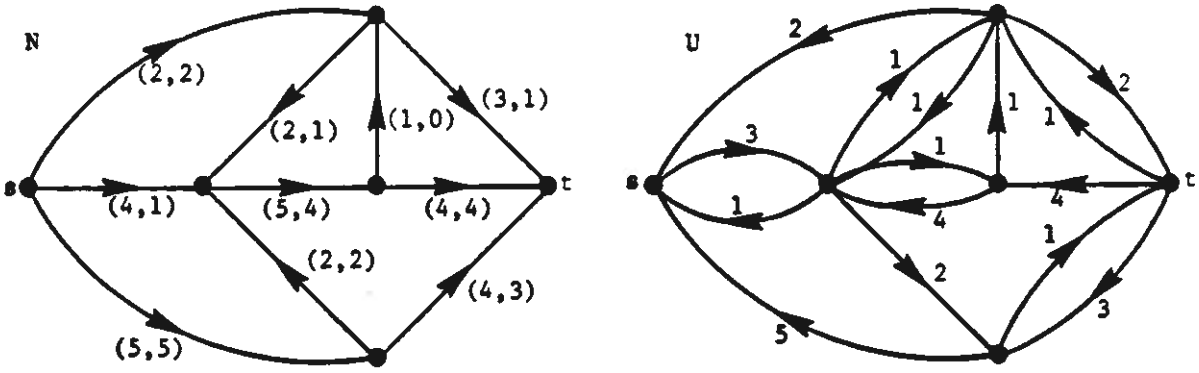
(1) $$\text{length } V_2 \geq \text{length } V_1 .$$

To be able to formulate a clear proof of (1) we shall introduce the concept extension network, which we shall also use in the next section.

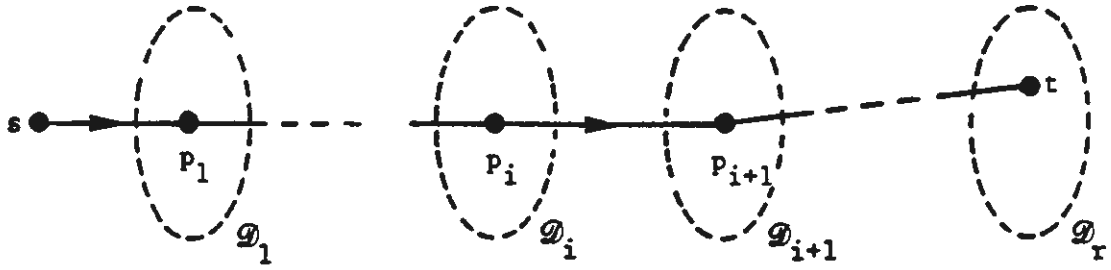
Let N be a transport network and let f be a flow in N. The extension network, $U = U(N, f)$ for N with respect to f, is defined as the network which is obtained from N in the following way: Consider an arbitrary $p \to q$ edge, k, in N with capacity c_k and flow f_k. If $0 < f_k < c_k$ we replace k with two edges, a $p \to q$ edge



with capacity $c_k - f_k$ and a $q \to p$ edge with capacity f_k. If $0 = f_k < c_k$, we only use the first mentioned edge and if $0 < f_k = c_k$, only the last mentioned edge. When this is done for all edges in N, we have constructed the extension network, U. Clearly, corresponding to any augmenting $s \to t$ path V in N with value Δ, there exists an $s \to t$ path W in U. The construction is illustrated in the diagram.

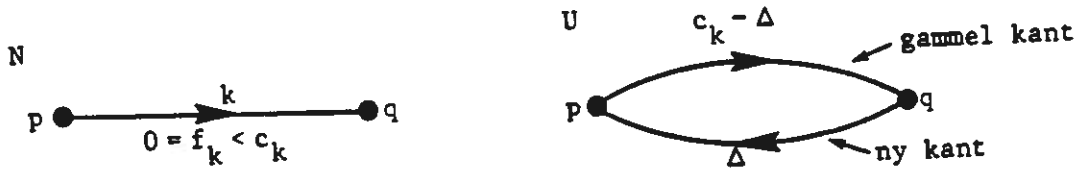


Now let $\mathscr{D}_0$, $\mathscr{D}_1$, $\mathscr{D}_2$,... denote the (directed) distance classes in U with respect to f_1 from s. So $\mathscr{D}_i$ is the set of all those vertices, p, in U for which the length of a shortest $s \to p$ path in U has length i. (In N this corresponds to the length of a shortest augmenting s, p path.) We shall investigate which changes will occur in U when we use the augmenting path V_1. If V_1 is a $(s, p_1, p_2, \ldots, p_r)$ path, $p_r = t$, then $p_i \in \mathscr{D}_i$ for $i = 1, 2, 3, \ldots, r$

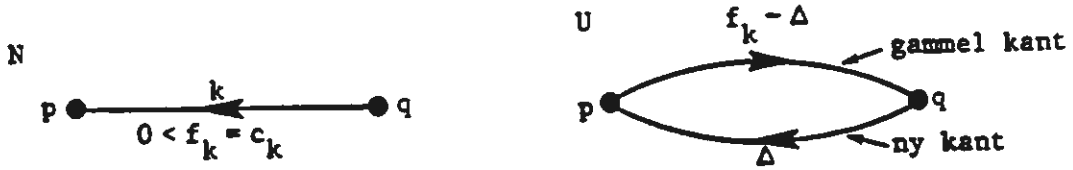


When V_1 is used, two things happen: First when an edge becomes saturated or empty, an edge will disappear from U , and secondly new edges can be created in U :

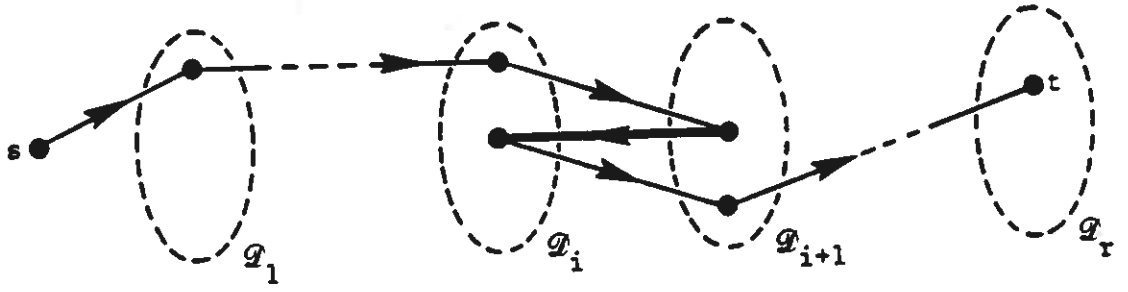
1°. If a $p \to q$ edge k , in N , is empty in f_1 and by applying V_1 , we send a flow Δ in k , then we create the new edge in U with capacity Δ which is shown in the figure.



2°. If a $p \to q$ edge k , in N is saturated in f and if by applying V we subtract flow Δ from f_k , then we create the new edge in U with capacity Δ , as shown in the diagram.



In U , p belongs to a set $\mathscr{D}_i$ and q belongs to $\mathscr{D}_{i+1}$ and son this case, we, again, create a new $\mathscr{D}_{i+1} \to \mathscr{D}_i$ edge and it is easy to see that such a new edge can have the consequence that in U we create a new $s \to t$ path of length $d(s, t) + 2$, but no shorter $s \to t$ path (cf. the figure). This proves (1).



We shall now prove that we do not use more than $|\mathcal{P}(N)| \cdot |\mathcal{K}(N)|$ augmenting paths.

Let

(2) $$V_1, V_2, \ldots, V_n$$

be the augmenting paths that we use. The length of these paths constitutes, according to (1), a non–decreasing sequence of numbers.

Let

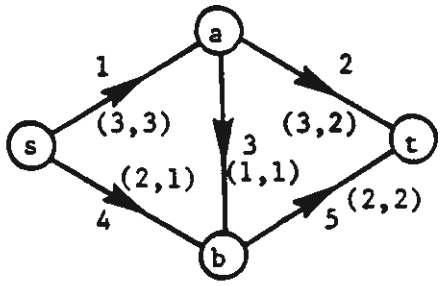
(3) $$V_i, V_{i+1}, V_{i+2}, \ldots, V_{i+n}$$

be a sub–sequence of paths with the same length m. Then $n \leq |\mathcal{K}(N)|$. Thyis follows from the fact that each path V contains at least one "bottleneck" k (an edge k we either saturate or make empty) and this edge will have it's direction changed in U when we use V. So when k is a bottleneck belonging to one of the paths in (3) k cannot belong to any other path in (3). It follows that there are at most $|\mathcal{K}(\Gamma)|$ paths in (3). In (2) the path lengths can only be constant through $|\mathcal{K}(N)|$ paths and since there can be at most $|\mathcal{P}(N)|$ different path lengths n, in (2), it follows that $n \leq |\mathcal{P}(N)| \cdot |\mathcal{K}(N)|$. □

You could fear that to obtain this good upper bound for the number of times we use line 4, which is mentioned in Edmonds–Karps theorem you had to pay a large price: Namely the trouble of finding the shortest augmenting path. However it turns out that the most natural way to carry out the MARKER procedure automatically leads to a shortest augmenting path. If you perform the labelling process by breadth first search in the extension network then each time you use line 4 in MAXIMAL FLOW we would use a shortest augmenting path. The algorithm occuring in this way is called Edmonds–Karps algorithm and we shall now describe this algorithm in detail.

As we have mentioned the MAXIMAL FLOW algorithm does not always stop. Therefore we cannot talk about the complexity of this algorithm. according to the Edmonds–Karps theorem, the Edmonds–Karps algorithm will always stop, therefore it is possible to calculate the complexity of the algoritm. Before we do that we shall choose a suitable way to represent a transport network N with a flow f. The edges of N are denoted by the numbers $1, 2, 3, \ldots, K$. N is represented by a set of neighbour lists in such a way that a $p \to q$ edge k is contained once in the neighbour

list of p and once in the neighbour list of q . With + and – we indicate that k starts in p and is constructed in q or conversely. The capacities and flow values are contained in two k–dimensional vectors c and f . The figure below should make it clear how the representation is carried out.



L(s) : (+, 1, a), (+, 4, b) .
L(a) : (–, 1, s), (+, 3, b), (+, 2, t) .
L(b) : (+, 5, t), (–, 3, a), (–, 4, s) .
L(t) : (–, 2, a), (–, 5, b)

c = (3, 3, 1, 2, 2) .
f = (3, 2, 1, 1, 2) .

Such as the procedure MARKER is described on page XXX, the edge set of N can be searched in "any order". We now define a new procedure MARKER WITH QUEUE, which carries out breadth first search in the extension network and thereby we can be sure that we determine a shortest augmenting path. Since in this case we need more information thatn we did in BREADTH FIRST SEARCH the label of the vertex has 4 coordinates denoted $M(p) = (\text{sign}(p), k(p), \pi(p), \Delta(p))$. Here $\pi(p)$ is the vertex the labelling came from, $k(p)$ is the name of the last edge we used, $\Delta(p)$ is the possible increase of the flow and sign (p) is + or – according to whether k was used with or against the arrow. M(s) is initialised as $(s, *, *, \infty)$. (don't have latin accents)

procedure: MARKER WITH QUEUE(f, s) .

1. **Put** $M(p) = *$(don't have this one!!) for all $p \neq s$ and put $Q = (s)$.
2. **while** Q is not empty, let p denote the first element in Q
3. Search L(p) once. For each element (sign, k, q) in L(P) ,
4. **if** $M(q) = *$,
5. **if** sign = + and $f_k < c_k$,
6. put $M(q) = (+, k, p, \min\{\Delta p, c_k - f_k\})$,
7. **if** sign = – and $f_k > 0$,
8. put $M(q) = (-, k, p, \min\{\Delta p, f_k\})$, and put q last in Q .
9. Delete the first element from Q .
10. When Q is empty, let $\mathcal{A}$ be the set of labelled vertices and let $\mathcal{B}$ be the set of unlabelled vertices.

This procedure is an extension of BREADTH FIRST SEARCH applied to the extension network. It is also and extension to MARKER (f, s) . Therefore t will be labelled if and only if there exists an augmenting s, t path in N and the augmenting s, t path which MARKER WITH QUEUE determines, when t has been labelled, is a shortest augmenting s, t path. The labellings can be stored in a $4 \times P$ matrix M . If for instance MARKER WITH QUEUE is applied to the example on page XXX we end up with the following labellings.

$$M = \begin{array}{c} \begin{matrix} s & a & b & t \end{matrix} \\ \begin{bmatrix} s & - & + & + \\ \emptyset & 3 & 4 & 2 \\ \emptyset & b & s & a \\ \infty & 1 & 1 & 1 \end{bmatrix} \end{array} .$$

By going backwards from t we find the augmenting path t, 2, a, 3, b, 4, s with value 1 .

When this new labelling procedure is substituted into the maximal flow algorithm we obtain

---

EDMONDS–KARPS **algorithm.**

Input: A transport network N with source s , terminal t and capacity $c_k \geq 0$ for each edge k .

Output: A flow f . A cut $\mathscr{S} = (\mathscr{A}, \mathscr{B})$.

Variables: p and q are vertices, k an edge, V a path. M(q) = (sign, k, p, Δq) is a labelling of the vertex q . sign is variable and can only take the values + and − . Q is a is a queue of vertices.

Initializing: f is the zero flow. M(s) = (s, *, *, ∞) .

1. MARKER WITH QUEUE(f, s).
2. **if** $t \in \mathscr{A}$,
3. use the augmenting $s \rightarrow t$ path, V , determined by the labellings such that you obtain a new flow f with a larger value.
4. **goto** line 1 .
5. **else**, if $t \notin \mathscr{A}$, STOP.

Theorem 21.1. EDMONDS–KARPS algorithm will always stop. In the output, f is a maximal flow and $\mathscr{S} = (\mathscr{A}, \mathscr{B})$ is a minimal cut. The complexity of the algorithm is $O(K^2 P)$.

Proof. That the algorithm stops follows from Edmonds–Karps theorem. Since Edmonds–Karps algorithm is a special way to carry out the MAXIMAL FLOW algorithm, f is a maximal flow and $\mathscr{S}$ is a minimal cut. Finally the complexity: First we shall determine the complexity of MARKER WITH QUEUE carried out once: Each vertex, p, will come into the queue, at most once and just before p is taken out of Q we shall search L(p) once. The treatment of each element in L(p) is one step. The work to search all the neighbour lists is, therefore, O(K) and since the work by moving vertices in and out of Q is O(P), the complexity of MARKER WITH QUEUE will be O(K).

The complexity of one execution of line 3 in the algorithm is O(P) and, therefore, the overall complexity of one execution of the large loop in the algorithm is $O(K) + O(P) = O(K)$. According to Edmonds–Karps theorem the loop is carried out, at most PK times and, therefore the overall complexity will be $O(K^2P)$.

□

22. **DINICS ALGORITHM.** The previous section concluded by finding that there exists an $O(K^2 P)$–algorithm, which finds a maximal flow and a minimal cut in a transport network with P vertices and K edges. Naturally one wonders if there exists an even better algorithm (and there does). Of the results obtained so far, we emphasize Dinic's algorithm, [1], with complexity $O(KP^2)$, which for dense networks (with $O(K) = O(P^2)$) is better than the EDMONDS–KARPS algorithm. We will also describe a variant with better complexity than Dinic's original algorithm. This improvement is due to Malhora, Pramodh Kumar and Maheshwari, [12], and has complexity $O(P^3)$.

Before a precise description of Dinic's algorithm is possible, we must first describe what is called a *minimal network*.

So, let N be a transport network represented by neighbour–lists L(p), $p \in \mathcal{P}(N)$ – as described at the foot of p.xxx – and let f be a flow in N . Corresponding to N and f we will define a data structure for the extension network U = U(N,f) and define the minimal network M = M(U) .

Dinic's algorithm, like the MPM–algorithm, uses the networks U and M .

We will represent U in the same way as N ; but the f–vector is omitted, as we shall not work with flows in U , and the c–vector will be 2K–dimensional, with zeroes in the places corresponding to non–existent edges. The neighbour–lists for U can be constructed in the following way: The neighbour–lists in the representation of N are searched once, and when we meet (+, k, q) in L(p) , we write:

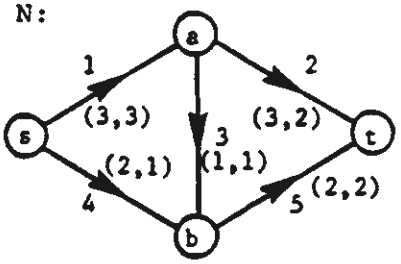
when $c_k > f_k$:
(+, +k, q) in $L_U(p)$,
(–, +k, p) in $L_U(q)$,
$c_k - f_k$ in the (2k–1)th entry in the c–vector.

when $f_k > 0$:
(–, –k, q) in $L_U(p)$,
(+, –k, p) in $L_U(q)$,
f_k in the (2k)th entry in the c–vector.

In this way one obtains the representation of U in O(K) steps.

The figure below shows an example.

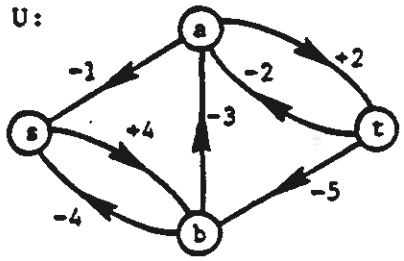
N:



L(s) : (+, 1, a), (+, 4, b) .
L(a) : (–, 1, s), (+, 3, b), (+, 2, t) .
L(b) : (+, 5, t), (–, 3, a), (–, 4, s) .
L(t) : (–, 2, a), (–, 5, b)

c = (3, 3, 1, 2, 2) .
f = (3, 2, 1, 1, 2) .

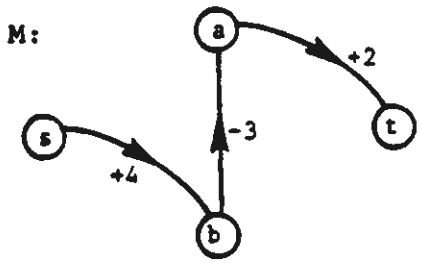
U:



$L_U(s)$: (–, –1, a), (+, +4, b), (–, –4, b) .
$L_U(a)$: (+, –1, s), (–, –3, b), (+, +2, t), (–, –2, t).
$L_U(b)$: (+, –4, s), (–, +4, s), (+, –4, a), (–, –2, t).
$L_U(t)$: (–, +2, a), (–, –2; a), (+, –5, b) .

c = (0, 3 ; +1, –1, +2, ··· 1, 2 ; 0, 1 ; 1, 1 ; 0, 2) .

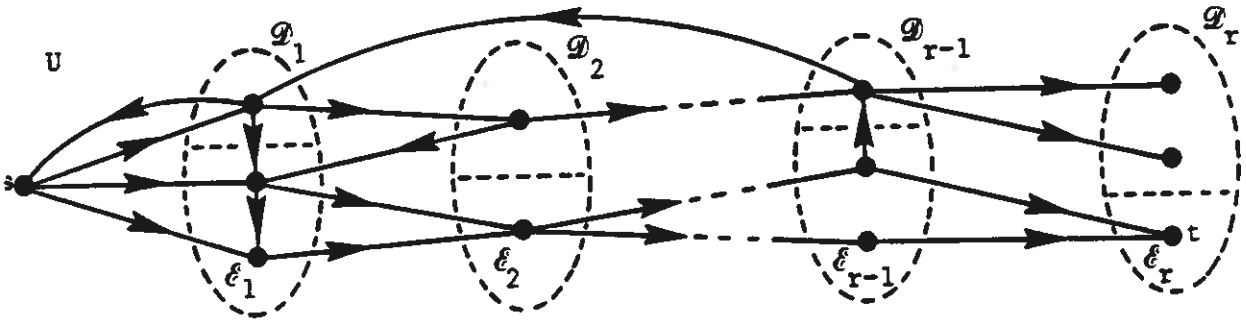
M:



$L_M(s)$: (+4, b)
$L_M(b)$: (–3, a)
$L_M(a)$: (+2, t)

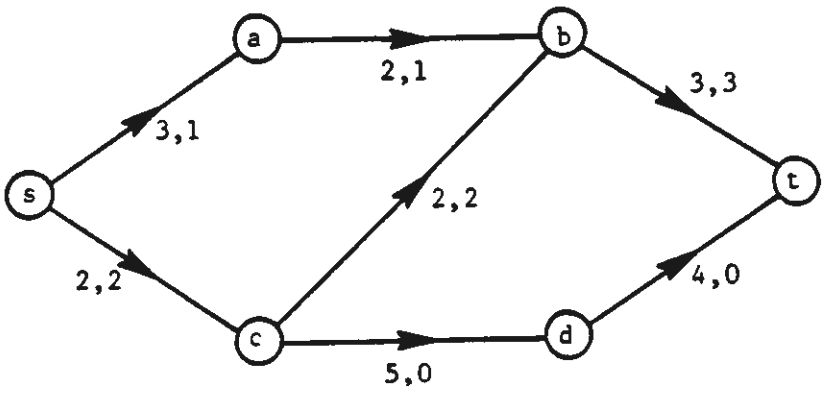
c_M = (0, 0; 1, 0; 0, 1; 1, 0; 0, 0)
f_M = (0, 0, 0, 0, 0)

The *minimal network* M is obtained from U by deleting all vertices and edges which do not belong to any shortest $s \rightarrow t$–path in U. The concepts are illustrated in the following diagrams, where the numbers at the edges are edge–numbers. Remark that U is represented by neighbour–lists in which each edge is represented twice, while M is represented by neighbour–lists where each edge is represented only once. M can be determined from U in the following way. First, the vertex–sets $\mathscr{D}_0, \mathscr{D}_1, \ldots, \mathscr{D}_r$ are determined, where $\mathscr{D}_i$ consists of the vertices p in U, for which a shortest $s \rightarrow p$–path in U has length i, and where $t \in \mathscr{D}_r$ (if such an r does not exist, f is a maximal flow). These vertex–sets can be found by breadth–first–search in U, in O(K) steps. It is clear that $\mathscr{P}(M) \subseteq \mathscr{D}_0 \cup \mathscr{D}_1 \cup \ldots \cup \mathscr{D}_r$.



Then the vertex–sets $\mathscr{E}_r = \{t\}, \ldots, \mathscr{E}_0 = \{s\}$ are determined, where $\mathscr{E}_i$ is the set of those vertices p in $\mathscr{D}_i$ for which there exists a $p \rightarrow t$–path in U of length $r - i$. The $\mathscr{E}$–sets can be found in O(K) steps by using breath–first–search against the arrows. Then $\mathscr{P}(M) = \mathscr{E}_0 \cup \mathscr{E}_1 \cup \ldots \cup \mathscr{E}_r$. $\mathscr{K}(M)$ can be determined simultaneously, since a $p \rightarrow q$–edge k in U belongs to $\mathscr{K}(M)$ if and only if there exists an i such that $p \in \mathscr{E}_i$ and $q \in \mathscr{E}_{i+1}$. The set $\mathscr{E}_i$ is called the *i–th layer* in M. During this process, one can establish a representation of M, and a 2K–dimensional capacity–vector c_m. The K–dimensional flow–vector f_m for M is initialised as 0. Consequently the representation of M can be determined in O(K) steps.

Now let M denote a minimal network. A flow f_M in M is called *complete* when every $s \rightarrow t$–path in M contains a saturated edge. Every maximal flow is obviously complete, but the converse is not true. The figure shows a complete flow that is not maximal.



It is easy to find a complete flow in M . One need only send flow in $s \to t$–paths until every $s \to t$–path contains a saturated edge. It is easy to see that every complete flow can be generated in this way. (cf. p.xxx)

We now consider an algorithm which we call the UMK–algorithm, because it uses the sequence extension network (U) , minimal network (M) , complete flow (K) .

algorithm: The UMK–ALGORITHM.

Input:	A transport network N with source s , terminus t , and a capacity $c_k \geq 0$ for every edge k .
Output:	A flow f .
Variables:	An extension network U , a minimal network M , and a flow f_M in M .
Initialisation:	f is the zero flow.

1. Find the extension network U w.r.t. f.
2. **if** U does not contain an $s \to t$–path , STOP.
3. Find the minimal network M w.r.t. f.
4. Find a complete flow f_M in M .
5. $f := f + f_M$.
6. **go to** 1 .

Theorem 22.1. When line 4 in the UMK–ALGORITHM is executed with an algorithm with complexity A , the algorithm stops, the output is a maximal flow, and the complexity is $O((A+K)P)$.

Proof. Let M_1 and M_2 be two minimal networks constructed in consecutive executions of line 3. Then $d_{M_2}(s,t) > d_{M_1}(s,t)$, according to formula (1) on p.xxx. Hence, it follows that the algorithm stops. When the algorithm does stop in line 2, there are no augmenting s,t–paths in N . Hence f is maximal. The loop from line 1 to line 6 can be performed at most $P = |\mathcal{P}(N)|$ times, because $d_M(s,t)$ will be at least one greater every time the loop is traversed. The individual lines have the following complexities:

1. O(K) .
2. O(K) .
3. O(K) .
4. O(A) .
5. O(K) .
6. O(1) .

From this follows the statement about the complexity. □

We conclude this section by describing three methods for executing line 4.

Method 1. EDMONDS – KARPS ALGORITHM. Repeat the following until there does not exist an $s \to t$–path in M :

(1) Find with breadth–first–search an $s \to t$–path V in M , and using V reduce the capacities and delete edges with zero capacity.

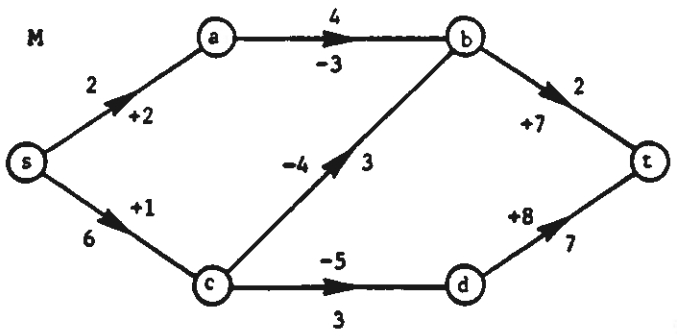
With every application of (1) at least one edge obtains capacity zero. Therefore (1) is executed at most K times, and every execution requires K steps. Therefore $A = O(K^2)$ and the total complexity will be $O(K^2P)$ (compare theorem 21.1, p.xxx).

Method 2. DINIC'S ALGORITHM. One constructs a complete flow in M by using the following procedure:

procedure: DINIC.

1. Start in s .
2. Go forwards through the layers in M , using only edges with capacity > 0 , until a vertex p is reached where no edge with capacity > 0 starts.
3. **if** $p = s$, STOP.
4. **if** $p = t$,
5. increase the value of f using the augmenting path which has been found, and reduce the capacities .
6. **otherwise** , (i.e. if $p \neq s, t$),
7. go back one step along the edge k one came from, and change the capacity of k in M to zero.
8. **go to** 2 .

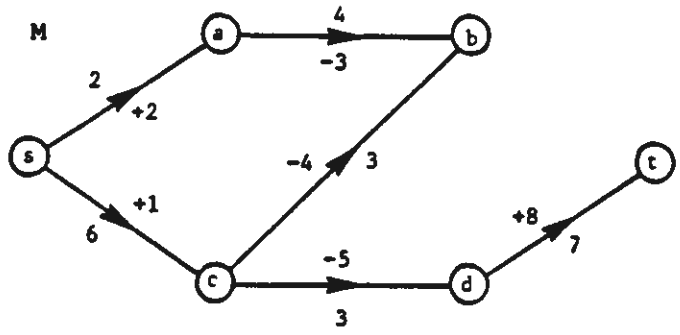
Let us illustrate the procedure by an example. In the figures, the numbers at the edges are edge numbers.



$L_M(s) : (+1, c), (+2, a)$.
$L_M(a) : (-3, b)$.
$L_M(b) : (+7, t)$.
$L_M(c) : (-4, b), (-5, d)$.
$L_M(d) : (+8, t)$.

$f_M = (0, 0, 0, 0, 0, 0, 0, 0)$

Line 2 in the procedure can be carried out in a very simple way by means of the edge–lists. Actually, it is only necessary to consider edges which are first elements in the edge–lists. First we find the path s, c, b, t with value 2. The b → t–edge is a bottleneck and we obtain on line 5:



$L_M(s) : (+1, c), (+2, a)$.
$L_M(a) : (-3, b)$.
$L_M(b) : \emptyset$.
$L_M(c) : (-4, b), (-5, d)$.
$L_M(d) : (+8, t)$.

$f_M = (2, 0, 0, -2, 0, 0, 2, 0)$

At the second execution of line 2, we find the path s, c, b, then we return to c and delete the element (–4, b) from $L_M(c)$. Finally, we determine in line 2 the path s, c, d, t with value 3 and obtain $f_M = (5, 0, 0, -2, -3, 0, 2, 3)$.

Edge 5, which is now first on $L_M(c)$, is deleted, and after 3 more passages of the loop the algorithm discovers that we have a complete flow.

Every execution of the loop (line 1 – line 8) results in at least one edge having its capacity reduced to zero. The loop can therefore be executed at most K times. The complexity of one execution of the loop is O(P) . Therefore $A = O(P \cdot K)$, and the overall complexity of Dinic's algorithm will be $O(P^2K)$, which is Dinic's result [1].

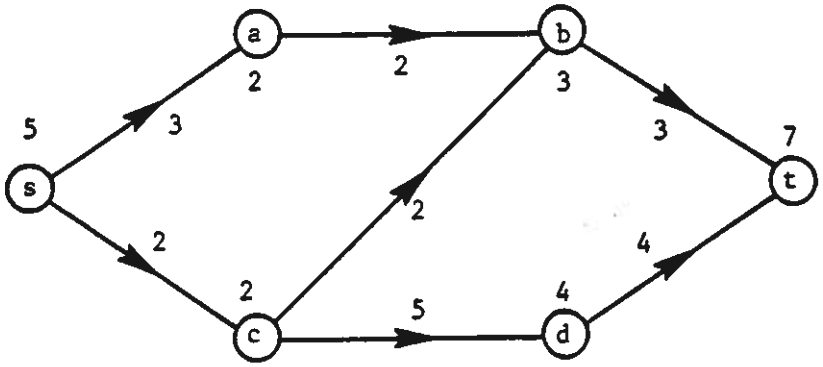
Method 3. MPM–ALGORITHM. This method, due to Malhotra, Pramodh Kumar and Maheshwari [12], needs a little more explanation. Let p be a vertex in M . We define the *in–value* $w_i(p)$ and the *out–value* $w_u(p)$ by the equations

$$w_i(p) = \sum_{k \text{ ends in } p} c_k \quad , \quad w_u(p) = \sum_{k \text{ begins in } p} c_k \quad ,$$

where $k \in \mathcal{K}(M)$. $w_i(p)$ [or $w_u(p)$] is an upper bound for how much flow in M can flow into p [or out of p]. Finally the value $w(p)$ is defined by the equation

$$w(p) = \begin{cases} \min \{w_i(p), w_u(p)\}, & p \neq s, t \ , \\ w_u(p) \quad , & p = s \quad , \\ w_i(p) \quad , & p = t \quad . \end{cases}$$

A vertex p in M is called a *bottleneck* in M when $w(p)$ has minimal value; that is, when $w(q) \geq w(p)$ for all $q \in \mathcal{P}(M)$. In the figure, the numbers by the edges give capacities, and the numbers by the vertices give $w(p)$. There are two bottlenecks in M , namely a and c .



The function $w(p)$, and thereby a bottleneck, can be determined in $O(K)$ steps.

The basic idea of the MPM–algorithm is the following:

When p is a bottleneck in M then one can increase the flow value by the amount $w(p)$, by sending a flow of total value $w(p)$ forward through the layers from p to t , and collect a corresponding flow from s by going backwards through the layers from p . Since $w(p)$ is minimal, one cannot get stuck on the way. This process is repeated until one has a complete flow.
To ensure the process works, and to keep a low complexity, there are two difficulties that must be cleared up:

1. When the process is executed once, vertices having value 0 arise. Such vertices must be deleted in M before the process is repeated. Such deletions can create new vertices with value 0 , which must also be deleted, etc.

2. If the flow value $w(p)$ is distributed in small amounts over many edges, one can (by repetition of the process) end up adding flow to the same edge a lot of times, so that the complexity will be large. One therefore aims to saturate or fill up the edges as much as possible.

The method is described in the following procedure:

procedure: MPM .

1. Calculate the value–function **w** .
2. Find a vertex p with minimal value.
3. **if** $w(p) = 0$,
4. **if** $p = s$ or $p = t$, STOP.
5. **else** delete p from M .
6. **go to** 2.
7. Assume that there is a surplus flow in p of size $w(p)$.
8. Perform breadth–first–search with the arrows out from p . Every time an $a \rightarrow b$–edge k is considered and there is surplus flow into a , saturate this edge k , if the surplus is large enough to do so. Otherwise send the remaining surplus flow out through k . When an equilibrium of flow is established, proceed immediately to the next vertex in the queue. The capacities and values are updated immediately, and the edges with capacity 0 are deleted from M .
9. Now there is a flow deficit in p of size $w(p)$.
10. Perform breadth–first–search against the arrows out from p . Every time an $a \rightarrow b$–edge k is considered and there is a flow–deficit in b , saturate this edge k , if the deficit is sufficiently big. Otherwise send the remaining flow–deficit in through k . When a flow–equilibrium is established, go on immediately to the next vertex in the queue. The capacities and values are updated immediately, and the edges with capacity 0 are deleted from M .
11. **Go to** 2.

We will prove the following three statements:

1. The procedure stops.
2. When the procedure is stopped, the total flow in M is complete.
3. The procedure has complexity $O(P^2)$.

Proof for 1. When line 10 is executed, p has value 0 . Therefore either the procedure stops in line 4, or p is deleted in line 5. When line 6 is executed, p is just deleted in line 5. One execution of the loop (line 1 – line 6) results therefore in one vertex being deleted. The procedure therefore stops.

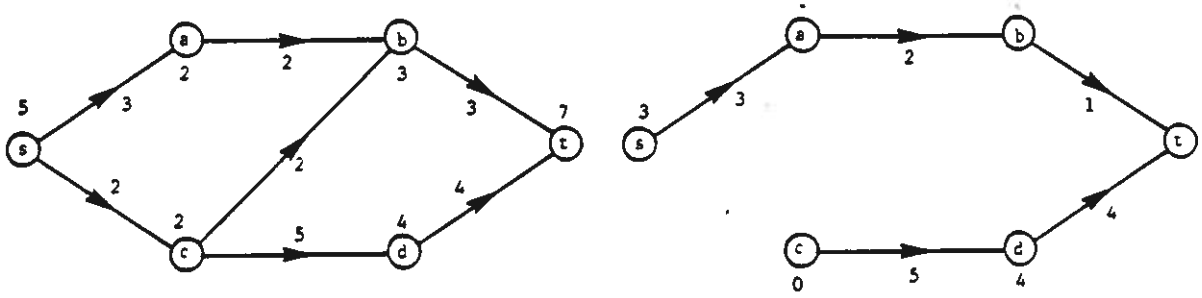
Proof for 2. When vertex p has value 0 , there exists no $s \rightarrow t$–path in M through p with value > 0 . When the procedure stops in line 4, the flow is therefore complete.

Proof for 3. If one calculates the complexity by multiplying the number of executions of the loop by the complexity of one execution, we will not get the required (correct) result. Instead we will calculate the total work carried out in each individual line by all executions of the two loops.

1. $O(K)$.
2. $O(P^2)$. Since every loop is executed at most p times.
3. $O(P)$.
4. $O(P)$.
5. $O(K)$. To delete a vertex is in fact the same as deleting the edge incident with the vertex. Since an edge can be deleted in one step in a double–list representation, line 5 requires $O(K)$ steps.
6. $O(P)$.
7. $O(P)$.
8. $O(P^2)$. Here one would expect $P \cdot K$. The argument why it is P is subtle. When an adge k is met in breadth–first–search there are two possibilities:
   1) k is made saturated.
   2) k is not made saturated but the flow is changed.
   By considering all executions of line 8, (1) can only happen K times, namely once for every edge. With one execution of line 8, (2) can occur at most P times, once for every vertex, that is P times in all. Since $P > K$, the result is $O(P^2)$.
9. 0 .
10. $O(P^2)$. Same as 8 .
11. $O(P)$.

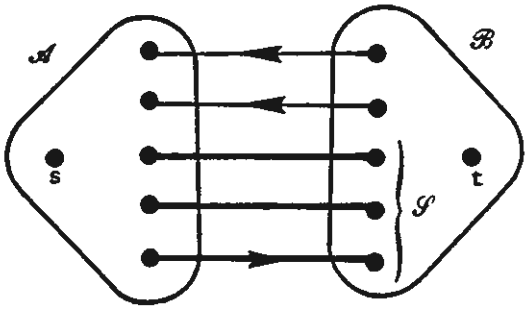
The complexity of the procedure will be the sum of these complexities, that is $O(P^2)$, since $P^2 > K$. From theorem 22.1 it follows that the complexity of the MPM–algorithm will be $O(P^3)$.

As an example of how the procedure works, we consider the network in the figure on the left. The value–function w is shown at the vertices, and we choose $p = c$ in line 2 . Since $w(c) = 2$ we send the flow through, for example, the path s, c, b, t . Hereby the network in the figure on the right is obtained with the given (updated) value–function. Next (in line 5), one deletes c , then d , and b will be the last bottleneck. When this is done, t has value 0 , and the algorithm stops in line 4 .



23. THREE TRIVIAL GENERALISATIONS. In the previous sections, we have considered only directed transport networks. However, it is easy to see that the developed theory is also valid for non–directed or mixed networks.

Let N be a non–directed (or mixed) graph, which contains two vertices, s (source) and t (terminal), and in which each edge has a capacity $c_k > 0$. A flow f in a network N is defined when, for each edge k , there is a number f_k (or f(k)) which satisfies the condition $0 \leq f_k \leq c_k$; if k is non–directed, there has to be an arrow which shows the direction in which the flow goes. Then f is called *a flow* if the total flow out of x is equal to the total flow into x for all $x \in \mathcal{P}(N)\backslash\{s,t\}$. The netflow out from s (which is equal to the netflow into t) is called the *value* of f and is denoted by $F(f) = F$. When we partition the vertices in a transport network N into two sets $\mathcal{A}$ and $\mathcal{B}$,such that $s \in \mathcal{A}$ and $t \in \mathcal{B}$, then the set of $\mathcal{A} \rightarrow \mathcal{B}$ edges united with the set of non–directed $\mathcal{A},\mathcal{B}$–edges is called a *cut* $\mathcal{S}$ in N . The Max–flow–min–cut thoerem is valid also for non–directed and mixed networks. Lemma 18.1 and lemma 19.2 are valid as well if one changes the definition of an augmenting path slightly: An s,t–path V , directed from s to s , is called an augmenting path w.r.t. f if the following 3 conditions are satisfied:



1) Each directed edge, which has the same direction as V, is unsaturated.
2) Each directed edge, which has the opposite direction w.r.t. V, has a flow greater than 0.
3) Each non–directed edge, in which there is a flow in the $s \to t$ direction, is unsaturated.

Then one can find a maximal flow in a mixed network by means of these augmenting paths. Also, Edmonds–Karps theorem remains valid.

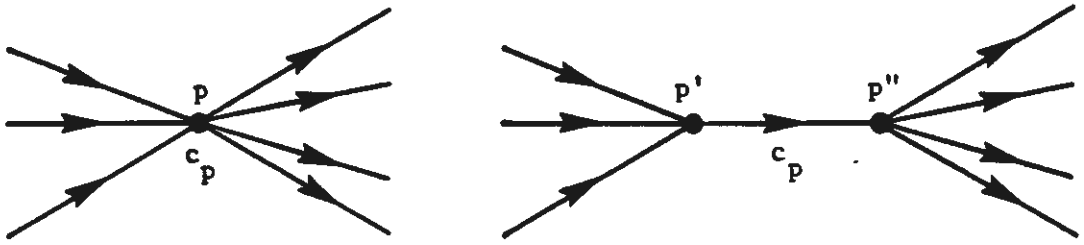
The other trivial generalisation which we shall shortly describe in this chapter is dealing with networks with several sources and several terminals.

Let N be a graph which contains a non–empty set of vertices $\mathscr{S} = \{s_1, s_2, ..., s_p\}$, the *sources*, and a non–empty set of vertices $\mathscr{T} = \{t_1, t_2, ..., t_q\}$, the *terminals*, such that $\mathscr{S} \cap \mathscr{T} = \emptyset$, and in which each edge k has a capacity $c_k > 0$. A flow f defined by a flow f_k in each edge k has to satisfy $0 \leq f_k \leq c_k$. For each vertex $p \in \mathscr{P}(N) \setminus (\mathscr{S} \cap \mathscr{T})$ the total flow into p has to be equal to the total flow out of p. The value $F = F(f)$ of the flow f is defined by the sum of netflows out of any source. One can find a maximal flow in such a network by constructing a new network N' which is obtained from N by adding two vertices s and t, the new source and new terminal respectively: connect s with all $s_i \in \mathscr{S}$ by an edge k_i with a large capacity (e.g. sum of capacities of all edges in N which are incident with s_i) and connect all $t_i \in \mathscr{T}$ with t by an edge k_i' with large capacity too. It is clear that for each flow f in N with value F there is a corresponding flow f' in N' with the same value F. It is also possible to find a maximal flow in N by applying the algorithm MAXIMAL FLOW to N'.

Finally in this section we shall consider the case where there is a bound for how much flow can pass through each vertex.

Let N be a directed transport network in which some (or all) vertices $p (p \neq s, p \neq t)$ have a capacity c_p. We consider only those flows for which the total flow out of p is $\leq c_p$ (therefore c_p is an upper bound for how much can flow through p). If we want to determine among these flows one with maximal value, we can use a network N' which is obtained from N in the following way: each vertex p with capacity c_p is replaced by two vertices p' and p'' and a $p' \to p''$ – edge with capacity c_p. Edges ending in p in N end in p' in N'. Edges in N which start in p start in p'' in N'. A corresponding method can be used in the case of a network with several sources (or terminals) if each source s_i has an upper bound for how much the netflow

is which can come out of s_i . As an example, one can think about the graph of production capacities for factories which manufacture the same article.



24. PATH FLOWS. In a transport network it is natural to consider a flow not only as something which flows in each edge separately but as something which flows along a one–way path from s to t . In this section we shall make precise this way to consider a flow.

Let N be a transportation network with a source s , a terminal t and capacities c_k . Let $\mathscr{V}$ be the set of $s \to t$ paths in N , $\mathscr{W}$ be the set of $t \to s$ paths and $\mathscr{C}$ the set of one–way circuits in N . A *path flow* g in N is defined as follows: For each element $V \in \mathscr{V}_0 = \mathscr{V} \cup \mathscr{W} \cup \mathscr{C}$ we choose a real number $g(V) \geq 0$ such that for each edge k in N we have

$$\sum_V g(V) \leq c_k , \tag{1}$$

where the summation is over all $V \in \mathscr{V}_0$ which contains the edge k .The *value* G of a path flow is defined as

$$G = \sum_{V \in \mathscr{V}} g(V) - \sum_{W \in \mathscr{W}} g(W) ,$$

i.e. netflow out of s .

When we put for each edge k

$$f_k = \sum g(V) ,$$

where the summation is again over all $V \in \mathscr{V}_0$ which contain the edge k , then (1) means that f satisfies the limit condition S2, page xxx. Since it is clear that f satisfies the equilibrium condition S1, page xxx, f is a flow in N . It is also clear that f and g have the same value. f is said to be the flow corresponding to the path flow g. Conversely, let f be a given flow in N . We shall show that there exists a

path flow g such that f corresponds to g . If there exists a $s \to t$ path V in which all edges k have $f_k > 0$ we put

$$g(V) = \min_{k \in \mathcal{K}(V)} f_k .$$

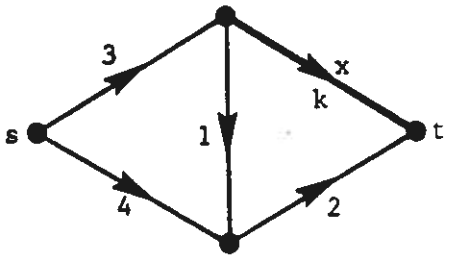
Then we form a new flow f' by subtracting $g(V)$ from the edges which belong to V . In this way, we continue until all $s \to t$ paths contain an empty edge. Having done this we treat the $t \to s$ path in the same way and we obtain a flow f_1 . Let $\mathcal{A}$ denote the set of those vertices p in N for which there exists a $s \to p$ path in N in which all edges have a flow > 0 . Since $t \in \mathcal{B} = \mathcal{P}(N) - \mathcal{A}$ and all $\mathcal{A} \to \mathcal{B}$ edges are empty, the value of f_1 must be ≤ 0 . By considering in the same way the set of those vertices p in N for which there exists an $p \to s$ path in N in which all edges have a flow > 0 we obtain that the value of f_1 is ≥ 0 . Therefore, f_1 has the value 0 . If f_1 is the zero flow we have finished. Otherwise, we consider an edge k with $f_k > 0$.

Since the equilibrium condition is satisfied now in all edges we can follow a flow > 0 around the network from the endvertex of edge k until we reach a vertex where we have already been before. Thus we have determined a one–way–circuit C in which all edges have a flow > 0 . We put $g(C) = \min_{k \in \mathcal{K}(V)} f(k)$ and form a new flow by subtracting $g(C)$ from each edge in C . In this way we can continue until reaching the zeroflow. Putting $g(V) = 0$ for an element $V \in \mathcal{V}_0$ which has not obtained any flow during this process we have defined a path flow g which is easily seen to correspond to the flow f . This completes the proof.

If we add a $t \to s$ edge with a flow $f = F(f)$ (≥ 0) the equilibrium condition will be satisfied in all vertices. The result we have just proved states that such a flow called a circulation can be described as a sum of flows in one–way circuits.

Finally, we shall remark that since flows in one–way circuits and $t \to s$ paths don't contribute to the value of the path flow, in every transport network there exists a maximal path flow in which all one–way circuits and $t \to s$ paths have a flow equal to zero. Naturally, the corresponding flow in N is also maximal.

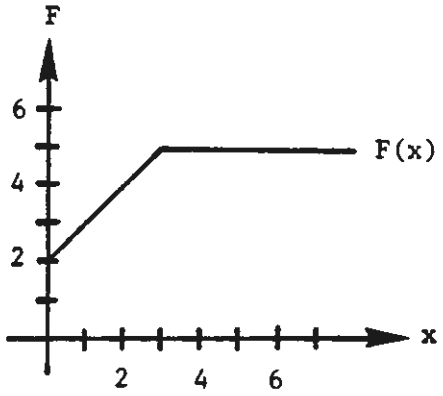
25. NETWORKS WITH AN EDGE WITH VARIABLE CAPACITY In this section we will show that the Max–flow–min–cut theorem makes it very easy to understand completely how the value of the maximal flow in a transport network will change if the capacity of one edge k is variable.



First, let us consider an example, shown in the figure on the previous page, where capacities of the edges are $1, 2, 3, 4$ and x. The smallest of the two cuts which contains the edge k has capacity $x + 2$ and the smallest of the two cuts which does not contain k has capacity 5. Then it follows from the Max–flow–min–cut theorem that the value $F(x)$ of the maximal flow is

$$F(x) = \min\{x + 2, 5\} = \begin{cases} x + 2 & \text{for } x \leq 3 \\ 5 & \text{for } x \geq 3 \end{cases}$$

Remark that for $x = 3$ there exists a minimal cut which contains k as well as a minimal cut which does not contain the edge k.



Now let us proceed to the general case. We consider an arbitrary transport network with one edge k which has a variable capacity $c_k = x$. Let the value of the maximal flow be denoted by $F(x)$. In the very simple case where k is an $s \to t$ edge we obtain $F(x) = x + F(0)$. If k ends in s or starts in t then $F(x) = F(0)$ for all x. In this case, the edge k is not contained in any cut.

In any other case, there exist both a cut which does not contain k and a cut which does contain k. Among the cuts which do not contain k there exists one which has minimal cacpacity denoted by $\mathscr{A}_0$. A cut $\mathscr{A} \cup \{k\}$, which contains k, has capacity $c(\mathscr{A} \cup \{k\}) = x + c(\mathscr{A})$, where $c(\mathscr{A})$ is the sum of the capacities of the other edges in

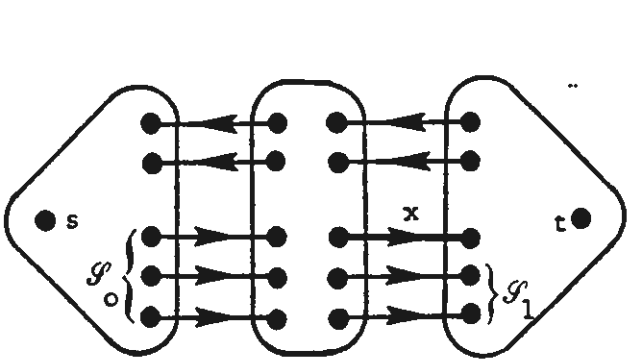
the cut. Among the cuts which contain k we now choose one with minimal capacity. This is denoted by $\mathscr{S}_1 \cup \{k\}$, its capacity is $c(\mathscr{S}_1 \cup \{k\}) = x + c(\mathscr{S}_1)$. It follows from the Max–flow–min–cut theorem that

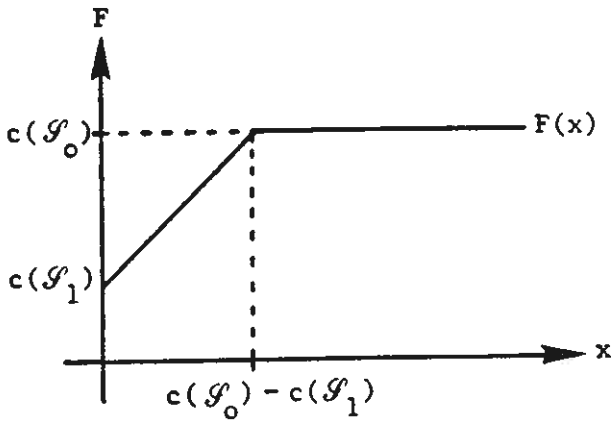
$$f(x) = \min \{c(\mathscr{S}_0) , x + c(\mathscr{S}_1)\} .$$

If $c(\mathscr{S}_0) \leq c(\mathscr{S}_1)$ then $F(x) = c(\mathscr{S}_0)$ for all x , i.e. constant.

If $c(\mathscr{S}_0) > c(\mathscr{S}_1)$ then

$$F(x) = \begin{cases} x + c(\mathscr{S}_1) & \text{for } 0 \leq x \leq c(\mathscr{S}_0) - c(\mathscr{S}_1) \\ c(\mathscr{S}_0) & \text{for } x \leq c(\mathscr{S}_0) - c(\mathscr{S}_1) \end{cases} .$$

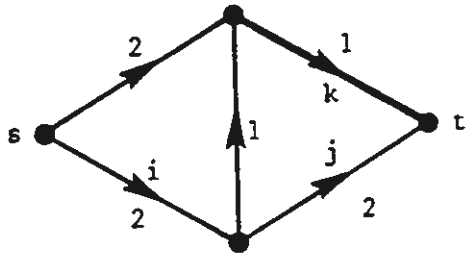




What happens is that when x becomes equal to $c(\mathscr{S}_0) - c(\mathscr{S}_1)$ the variable cut–capacity becomes larger than the constant cut–value $c(\mathscr{S}_0)$. This number x is called the *optimal capacity* for k . An edge k has optimal capacity if and only if there exist a minimal cut which contains edge k and another minimal cut which does not contain k .

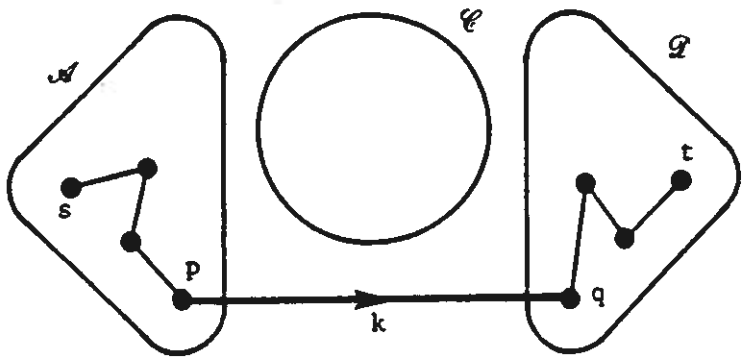
It is seen that the value of maximal flow will be sensitive to small changes of the capacity of an edge if the capacity of the edge corresponds to a point on the sloping line in the figure on the last page. This plays a role when one has to extend a transport network. Then one has, naturally, to increase the capacity of an edge on the sloping line (without the capacity being optimal). Formally this problem gives reason for the following definition:

Let N be a network and let k be an edge in N . The edge k is called *critical* if the value of a maximal flow increases when the capacity of k is made larger, i.e. when the capacity is smaller than the optimal capacity and corresponds to a point on the sloping line in the figure on page xxx.



As an example let us consider the network in the figure where the numbers are capacities. The value of a maximal flow is 3 . If the capacity of the thick edge k is made larger the value of the maximal flow will increase. Thus, k is critical. Moreover, it is the only critical edge in the network. This follows from the fact that the network contains two minimal cuts, namely {i,k} and {j,k} and k is the only common edge for them. It follows from the Max–flow–min–cut theorem that an edge k is critical if and only if k belongs to any minimal cut in N . However, this remark is not suitable for determination of critical edges in a network. It demands to find all minimal cuts in a network and that can become hard work (there can be 2^{P-2} minimal cuts in a transport network where $P = |\mathcal{P}(N)|$) .

However, we can give a good algorithm to determine all critical edges in a given transport network N . We proceed as follows: Let f be a maximal flow in N and let $\mathcal{A}$ be the set of those vertices p in N for which there exists an augmenting s,p—path w.r.t. f . Let the set of those vertices q in N for which there exists an augmenting q,t–path w.r.t. f be denoted by $\mathcal{D}$. $\mathcal{A}$ and $\mathcal{D}$ can be determined in O(K) steps by means of MARKER or MARKER WITH QUEUE algorithms if f is given (see page xxx).

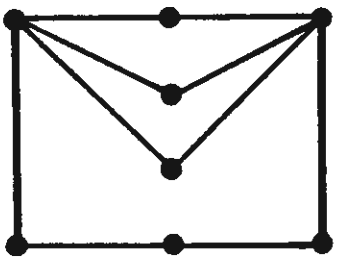


We will prove that an edge k is critical if and only if k is an $\mathcal{A} \rightarrow \mathcal{D}$–edge: If k is a $p \rightarrow q$ edge, $p \in \mathcal{A}$, $q \in \mathcal{D}$ and the capacity of k increases then the augmenting s,p–path and the augmenting q,t–path, together with k , form an augmenting s,t–path. Therefore, f cannot be maximal any longer. Conversely, if a p,q–edge is critical, an augmenting s,t–path V will be formed by increasing the capacity of k . Then the s,p–part of V will then be an augmenting s,p–path, thus $p \in \mathcal{A}$ and, in a similar way, it is seen that the q,t–part of V will become an augmenting path w.r.t. f and therefore $q \in \mathcal{D}$.

$\mathcal{A}$ can be characterised as the set of those vertices p in N with the property that if one adds a p,t–edge with capacity > 0 the value of the maximal flow will increase. Since this property does not depend on f the set $\mathcal{A}$ does not depend on which maximal flow we consider. Analogous, the set $\mathcal{D}$ of those vertices q from which MARKER(f,q) can reach t does not depend on which maximal flow we consider.

26. MATCHING A BIPARTITE GRAPH The max–flow–min–cut theorem is a central graph theoretical theorem. Among other things, this is implied by the fact that the theorem has many different applications. Some of these applications refer to a very special case of the theorem which deals with matchings in a bipartite graph and which is called König's theorem. This theorem was discovered by the Hungarian mathematician Denes König, the founder of modern graph theory and author of the first book about graph theory, published in 1931 [11]. In this section we shall prove König's theorem and describe a corresponding algorithm and we shall mention a couple of applications.

A *matching* in a graph Γ is a set $\mathcal{M}$ of edges in Γ such that each vertex in Γ is incident with not more than one edge in $\mathcal{M}$. The end vertices of the edges in $\mathcal{M}$ are called *paired*. A *maximal* matching in Γ is a matching such that $|\mathcal{M}|$ is maximal. A *perfect* matching is a matching in which all the vertices in Γ are paired.



Γ

The two heavy edges in the figure form a matching. This matching is relatively maximal, i.e. it is not a proper subset of any other matching but it is not maximal: It is easy to find a matching with 3 edges. Γ does not have a perfect matching. This example shows that it is not the number of vertices which determines how many edges are in a maximal matching. However, we can obtain an upper bound in the following way: We define a *node cover* in a graph Γ as a set $\mathcal{N}$ of vertices in Γ such that each edge in Γ is incident with at least one vertex in $\mathcal{N}$. Corresponding to lemma 18.3 we have

Lemma 26.1. Let $\mathcal{M}$ be a matching in a graph Γ and let $\mathcal{N}$ be a node cover. Then

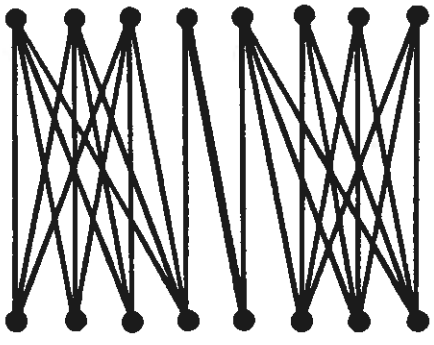
$$|\mathcal{M}| \leq |\mathcal{N}| .$$

Proof. For each edge in $\mathcal{M}$, at least one of the end vertices belongs to $\mathcal{N}$. Since these end vertices are all different, we obtain that $|\mathcal{N}| \geq |\mathcal{M}|$ □

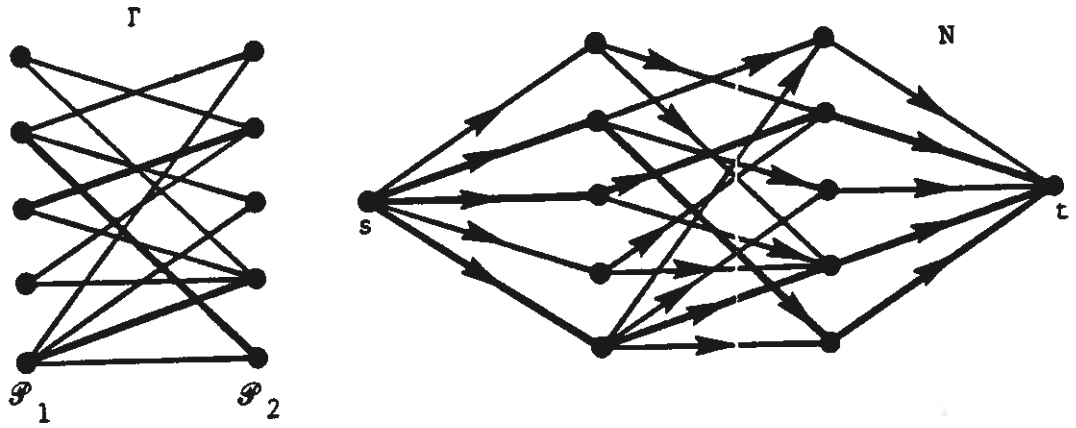
Inspired by the similarity of lemma 18.3 and lemma 26.1 it could be assumed that if $\mathcal{M}$ is a maximal matching and $\mathcal{N}$ a minimal node cover (i.e. a node cover with a least number of vertices) then $|\mathcal{M}| = |\mathcal{N}|$. This, however, is wrong, as can be seen when we consider, for example, a triangle where $|\mathcal{M}| = 1$ and $|\mathcal{N}| = 2$.

In connection with matchings there is an algorithmic problem (to find an algorithm to determine a maximal matching) and an existence problem (to find whether there exists a perfect matching). Both problems are solved, [2], but since they are very complicated we shall consider only the special case where the given graph is bipartite.

First we must notice that even for bipartite graphs neither the matching problem, the algorithm nor how to determine the size of the maximal matching is trivial. Let us illustrate it by means of some examples. A reasonable suggestion for an algorithm could be the following: Since those vertices which have the smallest valencies are the hardest to get paired we shall start to pair them two by two, if possible.



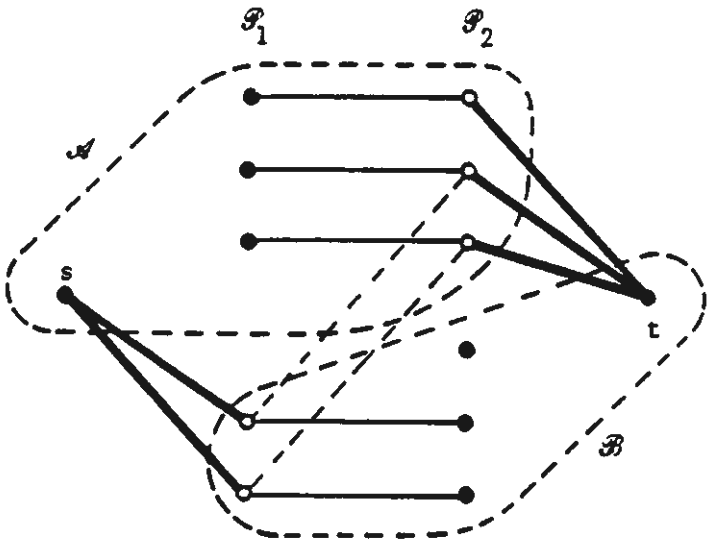
The figure shows a graph with a perfect matching (consisting of the horizontal edges). However, if we start in the described way, we have to take the heavy edge first and then we cannot add edges in order to obtain a perfect matching. Concerning the size of a perfect matching one can, naively, hope that if the number of vertices in $\mathcal{P}_1$ is equal to the number of vertices in $\mathcal{P}_2$ where $\mathcal{P}_1$ and $\mathcal{P}_2$ are the two in a connected bipartite graph then one can contain a perfect matching. However that is wrong as we can see in the figure on page XXX, graph Γ .



Now, let Γ be a bipartite graph with vertex–set $\mathcal{P}(\Gamma) = \mathcal{P}_1 \cup \mathcal{P}_2$, $\mathcal{P}_1 \cap \mathcal{P}_2 = \emptyset$, where each edge in Γ is a $\mathcal{P}_1$, $\mathcal{P}_2$ – edge. Now we form a directed transport network as follows: Each edge is directed from $\mathcal{P}_1$ into $\mathcal{P}_2$ and we give these edges capacity ∞ (or a capacity K which is chosen to be sufficiently large). Then we add a source s and a terminal t . s shall be connected with each vertex $p_1 \in \mathcal{P}_1$ by an $s \to p_1$–edge and t shall be connected with each vertex $p_2 \in \mathcal{P}_2$ by a $p_2 \to t$–edge. These edges are given capacity 1 . Let $\mathcal{M}$ be a matching in Γ . Corresponding to $\mathcal{M}$ we define a flow f in N such that the flow is equal to 1 in those edges which correspond to the edges in $\mathcal{M}$ and in those edges which are incident with either s or t and, furthermore, with a paired vertex. Obviously, the value of f is $|\mathcal{M}|$. Conversely, to each integer–valued flow f in N there corresponds a matching $\mathcal{M}$ in Γ consisting of those edges in Γ for which the corresponding edge in N has flow 1. $|\mathcal{M}|$ is the value of f . Therefore, the algorithmic problem can be solved by the following remark: We can find a maximal matching in Γ by finding a maximal integer flow in N . Using the special structure of N in this case Hopcraft and Karp [6] gave an $O(P^{\frac{5}{2}})$ algorithm to determine a maximal matching in a bipartite graph with P vertices. Now it is not surprising that one can characterize the number of edges in a maximal matching using the Max–flow–min–cut theorem. The result is

König's Theorem. Let Γ be a bipartite graph. Then the number of edges in a maximal matching is equal to the number of vertices in a minimal node cover.

Proof. Let f be a maximal integer flow in N . Then f has a certain value $F = |\mathcal{M}|$ where $\mathcal{M}$ is a maximal matching in Γ .



Now we apply the procedure MARKER (f,s) on N . Let $\mathcal{A}$ be the set of marked vertices and $\mathcal{B}$ be the set of unmarked vertices, $t \in \mathcal{B}$. N has a cut with capacity $|\mathcal{P}_1|$, therefore, if the capacity K of the $\mathcal{P}_1 \to \mathcal{P}_2$ –edges is chosen such that $K > |\mathcal{P}_1|$ then the cut $(\mathcal{A}, \mathcal{B}) = \mathcal{S}$ cannot contain any $\mathcal{P}_1 \to \mathcal{P}_2$ –edge . Therefore, there are no $(\mathcal{P}_1 \cap \mathcal{A}) \to (\mathcal{P}_2 \cap \mathcal{B})$ – edges in N and the set

$$\mathcal{N}_1 = (\mathcal{P}_1 \cap \mathcal{B}) \cup (\mathcal{P}_2 \cap \mathcal{A}) ,$$

which is marked by o in the figure above, forms a node cover in Γ . Since the vertices in $\mathcal{A}$ are labelled and the vertices in $\mathcal{B}$ are unlabelled, the edges shown in the figure with heavy lines must be saturated and the edges shown with the dotted lines must carry zero flow. Therefore, $F = |\mathcal{M}| = |\mathcal{N}_1|$. It follows from Lemma 26.1 that $\mathcal{N}_1$ is a minimal node cover and this completes the proof of König's theorem.

□

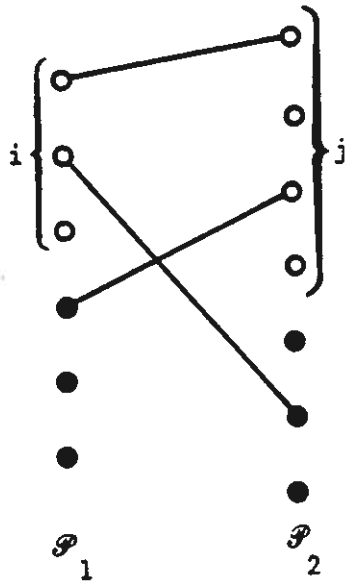
In addition, we obtain, from the proof, a method to determine a minimal node cover in Γ : The unlabelled vertices in $\mathcal{P}_1$ together with the labelled vertices in $\mathcal{P}_2$ form a node cover.

As an example of application of König's theorem, we consider an $m \times n$ matrix $A = (a_{ij})$. We define the normal rank $nr(A)$ as the largest number r for which there exist r non–zero elements in A, which belongs to r different rows in A and to r different columns in A. The normal rank of A is the largest rank that A can get by changing the non–zero elements of A. Corresponding to A we consider the bipartite graph Γ such that $\mathcal{P}(\Gamma) = \mathcal{P}_1 \cup \mathcal{P}_2$ where $\mathcal{P}_1 = \{ r_1, r_2, \ldots, r_m \}$ corresponds to the rows of A and $\mathcal{P}_2 = \{s_1, s_2, \ldots, s_n \}$ corresponds to the columns of A and where r_i is connected with s_j with an edge if and only if $a_{ij} \neq 0$. Obviously, $sr(A) = |\mathcal{M}|$ where $\mathcal{M}$ is a maximal matching in Γ. Thus the normal rank can be determined by means of the maximal flow algorithm. Moreover, applying König's theorem on Γ we obtain that the normal rank of a matrix A is equal to the smallest number of rows and columns in A which together contain all non–zero elements in A.

Applying König's theorem one can also find a condition for a bipartite graph $(\mathcal{P}_1, \mathcal{P}_2)$ to have a matching such that all vertices in $\mathcal{P}_1$ are paired. The condition is given in the following theorem found in 1935, [5]:

Hall's theorem. Let Γ be a bipartite graph with bipartation $(\mathcal{P}_1, \mathcal{P}_2)$. Then Γ contains a matching with $|\mathcal{P}_1|$ edges if and only if for every subset $\mathcal{P} \subseteq \mathcal{P}_1$ there exist $|\mathcal{P}|$ vertices in $\mathcal{P}_2$ each having a neighbour vertex in $\mathcal{P}$.

Proof. "only if": This is trivial, because, if there exists a matching $\mathcal{M}$ such that $|\mathcal{M}| = |\mathcal{P}_1|$ and if $\mathcal{P} \subseteq \mathcal{P}_1$ then each end–vertex (in $\mathcal{P}_2$) of an edge in $\mathcal{M}$ starting in $\mathcal{P}$ has a neighbour vertex in $\mathcal{P}$.



"if": Conversely, let every subset $\mathcal{P} \subseteq \mathcal{P}_1$ have at least $|\mathcal{P}|$ neighbours in $\mathcal{P}_2$. Let $\mathcal{N}$ be a minimal node cover in Γ shown with $\circ$ in the figure and put $i = |\mathcal{P}_1 \cap \mathcal{N}|$ and $j = |\mathcal{P}_2 \cap \mathcal{N}|$. Applying the given condition to the set of those vertices in $\mathcal{P}_1$ which are not in $\mathcal{N}$ it follows that

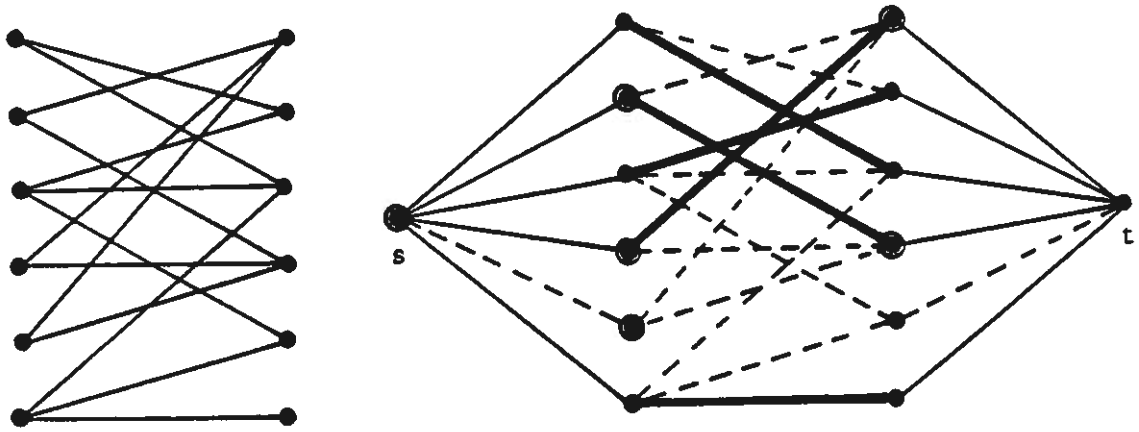
$$|\mathcal{P}_1| \leq i - j$$

and, therefore

$$|\mathcal{N}| = i + j \geq |\mathcal{P}_1| .$$

Since $\mathcal{P}_1$ is a node cover, it follows that $|\mathcal{N}| = |\mathcal{P}_1|$. Therefore, it follows from König's theorem that there exists a matching with $|\mathcal{P}_1|$ edges.

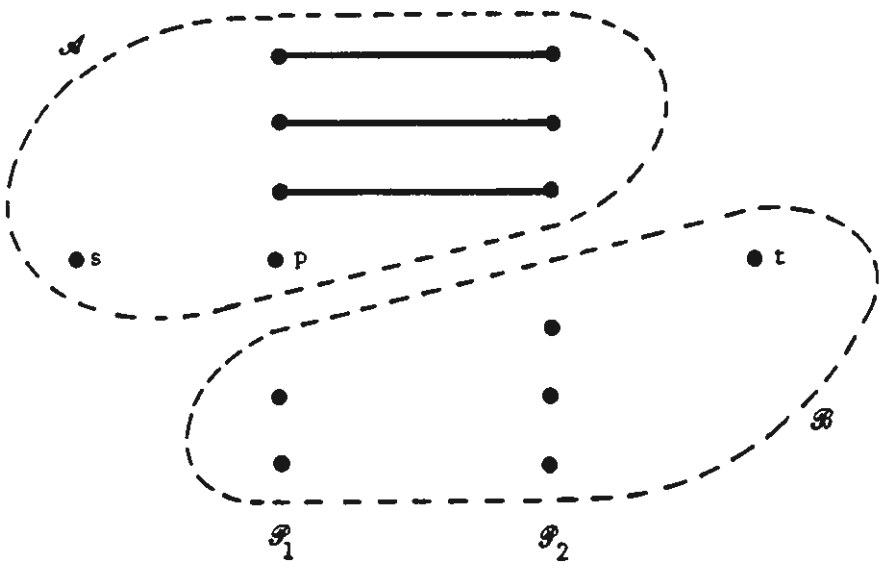
The interesting part of Hall's theorem states that if there does not exist a matching with $|\mathcal{P}_1|$ edges then it is because there exists a subset $\mathcal{P} \subseteq \mathcal{P}_1$ with less than $|\mathcal{P}|$ neighbours. The set $\mathcal{P}$ can in fact, always be determined by means of the labelling procedure: If we determine a maximal flow in $\mathcal{N}$ and afterwards apply MARKER (s,f) the set $\mathcal{P}$ labelled vertices in $\mathcal{P}_1$ (shown with circles in the figure to the right) has less than $|\mathcal{P}|$ neighbours. In the figure, the 3 labelled vertices in $\mathcal{P}_1$ have only 2 neighbours.



In general, the statement can be shown as follows: We consider the transport network described on page XXX (the figure to the right), determine a maximal flow f in N and finally, apply MARKER (f,s) .

1. Each vertex in $\mathcal{P}_2 \cap \mathcal{A}$ has to be paired (otherwise the labelling would reach t) with a vertex in $\mathcal{A} \cap \mathcal{P}_1$ (otherwise there is a vertex in $\mathcal{B} \cap \mathcal{P}_1$ which can be labelled).
2. $\mathcal{P}_1$ contains an unpaired vertex p . p can be labelled, therefore, $p \in \mathcal{P}_1 \cap \mathcal{A}$.
3. There cannot exist an $\mathcal{A} \cap \mathcal{P}_1 - \mathcal{B} \cap \mathcal{P}_2$ – edge.

From 1., 2. and 3. it follows that $\mathcal{P} = \mathcal{P}_1 \cap \mathcal{A}$ has less than $|\mathcal{P}|$ neighbours.



If every vertex in $\mathcal{P}_2$ is a person and every vertex in $\mathcal{P}_1$ symbolizes a committee consisting of its neighbours in $\mathcal{P}_2$ then we obtain the following formulation of Hall's theorem: It is possible to choose m different representatives for m committees if and only if k arbitrary committees contain at least k different memebers, $k = 1, 2, \cdots, m$.

By means of Hall's theorem, it is easy to derive the following theorem found by König in 1914:

Theorem 26.2. A v–regular bipartite graph $(v \neq 0)$ has a perfect matching.

Proof. If all vertices have valency v $(\geq 1\ !)$ out of $\mathcal{P} \subseteq \mathcal{P}_1$, there will be at least $v|\mathcal{P}|$ edges. These edges cannot all end in a set $\mathcal{R} \subseteq \mathcal{P}_2$ with $|\mathcal{R}| < |\mathcal{P}|$ since then there would exist at least one vertex in $\mathcal{R}$ with valency $> v$. Therefore, theorem 26.2 follows from Hall's theorem. □

Let us conclude this section with two applications of this theorem. First we consider an $n \times n$ matrix A. As mentioned on page 3.61, the structure rank of A can be determined by means of a bipartite graph. If Γ is a regular graph with valency v, then the rows and columns in A each contain v non–zero elements. From theorem 26.2 it follows that if an $n \times n$–matrix A has the same number of non–zero elements in each row and column then the structure rank of A is equal to n.

Another application of theorem 26.2 was formulated by König: There are n ladies and n gentlemen present at a dance, and none of them are willing to make acquaintances with new people. It so happens that every lady knows exactly k gentlemen and every gentleman knows exactly k ladies. Then, it is possible to arrange these ladies and gentlemen in pairs of a lady and a gentleman such that only persons who know each other in advance will be arranged in a pair.

27. MENGER'S THEOREM. As we have emphasized in previous chapters that the Max–flow–min–cut theorem has a lot of applications of different kind. In this section we shall show that Menger's theorem which we have formulated without proof on page XXX can be proved by means of the Max–flow–min–cut theorem.

Let Γ be a given graph and let $\kappa = \kappa(\Gamma)$ denote the largest number such that two arbitrary vertices s and t , $s \neq t$ in Γ are connected with at least κ disjoint paths. Disjoint means that the paths have no other vertices except s and t in common. Analogous, $\sigma = \sigma(\Gamma)$ is defined as the smallest number such that there exists a vertex cut in Γ with σ vertices. If two arbitrary vertices in Γ are connected with an edge such that there doesn't exist any vertex cut we put $\sigma = |\mathcal{P}(\Gamma)| - 1$. Then we have

Menger's theorem. For every graph Γ: $\kappa(\Gamma) = \sigma(\Gamma)$.

As mentioned on page XXX it is easy to see that $\sigma \geq \kappa$. Let us consider the more difficult part of Menger's theorem, namely, the inequality $\kappa \geq \sigma$ which is more interesting. What we have to do to prove this is to consider a graph Γ with a certain value σ . Then we know that Γ does not contain any vertex cut with less than σ vertices. By means of this knowledge we have to prove that two arbitrary vertices s and t are connected with at least σ disjoint paths. Then, it follows that $\kappa \geq \sigma$. Therefore, we shall construct somehow, these paths or at least show their existence. This shall be done by means of the Max–flow–min–cut–cut theorem.

The proof is based on the following theorem which is somewhere in between the Max–flow–min–cut theorem and Menger's theorem. A cut $\mathscr{S}$ is said to *seperate* s and t if s and t are in the same connected component in Γ but in two different connected components in $\Gamma \setminus \{\mathscr{S}\}$.

Theorem 27.1. Let s and t be two different vertices in a graph Γ, let κ be the largest number for which one can find κ edge–disjoint s,t–paths in Γ and let σ be the smallest number such that there exists an edge–cut $\mathscr{S}$ consisting of σ edges and which separate s and t. Then $\kappa = \sigma$.

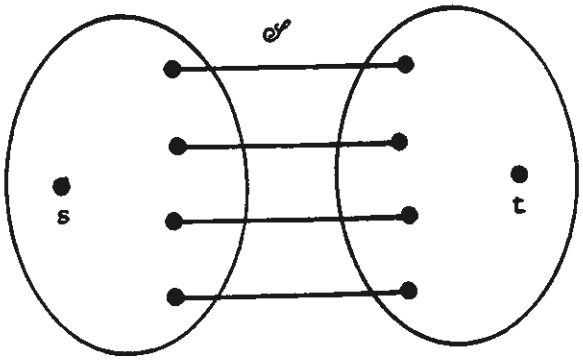
Proof. Let us take s as a source and t as a terminal and give each edge in Γ capacity 1. Γ becomes a non–directed transport network. From the integer–value–theorem and the theory about path flows in section 2.4, it follows that κ is the value of a maximal flow and $\mathscr{S}$ is a minimal cut. Then from the Max–flow–min–cut theorem we obtain that $\kappa = \sigma$. □

We shall also use a "directed" version of theorem 27.1. An edge–cut which separates s from t is a set $\mathscr{S}$ of edges such that every $s \to t$–path in Γ contains an edge in $\mathscr{S}$. Then theorem 27.1 holds for a directed graph if "s,t–path" is replaced by "s→t–path". The proof is the same, it just uses the "directed" version of the Max–flow–min–cut theorem.

Theorem 27.1 is very close to the theory of transport networks because the two vertices s and t are given in the theorem. In the following theorem which is an alternative formulation of theorem 7.3, this assumption is taken away.

Theorem 27.2. Let Γ be an arbitrary graph, let κ be the largest number for which two arbitrary vertices in Γ are connected with the least κ pairwise edge–disjoint paths and let σ be the smallest number for which σ be the smallest number for which Γ has an edge cut consisting of σ edges. Then $\kappa = \sigma$.

Proof. $\kappa \leq \sigma$ is trivial. For if $\sigma < \kappa$ we consider a cut $\mathscr{S}$ consisting of σ edges. Then two vertices s and t which belong to different connected compounds in $\Gamma \setminus \{\mathscr{S}\}$ cannot be connected with κ pairwise edge–disjoint paths.

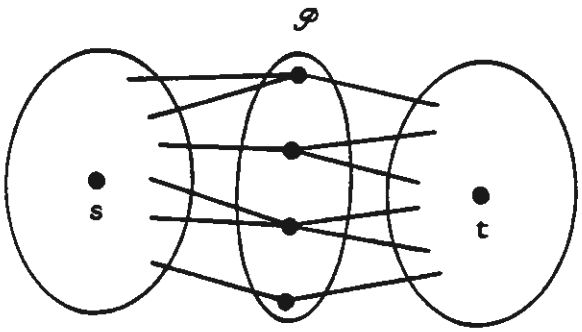


Conversely, we shall show that $\kappa \geq \sigma$ i.e. two arbitrary vertices s and t are connected with at least σ pairwise edge–disjoint paths. A smallest cut $\mathscr{A}$ in Γ for which s and t belong to different connected components in $\Gamma \setminus \{ \mathscr{A} \}$ satisfies $| \mathscr{A} | \geq \sigma$. The largest number κ_1 for which there exist κ_1 pairwise edge–disjoint s,t–paths in Γ satisfies $\kappa_1 = | \mathscr{A} |$ according to theorem 27.1. From these two relations, it follows that $\kappa_1 \geq \sigma$. This completes the proof. □

Finally we come to Menger's theorem (compare p.XXX and XXX).

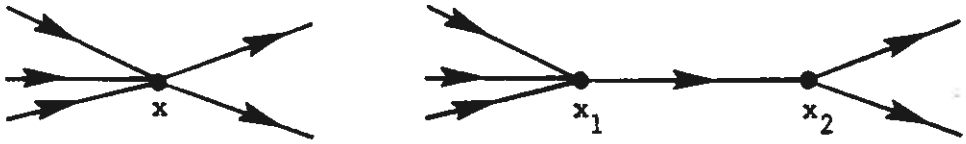
Theorem 27.3. Let Γ be an arbitrary graph in which there exists a vertex–cut (i.e. in which one can find two vertices which are not neighbours). Let κ be the largest number for which two arbitrary vertices s and t in Γ are connected with at least κ pairwise disjoint (not taking into account s and t) and let σ be the smallest number for which Γ has a vertex–cut consisting of σ vertices. Then $\kappa = \sigma$.

Proof. $\kappa \leq \sigma$ is trivial. For if $\sigma < \kappa$ two vertices s and t belonging to different connected components in $\Gamma \setminus \{ \mathscr{P} \}$ where $\mathscr{P}$ is a vertex cut consisting of σ vertices cannot be connected with κ pairwise disjoint paths.



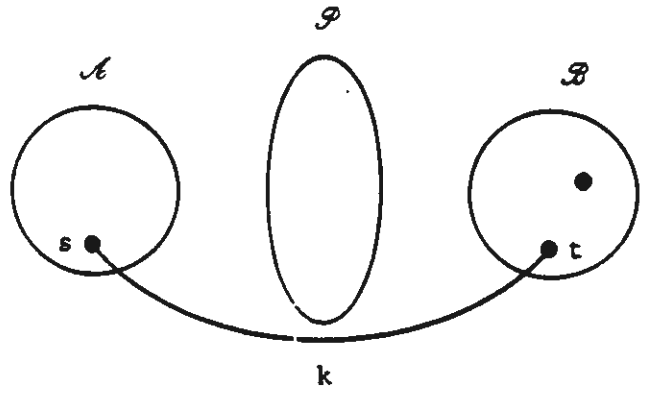
Conversely, we shall show that $\kappa \geq \sigma$, i.e. that two arbitrary vertices s and t are connected with at least σ disjoint paths. We consider two cases:

Case 1: s and t are not neighbours. Then we form a new graph as follows: First, we replace each edge by a double–edge with arrows in converse directions. Then we replace each vertex x by two vertices x_1 and x_2 which are connected as shown in the figure below.



If we denote the new graph by Γ_1 we obtain the following: For any s,t–path V in Γ we have a corresponding one–way $s \to t$–path V_1 in Γ_1 (and conversely) and two paths in Γ are disjoint if and only if, the corresponding paths in Γ_1 are edge–disjoint. In Γ_1, all edge–cuts which separate s from t have cardinality $\geq \sigma$. This can be proved as follows. Suppose that $\mathscr{S}$ is an edge–cut in Γ_1 which seperates s from t and which has cardinality $< \sigma$. If for every edge in $\mathscr{S}$ we take one end vertex different from s and t we obtain a vertex–cut in Γ_1 with cardinality $< \sigma$ which seperates s from t and the corresponding vertices in Γ then will become a vertex cut in Γ which seperates s and t and which has cardinality $< \sigma$. Contradiction. Since all edge–cuts in Γ_1 which separate s and t therefore have cardinality $\geq \sigma$ there exist σ edge–disjoint one–way $s \to t$–paths in Γ_1. This follows form the "directed version" of theorem 27.1. Therefore, the corresponding s,t–paths in Γ are disjoint.

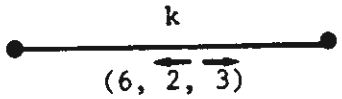
Case 2: s and t are neighbours connected with an edge k. Let Γ_1 be the graph obtained from Γ by taking away the s,t–edges. Then Γ_1 has no vertex–cut with cardinality $\leq \sigma-2$. This can be proved as follows: Suppose that $\mathscr{P}$ is a vertex–cut in Γ_1 with cardinality $\leq \sigma-2$. Let $\mathscr{P}$ separate the other vertices in the graph into two sets $\mathscr{A}$ and $\mathscr{B}$. Then it is clear that k must be an $\mathscr{A}$, $\mathscr{B}$–edge and we can suppose that $s \in \mathscr{A}$ and $t \in \mathscr{B}$. Either $|\mathscr{A}| \geq 2$ or $|\mathscr{B}| \geq 2$ because in other cases Γ would only have σ vertices. Suppose, for example, that $|\mathscr{B}| \geq 2$ as shown in the figure below. Then $\mathscr{P} \cup \{t\}$ would be a vertex–cut with $\sigma-1$ vertices and this is a contradiction.

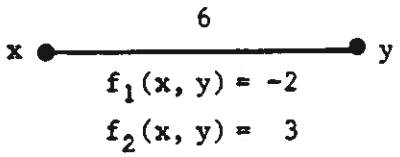


Therefore, in Γ_1, all vertex–cuts have cardinality $\geq \sigma-1$ and because of Case 1, there exist $\sigma-1$ vertex–disjoint s,t–paths in Γ. This completes the proof. □

28. TWO–COMMODITY NETWORKS . It is natural to try to generalise the theory of transport networks to the situation where we have n commodities, which flow in the same network. Each commodity then has its own source and its own terminal, and the purpose of the investigation is to maximise the sum of the flows of each single commodity. We assume that the sum of the flows of different commodities in an edge must not be larger than the capacity of that edge, and that in every vertex which is not a source or a terminal of commodity i , the flow into the vertex of commodity i is equal to the flow out of the vertex. This problem turns out to be very complicated, and a solution is not known; only the theory of non–directed 2–commodity flows is well established. The basic content of this theory will be described in the following.

A *2–commodity network* N consists of a non–directed graph in which 4 different vertices are appointed to be sources and terminals. The sources are denoted s_1 and s_2 , and the terminals t_1 and t_2 . Each edge k has a capacity $c_k > 0$. A flow f is described in the following way: For each edge k we denote the flow of commodity 1 by $f_1(k)$, the flow of commodity 2 by $f_2(k)$, and we use two arrows to show in which direction the flow of commodity 1 and the flow of commodity 2 actually take place. In the figure to the left, the capacity is 6, 2 units of commodity 1 are flowing to the left, and 3 units of commodity 2 are flowing to the right.





The edge is *unsaturated*, but if we change the commodity 2 flow to 4 it will be *saturated.* If k is an x,y–edge then we can also describe the direction of the flow by means of signs, as shown in the other figure.

The *Boundary Condition* then states that $|f_1(x,y)| + |f_2(x,y)| \le c_k$, and the *equilibrium condition* states that when i is 1 or 2, and when $x \neq s_i , t_i$, then

$$\sum_y f_i(x,y) = 0 ,$$

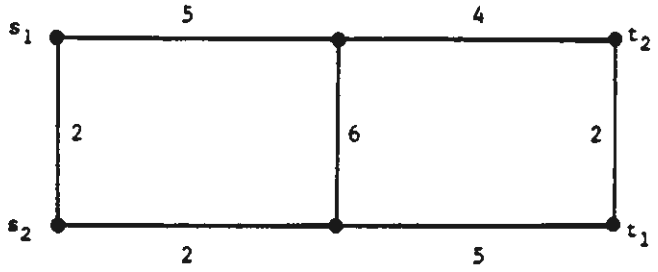
where the summation is over all neighbours y of x . The *value* $f = F(f)$ of the flow f is

$$F = \sum_x f_1(s_1,x) + \sum_y f_2(s_2,y) ,$$

where in the first sum we have to sum over all neighbours x of s_1, and in the other sum over all neighbours y of s_2.

The problem we are going to solve is the following: How can we find a maximal flow.

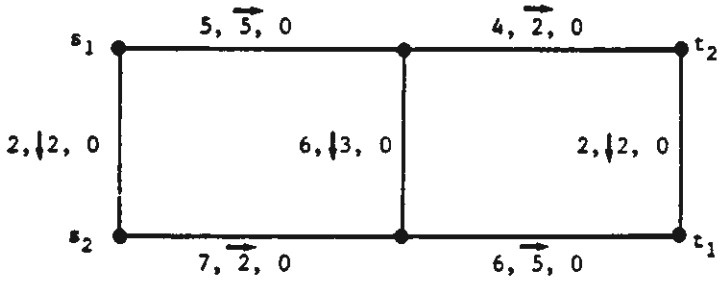
Let us illustrate the problem by means of the example in the figure, where the numbers are capacities.

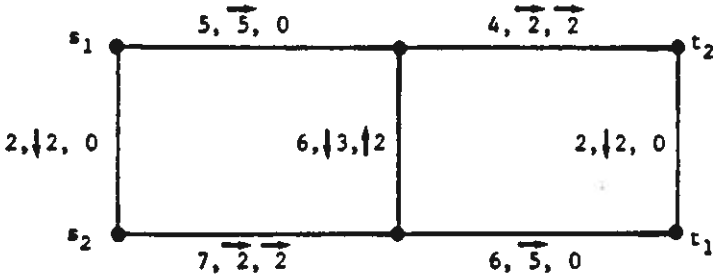


The most natural algorithm to start with is the following:

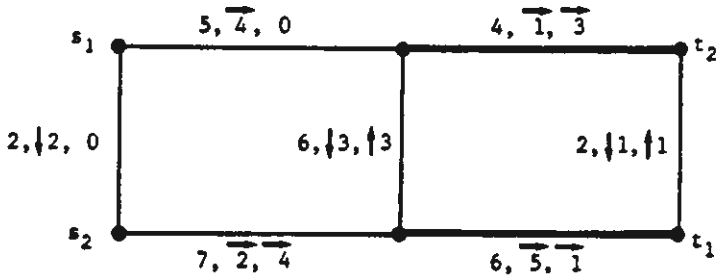
1) Find a maximal flow of commodity 1 from s_1 to t_1.
2) Reduce the capacities by the flow values found.
3) Find a maximal flow of commodity 2 from s_2 to t_2.

This process is shown in the two figures below.



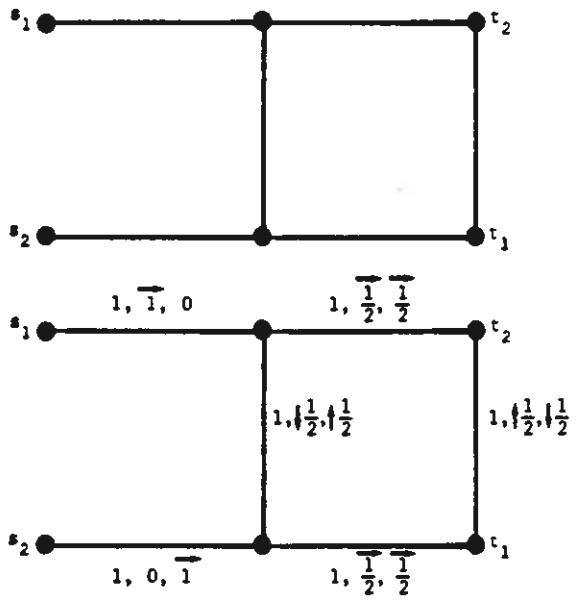


Observe that the resulting flow has value 9. However, this flow is not maximal because the figure below shows a flow of value 10. This flow is actually maximal, as can be seen from the two edges drawn in heavy lines. So a simple algorithm will not always work.

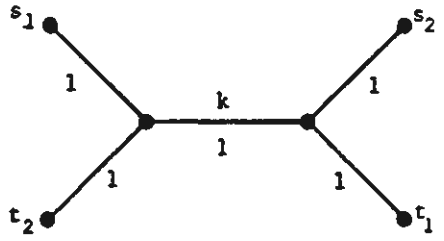


Among the reasons why it isn't easy to find a correct algorithm, is that the 2–commodity problem contains the following two difficulties:

1. In the network in the figure below to the left, all capacities are 1. If we assume that the flow in each edge is 0 or 1 it is easy to see that a non–zero flow of commodity 1 will not allow any flow of commodity 2, and conversely. Therefore a maximum integer flow has value 1. As shown in the figure to the right, there exists a flow with value 2. Therefore the integer value theorem is not valid for 2–commodity flows.



2. If we define a cut in the same way as for usual transport networks, then the max–flow–min–cut theorem is not valid for 2–commodity flows. For if a cut $\mathscr{S}$ is a set of $\mathscr{A}$, $\mathscr{B}$–edges, where $\{s_1, s_2\} \subseteq \mathscr{A}$ and $\{t_1, t_2\} \subseteq \mathscr{B}$, then the 2–commodity network in the figure contains only cuts with capacity 2 and 3 and it has a maximum flow of value 1.



It follows from the examples we have given that we have to find a new type of augmenting path and a new type of cut if we want an algorithm which works, and if we want some type of max–flow–min–cut theorem to be valid. We shall start with the cut problem.

Let N be a 2–commodity network. We then define a cut in N as a set $\mathscr{S}$ of edges in N for which every s_1, t_1–path contains an edge in $\mathscr{S}$, and every s_2, t_2–path contains an edge in $\mathscr{S}$. Then we have:

Theorem 28.1. If f is a flow in a 2–commodity network N and $\mathscr{S}$ is a cut in N, then

$$F(f) \leq c(\mathscr{S}) .$$

Proof. We consider each of the 2 flows f_1 and f_2 as path flows, and denote the corresponding paths $V_1, V_2, \ldots, V_n$. $F(f)$ is then the sum of the flows in these paths, and since each path contains at least one edge in $\mathscr{S}$, the path occupies a capacity in an edge in $\mathscr{S}$ which is at least the flow in that path. Therefore $c(\mathscr{S}) \geq$ the sum of the flows in the path, that is, the value of F. □

From Theorem 28.1 we have immediately

Corollary 28.2 When f is a flow and $\mathscr{S}$ is a cut in a 2–commodity network N, and when

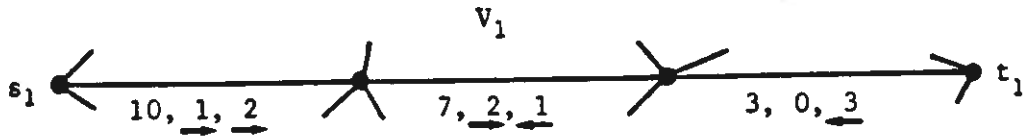
$$F(f) = c(\mathscr{S}) ,$$

then f is a maximal flow and $\mathscr{S}$ is a minimum cut.

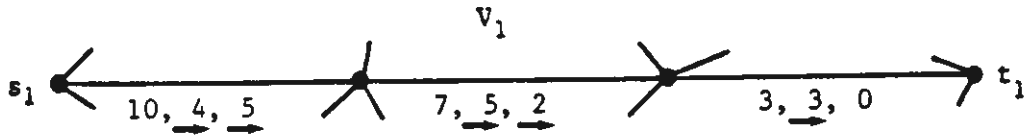
We next turn our attention to the augmenting paths.

Let N be a 2–commodity network and let f be a flow in N. As the example on page xxx shows, the usual augmenting paths are not good enough when we want to determine a maximum flow. The idea, which is due to Hu, ref. [7], is not trivial: We have to use pairs of two different kinds of paths, one which is called positive and the other which is called negative.

An s_1, t_1–path V_1 is called a *positive* s_1, t_1–*path* with value $\Delta_1 > 0$ when it is possible to increase both the commodity 1 flow and the commodity 2 flow in each edge of V_1, in the direction from s_1 to t_1, with the amount Δ_1. The figure shows an example with $\Delta_1 = 3$:

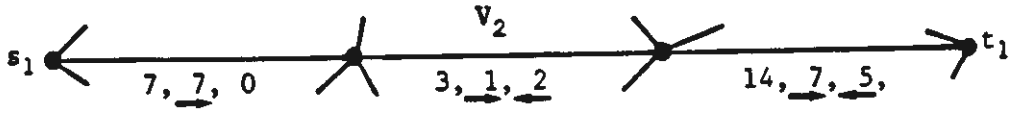


When we *use* V_1 , that is, carry out the increases mentioned, we obtain the following result:

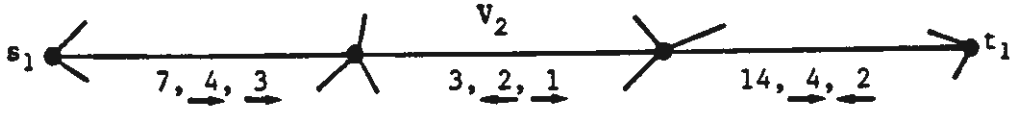


Clearly, when we use a positive s_1, t_1–path V_1 then we destroy the equilibrium condition for commodity 2 in s_1 and t_1 .

An s_1 ,t_1–path V_2 is called a *negative* s_1, t_1–*path* with value Δ_2 , if it is possible to increase both the commodity 1 flow in each edge of V_2 in the direction from s_1 to t_1 with the amount Δ_2 , and also the commodity 2 flow in each edge of V_2 in the direction from t_1 to s_1 with the value Δ_2 . The figure shows an example with $\Delta_2 = 3$:



When we *use* V_2 , that is, carry out the increases mentioned, we obtain the following result:



Clearly if we use a negative s_1, t_1–path V_2 , then we destroy the equilibrium condition for commodity 2 in s_1 and t_1 , but with the opposite sign to before.

We shall assume that the 2 numbers Δ_1 and Δ_2 are chosen as large as possible for the paths V_1 and V_2 . Neither positive nor negative paths work in themselves, since there is, as we have mentioned, an equilibrium condition which is destroyed. But if you take the paths in pairs, it works!

An s_1, t_1–*path pair* (V_1, V_2) with value $\Delta > 0$ consists of a positive s_1, t_1–path V_1 with value Δ_1, and a negative s_1, t_1–path V_2 with value Δ_2, where $\Delta = \min \{\Delta_1, \Delta_2\}$. By using the path pair (V_1, V_2) with value Δ, we understand to use V_1 with value Δ and then V_2 with value Δ (or conversely; the result will be the same). The construction is now so clever that when the pair is used, we have:

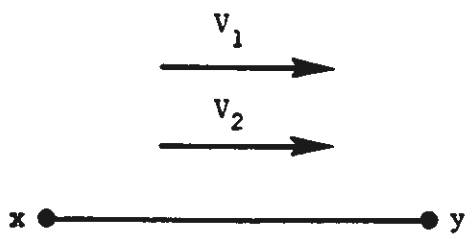
1. The equilibrium condition is now satisfied again.
2. All flows are smaller than or equal to the corresponding capacity.
3. The flow value has been increased by 2Δ.

Proof.

1. follows from the fact that V_1 sends Δ units of commodity 2 from s_1 to t_1, while V_2 sends Δ units of commodity 2 from t_1 to s_1.

2. is trivial when V_1 and V_2 have no common edge. If k is an x,y–edge which is common to V_1 and V_2, there are two cases:

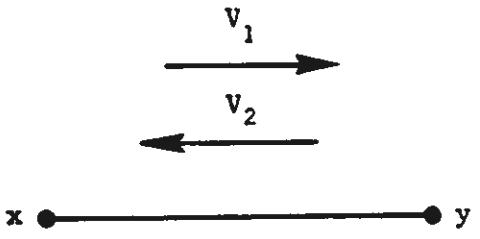
V_1 and V_2 use k in the same direction



V_1 : $f_1(x, y) : +\Delta$ $f_2(x, y) : +\Delta$

V_2 : $f_1(x, y) : +\Delta$ $f_2(x, y) : -\Delta$

V_1 and V_2 use k in opposite directions



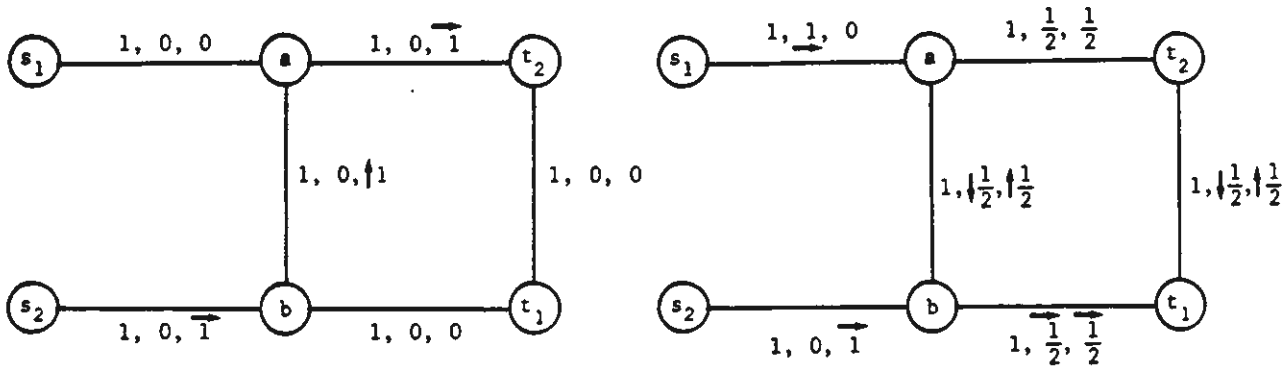
$V_1 : f_1(x, y) : +\Delta$ $f_2(x, y) : +\Delta$

$V_2 : f_1(x, y) : -\Delta$ $f_2(x, y) : +\Delta$

2 follows from these formulae.

Finally, 3 follows from the fact that the value of the commodity 1 flow is increased by the amount Δ both when V_1 and when V_2 are used, while the commodity 2 flow has the same value as before.

In the network in the figure below to the left, there are no usual augmenting paths. But there is a path pair consisting of the positive path $V_1 = s_1, a, b, t_1$ with value $\frac{1}{2}$, and a negative path $V_2 = s_1, a, t_2, t_1$ with value $\frac{1}{2}$. If we use this path pair, we obtain the flow in the figure to the right, which in this case is actually maximal.



It is easy either to find, or to prove the non–existence of, positive and negative paths (and also of path pairs) by using a usual labelling process in $O(K)$ steps.

A usual augmenting s_1, t_1–path N is in itself a path pair (V, V).

Obviously in a similar way we can define s_2, t_2–paths which are positive or negative and s_2, t_2–path pairs.

After these preparations, we can now prove a flow algorithm and at the same time show a maximal–flow–minimum–cut theorem for 2–commodity networks.

algorithm: MAXIMAL 2–COMMODITY FLOW.
Input: A non–directed 2–commodity network N.
Output: A maximal flow f in N.
Initialising: Let $f = (f_1, f_2)$ be the zero flow.

1. Find a maximal flow in N of commodity 1.
2. **While** there exists an s_2, t_2–path pair (V_1, V_2) with respect to f, use (V_1, V_2) to increase the value of f.
3. STOP when there are no more s_2, t_2–path pairs.

About this algorithm, we have the following theorem:

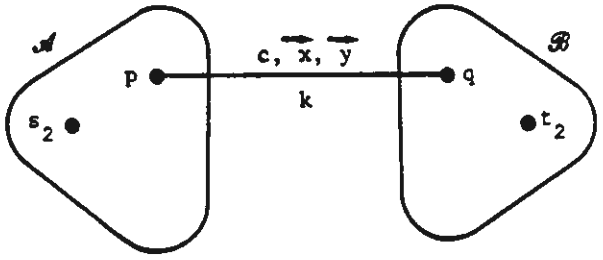
Theorem 28.3. When MAXIMAL 2–COMMODITY FLOW is stopped, then f is a maximal flow in N.

and furthermore

Max–flow–min–cut theorem. For every 2–commodity network, the value of a maximal flow is equal to the capacity of a minimum cut.

Proof of the two theorems. Assume that the algorithm has stopped. Then there exists no positive s_2, t_2–path, or there exists no negative s_2, t_2–path. Since those two cases can be treated in the same way, we need only consider the first case. So there are no positive s_2, t_2–paths in N with respect to f.

Let $\mathcal{A}$ denote the set of those vertices p in N for which there exists an s_2, p–path in N which is positive. Then s_2 is in $\mathcal{A}$ and t_2 is in $\mathcal{B} = P(N) \setminus \mathcal{A}$. Then every p,q–edge k (with capacity c) with $p \in \mathcal{A}$ and $q \in \mathcal{B}$ will be saturated by means of two flows x and y, both going from p to q, $c = x + y$. Otherwise, a positive s_2, p–path could be made to reach q.



Case 1: Assume there exists an $\mathcal{A}$, $\mathcal{B}$–edge with positive value of commodity 1, then it follows that $s_1 \in \mathcal{A}$ and $t_1 \in \mathcal{B}$. Therefore the set of $\mathcal{A}$, $\mathcal{B}$–edges is a 2 commodity cut $\mathcal{S}$ and $c(\mathcal{S}) = F(f)$, compare Lemma 17.1. From Corollary 28.2 it now follows that f is a maximal flow and $\mathcal{S}$ is a minimum cut, and hence the max–flow–min–cut theorem.

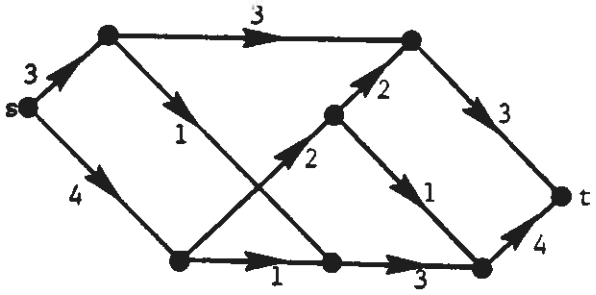
Case 2: All $\mathcal{A}$, $\mathcal{B}$–edges have flow 0 of commodity 1. From this it follows that $\{s_1, t_1\} \subseteq \mathcal{A}$ or $\{s_1, t_1\} \subseteq \mathcal{B}$. Set $\mathcal{S} = \mathcal{S}_1 \cup \mathcal{S}_2$, where $\mathcal{S}_1$ is a minimum cut when N is considered as a network with source s_1 and terminal t_1, and where $\mathcal{S}_2$ is the set of $\mathcal{A}$, $\mathcal{B}$–edges. Then $\mathcal{S}$ is a cut in N. This follows from the fact that every s_1, t_1–path contains an edge in $\mathcal{S}_1$ and every s_2, t_2–path contains an edge in $\mathcal{S}_2$. Furthermore, $c(\mathcal{S}) = F(f_1) + F(f_2) = F(f)$. From Corollary 28.2 it now follows that f is maximal and $\mathcal{S}$ is a minimum cut and therefore we have proved the max–flow–min–cut theorem in this case. □

Note that the existence of a maximal flow f in a 2–commodity network can be shown by means of an argument analogous to the argument on pp. 3.12 – 3.13 . It is not difficult to prove that the algorithm stops when the capacities are integers or just rational numbers.

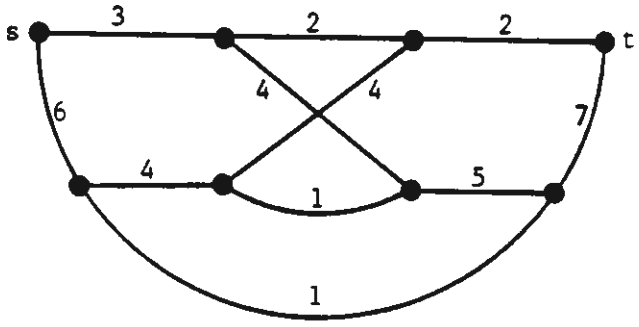
We shall conclude this chapter with a very important consideration. The max–flow–min–cut theorem gives a good characteristic of the maximal flow which can be found. Good in the sense that it may be difficult to find a maximal flow and a minimum cut, but once the flow and the cut have been found then it is possible without doing the work once more to make sure the result is correct. In the 2 cases we have considered (transport networks and non–directed 2–commodity networks) there exists a good characteristic as well as a good algorithm. It is very interesting that in the remaining cases (directed multi–commodity networks and non–directed networks with more than 2 commodities), no good characteristic and no good algorithm is known. This is not a pure coincidence. The proofs are constructed in such a way that it is not easy to prove the algorithm without at the same time proving the characteristic.

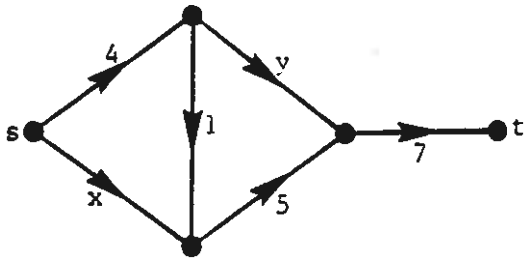
PROBLEMS FOR CHAPTER 3.

3.1. Find, by means of the algorithm, a maximal flow in the network shown in the figure (Use in every step a shortest augmenting path and draw the paths.) Find a minimal cut in the network.



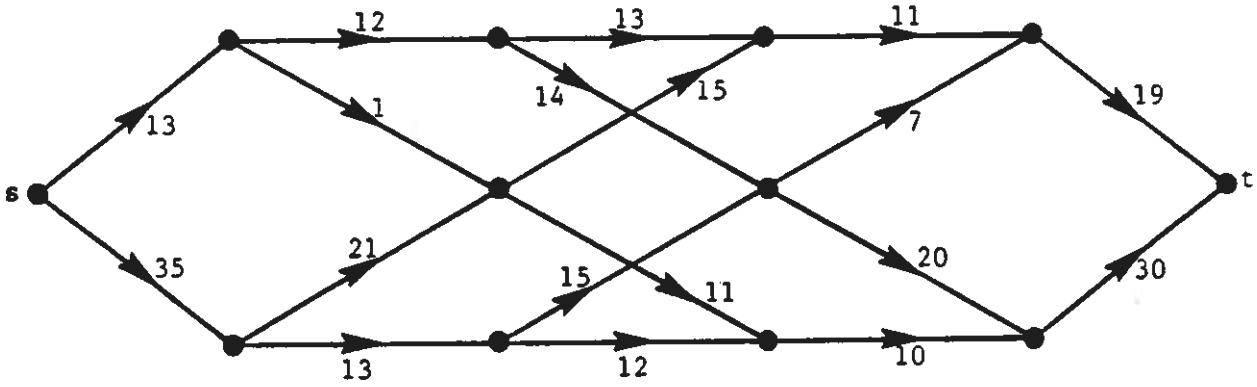
3.2. Find, by means of the algorithm, a maximal flow in the not–directed network in the figure. (Use in every step the shortest augmenting path and draw the paths you use) Find a minimal cut in the network.





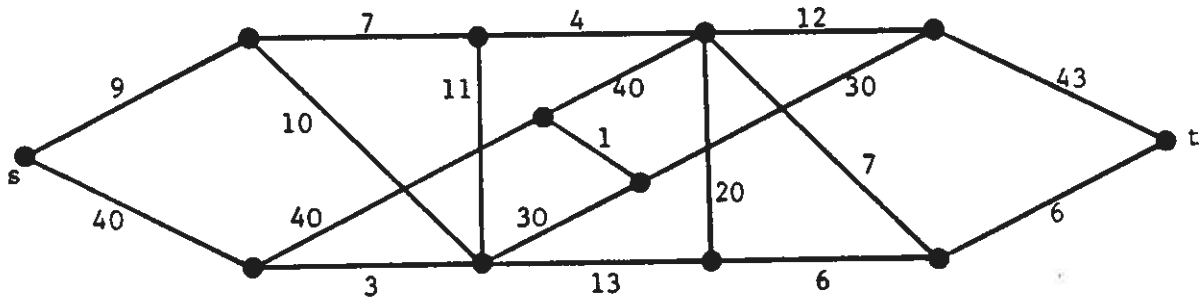
3.3. Find the value F(x, y) of a maximal flow in the network in the figure as a function of the two variable capacities x and y . Describe the answer by means of a figure in the xy–plane. (x and y are not necessarily integers.)

3.4.



Find a maximal flow and a minimal cut in the transport network in the figure. How many maximal integer flows are there ?

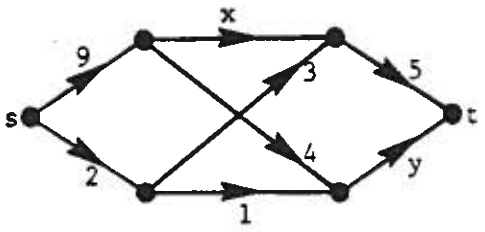
3.5.



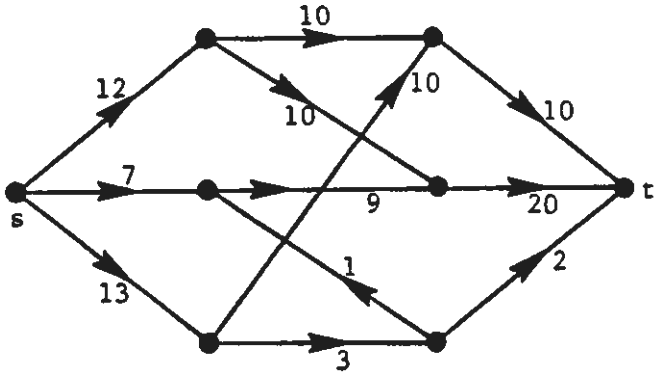
a) Find a maximal flow and a minimal cut in the not–directed network N in the figure.

b) Describe all maximal flows in the network.

3.6. Find the value F(x, y) of a maximal flow in the network in the figure as a function of the two variable capacities x and y . Describe the answer by means of a figure in the xy–plane (x and y are not necessarily integers.)



3.7. Let s(m) denote the maximal number of different cuts, which can exist in a transport network with m vertices. Find s(m) . Draw a transport network where the number of cuts is s(m) and another where the number of cuts are $< s(m)$. (Choose m yourself.)



3.8.

a) Find the critical edges in the network in the figure.

b) Find those edges which have optimal capacity.

3.9. Two water plants V_1 and V_2 provide 4 consumers F_1, F_2, F_3 and F_4 with water. V_1 can deliver 25 m^3/min and V_2 can deliver 20^3/min. The needs of the consumers are respectively 13 m^3/min, 6 m^3/min, 13 m^3/min and 13 m^3/min. The supply takes place via tubes to 3 pumping stations P_1, P_2 and P_3. The capacities of the tubes (in m^3/min) are the following:

	P_1	P_2	P_3
V_1	9	7	11
V_2	6	6	8

	F_1	F_2	F_3	F_4
P_1	7	8	0	0
P_2	7	4	9	8
P_3	0	0	6	10

Each pumping station has capacity 15 m^3/min. The flow takes place in the direction from water plant to pumping station to consumer.

a) Describe a maximal flow in the network and show that at least on econsumer cannot have its need covered.

b) Can the deficit be distributed equally between the consumers?

c) If the water in a tube does not move for a long time, there may start a rich seam of algae in this tube. Therefore, we want the water actually to flow in all the tubes. How can this be achieved?

d) Where can you change the network if you want to obtain a better network by increasing one of those capacities which are different from zero?

3.10. Find, by means of algorithm maximal, flow the structure rank of the matrix

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

3.11. Three power plants E_1, E_2 and E_3 with capacities 3 MW, 7 MW and 9 MW are going to provide 4 towns B_1, B_2 B_3 and B_4 with electricity. The needs of the towns are 3 MW, 4 MW, 8 MW and 4 MW . The supply takes place via 4 transformer stations T_1, T_2, T_3 and T_4 . The capacities of the high voltage lines used are given in the tables below.

	E_1	E_2	E_3
T_1	3	0	0
T_2	3	4	0
T_3	0	6	7
T_4	0	0	4

	T_1	T_2	T_3	T_4
B_1	2	3	0	0
B_2	3	3	2	0
B_3	0	4	3	2
B_4	0	0	2	4

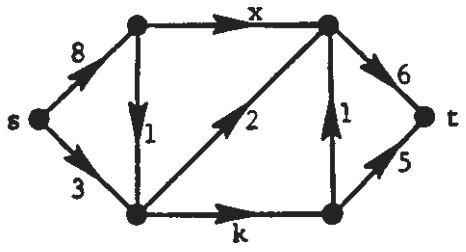
a) Find the maximal power Φ , which can be used by the consumers.

b) We want to increase Φ , by increasing the capacity of one of the high voltage lines by 1 MW . Which lines can be used for this purpose?

c) INstead of increasing Φ like in 2) you could transfer a certain power capacity of 1 MW from E_i to E_j . For which (i, j) will Φ become increased?

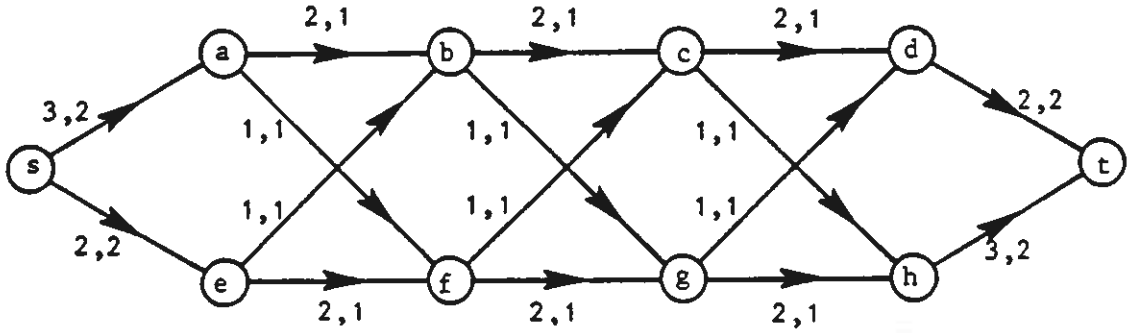
3.12. Find, by means of algorithm maximal flow, the structure rank of the matrix

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



3.13. Find the optimal capacity of the edge k as a function of the variable capacity x . (x is not necessarily an integer.)

3.14.



1) The figure shows a transport network with a flow f . Execute MARKER(f,s) , and use the augmenting path you have found to increase the flow value.

In the rest of this exercise, N in an arbitrary transport network with a flow f . N has K edges and P vertices.

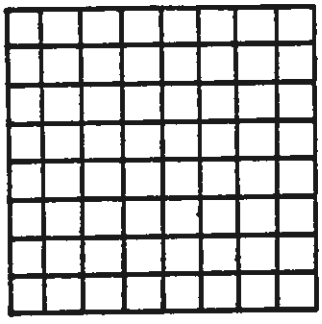
2) When MARKER(f,s) has stopped, have you then determined all augmenting paths with respect to f , which start in s ?

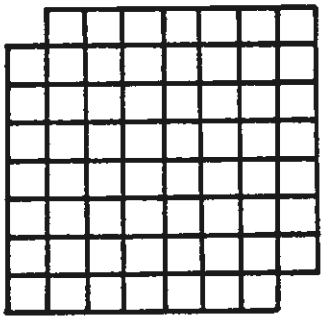
3) Assume that in advance we have ordered the edges in a certain cyclic order,

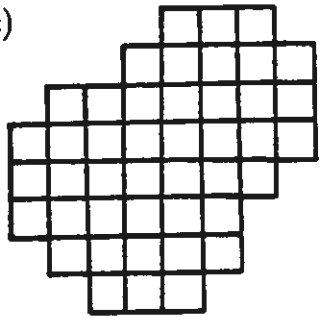
$$\boxed{k_1 \rightarrow k_2 \rightarrow \ldots \rightarrow k_K} .$$

When you want to investigate whether or not there exists an edge which satisfies the conditions of line 3 or line 5 in MARKER(f,s) , let us assume that we look upon these edges in cyclic order. Give an upper bound for the number of times you may have to consider each edge.

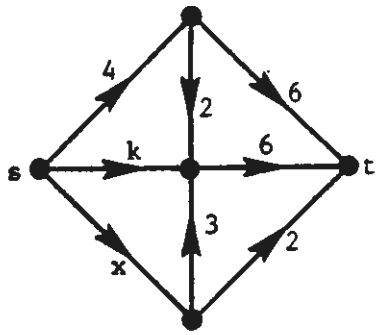
3.15. A domino is a figure like this:  . How many dominoes can be cut out of each of the figures below ?

a) 

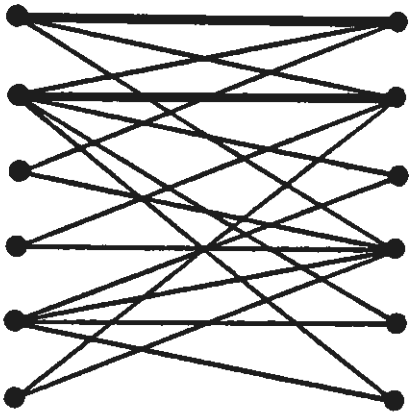
b) 

c) 

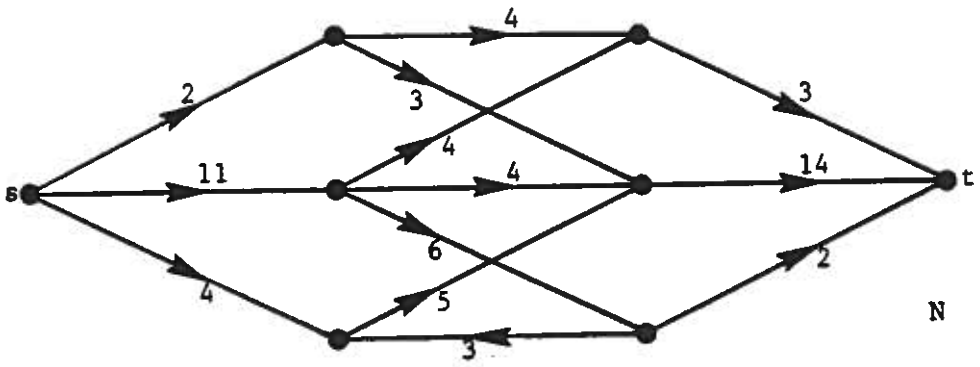
3.16. Find the optimal capacity of the edge k as a function of the variable capacity x . (x is not necessarily an integer.)



3.17. Use algorithm maximal flow to change the matching, which consists of the two heavy lined edges, to a maximal matching. (Show the augmenting paths you use.) Find a minimal notecover in the graph.



3.18.

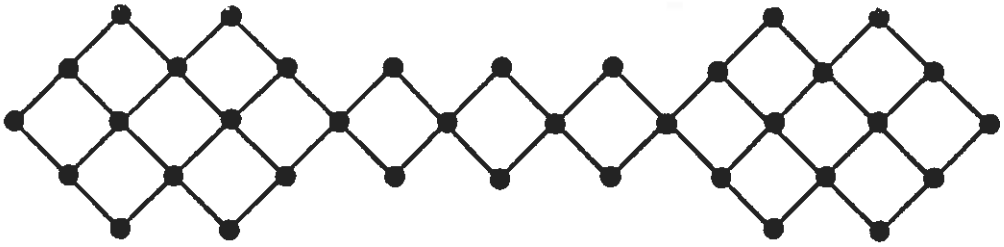


a) Find a maximal flow and the minimal cut in N .

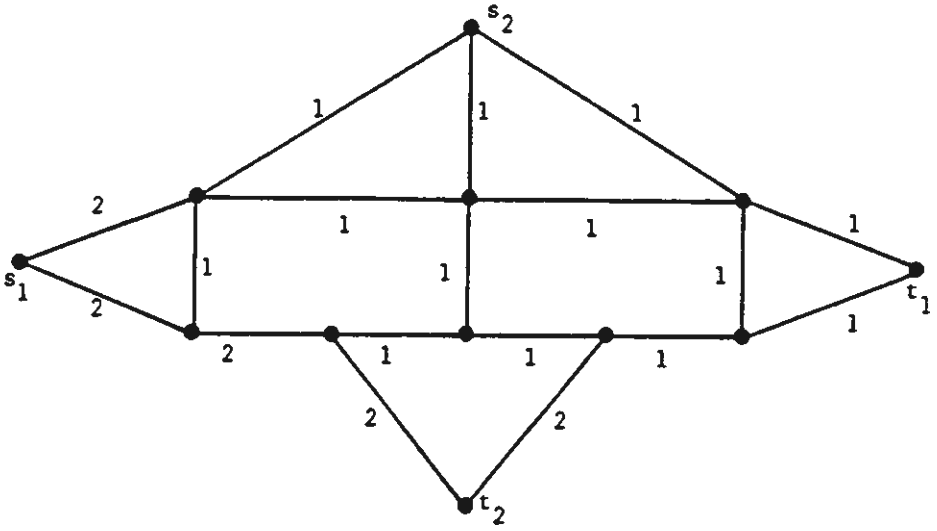
b) Which of the 3 edges, which start in s , has the following property: If we decrease the capacity of the edges by 1 then the value of a maximal flow decreases by 1 ?

c) Find among those cuts in N which contain the edge with capacity 6, one with the smallest possible capacity.

3.19. Find a maximal matching in the graph in the figure.

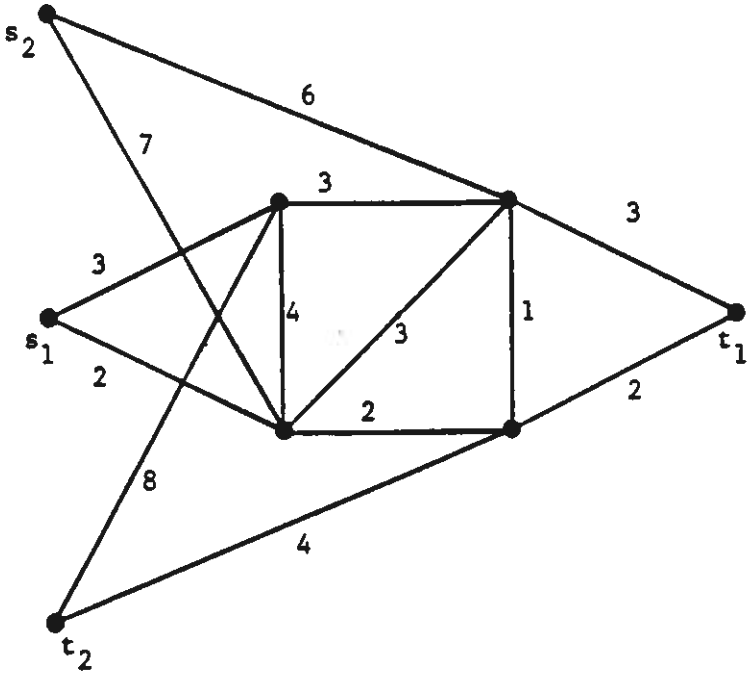


3.20.



Find, by means of algorithm two commodity maximal flow, a maximal flow in the two commodity netwrok in the figure. When the algorithm has stopped, show by means of labellings + (respectively –) those vertices which can be reached by using a positively (respectively –) path fron s_2 . Find a minimal cut in the network.

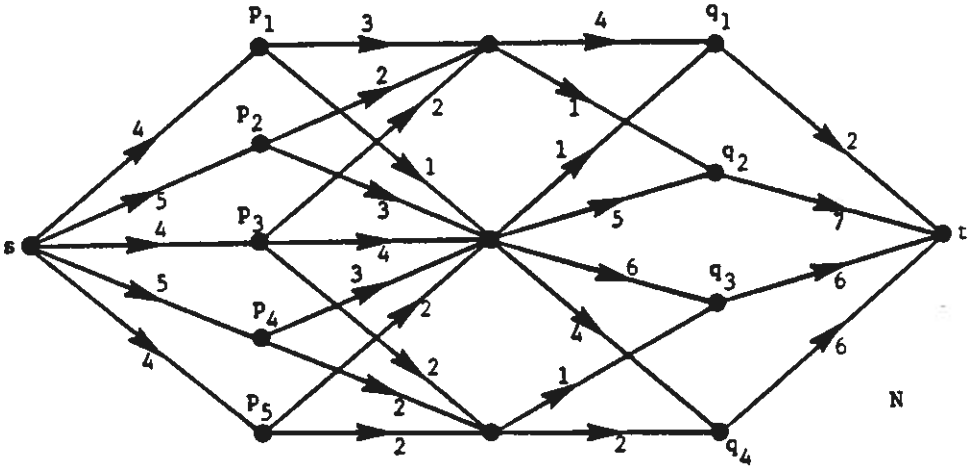
3.21.



Find a maaximal flow and a minimal in the two commodity flow network in the figure.

3.22. Consider the network of problem 3.4. Give the smallest and the largest flow value which can occur after one execution of the big loop in Dinics algorithm.

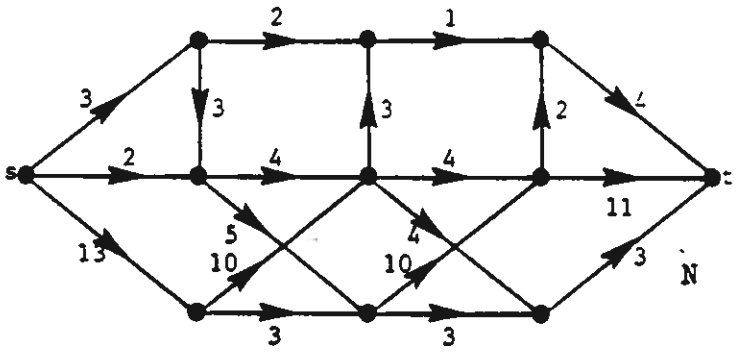
3.23.



a) Find a maximal flow in N and the value F of this flow.

b) A network N_{ij} is obtained from N by adding a $p_i \rightarrow p_j$-edge with capacity 1. For which values of i and j is the value of a maximal flow in N_{ij} greater than F ?

3.24.



a) Give a maximal flow and a minimal cut in the the transport network N in the figure.

b) Does there exist more than one minimal cut in N ? Find those edges which have optimal capacity.

c) We assume that we use Dinics algorithm to deetermine a maximal flow in N. After one execution of the big loop in the algorithm we have found the flow denoted f. Explain how the algorithm constructs f

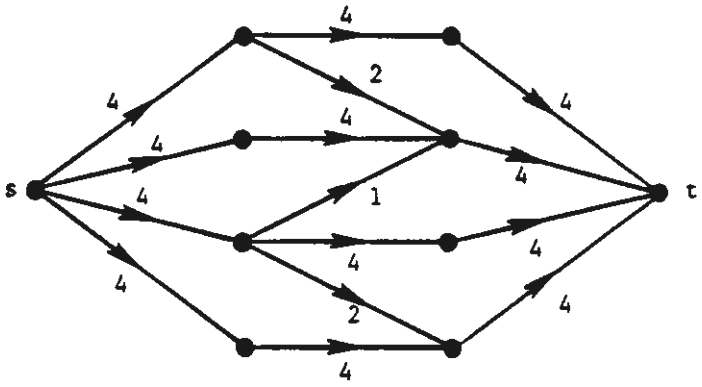
1) in the case where f has the largest value possible,
2) in the case where f has the smallest value possible.

Find in both cases the value of f.

3.25. There is given a transport network N with arbitrary real capacities > 0. Describe an algorithm to determine all those edges in N, which has optimal capacity. Try to do the algorithm as effectively as possible. Show that it works. What is the complexity of your algorithm ?

3.26. There is givena transport network N with arbitrary capacities > 0. Describe an algorithm to determine all edges which are empty in all maximal flows. Show that your algorithm works and give the complexity of the algorithm.

3.27. We assume that Edmonds-Karps algorithm is applied on the network in the figure. What is the smallest number of augmenting paths the algorithm can use. Find the largest number N, for which it can happen that the algorithm uses N paths. Draw the paths.



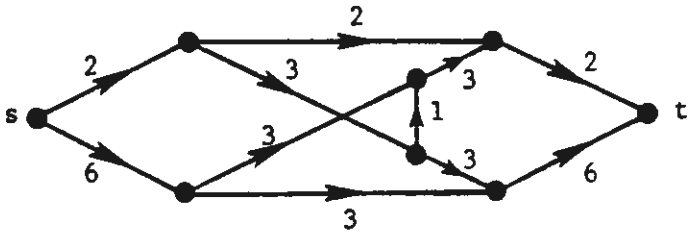
3.28. How many and how few minimal networks can be used when you use Dinics algorithm upon the network in problem 3.27.? Draw the minimal networks

3.29. 3 factories F_1, F_2 and F_3 produces 4, 2 and 22 units of a commodity each day respectively. 3 customers K_1, K_2 and K_3 needs respectively 3, 5 and 20 units each day. The transport system from the factories to the customers has capacities as given, measured in units per day. How are the commodities best distributed? How can it be done better?

	K_1	K_2	K_3
F_1	3	2	6
F_2	4	2	3
F_3	4	6	9

3.30. There is given a directed transport network N with arbitrary capacities > 0. Describe an algorithm which determines those edges in N, which are saturated in every maximal flow in N. Find the complexity.

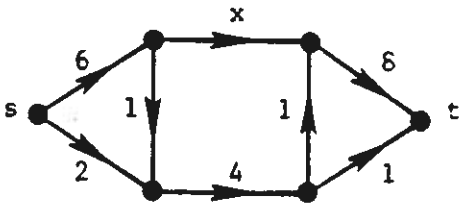
3.31. Find a maximal flow in the network in the figure. Use in the algorithm in each step the shortest augmenting path.



3.32.

a) Find the value of a maximal flow as a function of x .

b) For which values of f is the edge with capacity x critical, and for which has it optimal capacity ?



3.33. Prove the statement given just below the formulation of ALGORITHM MAXIMAL FLOW.

3.34. In section 26, we have explained how a matching $\mathcal{M}$ in a bipartite graph Γ corresponds to an integer flow f in a corresponding trnasport network N . Translate the concept augmenting path w.r.t. f i N to Γ with the matching $\mathcal{M}$.

3.35.

$$\mathbf{A} = \begin{bmatrix} \textcircled{1}1\,1\,1 \\ 0\,0\,1\,0 \\ 1\,1\,\textcircled{1}1 \\ 0\,0\,1\,0 \end{bmatrix}.$$

Find the structure rank of the matrix A by <u>exactly</u> copying the method in the book. Start with a matching in the bipartite graph, corresponding to the two elements in circles, find after that an augmenting path, use it and show after that by means of procedure MARKER that the structure rank is determined. Use the labelling to determine the minimal set of rows and columns which, taken together, contains all elements in A which are different from zero.

3.36. Given an $m \times n$ – matrix A . Describe an algorithm to determine the normal rank of A . This is going to take place in the following way: By changing MARKER(f,s) to a new procedure called MARKER($\mathcal{M}$,A) and changing algorithm MAXIMAL FLOW to an algorithm called NORMAL RANK. During the formulations you should make as few changes as possible, but obtain that your algorithm can work directly on the matrix A , without constructing any transport network. ($\mathcal{M}$ are thos elements in A , which have been taken out and whose elements are in $|\mathcal{M}|$ different rows in A and in $|\mathcal{M}|$ different columns in A .)

3.37. In a transport network N with integer capacities, there are given two optimal edges k_1 and k_2 and a maximal flow f with value F. Describe informally an algorithm which decides whether there exists a minimal $\mathscr{S}$ in N, such that both k_1 and k_2 belong to $\mathscr{S}$. Find the complexity of your algorithm.

3.38. When you want to decide whether an edge k in the transportation network N has optimal capacity or not, we have to find out whether there exists a minimal cut in N, which contains k, and at the same time there exists another minimal cut in N which doews not contain k. This is only possible if you knbow all minimal cuts, and to determine this is normally a very time–consuming task.

Another possibility is the following: First find a maximal flow and its value F. If k is not saturated it is not optimal. If k is saturated and not critical, we can increase the capacity c_k a little bit and find the value F_1 of a maximal flow in the new network. If $F_1 < F$ then k is optimal, otherwise not. This method demands the application of ALGORITHM MAXIMAL FLOW twice. But it can be done even simpler by means of the following algorithm:

algorithm: OPTIMAL EDGE

Input: A transport network N with a maximal flow f and a $x \rightarrow y$ – edge k with $c_k > 0$, which is not critical.

Input: It is decided whether k is optimal or not.

1. MARKER(f, x) .
2. **if** y is marked, then k is not optimal.
3. **if** y is not marked, then k is optimal.

1) Prove the statement in line 2.
2) Prove the statement in line 3.
NOTE:– Keep the two proofs sharply distinguished.

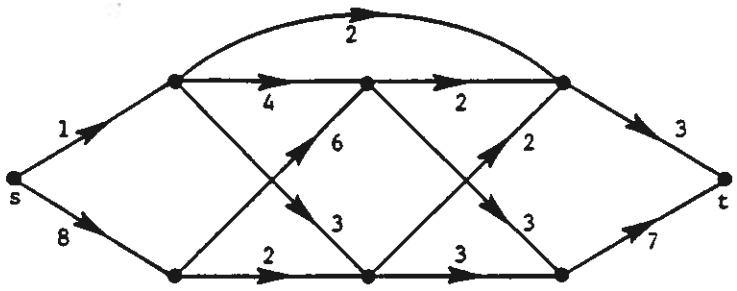
3.39. Draw a transport network, which contains an edge k with the following property: It can happen that EDMONDS – KARPS algorithm performs such that k is saturated and made empty 4 times (i.e. made saturated twice and made empty twice).

Is is possible that in a transport network with 20 vertices, it can happen that it contains an edge which becomes saturated or made empty, taken together 10 times, when EDMONDS – KARPS algorithm is used?

3.40. Show that Dinics algorithm obtains smaller complexity, if it is only used on networks in which all capacities are1 .

3.41. An edge k in a transport network N is called *harmful* if $f(k) = 0$ for every maximal flow f in N . The word harmful is appropriate becaue if there is a little flow through k , then the flow cannot be maximal.

1) Find the harmful edges in the network in the figure.



2) Show that when $\mathscr{S} = (\mathscr{A}, \mathscr{B})$ is a minimal cut, then every $\mathscr{B} \to \mathscr{A}$ - edge is harmful (- Then it is a good question whether there can be other harmful edges.)

Consider the following algorithm:

algorithm: HARMFUL EDGE.

Input: A transport network N with a maximal flow f and a $x \to y$ – edge k .

Output: It is decided whether k is harmful or not.

1. MARKER(f, y) .
2. ***if*** x is labelled, then k is not harmful.
3. **if** x is not labelled, then k is harmful.

3) Prove the statement of line 2.

4) Prove the statement of line 3.

NOTE:– Keep the two proofs sharply distinguished.

REMARK: – By the proof of 4) there appears necessarily a characteristic of the harmful edges.

LITERATURE FOR CHAPTER 3.

[1] Dinic, E.A.: Algorithm for Solution of a Problem of Maximal Flow in a Network with Power Estimation. Sov. Math. Dokl. 11 (1970), 1277-80.

[2] Edmonds, J.: Paths, Trees and Flowers. Canad. J. Math. 17 (1965), 449-67.

[3] Edmonds, J., R.M. Karp: Theoretical Improvements in Algorithmic Efficiency for Network Flow Problems. J. ACM 19 (1972), 248-64.

[4] Ford, L.K., Jr., D.K. Fulkerson: Flows in Networks. Princeton University Pres 1962.

[5] Hall, P.: On Representatives of Subsets. J. London Math. Soc. 10 (1935), 26-30.

[6] Hopcroft, J.E., R.M. Karp: An $n^{\frac{5}{2}}$ Algorithm for Maximum Mathings in Bipartite Graphs. Siam J. Comput. 2 (1973), 225.

[7] Hu, T.C.: Multi-Commodity Network Flows. Opns. Res. 11 (1963), 344--360.

[8] Hu, T.C.: Integer Programming and Network Flows. Addison-Wesley 1969.

[9] Itai, Alon: Two Commodity Flows. TECHNION - Israel Institute of Technology. Computer Science Dept. 1977.

[10] König, Dénes: Theorie der endlichen und undendlichen Graphen. Chelsea Publishing Company 1935.

[11] König, Dénes: Über trennende Knotenpunkte in Graphen (nebst anwendungen auf Determinanten und Matrizen). Acta Litterarum ac Scientiarum 6 (1933), 155-179.

[12] Malhotra, V.M. et al.: An $O(|V|^3)$ Algorithm for finding Maximum Flows in Networks. Indian Institute of Technology, 1978.

CHAPTER 4

GRAPH THEORETIC ALGORITHMS

29. DIJKSTRAS ALGORITHM. Many practical problems can be formulated as problems concerning shortest paths in graphs. For example, assume that we are interested in finding the shortest road between two given towns in the island of Fyn. Then we can consider a graph Γ whose vertices correspond to the towns and road crossings on Fyn and whose edges correspond to the roads on Fyn. For each edge k in the graph there is a number $\ell(k)$ which is called the length of the edge. $\ell(k)$ is equal to the real length of the corresponding shortest road between two towns or crossroads on Fyn. A shortest path between two towns then corresponds to a path V in Γ which connects the corresponding vertices, and for which $\sum_{k \in V} \ell(k)$ is as small as possible. One algorithm by means of which it is possible to determine a shortest path is the following: Represent each vertex in the graph Γ by a little sphere and each edge k by a string of the length $\ell(k)$ tied to the two spheres representing the end vertices of k. If you want to find the shortest path between the two towns you can take the two corresponding spheres and remove them as much as possible from each other. Those strings, which will become tied, correspond to the edges belonging to the shortest path. In the following sections, we shall describe an algorithm which is suitable for execution by a computer.

In section 12 we have treated the case where all edges have length 1 and seen that BREADTH FIRST SEARCH is an O(K)–algorithm which determines the distances from a given vertex s in a graph Γ to all the other vertices in Γ.

We shall now describe the more general concept of distance which we mentioned above: Suppose we have given a directed graph Γ. Furthermore for each edge k in Γ there is a number $\ell(K) > 0$ which is called the *length* of k. By the length of the path V in Γ we understand the number $\ell(V) = \sum \ell(K)$, where the summation is over all edges belonging to V. We want to find an algorithm which for each $p \in \mathcal{P}(\Gamma)$ finds the length $d(p)$ of a shortest $s \to p$–path Γ. $d(p)$ is called the *distance* from s to p. In many practical connections we will only be interested in a shortest path to one given vertex but no algorithm is known which, in the worst case, solves this problem faster than the more general problem. If there are no $s \to p$ paths we put $d(p) = \infty$. Let us assume that the vertices and the edges in Γ are labelled 1, 2,... . Then the edge lengths can be kept in a vector of length K and the distances in a vector of length P.

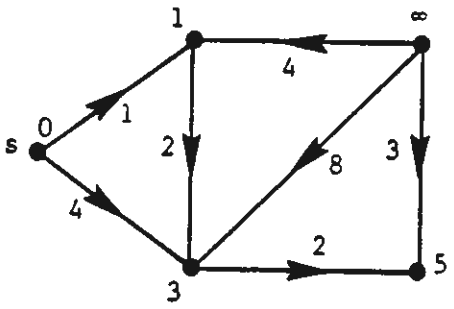
It is easy to see that the distance function $d(p)$ satisfies the following three conditions:

1) $d(s) = 0$.

2) For each vertex $q \neq s$ with a finite value of $d(p)$ there exist a $p \to q$–edge (*an equality edge*) such that $d(q) = d(p) + \ell(k)$.

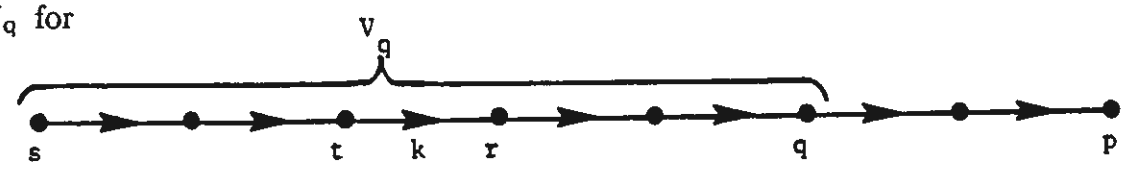
3) For each $p \to q$–edge k, $d(q) \leq d(p) + \ell(k)$.

1) is trivial. 2) follows from the fact that the last edge belonging to the shortest $s \to t$–path will be an equality edge and finally 3) is proved in the following way: If $d(q)$ is finite then $d(q)$ is the length of the shortest $s \to q$–path in V. But it is also possible to get to q by starting in s and going to p and after that using k. Since V is a shortest $s \to q$–path, 3) holds. The figure shows a example where the distances are written at the vertices.



Conversely we shall show that when $d(p)$, $p \in \mathcal{P}(\Gamma)$, is an *arbitrary* function which satsfies 1), 2) and 3) then $d(p)$ *must* necessarily be identical with the shortest distance from s to p, $p \in \mathcal{P}(\Gamma)$. This is proved as follows:

Let $d(p)$ satisfy 1), 2) and 3) and then let the path V_p, shown in the figure, be a shortest $s \to p$–path in Γ. For each vertex q then the $s \to q$–part V_q of V_p must be a shortest $s \to q$–path in Γ. If we start in q and walk against the arrows following equality edges as long as possible we must neccesarily stop in s. Therefore there exists an $s \to q$–path with the length $d(q)$. But since V_q was the shortest $s \to q$–path $\ell(V_q) \leq d(q)$. If $\ell(V_q) < d(q)$, then there must be a first vertex r on V_q for



which $\ell(V_r) < d(r)$. If t denotes the vertex just before r on V_r, we therefore have $\ell(V_t) = d(t)$, but then we have

$$d(r) > \ell(V_r) = \ell(k) + \ell(V_t) = \ell(k) + d(t),$$

contradicting in 3). Therefore $\ell(V_q) = d(q)$, for all q on V_q and in particular $\ell(V_p) = d(p)$. □

When you want to find the distances from s to all other vertices in Γ the work is done when we have found a function d satisfying 1), 2) and 3). This can be done very primitively but if you want low complexity you have to use

DIJKSTRAS ALGORITHM

Input: A directed graph Γ without multiple edges given by a set of neighbourlists in which every edge has a length $\ell(k) > 0$. The lengths are kept in a vector of length k . A vertex s in Γ .

Output: A function d defined on $\mathcal{P}(\Gamma)$ with values ≥ 0 . The values of the function are kept in a vector of length p .

Variables. $\mathcal{S}$ is a set of vertices described by means of a 0, 1 vector of length P .

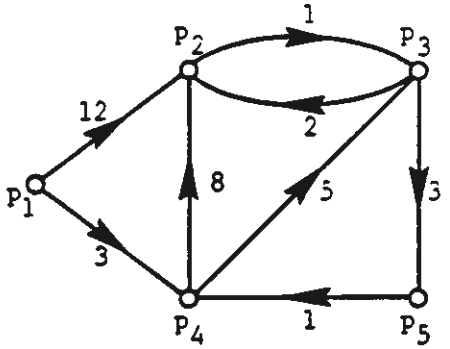
Initialising: $d(s) = 0$, $d(p) = \infty$ for $p \neq s$ and $\mathcal{S} = \{s\}$.

1. Put $d(p) = \ell(k)$ when there exists an $s \to p$–edge k .
2. **While** $|\mathcal{S}| < P$,
3. Choose a vertex $p \in \mathcal{P}(\Gamma) \setminus \mathcal{S}$ such that $d(p)$ is minimum.
4. If $d(p) = \infty$, STOP.
5. $\mathcal{S} = \mathcal{S} \cup \{p\}$.
6. For every $p \to q$–edge k where $q \in \mathcal{P}(\Gamma) \setminus \mathcal{S}$, put

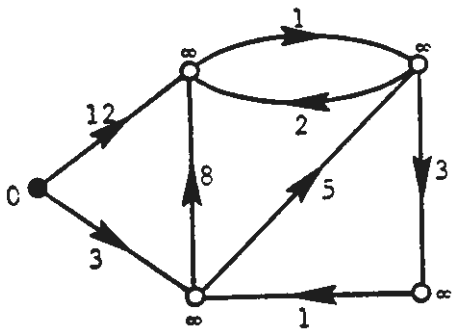
$$d(q) = \min\{d(q), d(p) + \ell(k)\} .$$

7. **STOP.**

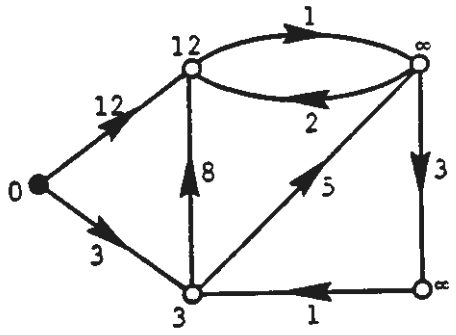
Let us illustrate the way the algorithm works by the example shown in the figure. In the following figures, the vertex p_i is labelled with a number $d(p_i)$, i.e. with the labelling we have found until the situation shown. The set $\mathcal{S}$ consists of the vertices coloured black, the vertex p has a circle around it.



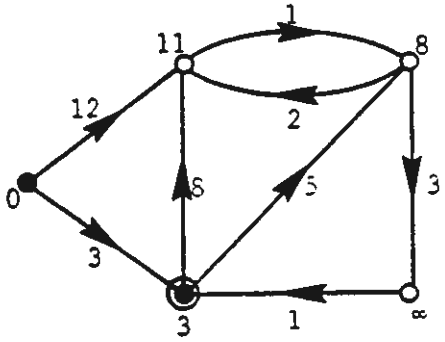
Initialising



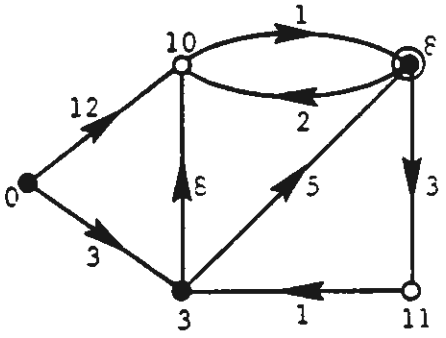
line 1



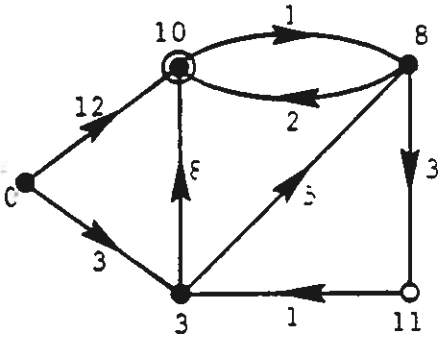
After the first execution of line 6.



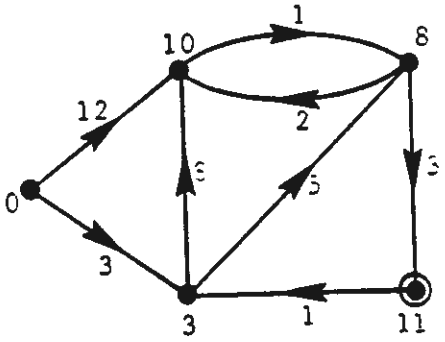
After the second execution of line 6



After the third execution of line 6

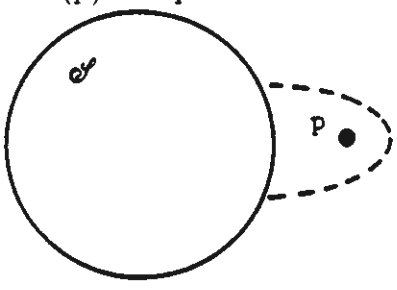


After the fourth execution of line 6



Note that all changes of d(p) decrease d(p) and that when a vertex is put into $\mathscr{S}$, then d(p) is changed no more.

Consider 2 vertices p and q and assume that q is put into $\mathscr{S}$ immediately after p . Then just before p is put into $\mathscr{S}$ $d(q) \geq d(p)$ according to the condition in line 3. Then we update d(q) in line 6 but it cannot be changed to something which is smaller than d(p) . When d(p) is put into $\mathscr{S}$ before q we therefore have $d(p) \leq d(q)$ in the output.

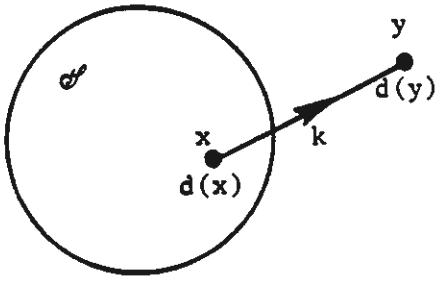


Theorem 29.1. In the output of DIJKSTRAS ALGORITHM $d(p)$ denotes the length of the shortest $s \to p$–path in Γ, $p \in \mathcal{P}(\Gamma)$. The complexity of the algorithm is $O(P^2)$.

Proof. Clearly, the algorithm stops. This follows from the fact that line 6 can be executed at most P times. The same fact determines the complexity since lines 3 to 6 can be executed once in $O(P)$ steps. Therefore the complexity is $O(P^2)$.

We shall next show that in the output, $d(p)$ is the length of a shortest $s \to p$–path in Γ. This is done by showing that $d(p)$ satisfies conditions 1), 2) and 3) from xxx page. Here 1) is completely trivial, 2) follows from the fact that each time the labelling of the vertex is changed in line 1 or line 6 then we create an edge from $\mathscr{S}$ which is an equality edge. We now only have to show that for every $x \to y$–edge k, in the output the inequality $d(y) \leq d(x) + \ell(k)$ will be satisfied. It is enough to show that 3) is valid at the time where the last of the vertices, x and y, is put into $\mathscr{S}$. There are 2 cases:

Case 1: y is put into $\mathscr{S}$ after x. Immediately after we executed line 5 with $p = x$, we updated $d(y)$ in line 6, to a value which satisfies 3). After this has happened we do not change $d(x)$ again and if $d(y)$ is changed it is changed to a smaller number. Therefore, 3) is satisfied in the algorithm.



Case 2: x is put into $\mathscr{S}$ after y. According to the remark just before theorem 29.1, $d(y) \leq d(x) < d(x) + \ell(k)$. Hence we have proved 3).

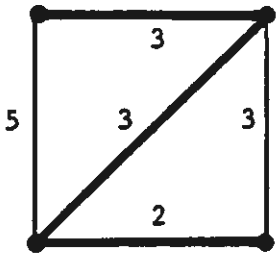
If, finally, $\mathscr{M}$ is the set of those vertices p for which $d(p) = \infty$ in the output, then there is no $\mathscr{S} \to \mathscr{M}$–edge and therefore no $s \to p$–edge in Γ.

It is possible to make DIKSTRAS algorithm a little better by using HEAPSORT at the execution of line 3. This is done in the following way: Those vertices in $p(\Gamma) \setminus \mathscr{S}$ which have finite labelling are kept in a heap ordered with the smallest d–value in the root. The value of $d(p)$ therefore, increases if we follow a path from the root to a leaf. We keep information about which vertices there are in the heap and their

address in the heap, in a vector of length P . This enables us to update $d(q)$ in line 6 in one step. Once this is done, it is possible to re–establish a correct ordering of p in $\log P$ steps. In this way line 3 can be executed in one step; p must be the root. Therefore, the complexity is determined by line 6 and for all possible executions of the while loop, line 6 will be executed, at most, once for each k in Γ , i.e. at most k times, and each execution of line 6 demands, as mentioned, at most, $\log P$ steps.

The result is that if in DIKSTRAS algorithm we use HEAPSORT then the complexity will be $O(K \log P)$. For dense graphs, with $K \geq c \cdot P^2$, this is worse than the original complexity of DIKSTRAS algorithm, for sparse graphs, with $K \leq c \cdot P$, the complexity will be $O(P \log P)$, which is better than DIKSTRAS original version of the algorithm.

30. KRUSKAL'S ALGORITHM. Let us start with a practical problem. A TV company wants to establish cable TV in the towns on Fyn. This is going to be done in the following way: certain pairs of the towns are going to be connected with a cable until we have a connected network connecting all the towns. And this should be done as cheaply as possible.



As you can measure in the map to the left, the two towns nearest to each other are Nyborg and Kerteminde. Therefore we could start by establishing a cable between Nyborg and Kerteminde, and after that continue such that in each step, we establish a cable between two towns which are the smallest possible distance apart, and which are not yet directly or indirectly connected. It is an interesting problem, whether the algorithm we have described here will always give a shortest network. You could imagine that when you start by establishing some short distances of cable, this would later make it necessary to use too many long distances. As we shall see later in this section, the problem can be solved by means of an algorithm which we shall now describe.

Let Γ be a connected graph without multiple edges in which each edge k has a *weight* $w(k)$, which is a real number. We want to determine a spanning tree T in Γ for which the number $w(T) = \sum_k w(k)$ is maximal. The summation is over all edges k in T. A spanning tree of this type is called a *maximal tree* in Γ. The figure above right shows an example where there are two maximal trees.

The edge–set for one of the trees is drawn with heavy lines in the figure. Since T is a spanning tree, the only thing we have to determine is obviously the edge–set $\mathcal{T} = \mathcal{K}(T)$.

The problem can be solved by means of Kruskal's algorithm: Start by sorting the edges in Γ according to decreasing weight. After that, put $\mathcal{T} = \emptyset$, and look at the edges one at a time, in the order mentioned. We add an edge to $\mathcal{T}$ if and only if this does not create a circuit in the part of the tree we have already constructed. We stop when we have a spanning tree. More formally, we can describe the algorithm as follows:

KRUSKAL'S ALGORITHM.

Input: A connected graph Γ represented by a set of edge–lists. A real-valued weight function $w(k)$, defined on $\mathcal{K}(\Gamma)$, represented in a k–dimensional vector. The vertices in Γ are denoted $p_1, p_2, \ldots, p_P$.

Output: The edge–set $\mathcal{T}$ for a maximal tree T in Γ.

Variable: i is an integer variable. $\mathcal{P}_1, \ldots, \mathcal{P}_P$ are vertex–sets described in a P–dimensional vector $\mathbf{v} = (v_1, v_2, \ldots, v_P)$, where $v_i = j \Leftrightarrow p_i \in \mathcal{P}_j$.

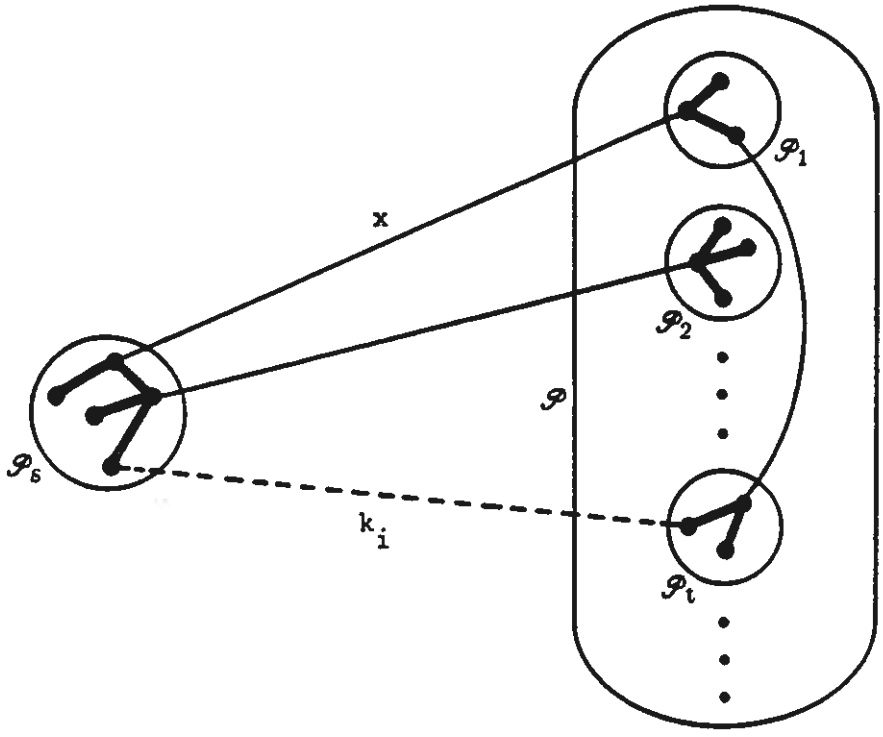
Initializing: $i = 1$, $\mathcal{T} = \emptyset$, $\mathcal{P}_j = \{p_j\}$, $j = 1, 2, \ldots, P$.

1. Sort and name the edges in Γ according to decreasing weight, $\mathcal{K}(\Gamma) = \{k_1, k_2, \ldots, k_K\}$, where $w(k_1) \geq w(k_2) \geq \ldots \geq w(k_K)$.
2. **while** $|\mathcal{T}| \leq P - 1$,
3. find the numbers s and t for which k_i is a $\mathcal{P}_s$, $\mathcal{P}_t$–edge.
4. **if** $s = t$, put $i := i + 1$.
5. if $s \neq t$, put
6. $\mathcal{T} := \mathcal{T} \cup \{k_i\}$
7. $\mathcal{P}_s := \mathcal{P}_s \cup \mathcal{P}_t$, $\mathcal{P}_t := \emptyset$,
8. $i := i + 1$,
9. **STOP.**

Remark, that in the algorithm we only uses the relative sizes of the weights, that is the order $k_1 , k_2 , \ldots , k_K$.

Theorem 30.1 The output $\mathcal{T}$ from KRUSKAL'S ALGORITHM is the edge–set of a maximal tree in Γ . The complexity of the algorithm is $O(\max(K \log K, P^2))$.

Proof. Clearly, the algorithm stops. Next, we shall show that *when a $\mathcal{T}$–set formed during the execution of the algoritm is contained in a maximal tree $\mathcal{T}_m$, then the new $\mathcal{T}$–set constructed in line 6, namely $\mathcal{T} \cup \{k_i\}$, is also contained in a maximal tree.* k_i is a $\mathcal{P}_s$, $\mathcal{P}_t$–edge. In the next figure, we have drawn $\mathcal{T}$ with heavy–lined edges. T_m is shown with full drawn, in particluarly heavy–lined, edges, and we have put $\mathcal{P} = \mathcal{P}(\Gamma) \setminus \mathcal{P}_s$.



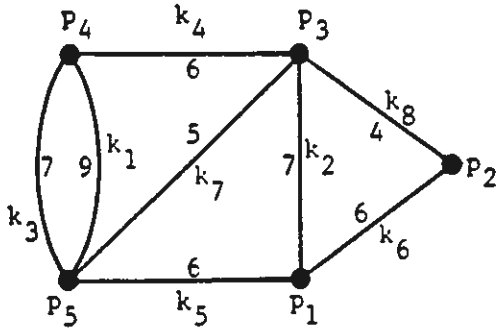
The path in T_m which connects the end–vertices of k_i must contain precisely one $\mathcal{P}_s$, $\mathcal{P}$–edge x . In the sorting, k_i must come before x ; otherwise we would have added x to $\mathcal{T}$ earlier. Therefore $w(k_i) \geq w(x)$. But then the spanning tree with the edge–set $\mathcal{K}(T_m) \setminus \{x\} \cup \{k_i\}$ must also be maximal (and $w(k_i)$ must be equal to $w(x)$), and this tree contains $\mathcal{T} \cup \{k_i\}$. So, we have proved the statement in italics.

Since we start with $\mathcal{T} = \emptyset$, which is contained in a maximal tree, by applying the statement P–1 times we obtain that we end up with the edge–set of a maximal spanning tree. Concerning the complexity, there are only two lines which cannot be executed in constant time. These are the sorting in line 1, which demands O(K log K) steps (theorem 15.1), and line 7, which demands O(P) steps. Since the **while** loop can be executed up to K times, but only can execute lines 6, 7, 8 P times, the complexity will be

$$O(K \log K) + O(K) + O(P^2) = O(K \log K) + O(P^2) . \qquad \square$$

For sparse graphs, the complexity will be $O(P^2)$, for dense graphs $O(K \log K)$.

As an example of the application of Kruskal's algorithm, we shall consider the graph in the figure.



The numbers are weights, and the labellings of the edges are chosen according to decreasing weight. Kruskal's algorithm runs as follows:

i		1	2	3	4	5	6
$\mathcal{T}$	$\emptyset$	k_1	k_1, k_2		k_1, k_2, k_4		k_1, k_2, k_4, k_6
$\lvert\mathcal{T}\rvert$	0	1	2	2	3	3	4
$\mathcal{P}_1$	P_1	P_1	P_1, P_3		P_1, P_3, P_4, P_5	x	P_1, P_2, P_3, P_4, P_5
$\mathcal{P}_2$	P_2	P_2	P_2		P_2		$\emptyset$
$\mathcal{P}_3$	P_3	P_3	$\emptyset$		$\emptyset$		$\emptyset$
$\mathcal{P}_4$	P_4	P_4, P_5	P_4, P_5		$\emptyset$		$\emptyset$
$\mathcal{P}_5$	P_5	$\emptyset$	$\emptyset$		$\emptyset$		$\emptyset$

If, instead of step 1, we sort the edges in Γ according to increasing weight, then the output from Kruskal's algorithm will be a minimal tree in Γ, that is a spanning tree in Γ with minimal sum of weights. This can be seen as follows: Let N be a number which is greater than all weights. If $\mathcal{K}(\Gamma) = \{k_1, k_2, k_3, \ldots, k_n\}$ is a sorting according to increasing weight, then the same sorting is a sorting by decreasing weight when you use the weight–function

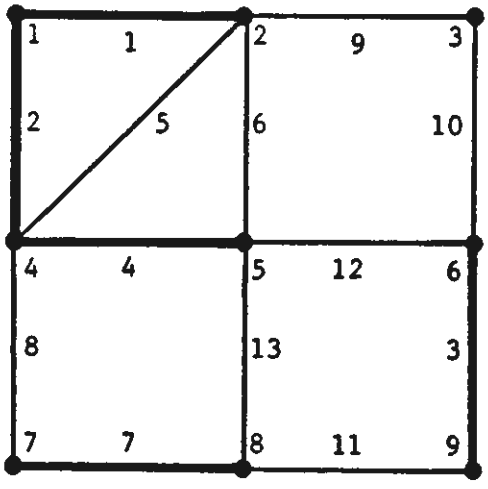
$$w'(k) = N - w(k) .$$

The output will therefore be a tree $\mathcal{T}$, which is maximal with respect to w', but since

$$\sum_{k \in \mathcal{T}} w'(k) = (P-1) \cdot N - \sum_{k \in \mathcal{T}} w(k) ,$$

and since $(P-1) \cdot N$ is a constant independent of $\mathcal{T}$, $\mathcal{T}$ must be minimal w.r.t. w. An example of the application of this minimal version of Kruskal's algorithm is the cable network example in the introduction to this section.

By choosing a more sophisticated data structure, it is possible also for sparse graphs to reduce the complexity to $O(K \log K) = O(P \log P)$. To obtain a simple description of this method, we assume that in line 1 we replace the name k_i with i; the edges in Γ are therefore denoted $1, 2, 3, \ldots, K$, ordered by decreasing weight. The data structure which we shall use is easiest to explain by means of an example. In the figure, we assume that the **while** loop is executed 7 times, and that $\mathcal{T}$ is the set of heavy–lined edges.



The data structure consists of 5 parts:

1) Edgelists;

$1 \to (1, 2)$, $2 \to (1, 4)$,
$3 \to (9, 6)$, $4 \to (4, 5)$,
$5 \to (4, 2)$, $6 \to (2, 5)$,
$7 \to (7, 8)$, $8 \to (7, 4)$,
$9 \to (2, 3)$, $10 \to (3, 6)$,
$11 \to (8, 9)$, $12 \to (6, 5)$,
$13 \to (5, 8)$

2) A P–dimensional vector $\mathbf{v}$, which describes the sets $\mathcal{P}_s$:

$\mathbf{v} = (1, 1, 3, 1, 1, 9, 7, 7, 9)$

3) A set of linked lists $L(S)$, each describing a P_s–set:

$L(1)$: $5 \to 4 \to 2 \to 1 \to \emptyset$
$L(3)$: $3 \to \emptyset$
$L(7)$: $7 \to 8 \to \emptyset$
$L(9)$: $6 \to 9 \to \emptyset$

4) A P–dimensional vector $\mathbf{S}$, which describes the cardinality of the P_s–sets:

$\mathbf{S} = (4, 0, 1, 0, 0, 0, 2, 0, 2)$

5) A K–dimensional vectore $\mathbf{T}$, which contains information about $\mathcal{T}$; furthermore a variable, which contains $|\mathcal{T}|$:

$\mathbf{T} = (1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0)$, $|\mathcal{T}| = 5$.

We shall now explain how many steps we need to execute one of the lines in the **while** loop once, including the updating of the data structure. It is immediately clear that each of the lines 2, 3, 4, 5, 6 and 8 can be executed in constant time, and since the loop can be exectued at most K times, this gives an overall complexity of $O(K)$. And now to line 7: We shall execute this line in the following way: First we determine $|\mathcal{P}_s|$ and $|\mathcal{P}_t|$ by using the vector $\mathbf{S}$. We may assume that $|\mathcal{P}_s| \geq |\mathcal{P}_t|$. It is now very important that *as a name for* $|\mathcal{P}_s| \cup |\mathcal{P}_t|$ *we apply the name of the*

large set, that is $\mathcal{P}_s$. This we do in the following 3 steps:

1) First we search through L(t) once, and for each of the vertices, we change the corresponding coordinate in **v** to s .

2) After that, we update **S** ; in particular, we put the t–th coordinate to 0 .

3) Finally, we update the list s by linking the t–th list after the s–th list.

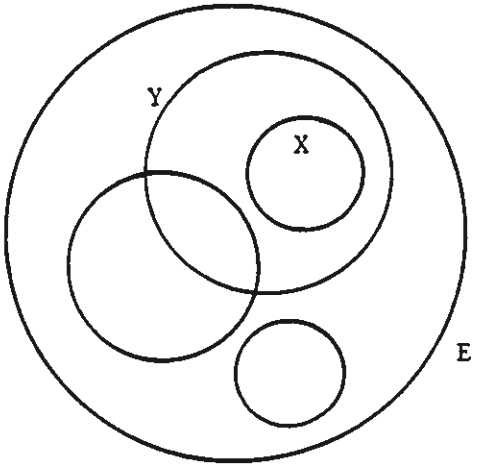
Here, 2) and 3) can be exectued in constant time, and therefore contribute (in all executions of the **while** loop) O(K) to the complexity. We shall now calculate the number of steps it takes to execute 1). And now comes the decisive point in the construction. Let us consider a fixed vertex $p \in \mathcal{P}(\Gamma)$. *Each time the* $\mathcal{P}_i$–set, *which* p *is contained in, gets its name changed in line 7, it becomes at least twice as big as it previously was.* If n therefore denotes the number of times the $\mathcal{P}_i$–set which p belongs to changes name in line 7, we have that $2^n \leq P$, from which $n \leq \log P$. During all the executions of line 7, p therefore can be met log P times in L(T) , and since this is valid for every vertex in Γ , all executions of line 7 can be done in O(P log P) steps. The result of these considerations is that the complexity is determined by line 1. Therefore the complexity is O(K log K) :

Theorem 30.2 Kruskal's algorithm can be performed such that it will have complexity O(K log K) .

The sorting in line 1 is best performed in such a way that we organise the edges in a heap (see section 15, last pages), and then select k_i in line 3 as the root of the heap.

There is a characteristic difference between Kruskal's algorithm on the one hand, and on the other hand, the algorithm described on p. 3.47 to determine a maximal matching in a bipartite graph: When you use the matching algorithm, you normally encounter the situation several times, where you have to change the choices you had already made. First you take in an edge in the matching algorithm, then you take it out again, and maybe later you take it in again. In Kruskal's algorithm, it is the other way around: once you have joined an edge to $\mathcal{T}$, you will never regret it. We shall therefore say that Kruskal's algorithm is a *greedy* algorithm. In the next section we shall characterise the problems which can be solved by means of greedy algorithms.

31. THE GREEDY ALGORITHM. This section does not deal with graphs but generalizes certain graph–theoretical algorithms, for example, Krudal's algorithm. We shall consider sets of subsets of a given set E. Every subset of E is denoted by a script letter.



Let E denote an arbitrary non–empty set with n elements and let $\mathcal{U}$ be a non–empty set of subsets of E. $\mathcal{U}$ is called an *independence system on a fundemental set* E (or just on E) if $Y \in \mathcal{U}$, and $X \subseteq Y$ implies that $X \in \mathcal{U}$. The sets $X \in \mathcal{U}$, $X \subseteq E$ are called *independent*; a set $Z \subseteq E$ such that $Z \notin \mathcal{U}$ is called *dependent*. Let us consider some examples, where $E = \{1, 2, 3, 4\}$:

$\mathcal{U}_1 = \{\emptyset, \{1\}, \{2\}, \{1,2\}\}$ is an independent system.
$\mathcal{U}_2 = \{\{1\}, \{2\}, \{1,2\}\}$ is not an independence system.
$\mathcal{U}_3 = \{\emptyset, \{1\}, \{2\}, \{3\}, \{1, 2\}, \{1, 3\}, \{1, 2, 3\}\}$ is not an independence system.
$\mathcal{U}_4 = \{\emptyset, \{1\}, \{2\}, \{3\}, \{4\}, \{1, 2\}, \{1, 3\}, \{2, 3\}\}$ is an independence system.

One meets sets of subsets of a given set in many mathematical disciplines:

For example, let V be a finite set of vectors from a vector space. If $\mathcal{U}_0$ is defined such that $X \subseteq V$ belongs to $\mathcal{U}_0$ if and only if X consists of linearly independent vectors, then $\mathcal{U}_0$ is an independence system on V. (From this example we have taken the name "independence system"). The set $\mathcal{A}_0$ of linearly dependent subsets of V is not an independent system on V.

Let us illustrate the concept of an independence system by means of some graph–theoretical examples. In the following examples, Γ denotes a given fixed graph. If $E = \mathcal{K}(\Gamma)$ and if $\mathcal{U}_c$ is the set of circuit free subsets of E, then $\mathcal{U}_c$ is an independence system. The set of those subsets of E which contain a circuit is not an independence system, but the set of those subsets of E each of which is contained in some circuit in Γ is an independence system if we add the empty set.

We can also take $E = \mathcal{P}(\Gamma)$. The set $\mathcal{U}$ of those subsets $\mathcal{U} \subseteq E$ for which there exists a matching in Γ , such that all vertices in $\mathcal{U}$ are matched is an independence system, but the set of vertex–sets of connected subgraphs in Γ is not an independence system.

In several of these examples it is natural to ask for an independent set with maximal cardinality and therefore, it is natural to look for an algorithm to solve this problem. However, the concept of an independence system is much too general to make it possible to find a good algorithm to determine an independent subset with maximal cardinality in an arbitrarily given independence system.

Before we proceed to describe an algorithm which can be used in many cases to determine a cardinality–maximal independent set in an independence system, we shall carry out a natural generalization of the problem.

Let $\mathcal{U}$ be an independence system on E and suppose that on E there is a given real–valued function $w: E \to \mathbb{R}$, such that $w(x) \geq 0$ for all $x \in E$. w is called a weight function and $(\mathcal{U}, w)$ is called a weighted independence system. We look for an independent set $X \subseteq E$, such that $w(X) = \sum_{x \in X} w(x)$, which is called the weight of X , is maximal.

Again the problem is formulated so general that, in general, one can solve it only by testing all possibilities, but in many special cases one can use a very fast algorithm which is called "The Greedy Algorithm", (GA).

THE GREEDY ALGORITHM.

Input: A weighted independence system $(\mathcal{U}, w)$ on a set E .

Output: A set $B_{GA} \subseteq E$.

Variables: A count–variable t .

Initialization: $t = 1$, $B_{GA} = \emptyset$.

1. Sort the elements in E according to decreasing wights, $E = \{e_1, e_2, \ldots, e_n\}$ where $w(e_i) \geq w(e_j)$ for $i < j$.
2. **While** $t < n + 1$,
3. if $B_{GA} \cup \{e_t\} \in \mathcal{U}$ put $B_{GA} := B_{GA} \cup \{e_t\}$.
4. $t := t + 1$.
5. **STOP.**

Remark: The output is not uniquely determined by the input. If there are several elements with the same weight, then line 1 can be executed in different ways. GA is said to *work on* $(\mathcal{U}, w)$ if, for every possible way to execute line 1, the output is an independent set with maximal weight sum.

Remark: The algorithm uses only relative sizes, the absolute values play no role.

Let us show how the Greedy algorithm works on a couple of examples: Let E be the set $\{v_1, v_2, v_3, v_4, v_5, v_6\}$ of 6 vectors in $\mathbb{R}^4$, where

$$\begin{aligned} v_1 &= (1, 1, 2, -1) \\ v_2 &= (2, 2, 4, -2) \\ v_3 &= (3, 0, 0, 1) \\ v_4 &= (0, 0, 0, 0) \\ v_5 &= (4, 1, 2, 0) \\ v_6 &= (1, 1, 1, 1) \end{aligned}$$

Let $\mathcal{U}$ be the set of systems of linearly independent vectors in E. Put $w(v_i) = 1$ for all i. If we assume that in line 1, the elements are sorted in the given order, line 3 can be carried out as follows:

t		1	2	3	4	5	6
B_{GA} :	$\emptyset$	$\{v_1\}$	$\{v_1\}$	$\{v_1, v_3\}$	$\{v_1, v_3\}$	$\{v_1, v_3\}$	$\{v_1, v_3, v_6\}$

Thus the output is $\{v_1, v_3, v_6\}$ which is a maximal system of linearly independent vectors in E. Note that the usual method of determining a maximal system of linearly independent rows in a matrix is a special case of the Greedy algorithm.

As another example, let us consider the independence system

$$\mathcal{U} = \{\emptyset, \{1\}, \{2\}, \{3\}, \{4\}, \{5\}, \{6\}, \{1, 2\}, \{3, 4\}, \{3, 5\}, \{3, 6\}, \{5, 6\}, \{3, 5, 6\}\},$$

with the following weight functions:

$$w(1) = 4\,,\ w(2) = 5\,,\ w(3) = 6\,,\ w(4) = 1\,,\ w(5) = 1\,,\ w(6) = \frac{1}{2}\,.$$

The sorting in line 1 can be carried out in two ways:

a) 3, 2, 1, 4, 5, 6 b) 3, 2, 1, 5, 4, 6

The corresponding output of the Greedy algorithm will be:

a) $B_{GA} = \{3, 4\}$. $w(B_{GA}) = 7$.

b) $B_{GA} = \{3, 5, 6\}$. $w(B_{GA}) = 7.5$.

But it can be done better: An independent set with maximal weight sum is $\{1, 2\}$ with $w(\{1, 2\}) = 9$.

In general it is impossible to determine the complexity of the algorithm; it depends on the complexity of the algorithm which is, in a concrete case, used to investigate whether or not a set is independent. It *is* clear however, that if this can be done in polynomial time then the Greedy algorithm is also polynomial (line 1 can be executed in O(n–log n) steps. If w(e) is either 0 or 1 for each $e \in E$ then w is called a {0, 1}–weight–function . A decisive property of the Greedy algorithm is expressed in the following theorem:

Theorem 31.1. Let $\mathcal{U}$ be an independent system on E with n elements. Then we have: GA works on every {0, 1}–weight–function if and only if GA works on every weight function.

Proof. $\Leftarrow$ is trivial. $\Rightarrow$: Consider an arbitrary weight–function w and let

$$E = \{e_1, e_2, \cdots, e_n\}$$

be the sorting obtained from line 1 where $w(e_i) \geq w(e_j)$, for $i < j$. Let

$$B_{GA} = \{e_{i_1}, e_{i_2}, \cdots, e_{i_p}\}$$

denote the output of GA where $(i_1 < i_2 < \cdots < i_p)$ and assume that B_{GA} is not an independent set with maximal weight sum.
Let

$$B_{OPT} = \{e_{j_1}, e_{j_2}, \cdots, e_{j_q}\},$$

where $j_1 < j_2 < \cdots < j_q$, be an independent set with maximal weight–sum.

(1) $$w(B_{OPT}) > w(B_{GA}).$$

If $p < q$ then we obtain a contradiction by using a $\{0, 1\}$–weight–function, where all weights are 1 and where the sorting is as before, because in this case, the output B_{GA} from GA will have weight–sum p which is not maximal because B_{OPT} has weight sum q. Therefore $p \geq q$. Then there exists a $k = 1 \leq k \leq q$ such that

(2) $$w(e_{j_k}) > w(e_{i_k})$$

Suppose namely, $w(e_{j_k}) \leq w(e_{i_k})$ for $k = 1, 2, \cdots, q$

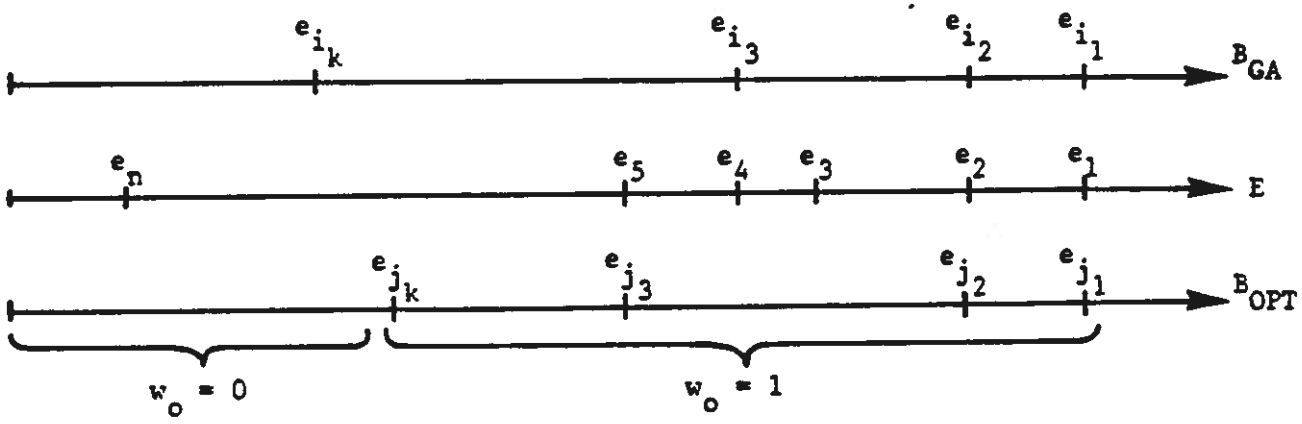
then we get by summation

$$w(B_{OPT}) = \sum_{k=1}^{q} w(e_{j_k}) \leq \sum_{k=1}^{q} w(e_{j_k}) \leq \sum_{k=1}^{p} w(e_{j_k}) = w(B_{GA}) ,$$

which is a contradiction to (1). Suppose now that k satisfies (2) and use a $\{0,1\}$–weight–function w_0 in GA which is defined such that

$$w_0(e_j) = \begin{cases} 1 & \text{for } i \leq j_k \\ 0 & \text{for } i > j_k \end{cases} ,$$

(compare the figures where the position of an element shows its weight in the weight–function w) and suppose that line 1 is executed as before. Then B_{GA} is still the output of GA and $w_0(B_{GA}) \leq k-1$, but $w_0(B_{OPT}) = k$; this is a contradiction to the fact that GA works for every $\{0, 1\}$–weight–function. □



Now we define a matroid M on E as a pair $(E, \mathcal{U})$ where $\mathcal{U}$ is a system of subsets of E which are called independent sets (in M) and which satisfies the following three axioms:

U1. $\emptyset \in \mathcal{U}$.

U2. If $X \subseteq Y \in \mathcal{U}$ then $X \in \mathcal{U}$.

U3. If $X \in \mathcal{U}$ and $Y \in \mathcal{U}$ and $|Y| = |X| + 1$ then there exists a $y \in Y \setminus X$ such that $X \cup \{y\} \in \mathcal{U}$.

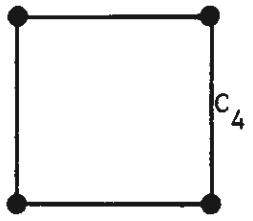
We write $M = (E, \mathcal{U})$. U2 means that every matroid is an independence system. A basis of M is a relatively maximal independent set, i.e. an independence set $B \subseteq E$ for which for all $e \in E \setminus B$ we have that $B \cup \{e\} \notin \mathcal{U}$. Let B_1 and B_2 be bases of M. If $|B_1| < |B_2|$ then it follows from U3 that there exists a $x \in B_1 \setminus B_2$ such that $B_2 \cup \{x\} \in \mathcal{U}$. Since this is a contradiction to the fact that B_2 is a maximal independent set, all bases of a matroid M have the same cardinality. This cardinality is called the rank of the matroid and is denoted by $\rho = \rho(M)$. Thus, we have

Theorem 31.2. All bases of a matroid have the same cardinality.

If $M = (E, \mathcal{U})$ is a matroid and $X \subseteq E$ we define the rank $\rho(X)$ as the number of elements in the largest possible independent subset of X. ρ is called the rank–function of the matroid.

Let us mention some important examples of matroids:

Graphical matroids. Let Γ be an arbitrary graph, put $E = \mathcal{K}(\Gamma)$ and let $\mathcal{U}$ denote the set of circuit free subsets of E. It is clear that U1 and U2 are satisfied and that U3 is also satisfied can be proved as follows: Let X and Y be given circuit free subsets of $\mathcal{K}(\Gamma)$ such that $|Y| = |X| + 1$. Then X is an edge set of a spanning forest S in Γ. Let T be one of the trees in S, ie. one of the connected components in S. The number of edges in Y which connect two vertices in T cannot be greater than the number of X–edges in T. But since Y is greater than X there must be an edge K in Y which connects two different trees in S. Thus, $X \cup \{k\} \in \mathcal{U}$. Therefore, $M = (e, \mathcal{U})$ is a matroid. It is called the *circuit–matroid* of Γ and is denoted by $M(\Gamma)$. If Γ is connected then a basis of $M(\Gamma)$ is the edge set of a spanning tree in Γ. A dependent set in $M(\Gamma)$ is a subset of $\mathcal{K}(\Gamma)$ which contains a circuit. A matroid is called *graphical* if it is a circuit–matroid for some graph.



Uniform Matroids. Let E be a set with n elements and let $k \geq 0$ be an integer $\leq n$. If $\mathcal{U}$ denotes the set of subsets of E which have $\leq k$ elements then $(E, \mathcal{U})$ is a matroid. It is denoted by $M_{n,k}$ and is called a k–*uniform matroid* on a set with n elements.

It is clear that the matroid axioms are satisfied. Matroids which can be obtained in this way are called uniform matroids. The circuit–matroid for C_4 is $M_{4,3}$ since k arbitrary edges form a circuit free subset of C_4 if $k \leq 3$. The matroid $M_{4,2}$ is not graphical because if $M_{4,2} = M(\Gamma)$, Γ would be a graph with 4 edges where 3 arbitrary edges form a circuit but such a graph does not exist. The basis of $M_{n,k}$ are subsets of E which have cardinality k and the circuits are those subsets of E with cardinality $k+1$. The rank function $\mathcal{S}$ of $M_{n,k}$ is defined such that

$$\rho(A) = \begin{cases} |A| & \text{for} \quad |A| \leq k \\ |k| & \text{for} \quad |A| > k \end{cases}.$$

Vector Matroids. Let V be a vector space, let E be a finite subset of V and let $\mathcal{U}$ be the set of linearly independent subsets of E. Then it is well–known from linear algebra that $M = (E, \mathcal{U})$ satisfies the 3 matroid axioms. Therefore, M is a matroid. Matroids which can be obtained in this way are called *vector matroids.* A basis is a relatively maximal linearly independent subset of E and the matroide–rank of a set $A \subseteq E$ is the usual rank of A.

Transversal matroides. Let Γ be a bipartite graph with colour classes $\mathcal{P}_1$ and $\mathcal{P}_2$, put $E = \mathcal{P}_1$ and define U such that $U \in \mathcal{U}$ if and only if there exists a matching in Γ in which the vertices in U are matched. It is clear that U1 and U2 are satisfied. U3 is also satisfied for if X and Y are independent subsets of $\mathcal{P}_1$ with $|Y| = |X| + 1$ we consider a matching $\mathcal{M}$ with $|X|$ edges such that each vertex is X matched. $\mathcal{M}$ is also a matching in $\Gamma \setminus (\mathcal{P}_1 \setminus (X \cup Y)$, but it cannot be maximal. If we make $\mathcal{M}$ larger by means of an augmenting path we obtain, in addition to X a vertex $y \in Y \setminus X$ matched. Therefore, $X \cup \{Y\}$ is independent. A matroid which can be constructed in this way is called a *transversal matroid.* The rank of $\mathcal{P}_1$ is the number of edges in a maximal matching in Γ.

Remark: The theorem which states that all bases in a vector space have the same cardinality and the theorem whihc states that all spanning trees in a connected graph have the same number of edges are special cases of theorem 31.2. If F is a subset of E and if $\mathcal{U}_F$ denotes the set of those subsets of $\mathcal{U}$ which are contained in F, then

it is trivial that $M_F = (E, \mathcal{U}_F)$ is a matroid on F ; it is called the *restriction* of M to F . With this remark we have proved the $\Rightarrow$ part of

Theorem 31.3. Let $\mathcal{U}$ be an independent system on a set E . Then it is valid that

$$M = (E, \mathcal{U}) \text{ is a matroid } \Leftrightarrow$$

For every subset $F \subseteq E$ all relatively maximal independent subsets of F have the same cardinality.

Proof for $\Leftarrow$. It is given that $\mathcal{U}$ is an independent system for which for every subset $F \subseteq E$ it is valid that all relatively maximal independent subsets of F have the same cardinality. Since U1 and U2 follow form the fact that $\mathcal{U}$ is an independent system we shall show U3 . Let X and Y be independent sets with $|Y| = |X| + 1$. Consider $Z = X \cup Y$. If X was a relatively maximal subset of Z then X would also be a cardinality–maximal, but this cannot be because Y is actually greater. Since X is not relatively maximal there exists a $y \in Z \backslash X = Y \backslash X$ such that $X \cup \{y\} \in \mathcal{U}$. □

Another formulation of theorem 31.1 and theorem 31.3 is

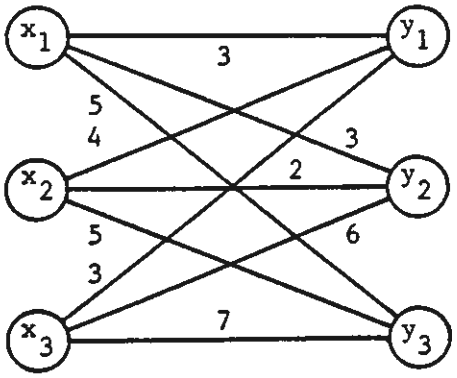
Theorem 31.4. Let $\mathcal{U}$ be an independent system on a set E . Then we have:

$$M = (E, \mathcal{U}) \text{ is a matroid on } E \Leftrightarrow$$

the Greedy algorithm works on $\mathcal{U}$ for every $\{0, 1\}$–weight function $\Leftrightarrow$ the Greedy algorithm works on $\mathcal{U}$ for every weight–function.

If one applies theorem 31.4 on a graphical matroid for a connected graph Γ we obtain that the edge–set for a maximal spanning tree in Γ can be determined by means of the Greedy algorithm. But since line 3 in GA corresponds exactly to line 6 in Krushals algorithm we have a given a new proof for the fact that Krushals algorithm works correctly.

32. MAXIMAL WEIGHT MATCHING. In section 26 we described a method to determine a maximal matching in a bipartite graph Γ. In this section we shall generalize this method to the more complicated case, where each edge k in Γ has a weight $w(k) \geq 0$ and where we want to determine a matching with maximal weight sum. For example, in the graph on the figure there are 6 maximal matchings, namely the following:



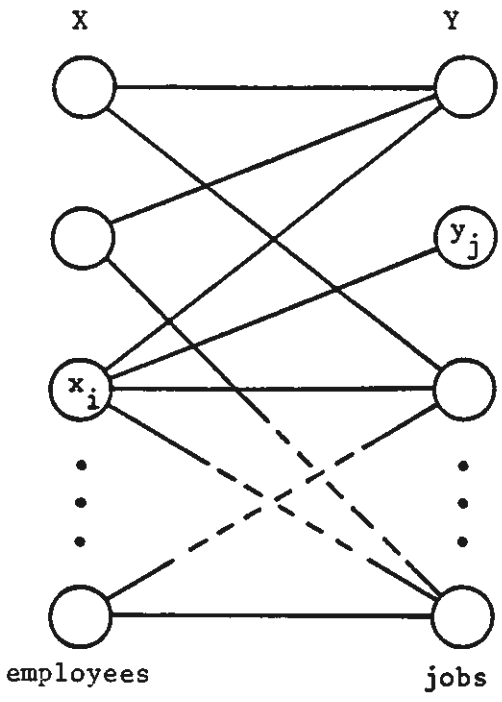
$x_1\,y_1$, $x_2\,y_2$, $x_3\,y_3$ with weight $3 + 2 + 7 = 12$,
$x_1\,y_1$, $x_2\,y_3$, $x_3\,y_2$ with weight $3 + 5 + 6 = 14$,
$x_1\,y_2$, $x_2\,y_1$, $x_3\,y_3$ with weight $3 + 4 + 7 = 14$,
$x_1\,y_2$, $x_2\,y_3$, $x_3\,y_1$ with weight $3 + 5 + 3 = 11$,
$x_1\,y_3$, $x_2\,y_1$, $x_3\,y_2$ with weight $5 + 4 + 6 = 15$,
$x_1\,y_3$, $x_2\,y_2$, $x_3\,y_1$ with weight $5 + 2 + 3 = 10$.

The matching $\mathcal{M} = \{x_1\,y_3, x_2\,y_1, x_3\,y_2\}$ has maximal weight, $w(\mathcal{M}) = \sum_{k \in \mathcal{M}} w(k) = 15$.

Here we have solved the problem, by constructing a list containing all maximal matching. But we can do better!

The problem, we are dealing with is called the *assignment problem* (– or job assignment problem). This name is derived from the following special case:

Assume, that in a company, there are n jobs to be distributed between the n employees. We shall describe the situation by means of a bipartite graph with



bipartition $\mathcal{P}(\Gamma) = X \cup Y$, where both X and Y contain n vertices. Each vertex in X corresponds to a person and each vertex in Y corresponds to a job. In Γ we draw an edge between x_i and y_j when the person x_i is qualified to the job y_j. If this is the case, we assume that a certain weight $w(x_i\, y_j) \geq 0$ is attached to the edge. $w(x_i\, y_j)$ can be a measure for the degree of qualification of person x_i to take care of job y_j. Or it can measure the profit resulting from letting x_i take job y_j. Or it can measure how much x_i wants job y_j. In any case it is natural to ask for an assignment algorithm, which finds a matching with maximal weight.

First note, that it is sufficient to consider the case, where Γ is the complete bipartite graph $K_{n,n}$, and where we are looking for a perfect matching with maximal weight. For if Γ is not $K_{n,n}$ we can add some new vertices and some new edges with weight 0, and create $K_{n,n}$. If $\mathcal{M}$ is a perfect matching with maximal weight in the new graph, then those edges of $\mathcal{M}$, which has weight > 0, is a maximal weight matching in the original graph.

So, in what follows, $\Gamma = K_{n,n}$, with color classes $X = \{x_1, x_2, \cdots, x_n\}$ and $Y = \{y_1, y_2, \cdots, y_n\}$, and each edge $k = xy$ in Γ has a real weight $w(k) = w(xy)$. The weight $w(\mathcal{M})$ of a matching $\mathcal{M}$ is defined to the

$$w(\mathcal{M}) = \sum_{k \in \mathcal{M}} w(k)\,,$$

and we shall find a matching with maximal weight. When we consider an edge $k = xy$ it is to be understood, that $x \in X$ and $y \in Y$.

Sometimes, in particular in examples, it is practical to formulate the problem by using an $n \times n$ – matrix, in which the element in row r and column s (from danish søjle = column) is $w(x_r\, y_s)$. In this formulation, the problem is to find n elements in the matrix, one in each row and one in each column, the weigh sum of which is maximal. For example, the problem above can be described by the matrix

$$\begin{bmatrix} 3 & 3 & \textcircled{5} \\ \textcircled{4} & 2 & 5 \\ 3 & \textcircled{6} & 7 \end{bmatrix},$$

and the solution is the three numbers in circles. The rows in the matrix correspond to the elements in X and the columns correspond to the elements in Y. Therefore we can write

$$\begin{array}{c} \\ x_1 \\ x_2 \\ x_3 \end{array} \begin{array}{c} \begin{array}{ccc} y_1 & y_2 & y_3 \end{array} \\ \begin{bmatrix} 3 & 3 & 5 \\ 4 & 2 & 5 \\ 3 & 6 & 7 \end{bmatrix} \end{array} .$$

Before we (in page 3.15) could prove algorithm maximal flow we had (in page 3.10) to introduce a concept which provides us with an upper bound for the value of a flow in a network. It was the concept of a cut.

Similarly, here we need a concept which can provide us with an upper bound for the weights of the matching in Γ . This concept is called a *vertex labelling.* A vertex labelling in Γ is a real function p , defined on $\mathcal{P}(\Gamma)$, for which

$$p(x) + p(y) \geq w(x, y) \tag{1}$$

for all $x \in X$ and $y \in Y$. It is not difficult to find a vertex labelling. For example $p(z) = \frac{1}{2} M$ for all $z \in \mathcal{P}(\Gamma)$, where M is the maximal edge weight. Or

$$p(x) = \begin{cases} M_z & \text{for } z \in Y , \\ 0 & \text{for } z \in X , \end{cases} \tag{2}$$

where M_z is the maximal weight of the edges incident to z . The *sum* $S(p)$ of a vertex labelling p is defined to be

$$\sum_{z \in \mathcal{P}(\Gamma)} p(z) ,$$

that is, the sum of all vertex labels. Corresponding to lemma 18.3 we now have

> ***Lemma 32.1.*** When p is a vertex labelling in Γ and $\mathcal{M}$ is a matching in Γ , then
>
> $$w(\mathcal{M}) \leq S(p) .$$

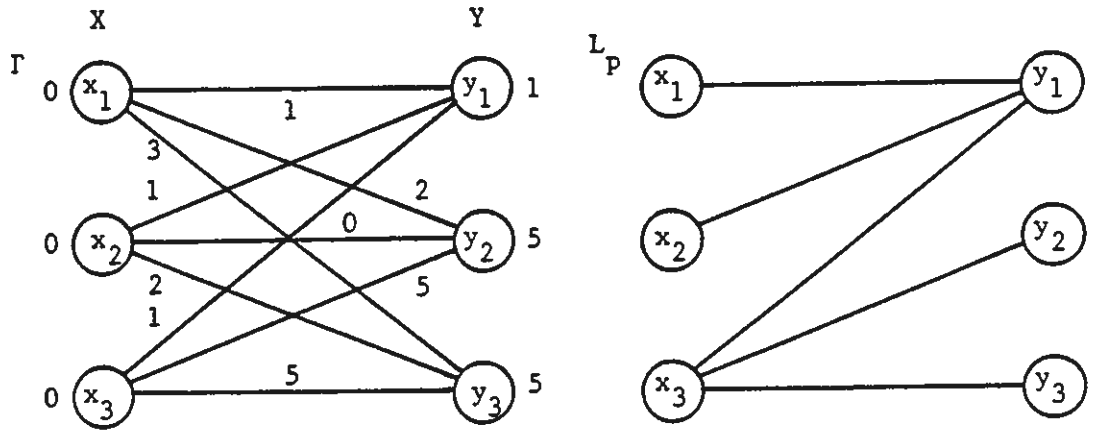
The lemma follows from the fact that already the sum of the vertex labels of the end vertices of the edges in $\mathcal{M}$ is $\geq w(\mathcal{M})$; this follows from (1).

Remark, that from the lemma it follows immediately, that if we encounter a matching $\mathcal{M}$ and a vertex labelling p such that $w(\mathcal{M}) = S(p)$, then $\mathcal{M}$ has maximal weight and p has minimal sum. This remark corresponds to corollary 18.4, page 3.12.

Now, let p be a vertex labelling in Γ. The *equality graph* for Γ is defined to be a subgraph L_p (danish Lighedsgraf = equality graph) in Γ. The vertex set of L_p is the vertex set of Γ and the edge set of L_p is the set of those edges xy in Γ for which

$$p(x) + p(y) = w(xy) \quad . \tag{3}$$

These edges are called equality edges. The figure shows an example:



The motivation for constructing L_p is that we have

> ***Corollary 32.2.*** If L_p has a perfect matching $\mathcal{M}$, then $\mathcal{M}$ is a matching with maximal weight in Γ and p is a vertex labelling with minimal sum.

Remember, that a perfect matching is a matching, where every vertex is incident to an edge in $\mathcal{M}$. The corollary follows immediately from the remark following lemma 32.1. For if $\mathcal{M}$ is a perfect matching in L_p then $w(\mathcal{M})$ is equal to the sum of all vertex labellings, that is, S(p).

Now, the idea in the assignment algorithm is the following: By means of a sequence of operations, S(p) is step wise made smaller and smaller and the matching $\mathcal{M}$ is made larger and larger, until $\mathcal{M}$ is a perfect matching in L_p. Then it follows from the corollary, that $\mathcal{M}$ has maximal weight. In the example below (3), there is no P(z) which can be made smaller such that we obtain a new vertex labelling with smaller sum. This shows, that it is not a trivial sequence of operations which we have to construct in the

algorithm: ASSIGNMENT.

Input: A bipartite graph $\Gamma = K_{n,n}$ with colour classes X and Y and a real weight function $w(k) \geq 0$, $k \in \mathcal{K}(\Gamma)$.

Output: A perfect matching $\mathcal{M}$ in Γ. A vertex labelling p in Γ.

Variables: L_p is a graph. α is a real variable. $\mathcal{L}$ is a set of edges.

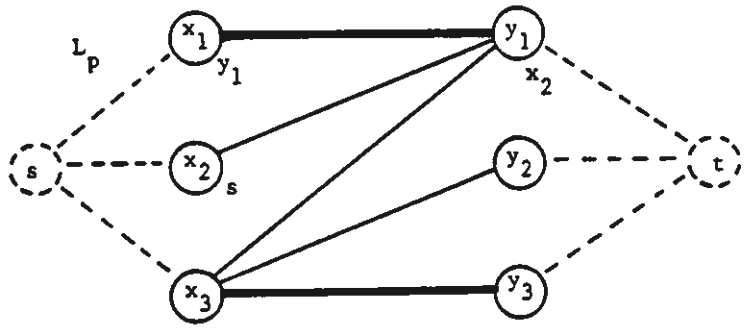
Initializing: p is any vertex labelling, for example determined by (2) p. 4.23.

1. Construct the equality graph L_p.
2. Find a maximal matching $\mathcal{M}$ in L_p.
3. **if** $\mathcal{M}$ is perfect, STOP.
4. MARKER (f, s) in the transport network described p. 3.48, where Γ (s. 3.48) is L_p, f is the flow determined by $\mathcal{M}$ and s is connected to the vertices in X. $\mathcal{A}$ is the set of labelled vertices and $\mathcal{B}$ is the set of unlabelled vertices.
5. **while** t is not labelled,
6. put $\alpha = \min\{p(x) + p(y) - w(xy)\}$, where $x \in \mathcal{A} \cap X$ and $y \in \mathcal{B} \cap Y$.
7. Let $\mathcal{L}$ be the set of those edges xy in Γ for which $x \in \mathcal{A} \cap X$, $y \in \mathcal{B} \cap Y$ and $\alpha = p(x) + p(y) - w(xy)$.
8. Let $p(z) := \begin{cases} p(z)+\alpha & \text{for } z \in \mathcal{A} \cap Y \\ p(z)-\alpha & \text{for } z \in \mathcal{A} \cap X \\ p(z) & \text{for } z \in \mathcal{B} \end{cases}$
9. Find the new equality graph L_p by adding $\mathcal{L}$ to the edge set, and deleting the $\mathcal{B} \cap X$, $\mathcal{A} \cap Y$ – edges.
10. Continue MARKER(f,s) in the new part of L_p, by starting in each vertex p of X, which is start vertex of an edge in $\mathcal{L}$.
11. Use the augmenting path found to make $\mathcal{M}$ larger.
12. **go to** 3.

Before we give the proof of the algorithm, we take two examples.

First, we consider the example from page 4.24 and describe the weights in a matrix:

$$\begin{array}{c} \\ \\ \rightarrow x_1 \\ \rightarrow x_2 \\ x_3 \\ \\ \end{array} \begin{array}{c} \downarrow \quad \downarrow \\ y_1 \ y_2 \ y_3 \\ \left[\begin{array}{ccc} 1 & 2 & 3 \\ 1 & 0 & 2 \\ 1 & 5 & 5 \\ 1 & 5 & 5 \end{array}\right] \end{array} \begin{array}{c} \\ \\ 0 \\ 0 \\ 0 \\ \\ \end{array}$$



The vertex labelling is also shown on the figure. It is determined by (2) p. 4.23. In line 1 we obtain the equality graph shown in the figure, and in line 2 we find the matching shown. Since $\mathcal{M}$ is not perfect, we proceed to line 4 and we label vertics s_i, x_2, y_1 , and, last x_1 . In the figure each vertex is labelled with the name of the vertex from which the label come. In line 6 we now have to form the minimum of

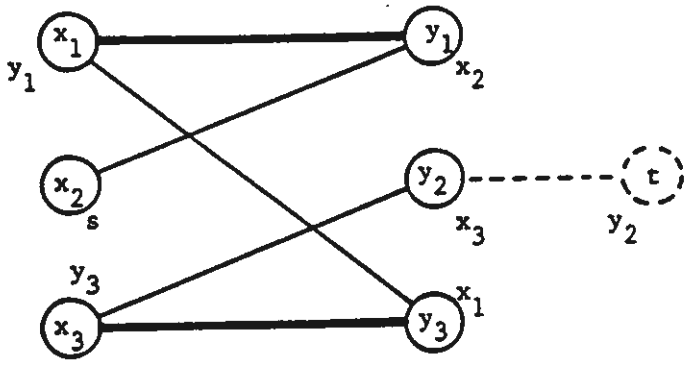
$$(4) \qquad p(x) + p(y) - w(xy)$$

taken over all edges xy , for which $x \in \{x_1, x_2\}$ and $y \in \{y_2, y_3\}$. These sets are shown with arrows in the matrix. The four values of (4) becomes:

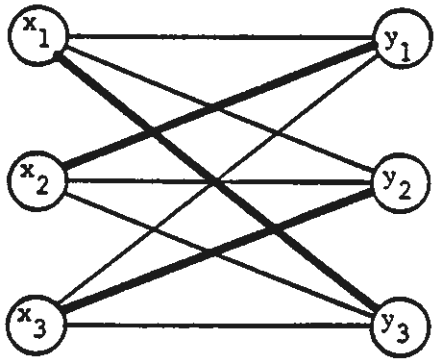
x	y	p(x) + p(y) − w(xy)
x_1	y_2	3
x_1	y_3	2
x_2	y_2	5
x_2	y_3	3

The minimum is 2, obtained only once, namely for the edge $x_1\, y_3$, $\mathcal{L} = \{x_1\, y_3\}$. In line 8 we shall now add 2 to $p(y)$, when y is labelled and subtract 2 from $p(x)$ when x is labelled:

$$\begin{array}{c} \\ x_1 \\ x_2 \\ x_3 \\ \\ \end{array} \begin{array}{c} y_1 \ y_2 \ y_3 \\ \left[\begin{array}{ccc} 1 & 2 & 3 \\ 1 & 0 & 2 \\ 1 & 5 & 5 \\ 3 & 5 & 5 \end{array}\right] \end{array} \begin{array}{c} \\ -2 \\ -2 \\ 0 \\ \\ \end{array}$$



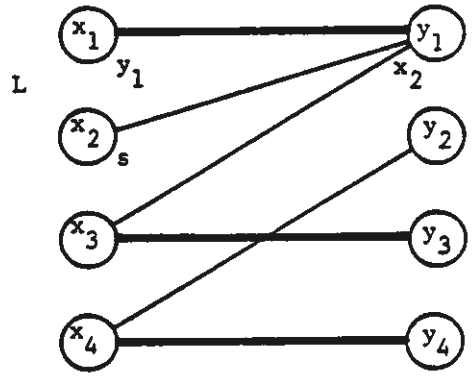
To L_p we now have to add those edges for which the minimum in line 6 is obtained, here only $x_1\, y_3$. Furthermore, the edge $x_3\, y_1$, has to be deleted from L_p . In line 10, the labelling can now proceed to y_3, x_3, y_2 and t as shown, and in line 11 we find the augmenting path (s), x_2, y_1, x_1, y_3, x_3, y_2, (t) and hereby the matching with weight



15 shown on the figure. $\mathcal{M}$ is perfect, hence according to the corollary it has maximal weight.

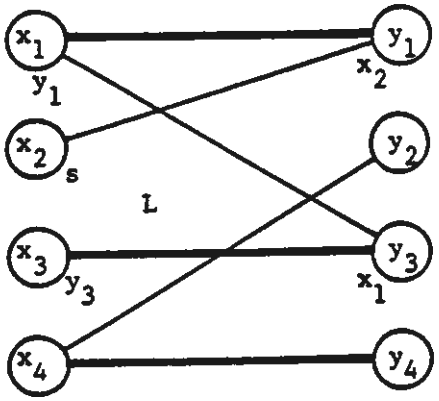
The other example we shall consider is the following

$$
\begin{array}{cc}
 & \begin{array}{cccc} \downarrow & \downarrow & \downarrow & \end{array} \\
\begin{array}{c} \rightarrow \\ \rightarrow \\ \\ \\ \end{array} & \left[\begin{array}{cccc|c} 3 & 1 & 3 & 2 & 0 \\ 3 & 1 & 1 & 1 & 0 \\ 3 & 1 & 4 & 3 & 0 \\ 2 & 4 & 1 & 4 & 0 \end{array}\right] \\
 & \begin{array}{cccc} 3 & 4 & 4 & 4 \end{array}
\end{array}
$$



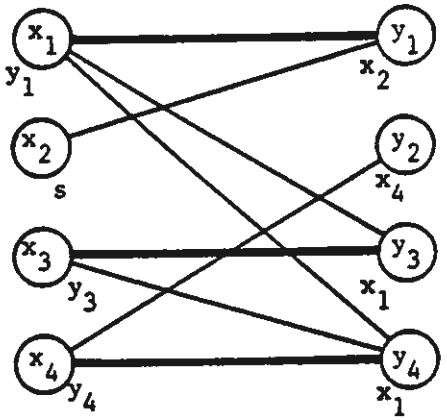
MARKER performs as shown and we determine a by looking at the $\{x_1, x_2\}$, $\{y_2, y_3, y_4\}$ – edges, shown with arrows. $\alpha = 1$, and the minimum is only obtained for the edge $x_1\, y_3$, $\mathcal{L} = \{x_1\, y_3\}$. In line 8 we now find a new vertex labelling:

$$
\begin{array}{cc}
 & \begin{array}{cccc} \downarrow & \downarrow & \downarrow & \end{array} \\
\begin{array}{c} \rightarrow \\ \rightarrow \\ \rightarrow \\ \\ \end{array} & \left[\begin{array}{cccc|c} 3 & 1 & 3 & 2 & -1 \\ 3 & 1 & 1 & 1 & -1 \\ 3 & 1 & 4 & 3 & 0 \\ 2 & 4 & 1 & 4 & 0 \end{array}\right] \\
 & \begin{array}{cccc} 4 & 4 & 4 & 4 \end{array}
\end{array}
$$



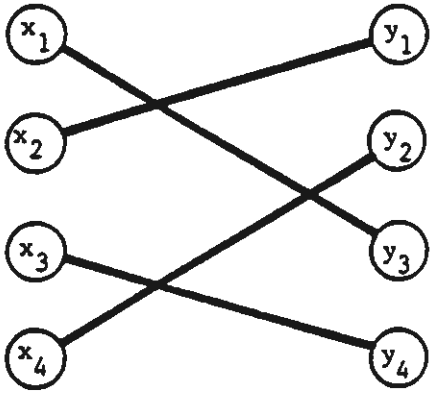
In the new equality graph, where the edge $x_3\, y_1$ is deleted, the labelling can proceed from x_1 to y_3 to x_3 as shown. t is not labelled, so we stay in the **while**–loop and define a new α in line 6. We get $\alpha = 1$ and $\mathscr{L} = \{x_1\, y_4, x_3\, y_4\}$ in line 7. The new vertex labelling and the equality graph is shown in the figure. An now the labelling

$$\begin{bmatrix} 3 & 1 & 3 & 2 \\ 3 & 1 & 1 & 1 \\ 3 & 1 & 4 & 3 \\ 2 & 4 & 1 & 4 \\ 5 & 4 & 5 & 4 \end{bmatrix} \begin{matrix} -2 \\ -2 \\ -1 \\ 0 \end{matrix}$$



continues from (for example) x_3 to y_4 to x_4 to y_2 to t . Therefore we leave the **while**-loop and find in line 11 the augmenting path $x_2, y_1, x_1, y_3, x_3, y_4, x_4, y_2$. Using this path we obtain a perfect matching, corresponding to the four circled elements in the matrix. Therefore the weight of a maximal weight matching is 13. Also, the last vertex labelling we found has sum 13.

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



Note, that once we have found a solution to an assignment problem, then it is easy (– also for someone, who do not know the algorithm) to control that the result is correct: If only $S(p) = w(\mathscr{M})$, we know from lemma 32.1 that $\mathscr{M}$ is a maximal weight matching and that p is a vertex labelling with minimal sum. By the way, this is also the case with algorithm MAXIMAL FLOW and MAXIMAL 2–COMMODITY FLOW: It is easy to check that a result is correct, without going through the whole algorithm.

We shall now prove:

Theorem 32.3. The ASSIGNMENT algorithm stops and in the output $\mathcal{M}$ is a maximal weight matching and p is a minimal sum vertex labelling. The complexity of the algorithm is $O(n^4)$.

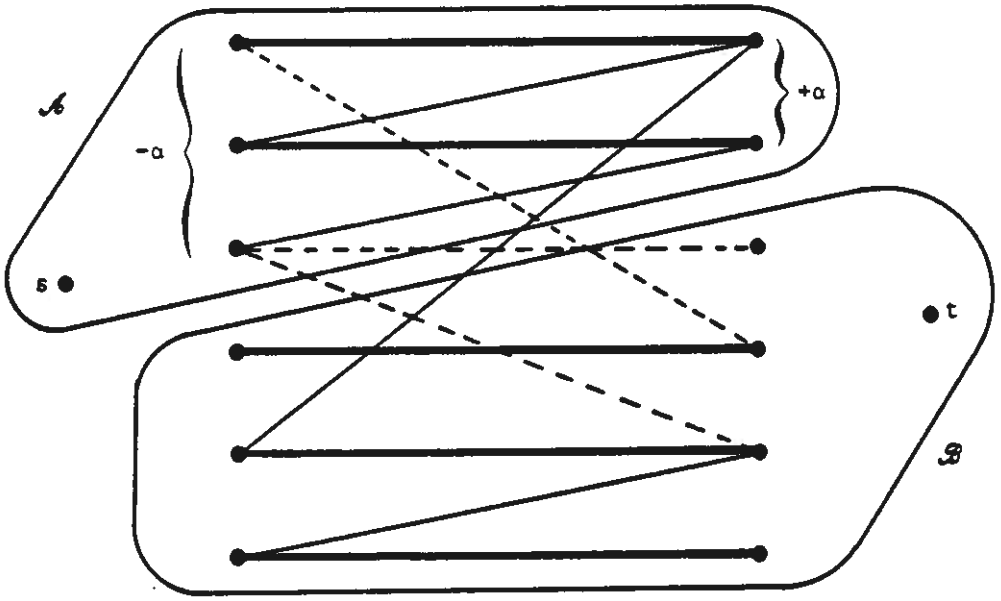
Proof: First, we shall show, that:

A. When a edge $k = xy \in \mathcal{M}$, then it is not deleted in line 9.

B. In line 8 we form a new vertex labelling.

From these two statements and line 3 it follows, that in the output $\mathcal{M}$ is a perfect matching in L_p. From corollary 32.2 it then follows, that $\mathcal{M}$ has maximal weight and that p has minimal sum.

Proof of A: Assume, that we are going to execute line 8 and that $k = xy \in \mathcal{M}$. If y is labelled, then x must clearly also be labelled. If x is labelled, the labelling did reach x from y (since $s\,x$ is saturated); therefore y is labelled. Hence



either $\{x, y\} \subseteq \mathcal{A}$ or $\{x, y\} \subseteq \mathcal{B}$, as shown in the figure. It follows, that $k = xy$ stays in L_p, when p is updated in line 8 since $p(x) + p(y)$ is not changed. This proves A.

Proof of B: We shall show, that the inequality

$$p(x) + p(y) \geq w(xy)$$

is also satisfied after the updating of p i line 8. The unequality stays correct for

1) $x \in \mathscr{A} \cap X$, $y \in \mathscr{A} \cap Y$, because in the case we don't change $p(x) + p(y)$.

2) $x \in \mathscr{B} \cap X$, $y \in \mathscr{B} \cap Y$, because this is how α is defined in line 6.

3) $x \in \mathscr{B} \cap X$, $y \in \mathscr{A} \cap Y$, because we make $p(y)$ larger but don't change $p(x)$.

4) $x \in \mathscr{B} \cap X$, $y \in \mathscr{B} \cap Y$, because neither $p(x)$ nor $p(y)$ is changed.

This proves B.

We shall now show, that the algorithm stops. The point is, that when α is defined in line 6, then $\alpha > 0$. Because, if $\alpha = 0$, there would be an $\mathscr{A} \cap X$, $\mathscr{B} \cap Y -$ edge $k = xy$ in L_p , shown with broken lines in the figure above. As we have seen in A, $k \notin \mathscr{M}$, and hence y can be labelled form x , which is a contradiction. Hence $\alpha > 0$. Each time the **while**-loop is executed, therefore at least one vertex in Y will be added to $\mathscr{A}$, and hence the **while**-loop can be executed at most n consequtive times. And each time the **go to** loop is executed once, an edge is added to $\mathscr{M}$. Therefore the **go to** loop can be executed at most n times. Hence the algorithm stops.

Finally, the complexity: We assume that the weights and the vertex labels are kept in a $(n+1)\times(n+1)$-matrix, as shown in the previous examples. The equality graph is assumed to be described by a double list, as shown in page 2.6. $\mathscr{M}$ can be described by underlining in the matrix. The complexity of each line then will be as follows:

1. $O(K) = O(n^2)$.
2. $O(n^3)$ (when we use the MPM-algorithm).

Totally $< n$ times:

3. $O(1)$, når $|\mathcal{M}|$ is kept in a variable.
4. $O(K) = O(n^2)$ when we use MARKER WITH QUEUE.
5. $O(1)$.
6. $O(K) = O(n^2)$.
7. $O(K) = O(n^2)$.
8. $O(n)$.
9. $O(K) = O(n^2)$.
10. $O(K) = O(n^2)$.

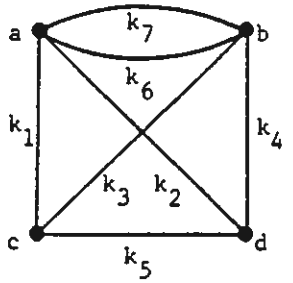
(5.–10.: n times for each execution of the **go to** loop)

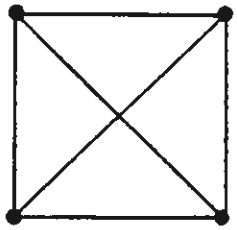
11. $O(n)$.
12. $O(1)$.

Therefore one execution of the **go to** loop can be carried out in $O(n^3)$ steps, and the overall complexity is $O(n^4)$.

At page 4.22 we mentioned that we could also start by finding all perfect matchings. This will give an $O(n!)$-algorithm. So, the ASSIGNMENT algorithm reduces this to $O(n^4)$. But the good algorithm provides us with one thing more namely an explanation why $\mathcal{M}$ can not possibly have lager weight than found. The reason is that there is a vertex labelling p , which has such a little sum, that $S(p) = w(\mathcal{M})$.

33. EULERIAN TOURS. There are many different applications of Graph Theory which leads to the following problem: Is it possible to draw a given connected, not directed graph, with pencil on a piece of paper, without lifting the pencil from the paper and without drawing the same edge more than once?





In the graph to the left it can be done, for example like this:

$$c, k_3, b, k_6, a, k_2, d, k_4, b, k_7, a, k_1, c, k_5, d .$$

This sequence of vertices and edges is called an (open) eulerian tour in the graph. In the graph to the right there does not exists an eulerian tour.

If a postman draw the roads, in which be has to deliver the mail as a graph, then it is best for him if the graph has an eulerian tour. Because in this case he can deliver the mail without being forced to walk longer than the sum of the lengths of the roads.

More formally, we define an *eulerian tour* in a not directed graph Γ without loops as a sequence

(1) $$E = p_1, k_1, p_2, k_2, \cdots, p_{n-1}, k_{n-1}, p_n \quad ,$$

where k_i is a p_i, p_{i+1}-edge for $i = 1, 2, \cdots, n-1$, and where each edge in Γ is mentioned exactly once in E. If $p_1 \neq p_n$ E is called an *open* eulerian tour, if $p_1 = p_n$ E is called a *closed* eulerian tour. Page 1.32 we have described a case, where there is no eulerian tour. This special case, the Bridges of Königsberg, was treated by Euler; this is the background for the name *Eulerian tour.*

Remark, that if p_1 and p_2 are different vertices in Γ and if Γ_1 is obtained from Γ by adding a new p_1, p_2-edge, the Γ has an eulerian tour with end vertices p_1 and p_2 if and only if Γ_1 has a closed eulerian tour.

Theorem 33.1.

(a) A connected graph Γ has an open eulerian tour if and only if Γ has exactly two vertices with odd valency.

(b) A connected graph Γ has a closed eulerian tour if and only if all vertices in Γ have even valency.

Proof of (a): The **"only if"** part. It is given that Γ has an open eulerian tour, (1). Let $t(p)$ be the number of times p is mentioned in (1). A vertex p, $p \neq p_1$ and $p \neq p_n$ has then valency $2t(p)$ which is even. If p is p_1 or p_n, the valency is $2(t(p)-1) + 1$, which is odd. Hence exactly two vertices have odd valency. The **"if"** part: It is given, that Γ contains exactly two vertices p_1 and p_n with odd valency. And we shall show that Γ has an eulerian tour with end vertices p_1 and p_n. The proof is constructive, it implies an algorithm which determines an eulerian tour:

Let us start in p_1 and chose any p_1, p_2- edge k_1. Next we chose any p_2, p_3- edge which we have not chosen before, and so on. In this way we obtain a finite sequence

(2) $$p_1, k_1, p_2, k_2, \cdots ,$$

where k_i is a p_i, p_{i+1}- edge and where $k_i \neq k_j$ for $i \neq j$. This process stops in p_n because any other vertex has (compare the valency conditions) the property, that if we arrive to q in (2), then it is also possible to leave q by a not used edge. So the result is a sequence

(3) $$p_1, k_1, p_2, k_2, \cdots , p_{n-1}, k_{n-1}, p_n$$

where, again, $k_i \neq k_j$ for $i \neq j$. If all edges appear in (3), we are finished. If there is an edge which do not appear in (3), then – since Γ is connected – there must exist a p_i (*) in (3) which is incident with an unused edge. We repeat the above process, but now with start in p_i and obtain a sequence

(4) $$p_i, k_i, p_{i+1}, k_{i+1}, \cdots$$

in which the edges have not been used before. This process must stop in p_i, and when it stops, we have used all edges incident with p_i. Therefore, we obtain a sequence

(5) $$p_i, k_i, p_{i+1}, k_{i+1}, \cdots , p_j, k_j, p_i \ .$$

Next we insert this sequence in (3) instead of the appearance mentioned (*). In this way we can continue, until we obtain an eulerian tour. This proves (a)

(b) follows now from (a) and the remark preceeding the theorem.

□

If a graph has an eulerian tour, then the above proof contains a description of a corresponding algorithm. For a graph in which all vertices have even valency the algorithm can be described as follows:

EULER'S ALGORITHM .

Input: A connected, not directed, graph without loops in which all vertices have even valency. A vertex s in Γ . Γ represented by a set of neighbour lists $L(p)$, $p \in \mathcal{P}(\Gamma)$, where there are arrows between the two elements representing the same edge.

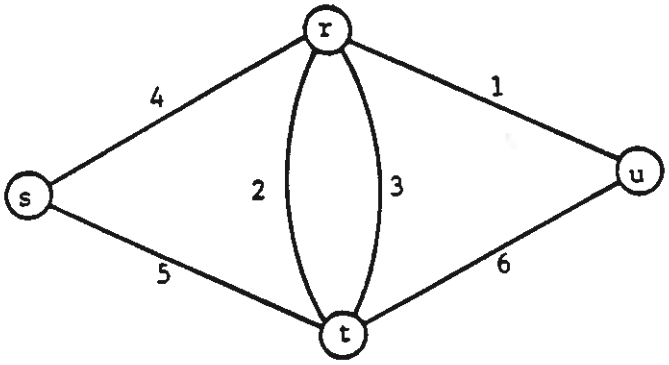
Output: A closed eulerian tour E in Γ .

Variables: q is a vertex, k an edge, x the address of an occurrence of a vertex in E , L is a list of pairs $(p, a(p))$, where $a(p)$ is the address of an occurrence of P in E . $m(p)$ is a labelling function, which can have the values "new" and "old".

Initializing: $E = s$, $L = (s, x)$, where x is the address of s in E . $m(p) =$ new for $p \neq s$, $m(s) =$ old.

1. **while** L is not empty
2. delete the first element (p, x) from L .
3. **while** $L(p)$ is not empty
4. let (k, q) be the first element on $L(p)$.
5. Delete the two elements containing k from the edge lists.
6. Insert (k, q) after the vertex with address x in E , and let now x be the address of q in E
7. **if** $m(q) =$ new,
8. add (q, x) to L and change $m(q)$ to old.
9. Put $p = q$.
10. **STOP.**

Let us illustrate how the algorithm works by an example. We don't give values for the addresses x .



L(r): 2, t → 4, s → 1, u → 3, t → Ø

L(t): 5, s → 2, r → 3, r → 6, u → Ø

L(u): 1, r → 6, t → Ø

L(s): 5, t → 4, r → Ø

vertex:	r	s	t	u
labelling:	new	old	new	old

L : s E: s

Remark, that only two of the arrows connecting occurrences at the same edge are shown on the figure. The algorithm proceeds as follows:

In line 2 we put $p = s$, $L = \emptyset$.

In line 3 – 9 , $E = s, 5, t$ and edge 5 is deleted from the lists. Furthermore $L = (t, x)$ and $m(t) = old$.

Then lines 3 - 9 is executed twice more: First we use edge 2, then edge 4. This gives $E = s, 5, t, 2, r, 4, s$, $L = (t, x), (r, y)$, edges 4, 2, 5 deleted and only $m(u) = new$. Now, $L(p) = L(s)$ is empty and we are finish with the while – loop in line 3 – 9. Then we continue in line 1. L is not empty $L(r, y)$, so in line 2 we delete (r, y) from L . Then $p = t$ and by two executions of the inner **while** – loop we insert first 6, u, then 1, r and, finally 3, t in E . (Here we use the addresses.) The result is

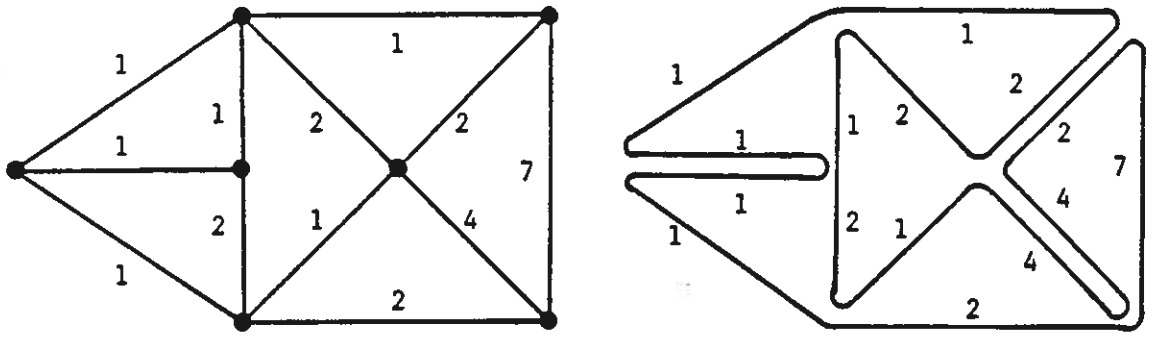
$$E = s, 5, t, \overline{6, u, 1, r, 3, t}, 2, r, 4, s \ .$$

Now, L(p) = L(t) in line 3 is empty and L = (r, x), (u, y) . Finally the while loop in line 1 is executed twice, without anything happening and the algorithm stops in line 10.

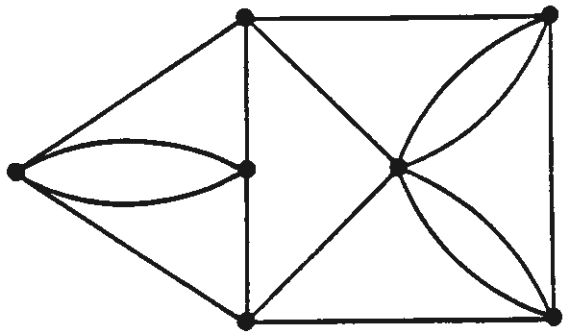
It follows from the proof of theorem 33.1 that the algorithm work correct. The overall work with the edge lists has complexity O(K) because each time an element is read in an edge list in line 4 it is deleted in line 5. The work with L is O(P) , since the labelling function ensures, that a vertex only enters L once. And finally, the work with the eulerian tour has complexity O(K) because we, by means of the addresses can insert the new element in E without searching. Therefore, the complexity is O(K) .

34. THE CHINESE POSTMAN. This strange name of a graph theoretic problem is due to the fact, that the problem was first formulated by a chinese mathematician [10].

There is a postman, who has to distribute the mail in a small town. He starts and ends in the post office and he wants to find a shortest possible route through all the streets. The network of streets forms a graph. If this graph has an eulerian tour, clearly this is a shortest possible route. The length of the route is the sum of the lengths of



the streets. If the graph does not have an eulerian tour, there must be some streets which the postman has to walk through several times. The problem is the following: How can these streets be chosen, if the sum of their lengths must be shortest possible? In the figure to the left, where the numbers are edge lengths, the tour can be chosen in many different ways, one of them with length 32 is shown to the right. It is not trivial to find a shortest route. But it is obvious that it is really not the tour itself we are looking for, it is the edges which should be traversed more than once. So, what we have to do is to find a set of edges, with smallest possible edge sum, such that if these edges are changed to double edges, then we obtain an *eulerian graph,* that is a graph in which all vertices have even valency.



The figure shows the eulerian graph, which corresponds to the tour described above. When we have found this eulerian graph, then we can find the tour itself by means of EULER'S ALGORITHM. – In advance, you could expect that it might be necessary to use the same edge trice (or even more). However, it is easy to see that this is not the case. We shall now start a formal treatment of the problem.

Let Γ be a connected graph without loops. For each k we assume a number $\ell(k) > 0$ is given. $\ell(k)$ is called the *length* of k. If M is a subgraph or a set of edges in Γ, we define the length $\ell(M)$ as the sum of the lengths of the edges in M. To double an x, y-edge k means to add a new x, y-edge to Γ, with the same length as k. By this process, the valencies of x and y increases by 1.

The problem we want to solve is the following: Find a set of edges in Γ having shortest possible length such that by doubling these edges we obtain an eulerian graph.

We shall prove, that the problem can be solved as follows:

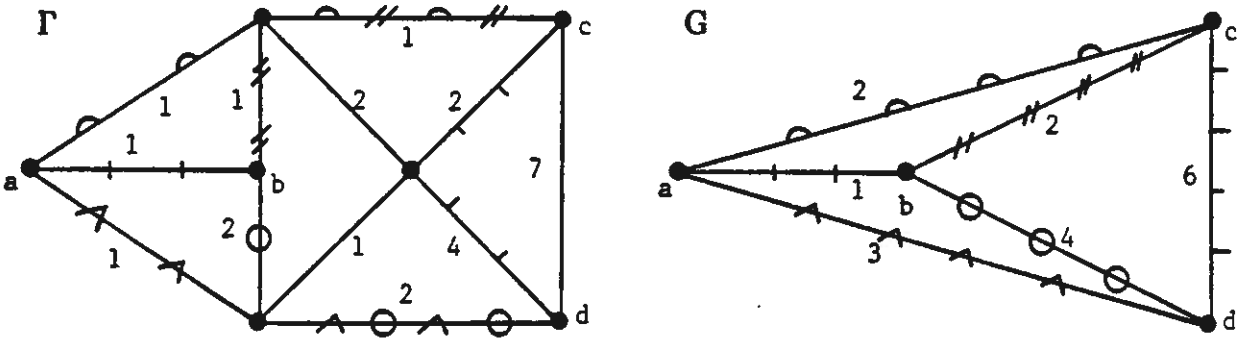
1. Construct an auxiliary graph G. The vertex set of G is the set of those vertices in Γ, which has odd valency, and two vertices x and y in G are connected with one edge k, the length of which is the length of an arbitrary, but fixed, shortest x, y–path V_k in Γ, $\ell(k) = \ell(V_k)$. (By theorem 2.1, $|\mathcal{P}(G)|$ is even).

2. Find a minimal weight perfect matching $\mathcal{M}$ in G.

3. For each edge k in $\mathcal{M}$ we double the edges of the corresponding path V_k in Γ. The graph obtained is the eulerian graph, we are looking for.

We have chosen not to call the method for an algorithm. The reason is, that we have not described, how 2. is carried out. For this purpose, the exist a famous $O(|\mathcal{P}(G)|^4)$- algorithm, which is due to Jack Edmonds, [5], [7]. This algorithm is complicated; it will not be treated here. In examples of moderate size, we can execute 2. by trying all possibilities. In advance, you could expect an edge of Γ to belong to several of the paths V_k. Actually, it is easy to see, that this can not happen, but this plays no rule for the following argument.

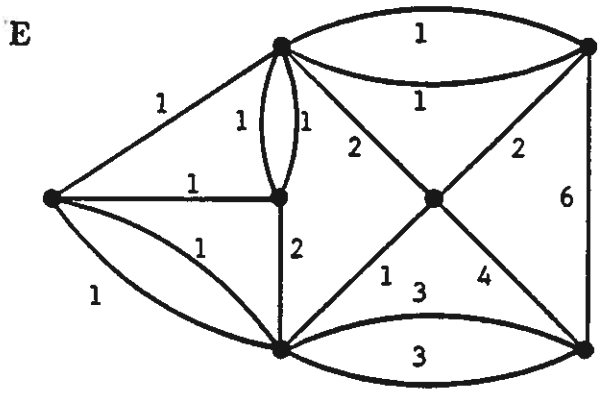
Before we prove, that the method is correct, we illustrate it with the example we considered above. An edge k in G and the corresponding path V_k in Γ are



marked in the same way. G has 3 perfect matchings:

ab , cd	with length	7,
ac , bd	with length	6,
ad , bc	with length	5.

So $\mathcal{M} = \{ad, bc\}$, and by doubling of the edges of the corresponding paths V_{ad} and V_{bc} we obtain the eulerian graph shown below. Here, the length is 31, which



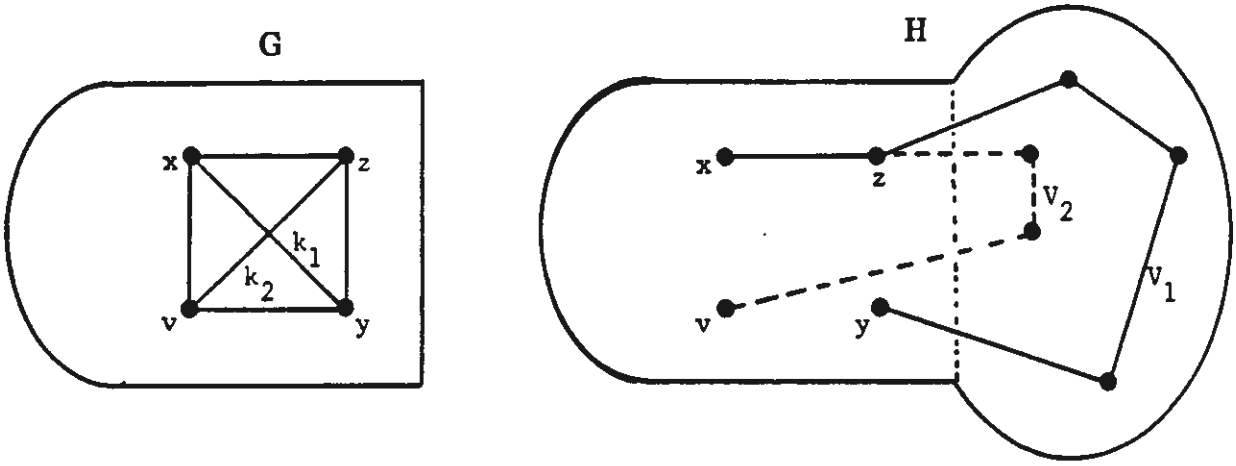
therefore is best possible.

We shall now prove in general, that the method works. In other words we shall show that if in an arbitrary way by means of doublings we construct an eulerian graph F, then

$$\ell(E_N) \leq \ell(F_N) \ ,$$

where E_N and F_N are the set of new edges in E respectively F.

Proof: Let H be the graph for which $\mathscr{P}(H) = \mathscr{P}(\Gamma)$ and $\mathscr{K}(H) = F_N = \mathscr{K}(F) \setminus \mathscr{K}(\Gamma)$. Since F is an eulerian graph, those vertices of H, which have odd valency in H are the same as those vertices in Γ, which have odd valency in Γ, that is, the vertices of G.



First we prove, that in H, there must be a path connecting two vertices with odd valency: Let x be a vertex in H with odd valency. According to theorem 2.1 the connected component in H which contains x will also contain a vertex $y \neq x$ with odd valency. And then x and y are connected by a path V_1 in H. If k_1 denotes the x, y-edge in G, then

$$\ell(k_1) \leq \ell(V_1) \ .$$

Now, put $H_1 = H - \mathscr{K}(V_1)$. x and y have even valency in H_1. If H_1 contains vertices with odd valency, again there must exist a path V_2 in H_1 which connects two vertices z and v with odd valency in H_1. If k_2 denotes the z,v-edge in G, then

$$\ell(k_2) \leq \ell(V_2) \ .$$

In this way we can continue until we have found $j = \frac{1}{2}|\mathscr{P}(G)|$ paths $V_1, V_2, \cdots V_j$ such that

$$\ell(k_i) \leq \ell(V_i) \ , \quad i = 1, 2, \cdots, j \ .$$

Since the matching $\mathcal{M}$ has minimal length, by adding these inequalitis we obtain

$$\ell(\mathcal{M}) \leq \sum_i \ell(k_i) \leq \sum_i \ell(V_i) \ .$$

But here $\sum_i \ell(V_i) \leq \ell(F_N)$, since the paths V_i are edge disjoint paths in H . And $\ell(\mathcal{M}) = \ell(E_N)$ since for each k_i in $\mathcal{M}$ we construct new edges in E_N with length $\ell(k_i)$. Therefore $\ell(E_N) \leq \ell(F_N)$. □

If two of the paths V_i had an edge k in Γ in common, in step 3 we should add two new edges k_1 and k_2 , both parallel to k , and then $E\backslash\{k_1 , k_2\}$ would be an eulerian graph with smaller length then E . The paths V_i are therefore edgedisjoint. This can also be seen directly, without using the proof above.

35. THE TRAVELLING SALESMAN. From an algorithmic point of view, comferiatorial problems are classified in two groups: First we have the problems which can be solved by polynomial algorithms, and on the other hand those problems, for which no polynomial algorithm is known. Here we list some important problems for which no polynomial algorithm is known:

1. Find the chromatic number of a graph.
2. Find a maximal cut in a transport network.
3 Find a maximal integer flow in a not directed 2–commodity network.
4. Find a maximal subset $\mathcal{M}$ of the vertex set of a graph Γ , such that no two vertices in $\mathcal{M}$ are connected by an edge in Γ ,
5. Given a logical expression which is a conjunction of disjunctions, each containing 3 terms, where negations are allowed. (For example $(a \vee b \vee \neg c) \wedge \neg(a \vee b \vee d) \wedge (\neg b \vee c \vee d)$). Is it then possible, to give each variable one of the values "true" or "false" such that the value of the expression becomes "true". This problem has played a major role in complexity theory. It is called 3–satisfyability.
6. The problem of the Chinese Postman if the graph is mixed i.e. where the graph contains directed as well as undirected edges.
7. The Travelling Salesman problem, which we are describing on the next page.

As mentioned no polynomial algorithm is known, which solves one of these problems. But it has been shown, that if there exists a polynomial algorithm A , which solves one of these problems, then by means of A , we could construct a polynomial algorithm for each of the other 6 problems. And, in fact, these 7 problems is a rather arbitrary selection from a larger list containing hundreds of combinatorial problems. In spite of many attempts through a period of many years no such algorithm has been found. This seems to indicate, that no such algorithm exist. But no proof of this statement seams to be within reach.

But even if we cant find a polynomial algorithm to solve these difficult problems, weaker results may be shown. As an example of this we shall give a little information about the travelling salesman problem. The problem is the following:

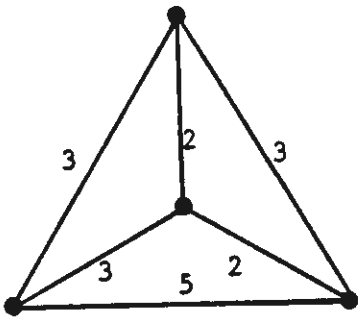
A salesman has to plan a round trip to some cities. He knows the distances and wants to find a shortest round trip including all the cities.

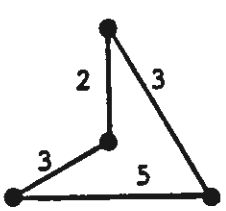
We shall formulate the problem in the language of graph theory: Consider the complete graph K_n with n vertices an in which for each x,y – edge k there is given a real number $\ell(k) = \ell(x,y)$. If M is a subgraph in K_n or a set of edges in K_n we define the length $\ell(M)$ as the sum of the lengths of the edges in M . The travelling salesmans problem is to find a circuit C_0 which contains all vertices in K_n and for which $\ell(C_0)$ is smallest possible. C_0 is called a solution to the travelling salesman problems and we write

$$C_0 = (p_1, p_2, \ldots, p_n) , \quad p_i \in \mathcal{P}(K_n) ,$$

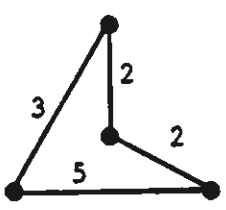
if C_0 consists of the p_1, p_2 – edge, the p_2, p_3 – edge, ..., p_n, p_1 – edge.

If for example we consider K_4 with the lengths given in the figure below, there are 3 circuits containing all four vertices. The solution is the circuit to the right.

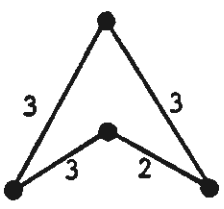




$\ell = 13$



$\ell = 12$



$\ell = 11$

However the number a_n of circuits containing the n vertices in K_n grows very fast with n. Actually, $a_n = \frac{1}{2}(n-1)!$. Therefore the number n can not be very large if we want to solve the problem by checking all possibilities. Even with a computer, the bound is something like $n = 15$.

As mentioned above no polynomial algorithm is known for the travelling salesman problem (TSP). But work has been done to find polynomial algorithms such that the output is a circuit containing all vertices for which the relative error,

$$\frac{\ell(C)}{\ell(C_0)} ,$$

can be proved to be under a not too large upper bound. But even this project is difficult. For example, let us consider the following simple algorithm which starts with $C = \mathscr{P}(K_n)$, $\mathscr{K}(C) = \emptyset$.

1. Chose a sorting of the edges in K_n,

$$k_1, k_2, ..., k_K , K = \tfrac{1}{2} n(n-1)$$

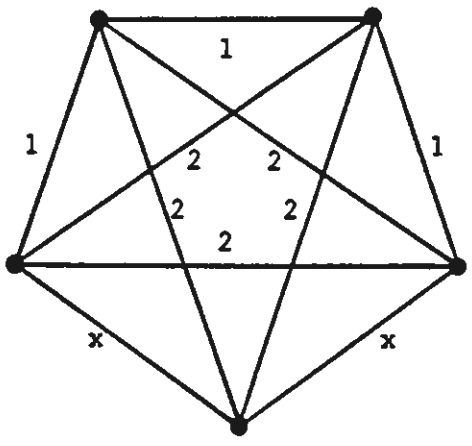
   such that $\ell(k_i) \geq \ell(k_j)$ for $i > j$.

2. Now consider the edges in this order. If we are considering k_i, and if each component in the graph obtained from C by adding k_i is a path, then we add k_i, to C. We stop this process when $|K(C)| = n-1$.

3. C is now a path of length n. Add the edge, which connects the end vertices of C to C.

What the algorithm does is to try to push as many short edges into C as possible.

But let us try the algorithm on the graph in the figure, where $x > 2.5$. First we add



the three length 1 edges to C. But then no edge of length 2 can be added to C. Finally, we add the two edges with length x. Since $\ell(C_0) = 8$, the relative errors is

$$\varphi = \frac{\ell(C)}{\ell(C_0)} = \frac{3+2x}{8} .$$

Since $\varphi \to \infty$ for $x \to \infty$ there is no upper bound, independent of x, for the relative error!

For the algorithm we are going to describe now, there is an upper bound for the relative error. But this algorithm demands an extra assumption, namely that the triangle inequality,

$$\ell(x\,y) \leq \ell(x\,z) + \ell(z\,y)$$

is satisfied for 3 arbitrary vertices x, y and z in K_n. If the vertices in K_n are cities and if $\ell(x\,y)$ is the distance between x and y, then the triangle inequality is satisfied. But if $\ell(x, y)$ is the time it takes to travel between x and y, then the triangle inequality is not necessarily satisfied.

We now consider the following attempt to solve TSP:

2–OPTIMAL ALGORITHM

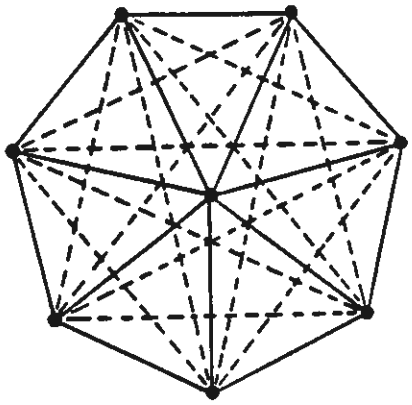
Input: The complete graph K_n, in which every x, y – edge k has a real length $\ell(k) > 0$, such that the triangle inequlity is satisfied..

Output: A circuit $C = (q_1, q_2, ..., q_n)$ in K_n, $q_i \in \mathcal{P}(K_n)$.

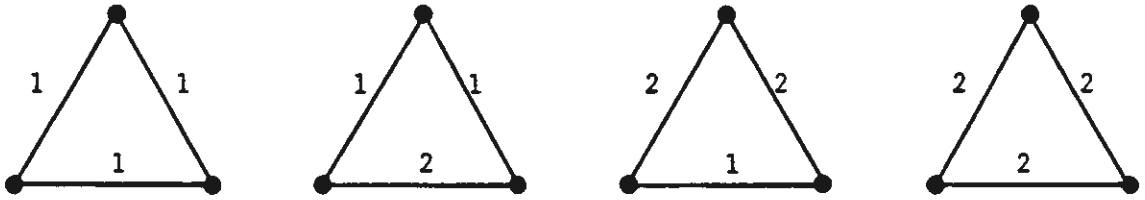
Variables: T is a tree in K_n, G is a graph, E is an eulerian tour in G, p_j, q_j, x and y are vertices and k_j and k are edges.

1. Find by means of KRUSKALS ALGORITHM a spanning tree T in K_n with minimal length.
2. Let G be the graph obtained from T by replacing each x, y – edge with two parallel x, y – edges.
3. Find by means EULER'S ALGORITHM a closed eulerian tour $E = (p_1, k_1, p_2, k_2, ..., p_{n-1}, k_{n-1}, p_1)$ in G.
4. For $i = 1, 2,..., n-1$, let q_i be the i^{th} vertex which is met for the first time when we traverse E as described. This implies, that $q_i \neq q_j$ for $i \neq j$.
5. $C = (q_1, q_2,..., q_{n-1}, q_1)$ is then a circuit containing the n vertices of K_n.

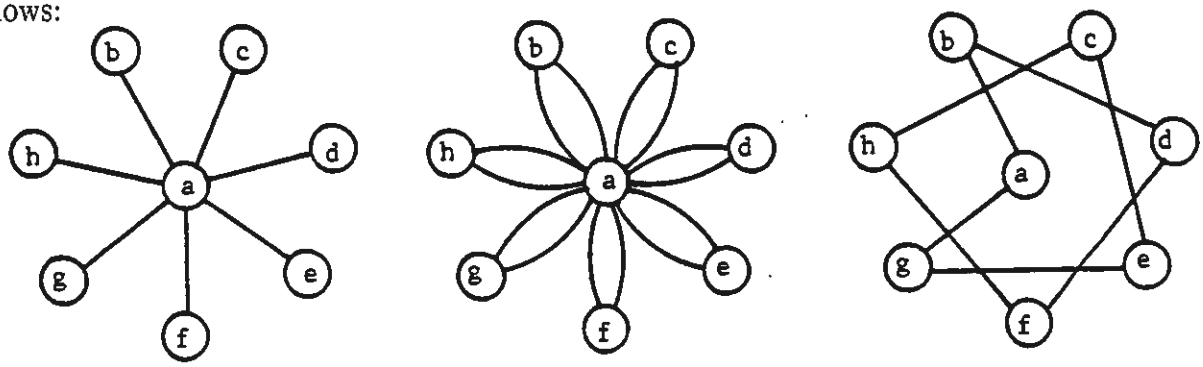
Let us illustrate the way the algorithm works by using it on the graph in the figure



where each full drawn line has length 1 and the dotted lines have length 2.



There are four types of triangles contained in the figure and it is easy to check, that the triangle inequality is satisfied for all of them. The algorithm can proceed as follows:



E = (a,b,a,d,a,f,
a,h,a,c,a,e,a,g,a)

C = (a,b,d,f,h,c,e,g)

C has weight 14 and C_0 has weight 8 . Hence the relative error is $\varphi = \frac{14}{8} = \frac{7}{4}$; this agrees with the theorem since $\frac{7}{4} < 2$.

Theorem 35.1. If the weight function satisfies the triangle inequality, and if C is the output of the 2-OPTIMAL ALGORITHM, and if C_0 is a solution to TSP, then

$$\frac{\ell(C)}{\ell(C_0)} < 2 .$$

The algorithm has complexity $O(n^2 \log n)$.

Proof. In the graph G constructed in line 2 all vertices have even valency. Therefore it is possible to execute line 3 and E contains all vertices in K_n. In line 4 therefore we obtain a circuit containing all vertices of K_n. If we delete an edge from C_0 we obtain a spanning tree T_0 in K_n. Since T is a tree with minimal length,

$$\ell(T) \leq \ell(T_0) < \ell(C_0) .$$

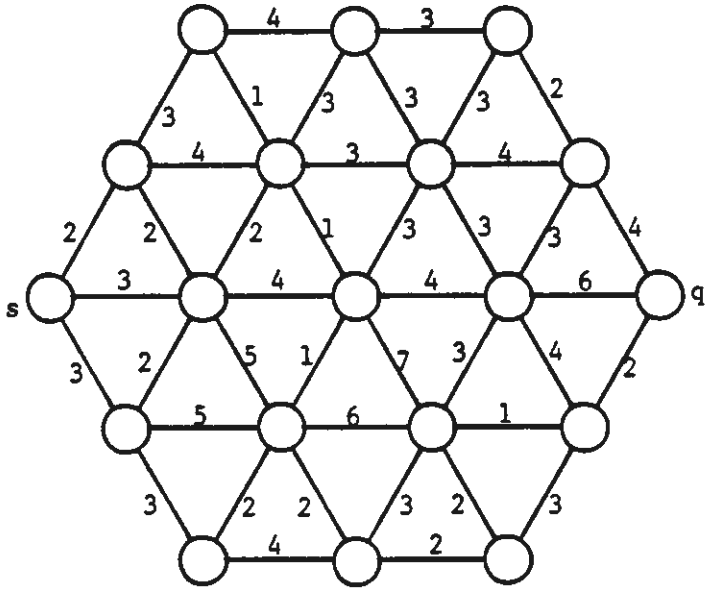
It follows that

$$\ell(E) = \ell(G) = 2\,\ell(T) < 2\,\ell(C_0) .$$

An x, y – edge k in C is either an edge in E or it will replace a sequence V of edges in E. By repeated application of the triangle inequality we obtain $\ell(k) \leq \ell(V)$. Therefore $\ell(C) \leq \ell(E) \leq 2\,\ell(C_0)$. (For example, in the example above, the edge bd replace ba and ad in E.) The complexity of line 1 is $O(n^2 \log n)$, (Theorem 30.1), while the complexity of each of the other steps is $O(n)$. Therefore the overall complexity is $O(P^2 \log P)$. □

PROBLEMS TO CHAPTER 4.

4.1. Write in each circle p in the graph Γ in the figure the length ℓ_p of a shortest s, p – path in Γ. Direct some edges in Γ such that for each $p \neq s$ $v_u(p)$ (– the out valency) = 1 and such that Γ contains one and only one $p \to s$ – path V, and this path satisfies $\ell(V) = \ell_p$. Show in the figure a shortest s, q – path.



4.2. Does the greedy algorithm work for every 0,1–weight function of the following independence system:

a) $\mathcal{U}_1 = \{X \subseteq E \mid |x| \leq 2 \text{ and } X \notin \{\{3,4\}\ \{3,5\}\}\}$.

b) $\mathcal{U}_2 = \{X \subseteq E \mid |x| \leq 2 \text{ and } X \notin \{\{3,4\}, \{3,5\}, \{4,5\}\}\}$.

c) $\mathcal{U}_3 = \{X \subseteq E \mid |x| \leq 2 \text{ and } 5 \notin X\}$.

4.3. On the set $E = \{a,b,c,d,e,f,g,h\}$ there is given a weighted independence system $(\mathcal{U}, w)$ where $\mathcal{U}$ consists of the sets

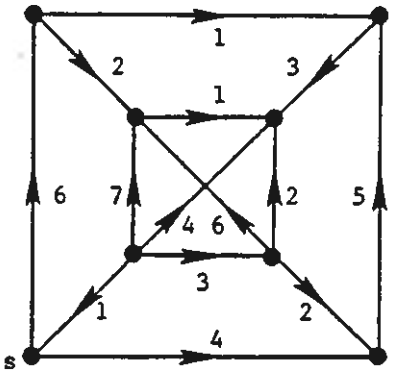
$$\{a,f,g,h\},\ \{b,c,g,h\},\ \{b,d,e,h\},\ \{c,d,e,f\}$$

and all their subsets, and where the weight function is

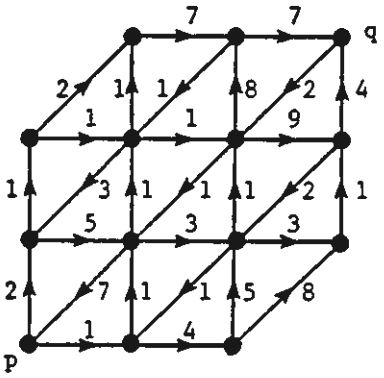
$$w(a) = w(b) = 5,\ w(c) = w(d) = 4,\ w(e) = w(f) = 3,\ w(g) = w(h) = 1\ .$$

Find an independent set B_{OPT} with maximal weight. Find the weight of any set, which can be the output when GA is applied to $(\mathcal{U}, w)$.

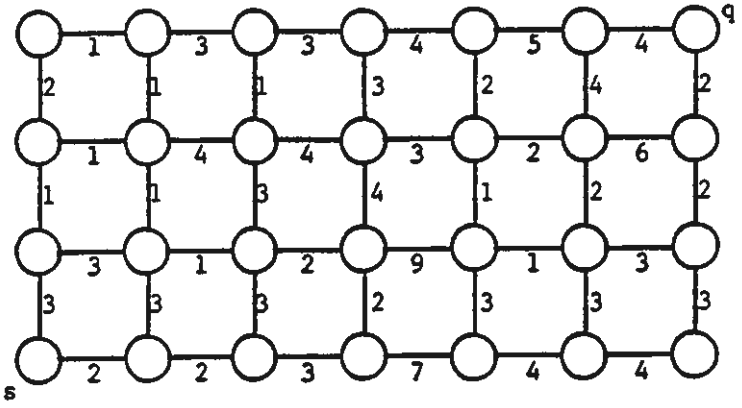
4.4. Show as on the figures page 4.4 how Dijkstra's algorithm may proceed on the graph in the figure.



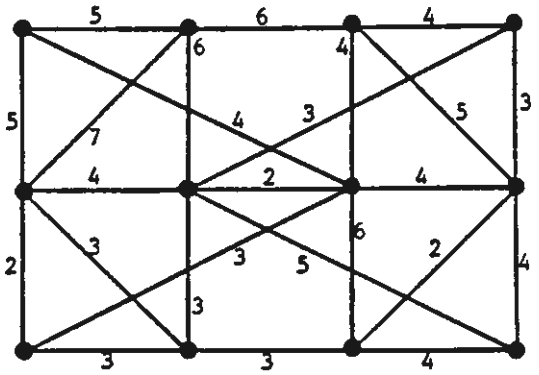
4.5. Find the length of a shortest $p \to q$ – path in the graph in the figure.



4.6. Dijkstra's algorithm can also be used when the input is a not directed graph Γ . The only change necessary is to replace "$\neg$" with "," at two places in the algorithm. Write in each circle in the figure below the distance from s . Use Dijkstra's algorithm, but use only one figure. Show finally on the figure a shortest s, q – path.



4.7. a) Find a spanning tree with maximal weight in the graph in the figure.
b) Find a spanning tree with minimal weight.



4.8. Solve the assignment problem for the matrix

$$\begin{Bmatrix} 5 & 2 & 3 & 3 \\ 3 & 4 & 1 & 1 \\ 6 & 4 & 2 & 1 \\ 2 & 4 & 1 & 1 \end{Bmatrix}.$$

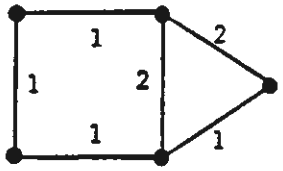
4.9. Solve the assignment problem for the matrix

$$\begin{Bmatrix} 8 & 6 & 7 & 10 \\ 9 & 8 & 8 & 6 \\ 4 & 2 & 1 & 10 \\ 8 & 6 & 6 & 10 \end{Bmatrix}.$$

4.10. Let Γ be a graph in which every edge k has a weight $w(k) \geq 0$. An independence system $\mathcal{U}_0$ on $\mathcal{K}(\Gamma)$ is defined by

$$U \in \mathcal{U}_0 \Leftrightarrow U \text{ contain at most 2 circuits.}$$

1) The figure shows a graph with 3 circuits.
The weights $w(k)$ of the edge are given.
GA is applied for this graph Γ on $(\mathcal{U}_0, w)$.
Find all the edge sets which can be output.
Does GA work in this case?



2) Is it true that for every graph Γ and every weight function $w(k)$ (≥ 0 for all $k \in \mathcal{K}(\Gamma)$), that GA works on $(\mathcal{U}_0, w)$? Motivate your answer.

4.11. Let M be the matroid on the set of rows in A, in which a set is independent, if and only if the corresponding rows in A are linearly independent

$$\mathbf{A} = \begin{bmatrix} 1 & 2 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 3 & 1 \\ 0 & 1 & 1 & 0 \\ 5 & 5 & 5 & 5 \\ 2 & 2 & 2 & 2 \\ 1 & 2 & 3 & 4 \end{bmatrix}$$

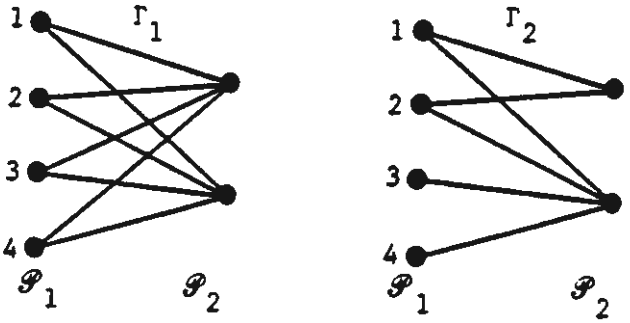
Draw a graph Γ, such that M is isomorphic to the circuit matroid of Γ.
Is M cographic?

4.12. For each integer i $(0 \leq i \leq 5)$, either draw a graph Γ, such that the uniform matroid M_{5i} is the circuit matroid of Γ, or prove that M_{5i} is not graphic.

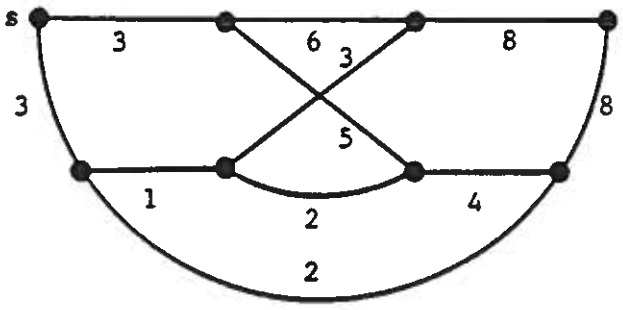
4.13. Which of the set systems below is the system of bases for a matroid on the set $E = \{ 1, 2, 3, 4, 5 \}$:

a) $\mathscr{B}_1 = \{\{1, 2, 4\}, \{1, 3, 4\}, \{2, 3, 4\}\}$.
b) $\mathscr{B}_2 = \{\{1, 3\}, \{1, 4\}, \{1, 5\}, \{2, 3\}, \{2, 4\}, \{2, 5\}\}$.
c) $\mathscr{B}_3 = \{\{1. 3\}, \{1, 4\}, \{1, 5\}, \{2, 3\}, \{2, 4\}\}$.

4.14 Let M_1 and M_2 denote the transversal matroids on the set $\mathscr{P}_1$ for Γ_1, respectively Γ_2. Is M_1 graphic? Is M_2 graphic?



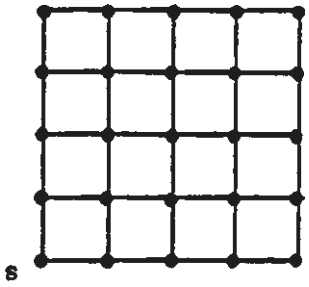
4.15.



a) Determine a spanning tree with maximal weight in the graph in the figure. Explain, at least a couple of times, which sets the $\mathscr{P}_i$'s are. Find a spanning tree in the graph with minimal weight.

b) Find, by means of a variation of Dijkstra's algorithm, the distances from s to the other vertices in the graph in the figure. If p is a vertex in the graph, it is then also easy to find a shortest s,p-path in the graph.

4.16.



In a not–directed connected graph Γ, where all edge lengths are 1, a vertex s is given. Describe an algorithm, which for every vertex p in Γ determines the number of shortest s,p-paths in Γ. Use your algorithm on the graph in the figure. Find the complexity of you algorithm.

LITERATURE FOR CHAPTER 4.

[1] Aho, Alfred V., Hopcroft, John E. and Ullman, Jeffrey D.: The Design and Analysis of Computer Algorithms. Addison–Wesley Publishing Company 1975.

[2] Bellman, R. and Cooke, K.L.: The Konigsberg bridges problem generalized. Jl. of Math. Anal. and Appl., 25, p. 1, 1969.

[3] Christofides, Nicos: Graph Theory. An Algorithmic Approach. Academic Press, New York, London, San Francisco, 1975.

[4] Dijkstra, E.W.: A note on two problems in connexion with graphs. Numerische Mathematik 1, 269–271, 1959.

[5] Edmonds, J.: Paths, trees and flowers. Canadian Math. Jl., 17, p. 449, 1965.

[6] Edmonds, J.: Matroids and the Greedy Algorithm. Mathematical Programming, 1 (1971) 127–136.

[7] Edmonds, J.: The Chinese Postman Problem. Operations Research, 13, Suppl. 1 (1965) 373.

[8] Euler, L.: Solutio problematis ad geometriam situs pertinentis. Comm. Acad. Sci. Imp. Petropol. 8 (1736), 128–140.

[9] Kruskal, J.B., Jr.: On the shortest spanning subtree of a graph and the traveling salesman problem. Proc. Amer. Math. Soc. 7:1, 48–50, 1956.

[10] Kwan, M–K.: Graphic programming using odd or even points, Chinese Mathematics, 1, 273, 1962.

[11] Lawler, Eugene L.: Combinatorial Optimization: Networks and Matroids. Holt, Rinehart and Winston. 1976.

CHAPTER 5

CIRCUITS, CUTSETS AND TREES.

36. THE INCIDENCE MATRIX. In this chapter, we shall describe some matrices which in a natural way is obtained from a graph. Those matrices are used in the theory of electrical networks, and the description here leads directly to this application. But the matrices can also be used in other connections. For example in section 10 we used the incidence matrix of a graph to represent the graph on a computer. In section 40 we describe to vector spaces obtained from a given graph. These vector spaces are closely connected to the theory of electrical networks.

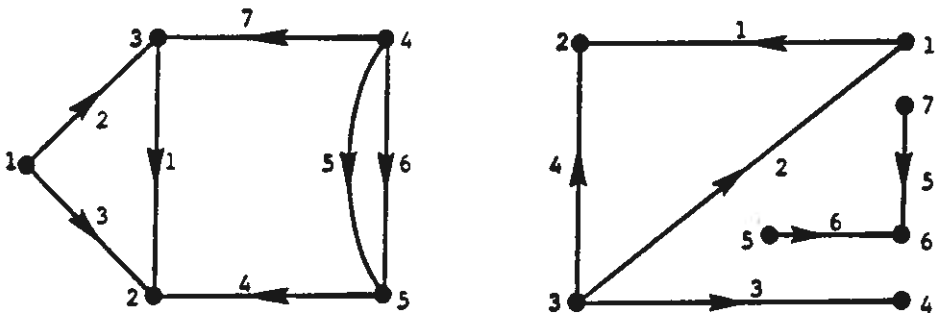
In the rest of this chapter we consider a directed graph and assume that the vertices of Γ are labelled with the numbers $1, 2, ..., P$ and that the edges of Γ are labelled $1, 2, ..., K$. Therefore $P = |\mathscr{P}(\Gamma)|$ and $K = |\mathscr{K}(\Gamma)|$.

It is necessary to have directed edges. But the problems we are dealing with here are such that the result of a change of a direction of an edge is only some trivial changes of signs in the results. Therefore we can change directions as we want without creating an essential change in the problem. (– In the theory of transport networks this is not the case.)

The incidence matrix for a graph Γ without loops is defined to be the $P \times K$ – matrix $A = \{a_{rs}\}$ for which

$$a_{rs} = \begin{cases} 1 & \text{when vertex } r \text{ is the start vertex of edges } s\,, \\ -1 & \text{when vertex } r \text{ is final vertex of edges } s\,, \\ 0 & \text{when vertex } s \text{ not incident to edge } s\,. \end{cases}$$

On the figure below, we have drawn two graphs and written the incidence matrices.



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

$\mathbf{A}$ is called the incidence matrix of Γ even though $\mathbf{A}$ depends on the order in which the vertices and edges are labelled. A new labelling will result in some row and column interchanges.

Each element of the incidence matrix of a graph is 0,1 or -1 . In each column of $\mathbf{A}$ there are exactly two elements $\neq 0$, a 1 and a -1 . Conversely, every matrix with these properties will be incidence matrix of a graph Γ . $\mathbf{A}$ determines Γ uniquely, up to isomorphism.

Theorem 36.1. Let $\mathbf{A}$ denote the incidence matrix of a directed graph Γ (without loops) with P vertices and S connected components. Then the rank $\rho(\mathbf{A}) = P - S$. When Γ is connected, $P - 1$ arbitrary rows in $\mathbf{A}$ are linearly independent.

Proof. First we shall show that if Γ is connected, then $\rho(\mathbf{A}) = P - 1$. $\mathbf{A}$ has P rows, the sum of which is $(0\ 0\ \ldots\ 0)$. Hence $\rho(\mathbf{A}) \leq P - 1$. We shall now prove that $P - 1$ arbitrary rows in $\mathbf{A}$ are linearly independent. The proof is indirect. Therefore assume that there exist a system of at most $P - 1$ rows which are linearly dependent. By changing the order of the vertices in Γ we can reach a situation where there exists a number q $(1 \leq q \leq P - 1)$ such that there are numbers $t_1, t_2, \ldots, t_q$, all $\neq 0$, for which

$$t_1\mathbf{r}_1 + t_2\mathbf{r}_2 + \ldots + t_q\mathbf{r}_q = \mathbf{0} ,$$

where $\mathbf{r}_i$ is row i in $\mathbf{A}$. It follows that each column in the matrix

$$\mathbf{A}' = \begin{bmatrix} \mathbf{r}_1 \\ \mathbf{r}_2 \\ \vdots \\ \mathbf{r}_q \end{bmatrix}$$

contains no or two elements $\neq 0$. If all columns in A' contained a 1 as well as a -1 , then vertex P would be isolated, contradicting the fact, that Γ is connected. Hence at least one of the columns in A' contains only 0's. All columns in A' can not be 0's, because then vertex 1 would be isolated. We can therefore change the labelling of the edges in Γ , such that A' gets the form

$$A' = (A_1\ 0)\ ,$$

where each column in A_1 contains as well a 1 as a -1 . But then A itself must have the structure

$$A = \begin{bmatrix} A_1 & 0 \\ 0 & A_2 \end{bmatrix} .$$

This shows, that Γ is not connected. This contradiction shows, that if Γ is connected, then $\rho(A) = P - 1$.

Next, let us consider a graph with S connected components, $\Gamma_1, \Gamma_2, ..., \Gamma_S$. Let us put $P_i = |\mathscr{P}(\Gamma_i)|$, $K_i = |\mathscr{K}(\Gamma_i)|$ for $i = 1, ..., S$. Now we chose the labelling of $\mathscr{P}(\Gamma)$ and $\mathscr{K}(\Gamma)$ such that the vertices and edges of Γ_1 have labels $1, 2, ..., P_1$, and $1, 2, ..., K_1$, such that the vertices and edges of Γ_2 have labels $P_1 + 1, P_1 + 2, ..., P_1 + P_2$ and $K_1 + 1, K_1 + 2, ..., K_1 + K_2$, and so on. Then the incidence matrix of Γ obtains the structure

$$A = \begin{bmatrix} A_1 & 0 & \dots & 0 \\ 0 & A_2 & \dots & 0 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \dots & A_S \end{bmatrix} ,$$

where A_i is incidence matrix for Γ_i , $i = 1, 2, ..., S$. It follows, that

$$\rho(A) = \rho(A_1) + \rho(A_2) + ... + \rho(A_S) = P_1 - 1 + P_2 - 1 + ... + P_S - 1 = P - S\ .$$

□

Later we shall see, that in the theory of electrical networks A occurs as a coefficient matrix for a system of linear equations. Therefore it is interesting to characterize the regular submatrices of A . Before we can do that we must prove

Lemma 36.2. Let C be a circuit in a directed graph Γ without loops. Then the columns in A which corresponds to the edges of C are linearly dependent.

Proof. If we change the direction of one of the edges in C, then the elements in the corresponding column in A changes sign. Since this has no influence on linear dependence, we may assume that C is a one way circuit. Let $c_1, c_2, \ldots, c_t$ denote the columns in A which corresponds to the edges of C. In the $P \times t -$ matrix

$$(c_1, c_2, \ldots, c_t)$$

each row then contains one 1 and one -1. Therefore $c_1 + c_2 + \ldots + c_t = 0$, and the columns are linearly dependent.

□

Theorem 36.3. Let A denote the incidence matrix for a directed, connected graph Γ (without loops) with P vertices. Let A' be a $(P-1) \times (P-1) -$ submatrix in A. Then A' is regular (that is $\det A' \neq 0$) if and only if the columns of A' corresponds to the brances of a spanning tree in Γ.

Proof. If the columns in A' corresponds to the edges of a spanning tree T, then the rows in A' are $P-1$ rows in the incidence matrix of T. Then it follows from the last part of theorem 36.1 that the rows in A' are linearly independent and hence A' is regular.

Conversely, if A' is regular, then the columns of A' are linearly independent. If T is the spanning subgraph in Γ, whose edge set corresponds to the columns of A', then it follows from the lemma, that T contains no circuit. Since $|\mathscr{K}(T)| = P - 1$ it follows from theorem 4.1 that T is a spanning tree.

It follows from theorem 36.1, that in many cases, in a connected graph we can use A' instead of A, where A' is obtained from A by deletion of a row. More general, for an arbitrary graph we can consider a submatrix A_r of A consisting of $P - S$ linearly independent rows. A_r is called a *reduced incidence matrix* for Γ. An important property of A_r is, that if i denotes a vector of n unknown quantities $(i_1, i_2, \ldots, i_n)$, then the equation systems

$$A\,i = 0 \quad \text{and} \quad A_r\,i = 0$$

are equivalent. A_r can be obtained from A by chosing a vertex in each component of Γ and deleting the rows of A corresponding to these vertices.

When Γ is a graph with P vertices, Q edges and S connected components, then the number $\rho = \rho(\Gamma) = P - S$ is called the *rank* of Γ. So, the rank of Γ is the same as the rank of the incidence matrix of Γ.

A matrix is called totally unimodular when each subdeterminant is 0 or ± 1. A matrix is unimodular when every $r \times r -$ subdeterminant is 0 or ± 1, where r is the rank of the matrix. The fact that the incidence matrix of a directed graph is totally unimodular is important as well for application in operations research as in the theory of electrical networks.

Theorem 36.4. The incidencematrix for a directed graph without loops is totally unimodular.

Proof. Let $\mathbf{A}$ be the incidence matrix of Γ and let $\mathbf{M}$ be a $t \times t -$ submatrix of $\mathbf{A}$. We shall show, that $\det\mathbf{M}$ is 0 or ± 1. The proof is by induction with respect to t. The statement is true for $t = 1$. Next assume that every $(t-1) \times (t-1) -$ submatrix in $\mathbf{A}$ has determinant 0 or ± 1. If $\mathbf{M}$ contains a column of 0's, then $\det\mathbf{M} = 0$. If every column in $\mathbf{M}$ contains as well a 1 as a -1, then the sum of the rows is 0 and $\det\mathbf{M} = 0$. Finally assume, that there is a column in $\mathbf{M}$ which contains exactly one element $\neq 0$. Then $\det\mathbf{M} = \pm \det\mathbf{M}_1$, where $\mathbf{M}_1$ is a $(t-1) \times (t-1)$ submatrix in $\mathbf{A}$. By the induction hypothesis, $\det\mathbf{M}_1$ is 0 or ± 1, and the theorem is proved. □

Remark, that if we write the equilibrium condition for a transport network such as it is done on page 3.22 then the coefficient matrix is the incidence matrix.

37. KIRCHHOFFS TREE THEOREM. When an application of mathematics is closely connected to a mathematical theory, then it happens, that the application by necessity introduces some concepts which – from a mathematical point of view – is less natural. For example, let us consider the relation between Königs theorem (or Mengers theorem) and the max–flow–min–cut theorem. The theorem of König and Menger deals with not directed graphs without edge – capacities; before we can state the max–flow–min–cut theorem, which is near to applications, we have to introduce directions, capacities and source and terminal. It is the same with Kirschhoff's tree theorem: There is a simple and nice version, but the applications in the theory of electrical networks is more complicated, but not deeper.

There is a group of theorems, which are closely related to Kirchhoffs tree theorem. The basic tool in the proofs of all these theorems is a theorem from linear algebra, which is called Binet–Cauchy's theorem. Consider two matrices $\mathbf{M}$ and $\mathbf{N}$, where $\mathbf{M}$ is a $m \times n -$ matrix, and $\mathbf{N}$ is a $n \times m -$ matrix, where $n \geq m$. If we delete $n - m$ columns in $\mathbf{M}$, the result is a square matrix $\mathbf{M}_i$. The *corresponding* submatrix $\mathbf{N}_i$ in $\mathbf{N}$ is obtained from $\mathbf{N}$ by deleting those rows which have the same numbers as the columns we deleted in $\mathbf{M}$. Then we have

Binet – Cauchy's theorem. When $\mathbf{M}$ is an $m \times n -$ matrix and $\mathbf{N}$ is an $n \times m -$ matrix, $n \geq m$, then

$$\det \mathbf{MN} = \sum_i \det \mathbf{M}_i \det \mathbf{N}_i ,$$

where the summation is over all pairs $(\mathbf{M}_i, \mathbf{N}_i)$ of corresponding $m \times m -$ submatrices in respectively $\mathbf{M}$ and $\mathbf{N}$.

For example if

$$\mathbf{M} = \begin{bmatrix} 1 & 2 & -1 \\ 3 & 1 & 1 \end{bmatrix}, \qquad \mathbf{N} = \begin{bmatrix} 1 & 2 \\ 0 & 1 \\ 1 & -2 \end{bmatrix} .$$

we find that

$$\det \mathbf{MN} = \det \begin{bmatrix} 0 & 6 \\ 4 & 5 \end{bmatrix} = -24 ,$$

and

$$\sum_i \det \mathbf{M}_i \det \mathbf{N}_i = \begin{vmatrix} 1 & 2 \\ 3 & 1 \end{vmatrix} \begin{vmatrix} 1 & 2 \\ 0 & 1 \end{vmatrix} + \begin{vmatrix} 1 & -1 \\ 3 & 1 \end{vmatrix} \begin{vmatrix} 1 & 2 \\ 1 & -2 \end{vmatrix} + \begin{vmatrix} 2 & -1 \\ 1 & 1 \end{vmatrix} \begin{vmatrix} 0 & 1 \\ 1 & -2 \end{vmatrix}$$

$$= -5 \cdot 1 + 4 \cdot (-4) + 3 \cdot (-1) = -5 - 16 - 3 = -24 .$$

Remark, that if $\mathbf{R}$ is an $n \times n -$ diagonalmatrix, then it follows Binet – Cauchys theorem, that

$$\det \mathbf{M}\, \mathbf{R}\, \mathbf{N} = \sum_i \det \mathbf{M}_i \cdot \mathbf{R}_i \cdot \det \mathbf{N}_i , \tag{1}$$

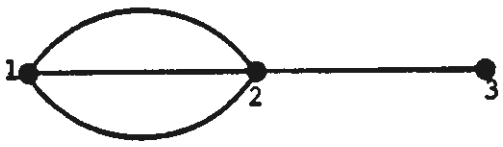
where R_i is the product of the diagonal elements in those rows of $\mathbf{R}$, which have the same numbers as those columns in $\mathbf{M}$ which constitutes $\mathbf{M}_i$.

We shall not prove Binet – Cauchy's theorem here. Many textbooks in linear algebra contains a proof, [2].

Let us consider a graph without loops in which the vertices are labelled $1, 2, ..., P$. Then the conductance matrix of Γ is defined as the $P \times P$ – matrix $\mathbf{K}$ for which the element k_{rs} in row r and column s is

$$k_{rs} = \begin{cases} (-1) \cdot (\text{no. of } r,s\text{–edges in } \Gamma) \text{ when } r \neq s . \\ \text{the valency of vertex } r \text{ when } r = s . \end{cases}$$

Below we show an example:



$$\mathbf{K} = \begin{bmatrix} 3 & -3 & 0 \\ -3 & 4 & -1 \\ 0 & -1 & 1 \end{bmatrix}$$

Clearly, if Γ is directed and if $\mathbf{A}$ is the incidence matrix of $\mathbf{A}$, then

$$\mathbf{K} = \mathbf{A}\,\mathbf{A}^{\mathsf{T}} .$$

The number of spanning trees in a graph Γ is denoted $\tau(\Gamma)$, the *tree number* of Γ. Even small graphs have normally large tree numbers. But these numbers can easily be computed using the following theorem. It is called Kirchhoff's tree theorem, even through is is only found implicitly in the works of Kirchhoff.

Kirchhoff's Tree theorem. Let $\mathbf{K}$ be the conductance matrix for a connected graph Γ and let $\mathbf{K}_i$ be the matrix obtained from $\mathbf{K}$ by deleting row i and column i. Then $\det \mathbf{K}_i$ is independent of i and

$$\det \mathbf{K}_i = \tau(\Gamma) ,$$

where $\tau(\Gamma)$ is the number of spanning trees in Γ.

As an example we consider the graph in the figure above:

$$\det K_1 = \begin{vmatrix} 4 & -1 \\ -1 & 1 \end{vmatrix} = 3\,, \quad \det K_2 = \begin{vmatrix} 3 & 0 \\ 0 & 1 \end{vmatrix} = 3\,, \quad \det K_3 = \begin{vmatrix} 3 & -3 \\ -3 & 4 \end{vmatrix} = 3\,,$$

and Γ has 3 spanning trees.

Proof: Let us direct the edges of Γ in an arbitrary way and consider the matrix A_i obtained from the incidence matrix be deleting row i. The it is easy to see, that

$$K_i = A_i\, A_i^T$$

and hence by Cauchy – Binets theorem

$$\det K_i = \det A_i\, A_i^T = \sum_j \det A_i^j \det A_i^{jT}\,,$$

where the summation is over all $(P-1) \times (P-1)$ – submatrices A_i^j in A_i. It follows from theorem 36.3, the number of terms $\neq 0$ in this sum is $\tau(\Gamma)$, and according to theorem 36.4 each if these terms has value 1.

□

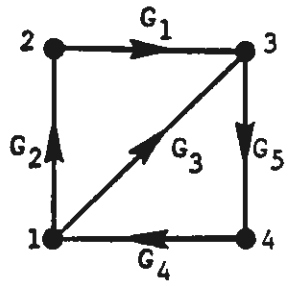
It is easy to see that all P^2 complements in K has value $\tau(\Gamma)$. The following generalisation of Kirchhoff's theorem shows that it is also possible to obtain a determinant the value of which is a "list" containing the spanning trees in Γ. For this purpose we attach to each edge k a variable G_k and consider the matrix

$$G = \begin{bmatrix} G_1 & 0 & \cdots & 0 \\ 0 & G_2 & \cdots & 0 \\ \cdot & \cdot & \cdots & \cdot \\ 0 & 0 & \cdots & G_K \end{bmatrix}.$$

Let A_r be the incidence matrix of Γ. Then the matrix $A_r\, G$ is obtained from A_r by multiplying column j by G_j for $j = 1, 2, ..., K$. By making a few changes in the proof of Kirchhoff's tree theorem, we obtain the equation

$$\det A_r\, G\, A_r^T = \sum \prod_{G_j \in \mathcal{T}} G_j\,, \tag{1}$$

where the summation is over all edge sets $\mathcal{T}$ for spanning trees in Γ. In connection with applications in the theory of electrical networks, G_j is called the *conductance* of edge k and the matrix $A_r\, G\, A_r^T$ is called the *node conductance matrix.* By (1) the value of this determinant is independent of the choice of reference vertex.



Since $(A_r\ G\ A_r^T)^T = A_r\ G\ A_r^T$, the node conductance matrix is symmetric. As an example we consider the graph above:

$$\det\begin{bmatrix} 0 & 1 & 1 & -1 & 0 \\ 1 & -1 & 0 & 0 & 0 \\ -1 & 0 & -1 & 0 & 1 \end{bmatrix}\begin{bmatrix} G_1 & 0 & 0 & 0 & 0 \\ 0 & G_2 & 0 & 0 & 0 \\ 0 & 0 & G_3 & 0 & 0 \\ 0 & 0 & 0 & G_4 & 0 \\ 0 & 0 & 0 & 0 & G_5 \end{bmatrix}\begin{bmatrix} 0 & 1 & 1 \\ 1 & -1 & 0 \\ 1 & 0 & -1 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{vmatrix} G_2+G_3+G_4 & -G_2 & -G_3 \\ -G_2 & G_1+G_2 & -G_1 \\ -G_3 & -G_1 & G_1+G_3+G_5 \end{vmatrix}$$

$$= G_1G_2G_4 + G_1G_2G_5 + G_1G_3G_4 + G_1G_4G_5 + G_2G_3G_4 + G_2G_3G_5 + G_2G_4G_5\,.$$

Observe that the matrix is symmetric and that the terms on the right hand side corresponds to the trees in Γ .

Formula (1) can be used to calculate the determinant by graph theoretic methods. For this purpose algorithms has been constructed which generates all trees in a given graph.

38. CIRCUITS AND CUT SETS. In the remaining part of this chapter we shall prepare the treatment of electrical networks in the next chapter. We shall try to obtain a high degree of symmetri between the concepts relating to electrical current on one side, and the corresponding concepts relating to voltage on the other side. A pair of concepts, which in this way correspond to each other are called dual. For example, deletion and contraction of an edge are dual concepts. If, for example, we want to give a generator the value 0 , then for a voltage generator you may contract the corresponding edge in the graph, whereas for a current generator, you have to delete the edge.

In the rest of this chapter we shall not distinguish between a subgraph D of Γ and its edge set $\mathscr{D}$. Except that in the name we replace D by $\mathscr{D}$. So, when we speak about a circuit $\mathscr{C}$ then we refer to the edge set of a circuit C . Similarly $\mathscr{T}$ is the edge set of a spanning tree T in Γ , in the case were Γ is connected. If Γ has S connected components we define *a tree* $\mathscr{T}$ *in* Γ as the edge set of a maximal

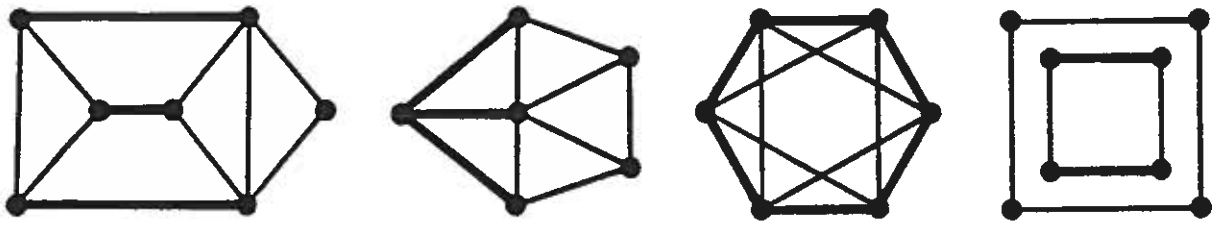
spanning forest in Γ , that is a subgraph consisting of a spanning tree in each connected component in Γ . $|\mathcal{T}| = P - S$. The complement $\mathcal{T}^* = \mathcal{K}(\Gamma)\backslash\mathcal{T}$ is called a *cotree* in Γ , $|\mathcal{T}^*| = K - P + S$. The numbers

$$\sigma = \sigma(\Gamma) = P - S \quad \text{and} \quad \sigma^* = \sigma^*(\Gamma) = K - P + S$$

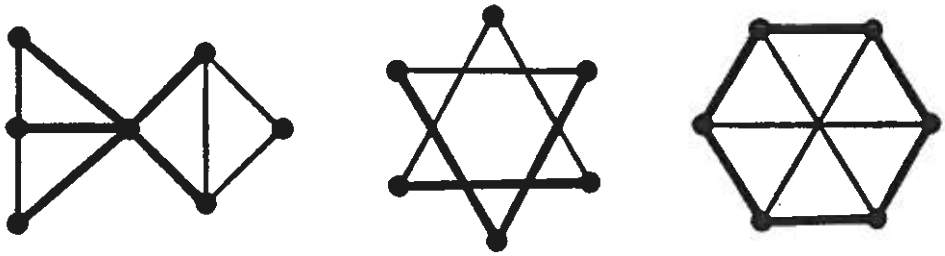
are called the *rank*, respectively the *corank* or *nullity* of Γ . The nullity is also denoted $\mu = \mu(\Gamma)$.

When $\mathcal{T}$ is a tree in Γ , then an edge $k \in \mathcal{T}$ is called a *branch* in $\mathcal{T}$ and an edge $k \in \mathcal{K}(\Gamma)\backslash\mathcal{T}$ is called a *chord* (w.r.t. $\mathcal{T}$).

A set $\mathcal{D}$ of edges on a graph Γ is called a *cutset* (or *cocircuit*) in Γ , when $\Gamma \setminus \mathcal{D}$ has more connected components then Γ , and when, for every proper subset $\mathcal{D}'$ of $\mathcal{D}$, Γ and $\Gamma \setminus \mathcal{D}'$ have the same number of connected components. In each of the four graphs in the figure the set $\mathcal{D}$ of edges drawn with heavy lines is a cutset.



This is not the case in any of the three graphs below. In all three cases $\Gamma - \mathcal{D}$ has more connected components than Γ , but $\mathcal{D}$ is not minimal w.r.t. this property. Note, that if x is not a cut vertex, the edges incident to x is a cutset.



In general, at set $\mathcal{M}$ is called relatively minimal w.r.t. a property E when $\mathcal{M}$ has the property E and $\mathcal{M}$ has no proper subset with property E . $\mathcal{M}$ is called cardinality minimal w.r.t. E if $\mathcal{M}$ has property E and no set with fewer elements than $|\mathcal{M}|$ has property E . Corresponding definitions can be applied when "minimal" is replace by maximal. Using this terminaligy, a cutset can be defined as a relatively minimal edgecut.

A circuit $\mathscr{C}$ could be defined as a relatively minimal subset of $\mathscr{K}(\Gamma)$ which is not contained in any tree in Γ and a cutset could be defined as a relatively minimal subset of $\mathscr{K}(\Gamma)$ which is not contained in any cotree in Γ. Because the symmetry in these definitions we consider tree and cotree as dual concepts. Circuit and cutset are also dual and so are rank and corank.

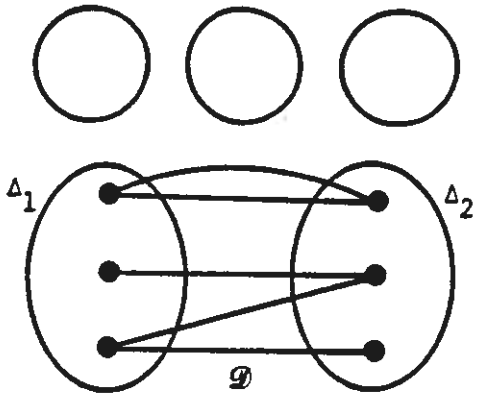
A tree in Γ can be characterized as well as a relatively maximal as a cardinality maximal circuit free set of edges. Correspondingly, a cotree is a relatively maximal and a cardinality maximal cutset free set of edges.

If $\mathscr{M}$ is an edge cut in a graph Γ (that is, if $\Gamma - \mathscr{M}$ contains more connected components than Γ), then $\mathscr{M}$ contains a cutset $\mathscr{D}$. $\mathscr{D}$ is obtained from $\mathscr{M}$ by deleting edges from $\mathscr{M}$ until $\mathscr{M}$ becomes a relatively minimal cutset.

In the following theorem we describe cutsets in a new way:

Theorem 38.1. A set $\mathscr{D}$ of edges in a graph Γ is a cutset in $\Gamma \Leftrightarrow$ there exists a connected component Δ in Γ and two disjoint connected subgraphs Δ_1 and Δ_2 in Δ, such that $\mathscr{P}(\Delta) = \mathscr{P}(\Delta_1) \cup \mathscr{P}(\Delta_2)$ such that $\mathscr{D}$ is the set of Δ_1, Δ_2 – edges in Γ.

Proof: $\Longleftarrow$. Clearly, see the figure, a set $\mathscr{D}$ of edges in Γ obtained in the way



described in the theorem is a cutset in Γ.

$\Longrightarrow$. Conversely, let $\mathscr{D}$ be a cutset, $\mathscr{D} = \{k_1, k_2, ..., k_n\}$. Then the graph $\Gamma' = \Gamma - \{k_1, k_2, ..., k_{n-1}\}$ has the same no of connected components as Γ and k_n must be bridge in Γ'. $\Gamma - \mathscr{D} = \Gamma' - k_n$ has therefore two new connected components Δ_1 and Δ_2 and since $\mathscr{D}$ is a relatively minimal edge cut $\mathscr{D}$ is the set of Δ_1, Δ_2 – edges in Γ.

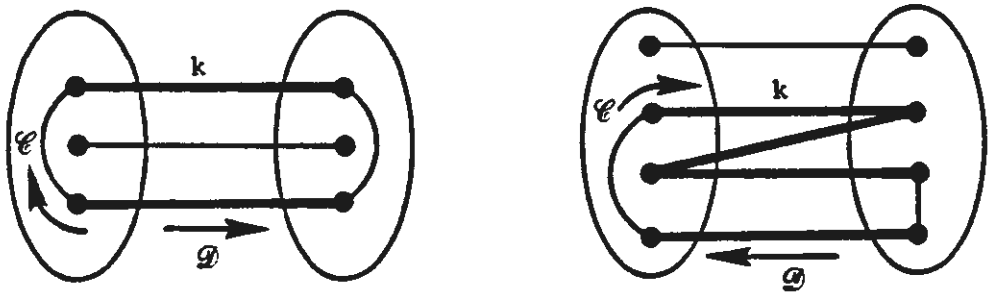
□

From theorem 38.1 it follows, that if $\mathscr{D}$ is a cutset in Γ, then all edges of $\mathscr{D}$ is in the same component of Γ and the number of connected components in $\Gamma - \mathscr{D}$ is exactly one more than in Γ.

An important relation between circuits and cutsets is expressed in the following theorem:

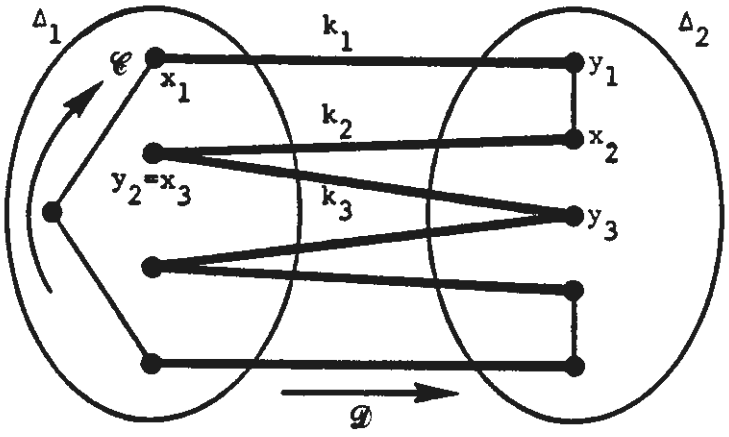
Theorem 38.2. A circuit and a cutset in a graph has always an even number of edges in common.

We shall prove this theorem together with the following theorem 38.3. But before we can do that we mention, that we can direct a circuit $\mathscr{C}$ in a graph Γ by drawing an arrow along $\mathscr{C}$ in a drawing. In a similar way we can direct a cutset $\mathscr{D}$ of Δ_1, Δ_2 – edges as in theorem 38.1 by drawing an arrow from Δ_1 to Δ_2 or conversely. If an edge k belongs to as well $\mathscr{C}$ as $\mathscr{D}$, then the two directions can agree along k as in the figure to the right, or they can disagree as in the figure to the right.



Theorem 38.3. Assume, that in a graph Γ a circuit $\mathscr{C}$ and a cutset $\mathscr{D}$ are both directed. The the directions agree along precisely half the edges of $\mathscr{C} \cap \mathscr{D}$.

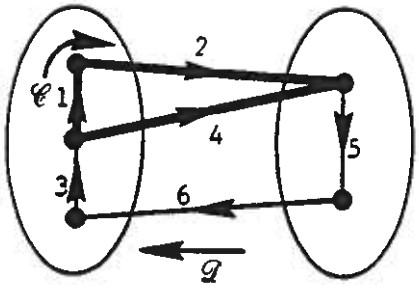
Proof of the theorems: We may assume, that Γ is connected and that $\mathscr{C}$ and $\mathscr{D}$ has at least one edge in common. Let us traverse $\mathscr{C}$, in the direction of $\mathscr{C}$ starting in an arbitrary vertex. Let $k_1 = (x_1, y_1)$, $k_2 = (x_2, y_2)$, ..., $k_n = (x_n, y_n)$ denote the edges of $\mathscr{C} \cap \mathscr{D}$ in the order we meet them, compare the figure. $\Gamma - \mathscr{D}$ has two connected components Δ_1 and Δ_2. If, for example, $x_1 \in \mathscr{P}(\Delta_1)$, then $x_2 \in \mathscr{P}(\Delta_2)$, $x_3 \in \mathscr{P}(\Delta_3)$ and so on. Since we have to end in Δ_1, n must be even, $n = 2t$, and it



is obvious that if the directions of $\mathscr{C}$ and $\mathscr{D}$ agree along k_1, they agree also along $k_3, k_5,\ldots, k_{2t-1}$. And the directions do not agree along the rest of the edges. This proves the theorems. □

For the applications in network theory we need a directed version of theorem 38.3. But first we have to introduce incidence vectors. Let Γ be a directed graph in which the edges are labelled $1, 2,\ldots, K$, and let $\mathscr{C}$ and $\mathscr{D}$ be a circuit and a cutset in Γ.

Then the incidence vectors



$$\underrightarrow{\mathscr{C}} = (c_1, c_2,\ldots,c_K) \text{ and } \underrightarrow{\mathscr{D}} = (d_1, d_2,\ldots,d_K)$$

are defined as follows:

$$c_i = \begin{cases} 0 \text{ if edge } i \text{ does not belong to } \mathscr{C}, \\ +1 \text{ if edge } i \in \mathscr{C} \text{ and the directions agree}, \\ -1 \text{ if edge } i \in \mathscr{C} \text{ and the directions disagree}, \end{cases}$$

$$d_i = \begin{cases} 0 \text{ if edge } i \text{ does not belong to } \mathscr{D}, \\ +1 \text{ if edge } i \in \mathscr{D} \text{ and the directions agree}, \\ -1 \text{ if edge } i \in \mathscr{D} \text{ and the directions disagree}, \end{cases}$$

For the circuit $\mathscr{C}$ and the cutset $\mathscr{D}$ in the figure we obtain

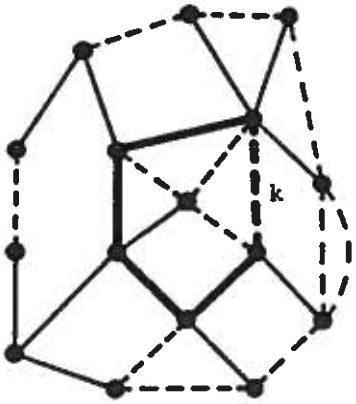
$$\underrightarrow{\mathscr{C}} = (1\ \ 1\ \ 0\ -1\ \ 0\ \ 0) \quad , \quad \underrightarrow{\mathscr{D}} = (0\ -1\ \ 0\ -1\ \ 0\ \ 1)\,.$$

If i is a common edge for $\mathscr{C}$ and $\mathscr{D}$ and if the directions of $\mathscr{C}$ and $\mathscr{D}$ agree along i , then $\underrightarrow{\mathscr{C}}$ and $\underrightarrow{\mathscr{D}}$ has the same i^{th} coordinate. If the directions disagree, the i^{th} coordinates are opposite. Therefore the contents of theorem 38.3 is that $\underrightarrow{\mathscr{C}} \cdot \underrightarrow{\mathscr{D}} = 0$:

> ***The Orthogonality Theorem:*** If $\mathscr{C}$ is a circuit in Γ and $\mathscr{D}$ is a cutset in Γ , then $\underrightarrow{\mathscr{C}} \cdot \underrightarrow{\mathscr{D}} = 0$.

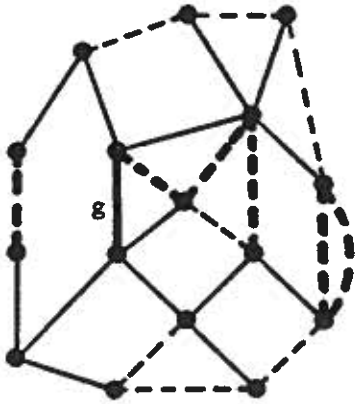
39. FUNDAMENTAL CIRCUITS AND FUNDAMENTAL CUTSETS. In the preceeding section we have described

a tree $\mathscr{T}$ in a graph Γ as a relatively maximal circuit free subset of $\mathscr{K}(\Gamma)$. Therefore, if $k \in \mathscr{K}(\Gamma) \backslash \mathscr{T}$, then $\mathscr{T} \cup \{k\}$ contains at least one circuit. We shall show, that $\mathscr{T} \cup \{k\}$ contains exactly one circuit. This circuit is called the *fundamental circuit* for $\mathscr{T}$ determined by k and it is denoted $\mathscr{C}(k, \mathscr{T})$.



The figure shows a chord k w.r.t. a tree $\mathscr{T}$ (full lines). The fundamental circuit is shown with heavy lines.

a cotree $\mathscr{T}^*$ in a graph Γ as a relatively maximal cutset free subset of $\mathscr{K}(\Gamma)$. Therefore, if $g \in \mathscr{K}(\Gamma) \backslash \mathscr{T}^*$, then $\mathscr{T}^* \cup \{g\}$ contains at least one cutset. We shall show, that $\mathscr{T}^* \cup \{g\}$ contains exactly one circuit. This circuit is called the *fundamental cutset* for $\mathscr{T}^*$ determined by g and it is denoted $\mathscr{D}(g, \mathscr{T})$.



The figure shows a branch g w.r.t a tree $\mathscr{T}$ (full lines). The fundamental cutset is shown with heavy lines.

It is possible to give dual proofs of the two statements. However, it is more natural to give different proofs:

We shall first prove that $\mathcal{T} \cup \{k\}$ contains only one circuit: Since $\mathcal{T}$ contains no circuit every circuit $\mathcal{C}$ in $\mathcal{T} \cup \{k\}$ must contain k . Then $\mathcal{C} - k$ is a path in $\mathcal{T}$ connecting the end vertices of k . Since there exists only one such path, $\mathcal{C} \cup \{k\}$ contains only one circuit.

Next we shall prove that $\mathcal{T}^* \cup \{g\}$ contains only one cutset: $\Gamma - \mathcal{T}^*$ consists of a spanning tree in each connected component of Γ . One of these trees, say T_0 , contains g . $T_0 - g$ has two connected comparents T_0' and T_0'' and these are the only connected components in $\Gamma - (\mathcal{T}^* \cup \{g\})$ which is not a spanning tree in a connected component of Γ . A cutset in Γ , contained in $\Gamma^* \cup \{g\}$, must therefore necessarily be the set of T_0' , T_0'' – edges in Γ . Hence $\mathcal{T}^* \cup \{g\}$ contains only this cutset in Γ .

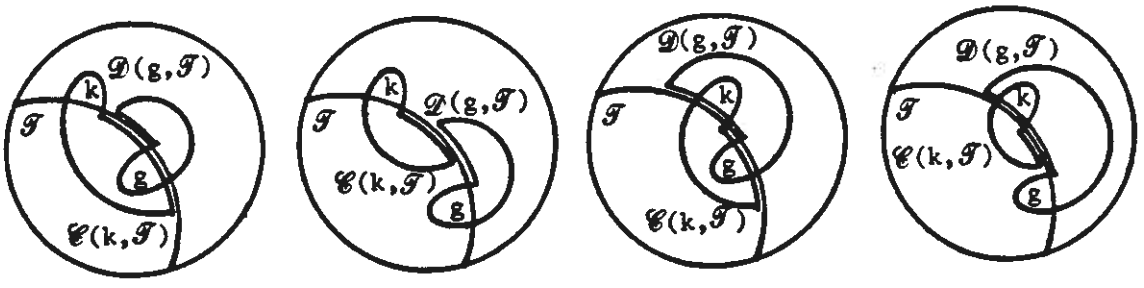
If Γ is a graph and $\mathcal{T}$ is a spanning tree in Γ , then there are $\rho(\Gamma)$ fundamental cutsets for $\mathcal{T}$ and $\rho^*(\Gamma)$ fundamental circuits for $\mathcal{T}$. The following theorem, which is self dual, characterize the edges belonging to the fundamental circuits and the fundamental cutsets:

Theorem 39.1. Let $\mathcal{T}$ be a tree in a graph Γ , let $g \in \mathcal{T}$ and $k \in \mathcal{K}(\Gamma) \backslash \mathcal{T}$. Then if $\mathcal{C}(k,\mathcal{T})$ is the fundamental circuit determined by k and if $\mathcal{D}(g,\mathcal{T})$ is the fundamental cutset determined by g , we have:

$$g \in \mathcal{C}(k,\mathcal{T}) \Leftrightarrow k \in \mathcal{D}(g,\mathcal{T}) .$$

If g and k are directed and determines the same direction (or opposite directions) of $\mathcal{C}(k,\mathcal{T})$, then g and k determines opposite directions (or the same direction) of $\mathcal{D}(g,\mathcal{T})$.

Proof: Consider $\mathcal{C}(k,\mathcal{T})$ and $\mathcal{D}(g,\mathcal{T})$. In advance, we could imagine the four possibilities shown in the figures, where the circle symbolizes $\mathcal{K}(\Gamma)$. However, in the first and the last case $\mathcal{C}(k,\mathcal{T})$ and $\mathcal{D}(g,\mathcal{T})$ has exactly one element in common,

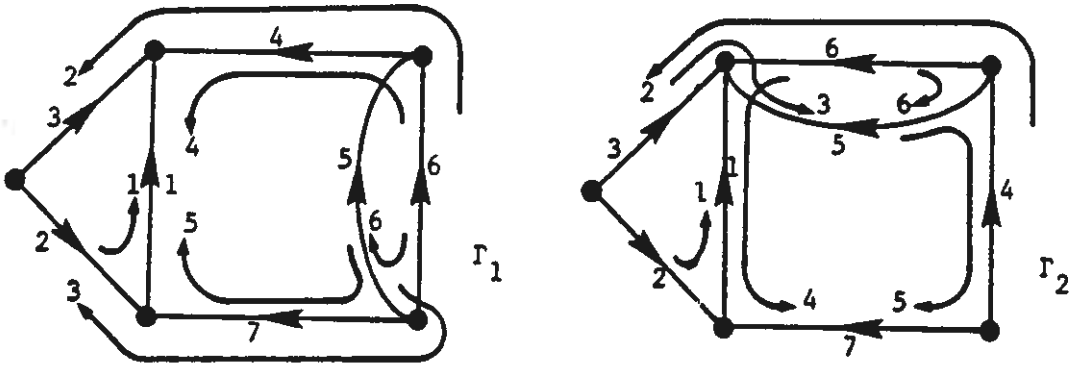


contradicting theorem 38.2. This proves the first part of theorem 39.1. The last part of the theorem is easy to prove, either be means of a figure or by means of theorem 38.3. □

Assume now, that the circuits in Γ are labelled by the numbers 1, 2, ... , c , and that each circuit is directed (by means of an arrow). Then we define the *circuit matrix* of Γ as the $c \times K$–matrix C , in which the element c_{rs} in row r and column s is defined by

$$c_{rs} = \begin{cases} 1 & \text{if edge s belongs to circuit r and the directions agree,} \\ -1 & \text{if edge s belongs to circuit r and the directions disagree} \\ 0 & \text{if edge s do not belong to circuit r .} \end{cases}$$

In the figure below we have drawn two non isomorphic graphs Γ_1 and Γ_2 , having the same circuit matrix C .



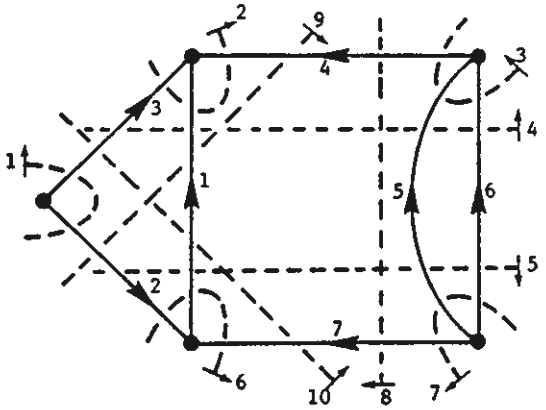
$$C = \begin{bmatrix} 1 & 1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 1 & -1 & 1 & 0 & 1 & -1 \\ 0 & -1 & 1 & -1 & -1 & 0 & 1 \\ -1 & 0 & 0 & 1 & 0 & 1 & -1 \\ 1 & 0 & 0 & -1 & -1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & -1 & 0 \end{bmatrix}$$

In general, it is complicated to decide whether a given matrix C is circuit matrix for same graph.

Again, let Γ be a directed graph and assume that the cutsets in Γ are labelled 1, 2, ..., d . Denote cutset i by $\mathscr{D}_i$. Assume also that each cutset is directed, as explained p 5.12. Then the cutset matrix of Γ is the $d \times K$ – matrix D in which the element in row r and column s is defined by

$$d_{rs} = \begin{cases} 1 & \text{if edge s belongs to cutset r and the directions agree,} \\ -1 & \text{if edge s belongs to cutset r and the directions disagree,} \\ 0 & \text{if edge s do not belong to cutset r .} \end{cases}$$

The concept is illustrated in the figure:



$$\mathbf{D} = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ -1 & 0 & -1 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 1 & 0 \\ -1 & 1 & 0 & 0 & -1 & -1 & 0 \\ 1 & -1 & 0 & 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ -1 & 1 & 0 & -1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & -1 \end{bmatrix}$$

Alternatively, $\mathbf{C}$ could be defined as a matrix in which the rows are incidence vectors of the circuits and $\mathbf{D}$ could be defined as a matrix in which the rows are incidence vectors of the cutset. Then by the orthogonality theorem,

Theorem 39.2. If $\mathbf{C}$ is the circuit matrix of Γ and if $\mathbf{D}$ is the cutset matrix of Γ, then

$$\mathbf{C}\,\mathbf{D}^{\mathsf{T}} = 0\,,$$

where $\mathbf{D}^{\mathsf{T}}$ denotes the matrix obtained from $\mathbf{D}$ by interchanging rows and columns.

In the theory of electrical networks the matrices $\mathbf{C}$ and $\mathbf{D}$ occurs as coefficient matrices in certain system of linear equations. Therefore, it is of interest to find the rand of these matrices. The answer is given in the following theorem. It is surprising by only depending in a very simple fashion on the structure of the graph.

Theorem 39.3. Let Γ be a directed graph with P vertices, K edges and S connected components.

Then the rank if the circuit $\mathbf{C}$ of Γ is $\mu=\rho(\mathbf{C})=K-P+S=\rho^*(\Gamma)=\mu(\Gamma)$.

Then the rank of the cutset matrix $\mathbf{D}$ of Γ is $\rho=\rho(\mathbf{D})=P-S=\rho(\Gamma)$.

Proof. Take a tree $\mathcal{T}$ in Γ and label the edges in Γ such that the chords of Γ gets the labels 1, 2, ..., K–P+S and the branches the labels K–P+S+1, ..., K . Furthermore we give the corresponding fundamental

circuits the labels 1, 2,..., K–P+S and assume that the directions of the circuits agrees with the directions of the corresponding chards. Then $\mathbf{C}$ gets the form

$$\mathbf{C} = \begin{bmatrix} 1 & 0 & \cdots & 0 & \\ 0 & 1 & \cdots & 0 & \mathbf{C}_{12} \\ \cdot & \cdot & \cdots & \cdot & \\ 0 & 0 & \cdots & 1 & \\ & \mathbf{C}_{21} & & & \mathbf{C}_{22} \end{bmatrix},$$

where the unity matrix is of order K–P+S . Since the first K–P+S rows are linearly independent we have

(1) $\rho(\mathbf{C}) \geq K - P + S$.

cutsets the labels 1, 2,..., P–S and assume that the directions of the cutset agrees with the directions of the corresponding cutsets. Then $\mathbf{D}$ gets the form

$$\mathbf{D} = \begin{bmatrix} & 1 & 0 & \cdots & 0 \\ \mathbf{D}_{11} & 0 & 1 & \cdots & 0 \\ & \cdot & \cdot & \cdots & \cdot \\ & 0 & 0 & \cdots & 1 \\ \mathbf{D}_{21} & & \mathbf{D}_{22} & & \end{bmatrix},$$

where the unity matrix is of order P–S . Since the first P–S rows are linearly independent we have

(1) $\rho(\mathbf{D}) \geq K - S$.

Let $\mathbf{d}_1, \mathbf{d}_2, \ldots, \mathbf{d}_\rho$ be an orthogonal base of the vector space spanned by the rows of $\mathbf{D}$ and let $\mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_\mu$ be an orthogonal base of the vector space spanned by the rows of $\mathbf{C}$. According to the Orthogonality Theorem the $\rho + \mu$ vectors

$$\mathbf{d}_1, \mathbf{d}_2, \ldots, \mathbf{d}_\rho, \mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_\mu$$

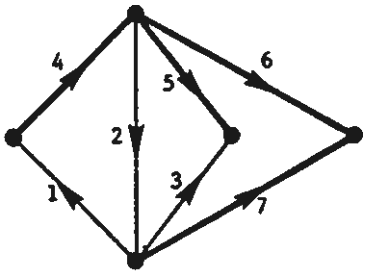
are pairwise orthogonal, and therefore linearly independent. Therefore $\rho + \mu \leq K$. But by (1) $\rho + \mu \geq K$. Therefore we can have no sharp inequality in any of the formulas (1).

□

A collection of circuits (or cutsets) in a directed graph Γ is called *linearly independent* if the corresponding rows in the circuit matrix (or the cutset matrix) are linearly independent. If the rows are linearly dependent, then the circuits (cutsets) are also called linearly dependent.

A *reduced circuit matrix* is a matrix consisting of $\mu(\Gamma)$ linearly independent rows in the circuit matrix $\mathbf{C}$ for Γ. A special kind of reduced circuit matrix is a *fundamental circuit matrix* $\mathbf{C}_f$, in which the rows are incidence vectors for the fundamental circuits w.r.t. a tree in Γ, and where the labellings and directions are chosen such that the first $\mu(\Gamma)$ columns is a unity matrix.

A *reduced cutset matrix* is a matrix consisting of $\rho(\Gamma)$ linearly independent rows in the cutset matrix $\mathbf{D}$ for Γ. A special kind of reduced cutset matrix is a *fundamental cutset matrix* $\mathbf{D}_f$, in which the rows are incidence vectors for the fundamental cutsets w.r.t a tree in Γ, and where the labellings and directions are chosen such that the last $\rho(\Gamma)$ columns is a unity matrix.



In the figure we have shown a graph Γ and four matrices $\mathbf{C}_f$, $\mathbf{C}_r$, $\mathbf{D}_f$ and $\mathbf{D}_r$ are written. In this case, the rows in $\mathbf{C}_r$ ($\mathbf{D}_r$) do not correspond to fundamental circuits (cutsets) for a tree.

$$\mathbf{C}_f = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 1 & -1 \\ 0 & 1 & 0 & 0 & 0 & -1 & 1 \\ 0 & 0 & 1 & 0 & -1 & 1 & -1 \end{bmatrix} \qquad \mathbf{D}_f = \begin{bmatrix} -1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ -1 & 1 & -1 & 0 & 0 & 1 & 0 \\ 1 & -1 & 1 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{C}_r = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 & -1 & 1 & -1 \end{bmatrix} \qquad \mathbf{D}_r = \begin{bmatrix} 0 & 1 & 0 & -1 & 1 & 1 & 0 \\ -1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & -1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 \end{bmatrix}$$

A moments reflection will show, that in terms of matrices theorem 39.1 becomes

Theorem 39.4. Let

$$C_f = (E_\mu\ C') \quad \text{and} \quad D_f = (D'\ E_\rho)$$

be a fundamental circuit matrix and a fundamental cutset matrix w.r.t. the same tree. E_μ and E_ρ denote unity matrices of order μ and ρ. Then

$$-C' = D'^{T} \ .$$

Finally, let us emphasize a couple of important relations between the incidence matrix A and the cutset matrix D for a graph Γ. Let us consider the row a_p in A, which corresponds to a vertex $p \in \mathcal{P}(\Gamma)$. If p is not a cut vertex in Γ, a_p is incidence vector for the cutset consisting of the edges incident with p. If p is a cutvertex, then the set of edges incident with p is disjoint union of cutsets, and a_p is sum of the corresponding incidence vectors. Hence every row in A is linear combination of rows in D. Since the two matrices have the same rank, the opposite is also true: Every row in D is linear combination of rows in A. If i is an unknown column vector, the equations $A\,i = 0$ and $D\,i = 0$ are equivalent.

40. CURRENT SPACE AND VOLTAGE SPACE. In this section we shall describe two vector spaces belonging to a graph. They are called the current space and the voltage space of the graph. We assume that the edges of Γ are labelled $1, 2, \ldots, K$.

A *voltage* in Γ is defined as a vector $v = (v_1, v_2, \ldots, v_K)$ for which $v \cdot \vec{\mathcal{C}} = 0$ for all circuits $\mathcal{C}$ in Γ. v_j is called the voltage across edge j. Let C_r be a reduced circuit matrix for Γ. Then v is a voltage iff

$$C_r\,v = 0 \ .$$

A current in Γ is defined as a vector $i = (i_1, i_2, \ldots, i_K)$ for which $i \cdot \vec{\mathcal{D}} = 0$ for all cutset $\mathcal{D}$ in Γ. i_j is called current in edge j. Let D_r be a reduced cutset matrix for Γ. Then i is a current iff

$$D_r\,i = 0 \ .$$

It follows, that the set of voltages in Γ is a $K-(K-P+S) = (P-S)$–dimensional subspace in $\mathbb{R}^K$. It is called the *voltage space* of Γ and denoted V_v. According to the orthogonality theorem every row in $\mathbf{D}_r$ is a voltage in Γ and since $\rho(\mathbf{D}_r)=P-S$ these rows constitutes a base of V_v.

It follows, that the set of current in Γ is a $K-(P-S) = (K-P+S)$–dimensional subspace in $\mathbb{R}^K$. It is called the *current space* of Γ and denoted V_i. According to the orthogonality theorem every row in $\mathbf{C}_r$ is a current in Γ and since $\rho(\mathbf{C}_r)=K-P+S$ these rows constitutes a base of V_i.

From this follows:

Theorem 40.1.

The currents in Γ is the set of linear combinations of incidence vectors for circuits in Γ.

The voltages in Γ is the set of linear combinations of incidence vectors for cut–set in Γ.

Since each row in $\mathbf{C}_r$ is orthogonal to each row in $\mathbf{D}_r$ we have

Tellegens Theorem. Let Γ be a directed graph with K labelled edges, let $\mathbf{i}$ be a current in Γ and let $\mathbf{v}$ be a voltage in Γ. Then

$$\mathbf{i} \cdot \mathbf{v} = 0 \ .$$

Furthermore, since $\dim V_i + \dim V_v = K$ the vector spaces V_i and V_v are orthogonal complements in $\mathbb{R}^K$, i.e. each of them consists of those vectors, which are orthogonal to all vectors in the other space:

Theorem 40.2. Let Γ be a directed graph.

The voltage space V_v has dimension $\rho(\Gamma)$. It is spanned by the rows of a reduced cutset matrix of Γ

The current space V_i has dimension $\mu(\Gamma)$. It is spanned by the rows of a reduced circuit matrix of Γ

V_i and V_v are orthogonal complements.

Let $\mathbf{v}$ be a voltage in Γ and let

$$\underrightarrow{\mathscr{D}}_1,\ \underrightarrow{\mathscr{D}}_2,\ldots,\ \underrightarrow{\mathscr{D}}_\rho$$

denote the rows in a reduced cutset matrix for Γ , that is, linearly independent incidence vectors for $\rho=\rho(\Gamma)$ cutsets in Γ . By theorem 40.1, there exists one and only one vector

$$\mathbf{v}^s = (v_1^s, v_2^s,\ldots,v_\rho^s)$$

such that

$$\mathbf{v} = v_1^s\ \underrightarrow{\mathscr{D}}_1 + v_2^s\ \underrightarrow{\mathscr{D}}_2 +\ldots+ v_\rho^s\ \underrightarrow{\mathscr{D}}_\rho\ .$$

The number v_j^s is called the *cutset voltage* of $\mathbf{v}$ across the cutset $\mathscr{D}_j$.

Let $\mathbf{i}$ be a current in Γ and let

$$\underrightarrow{\mathscr{C}}_1,\ \underrightarrow{\mathscr{C}}_2,\ldots,\ \underrightarrow{\mathscr{C}}_{\rho*}$$

denote the rows in a reduced circuit matrix for Γ , that is, linearly independent incidence vector for $\rho^*=\rho^*(\Gamma)$ circuits in Γ . By theorem 40.1 there exists on end only one vector

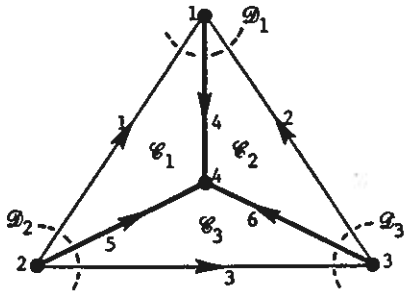
$$\mathbf{i}^m = (i_1^m, i_2^m,\ldots,v_\mu^m)$$

such that

$$\mathbf{i} = i_1^m\ \underrightarrow{\mathscr{C}}_1 + i_2^m\ \underrightarrow{\mathscr{C}}_2 +\ldots+ i_\mu^m\ \underrightarrow{\mathscr{C}}_\mu$$

The number i_j^m is called the *circuit current* of $\mathbf{i}$ across the circuit $\mathscr{C}_j$.

As an example, let us consider the graph in the figure, where $\mathscr{T} = \{4,5,6\}$.



$$\mathbf{D}_f = \begin{bmatrix} -1 & -1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & -1 & 0 & 0 & 1 \end{bmatrix} .$$

If the rows in $\mathbf{D}_r$ are denoted $\underset{\rightarrow}{\mathscr{D}}_1$, $\underset{\rightarrow}{\mathscr{D}}_2$ and $\underset{\rightarrow}{\mathscr{D}}_3$ we have that

$$\mathbf{v} = -\frac{1}{2}\,\underset{\rightarrow}{\mathscr{D}}_1 + \underset{\rightarrow}{\mathscr{D}}_2 + 2\,\underset{\rightarrow}{\mathscr{D}}_3$$

$$= (\tfrac{3}{2}, \tfrac{5}{2}, -1, -\tfrac{1}{2}, 1, 2)$$

is a voltage in Γ . The cutset voltages for $\mathbf{v}$ w.r.t. $\mathscr{D}_1$, $\mathscr{D}_2$ and $\mathscr{D}_3$ are $-\frac{1}{2}$, 1, 2.

$$\mathbf{C}_f = \begin{bmatrix} 1 & 0 & 0 & 1 & -1 & 0 \\ 0 & 1 & 0 & 1 & 0 & -1 \\ 0 & 0 & 1 & 0 & -1 & 1 \end{bmatrix} .$$

If the rows in $\mathbf{C}_f$ are denoted $\underset{\rightarrow}{\mathscr{C}}_1$, $\underset{\rightarrow}{\mathscr{C}}_2$ and $\underset{\rightarrow}{\mathscr{C}}_3$ we have that

$$\mathbf{i} = -2\,\underset{\rightarrow}{\mathscr{C}}_1 + 2\,\underset{\rightarrow}{\mathscr{C}}_2 + \underset{\rightarrow}{\mathscr{C}}_3$$

$$= (-2, 2, 1, 0, 1, -1)$$

is a current in Γ . The circuit currents for $\mathbf{i}$ w.r.t. $\mathscr{C}_1$, $\mathscr{C}_2$ and $\mathscr{C}_3$ are $-2, 2, 1$.

Observe that $\mathbf{v} \cdot \mathbf{i} = -3 + 5 - 1 + 1 - 2 = 0$, in accordance with Tellegen's theorem.

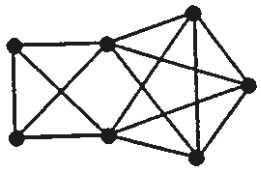
If $\mathbf{A}$ is the incidence matrix for a connected graph Γ , and if we delete the row in $\mathbf{A}$ which corresponds to a vertex p , then we obtain a matrix $\mathbf{A}_r$ with ρ linearly independent rows. These rows spans the same vector space as the rows in the cutset matrix, namely the voltage space of the graph. Therefore any voltage $\mathbf{v}$ in Γ can be written in the form

$$\mathbf{v} = v_1^p \mathbf{a}_1 + v_2^p \mathbf{a}_2 + \ldots + v_\rho^p \mathbf{a}_\rho \quad . \tag{1}$$

v_i^p is called the *node voltage* for the vertex corresponding to $\mathbf{a}_i$.

The vertex p , which is called the *reference vertex*, is given the node voltage 0 . Equation (1) expresses that the voltage across an edge is the difference between the node voltage of the start vertex and the node voltage of the end vertex. If Γ is not connected, we have to chose a reference vertex in each connected component. The example above is chosen such that $\underset{\rightarrow}{\mathscr{D}}_1$, $\underset{\rightarrow}{\mathscr{D}}_2$ and $\underset{\rightarrow}{\mathscr{D}}_3$ are three rows in the incidence matrix of the graph. Vertex 4 is reference vertex and $\mathbf{v}$ has the node voltages $-\frac{1}{2}$, 1 , 2 .

PROBLEMS TO CHAPTER 5.



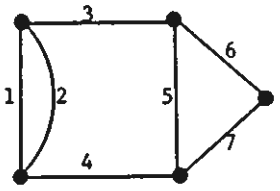
5.1. Find the number of spanning trees in the graph in the figure.

5.2. Find the number of trees in the graph K_m.

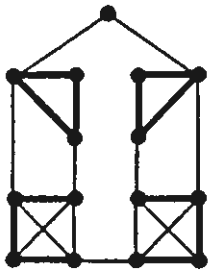
5.3. Find the number of trees in the graph $K_{4,3}$.

5.4. How many cutsets are there in K_5 and $K_{3,3}$.

5.5. *Feussners Method.* The trees in a connected graph Γ can be found by an algorithm, due to Feussner, [1]: Take any edge k in Γ, which is neither bridge or loop. Those trees in Γ, which do not contain k, are precisely the trees in $\Gamma - k$. And those trees in Γ, which contains k, are obtained from the trees in $\Gamma \div k$, by adding k. This process is repeated until we have obtained trees so small, that the trees can be found immediately. Determine, by Feussner's method, all the trees in the graph in the figure

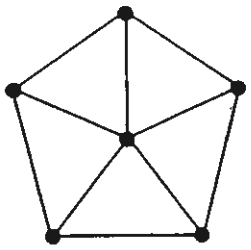


5.6. Let $\mathscr{A}$ be the set of thick edges in the graph Γ in the figure. Draw $\Gamma - \mathscr{A}$ and $\Gamma \div \mathscr{A}$.

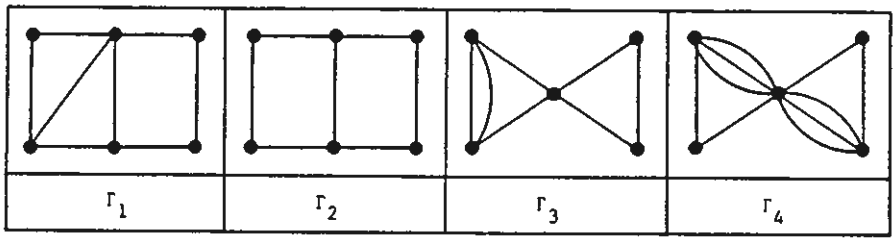


5.7. Find the number of circuits and cutsets in K_m .

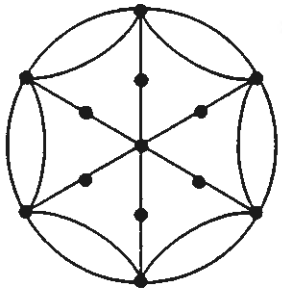
5.8. Find the number of circuits and cutsets in a wheel with P vertices. The figure shows a wheel with 6 vertices.

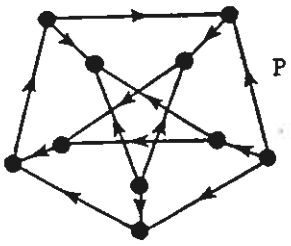


5.9. Answer for $i = 1, 2, 3, 4$ the following question. Does the graph Γ_i in the figure contain two circuits, which are not fundamental circuits w.r.t. the same tree? Does Γ_i contain two cutsets, which are not fundamental cutsets w.r.t. the same tree?



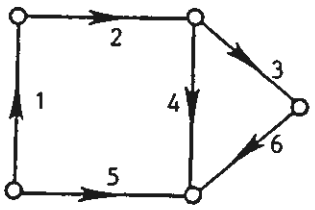
5.10. Let c denote the number of circuits and d the number of cutsets in the graph in the figure to the left. Is $c > d$ or $c = d$ or $c < d$?





5.11. Find the rank of the incidence matrix **A** , the circuit matrix **C** and the cutset matrix **D** for graph P in the figure to the right.

5.12. Find 3 different currents and 3 different voltages in the graph in the figure.



Check Tellegen's theorem. Find the dimensions of the current space, and the voltage space. Chose one of the currents $\mathbf{i}$ and one of the voltages $\mathbf{v}$. Does there exist a set of components, resistors and generators which can be inserted in the edges such that we obtain an electrical network with solution $\mathbf{v}, \mathbf{i}$? Can it happen, that all components can be resistors?

LITERATURE TO CHAPTER 5.

[1] Feussner, W.: Über Stromverzweigung in Netzförmigen Leitern. Ann. Phys. 9 (1902), 1304 – 1329.

[2] Gantmacher, F.R.: Matrizenrechung. Bd. 1. VEB Deutscher Verlag der Wissenschaften 1966.

[3] Seshu, S.; Reed, M. B.:Linear Graphs and Electrical Networks. Addison Wesley, 1961.

[4] Welsh, D. J. A.: Matroid Theory. Academic Press, 1976.

[5] Whitney, H: On the Abstract Properties of Linear Dependence. Amer. J. Math. (1935), 509 – 533.

CHAPTER 6

ELECTRICAL NETWORKS.

41. REGULAR AND SINGULAR NETWORKS OF RESISTORS. The concept of a graph is a very simple mathematical concept. Very much simpler than for example the fundamental geometrical concepts line and plane. And also very much simpler than, say, the real numbers. Nevertheless, graphs occur not until late in the history of mathematics. A systematic study of graphs was not initiated until late in the 19th century. It was inspired of problems from areas of pure mathematics, but also of problems in applied mathematics, in particular in organic chemistry and in the theory of electrical networks. The problems, which inspired to mathematical research before that had no natural relation to graph theory.

One of the first important contributions to graph theory was due to G. Kirchhoff (1824 – 1887), who, in a paper from 1847, emphasized, that there is a narrow connection between the properties of an electrical network and the structure of the graph of the network. In this chapter we shall describe some of these relations from a mathematical point of view. As we shall see, an electrical network can be considered as a mathematical object. The model we shall consider is pretty close to the real world networks.

A *network of resistors* is a directed graph N, called the *network graph,* the edge set of which is partitioned into three disjoint sets,

$$\mathcal{K}(N) = \mathcal{V} \cup \mathcal{R} \cup \mathcal{J};$$

Each element in $\mathcal{V}$ is called *voltage generator,* each element in $\mathcal{R}$ is called a *resistor*, and each element in $\mathcal{J}$ is called a *current generator.* For each $k \in \mathcal{V}$ there is given a real number V_k, called the *generator voltage* of edge k. For each $k \in \mathcal{R}$ there is given a number $R_k > 0$, called the *resistance* of k. And for each $k \in \mathcal{J}$ there is given a real number I_k, called the *generator current* of edge k.

– In the following we shall always assume, the edges of N are denoted by the numbers $1, 2, ..., K$, $K = |\mathscr{K}(N)|$. The numbers

$$\rho = \rho(N) = P - S \quad \text{and} \quad \mu = \mu(N) = K - P + S$$

are the *rank* and *corank* or *nullity* of N, where P is the number of vertices in N and S is the number of connected components in N. For $k \in \mathscr{R}$, the number $G_k = \frac{1}{R_k}$ is called the *conductance* of k.

In particular, a voltage generator with generator voltage 0 is called a *short circuit* and a current generator with generator current 0 is called an *open circuit.* In figures, we use the following symbols:

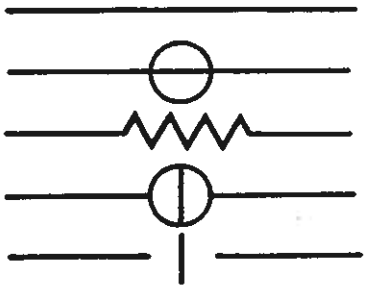
Short circuit:

Voltage generator:

Resistor:

Current generator:

Open circuit:



Now, let N be a network of resistors. We shall investigate whether there exist two vectors $\mathbf{v} = (v_1, v_2, ..., v_n) \in \mathbb{R}^n$, and $\mathbf{i} = (i_1, i_2, ..., i_n) \in \mathbb{R}^n$, $n = |\mathscr{K}(N)|$, which satisfies thefollowing three *network conditions:*

N 1. (Kirchhoff's voltage low.) For every circuit $\mathscr{C}$ in N we have that $\underrightarrow{\mathscr{C}} \cdot \mathbf{v} = 0$.

N 2. (Kirchhoff's current low.) For every cutset $\mathscr{D}$ in N we have that $\underrightarrow{\mathscr{D}} \cdot \mathbf{i} = 0$.

N 3. For $k \in \mathscr{V}$, $v_k = V_k$. For $k \in \mathscr{R}$, $v_k = R_k i_k$. For $k \in \mathscr{I}$, $i_k = I_k$.

Note, that N 1 expresses that $\mathbf{v}$ is a voltage in the network graph and that N 2 expresses that $\mathbf{i}$ is a current in the network graph. N 2 is equivalent with the equation $\mathbf{Ai} = 0$, where $\mathbf{A}$ is the incidence matrix of the network graph.

If the exists one and only one pair $(\mathbf{v}, \mathbf{i})$ of vectors, satisfying N 1, N 2 and N 3 we shall call N a *regular network,* $\mathbf{i}$ is called the *current vector* and $\mathbf{v}$ the *voltage vector* of N . v_k is called the *voltage across edge* k and i_k is called the *current in edge* k . If N is not regular it is called *singular.*

The three network equations constitutes a system of linear equations in the 2n unknown $v_1, v_2, \ldots, v_n ; i_1, i_2, \ldots, i_n$. These equations are not linearly independent. Let $\mathcal{T}$ be a tree in N . According to theorem 39.3 and the following proof N 1 is equivalent to the equation

$$\mathbf{C_f v} = \mathbf{0} , \tag{1}$$

where $\mathbf{C_f}$ is the fundamental circuit matrix of N w.r.t. $\mathcal{T}$. Similarly N 2 is equivalent with the equation

$$\mathbf{D_f i} = \mathbf{0} , \tag{2}$$

where $\mathbf{D_f}$ denotes the fundamental cutset matrix for N w.r.t. $\mathcal{T}$.

(1) contains $\mu(N)$ equations,
(2) contains $\rho(N)$ equations, and
(3) N 3 contains n equations.

Since $\mu(N) + \rho(N) + n = 2n$ we have formulated the network conditions as a square system, the *network equations.* The network is regular or singular if the determinant Δ of the system of equations is $\neq 0$, or $= 0$, respectively.

The following theorem shows, that regularity of a network is determined by the network graph alone. The values of the resistors play no rôle in this connection.

Theorem 41.1. Let N be a network of resistors, let $\mathcal{V}$ be the set of voltage generators in N and let $\mathcal{J}$ be the set of current generators in N . Then
N is regular $\Leftrightarrow$ $\mathcal{V}$ contains no circuit in N and $\mathcal{J}$ contains no cutset in N .

The proof of $\Rightarrow$ is rather easy. Assume that N contains a circuit $\mathscr{C}^v$ of voltage generators. If there exist a solution $(\mathbf{v}_0, \mathbf{i}_0)$ to the network conditions, then

$$(\mathbf{v}, \mathbf{i}) = (\mathbf{v}_0, \mathbf{i}_0 + t\underrightarrow{\mathscr{C}}^v)$$

is also a solution to the network equations: N 1 is satisfied since $\mathbf{v} = \mathbf{v}_0$. N 3 is satisfied because we have only changed the current in certain voltage generators. And, finally, N 2 is also satisfied, since for any cutset $\mathscr{D}$ in N

$$\underrightarrow{\mathscr{D}} \cdot \mathbf{i} = \mathscr{D} \cdot (\mathbf{i}_0 + t\underrightarrow{\mathscr{C}}^v) = \underrightarrow{\mathscr{D}} \cdot \mathbf{i}_0 + t\underrightarrow{\mathscr{D}} \cdot \underrightarrow{\mathscr{C}}^v = 0$$

Here we use that $\mathbf{i}_0$ satisfied N 2 and the Orthogonality theorem. So, if there exists just one solution, there are infinitely many, so N is not regular. By the dual arguments we can show, that if N contains a cutset of current generators, then N is not regular.

Proof of $\Leftarrow$: Assume that N contains no circuit of voltage generators and no cutset of current generators. We shall show that there is exactly one solution to the network equations. If this was not the case, then the determinant Δ would be 0. Putting all generator values equal to 0 would then give us a network with infinitely many solutions. Let $(\mathbf{i}, \mathbf{v}) \neq (\mathbf{0}, \mathbf{0})$ be a non–zero solution. According to Tellegens theorem

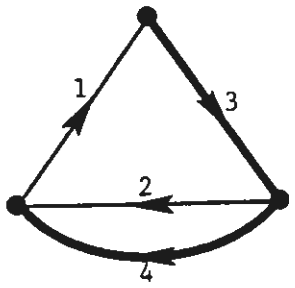
$$\mathbf{v} \cdot \mathbf{i} = v_1 i_1 + v_2 i_2 + \ldots + v_n i_n = 0 .$$

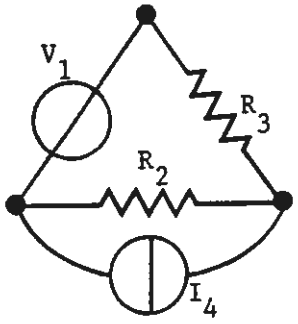
The terms in this sum which corresponds to a generator is 0, since the generator values are 0. Therefore

$$\sum_{k \in \mathscr{R}} v_k i_k = \sum_{k \in \mathscr{R}} R_k i_k^2 = 0 ,$$

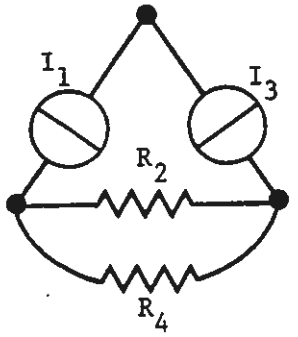
which together with $R_k > 0$ implies that $i_k = v_k = 0$ for every resistor k. If $\mathbf{i} \neq \mathbf{0}$ it is easy (as in section 24) to find a circuit $\mathscr{C}$ such that $i_k \neq 0$ for all $k \in \mathscr{C}$. Then $\mathscr{C}$ must consist of voltage generators contradicting the assumption. Dual arguments shows, that $\mathbf{v} \neq \mathbf{0}$ also contradicts the assumption. □

As illustration of the theorems we show first a regular, then two singular networks. In all three cases, the network graph and the tree $\mathcal{T}$ used when writing the equations is shown in the first figure.

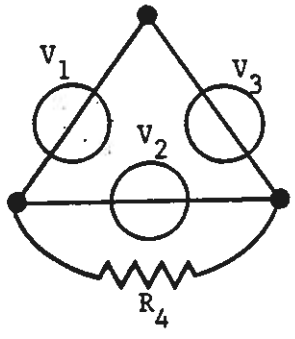




$$\begin{pmatrix} 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & -1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & R_2 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & 0 & R_3 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \\ i_1 \\ i_2 \\ i_3 \\ i_4 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ V_1 \\ 0 \\ 0 \\ I_4 \end{pmatrix} ;$$



$$\begin{pmatrix} 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & -1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & R_2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 & 0 & R_4 \end{pmatrix} \begin{pmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \\ i_1 \\ i_2 \\ i_3 \\ i_4 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ I_1 \\ 0 \\ I_3 \\ 0 \end{pmatrix} .$$

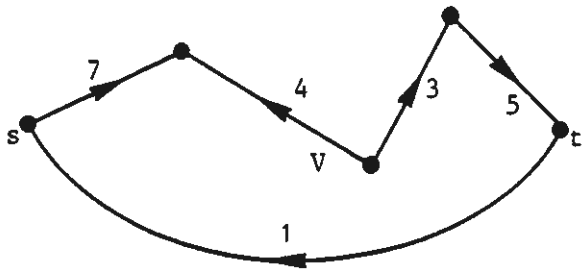


$$\begin{pmatrix} 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & -1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 & 0 & R_4 \end{pmatrix} \begin{pmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \\ i_1 \\ i_2 \\ i_3 \\ i_4 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ V_1 \\ V_2 \\ V_3 \\ 0 \end{pmatrix} .$$

Remark, that while it is easy to test regularity the means of theorem 41.1, it needs some considerations to obtain the result (regular or singular) directly from the matrices.

42. NETWORKS WITH ONE CURRENT GENERATOR. THE POWER THEOREM. In this section we shall give a new proof for theorem 41.1 in the special case where N consists of K-1 resistors and one current generator with generator current I . We many assume that $I \geq 0$, otherwise we just change the arrow. Also we assume, that the current generator is in edge 1 .

It is easy to see, that the network equations have at most one solution: If $(\mathbf{i}', \mathbf{v}')$ and $(\mathbf{i}'', \mathbf{v}'')$ are different solutions, then $\mathbf{i}'-\mathbf{i}'' \neq \mathbf{0}$ is a current in N with $i_1'-i_1'' = 0$. Again, as om section 24, by a possible change of some arrows, we can find a one–way cirant $\mathscr{C}$ of resistors in which all currents are > 0 . But then $\mathscr{C}$ does not satisfy N_1 . This contradiction shows the uniquess. We now proceed to show, that there exists a solution to the network equations. Let s denote the end vertex of edge 1 , and t the start vertex. Furthermore, let $\mathbf{i} = (I_1, i_2, i_3, \cdots, i_K)$ be any current in N , with $i_1 = I$; we shall call a current of this type a current of *value* I . We now chose a fixed s, t – path



V in $N\backslash\{1\}$. – Of course we assume, that edge 1 is not a bridge in N . Corresponding to the current i in N with value I we define a vector $\mathbf{v} = \mathbf{v}(\mathbf{i})$ by

$$(1) \qquad v_j = \begin{cases} R_j i_j & \text{for } j > 1 \ , \\ -\sum_{k \in V} \pm R_k \, i_k \ = \ -\sum_{k \in V} \pm v_k & \text{for } j = 1 \ ; \end{cases}$$

we use + if k is directed from s towards t on V , – otherwise. For example, in the figure above

$$v_1 = -(R_7 \, i_7 - R_4 \, i_4 + R_3 \, i_3 + R_5 \, i_5) = -(v_1 - v_4 + v_3 + v_5) \ .$$

The definition ensures that v satisfies N 1 for the circuit $V \cup \{1\}$. We shall show, that it is possible to chose $\mathbf{i} = \mathbf{i}_0$ with value I , such that $\mathbf{v}_0 = \mathbf{v}(\mathbf{i}_0)$ also satisfies N 1 for all other circuits. The point is, for a current i with value I , to consider the *power*

$$P(\mathbf{i}) = \sum_{k=2}^{K} R_k\, i_k^2 = \sum_{k=2}^{K} i_k\, v_k \ ,$$

and among all currents with value I to chose $\mathbf{i}_0$ as one for which the power

$$P(\mathbf{i}_0) = \sum_{k=2}^{K} R_k\, i_{0k}^2$$

is minimal. We shall show, that with this choice of $\mathbf{i}_0$, $\mathbf{v}_0 = \mathbf{v}(\mathbf{i}_0)$ will satisfy N 1 for every circuit $\mathscr{C}$ in N , $\underset{\rightharpoonup}{\mathscr{C}} \cdot \mathbf{v}_0 = 0$. We can assume, that $\mathscr{C}$ is a one way circuit directed as the edge arrows. This can be achieved by changing some edge arrows on $\mathscr{C}$ and change sign for the corresponding coordinates in $\mathbf{i}_0$ and $\mathbf{v}_0$. This leaves v_1 unchanged. We consider two cases:

Case 1. Edge 1 does not belong to $\mathscr{C}$. The proof is indirect. So assume that $\mathbf{v}_0 \cdot \underset{\rightharpoonup}{\mathscr{C}} \neq 0$, that is

$$\mathbf{v}_0 \cdot \underset{\rightharpoonup}{\mathscr{C}} = \sum_{k \in \mathscr{C}}{}' R_k\, i_{0k} \neq 0 \ . \tag{2}$$

Now, take any real number x and consider the current

$$\mathbf{i} = \mathbf{i}_0 + x\, \underset{\rightharpoonup}{\mathscr{C}} \ ,$$

which differs from $\mathbf{i}_0$ only in positions corresponding to the edges of $\mathscr{C}$. We nowconsider the power $P(\mathbf{i})$ as a function of x , $P(x) = P(\mathbf{i})$. The terms in $P(\mathbf{i})$ which depends on x corresponds to the edge of $\mathscr{C}$. Therefore

$$P'(x) = 2\sum_{k \in \mathscr{C}}{}' R_k(i_{0k} + x) = 2\sum_{k \in \mathscr{C}}{}' R_k\, i_{0k} + x\sum_{k \in \mathscr{C}}{}' R_k .$$

It follows that

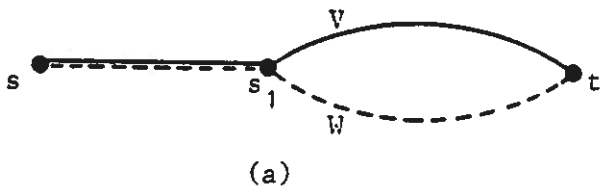
$$P'(0) = 2\sum_{k \in \mathscr{C}}{}' R_k\, i_{0k} \neq 0 \ .$$

But then the current $\mathbf{i}_0$ can not have minimal power: If namely $P'(0) > 0$ then $P(\mathbf{i}) < P(\mathbf{i}_0)$ when x is a numerical small, negative number and if $P'(0) < 0$, $P(\mathbf{i}) < P(\mathbf{i}_0)$ when x is a small positive number. This contradiction ends case 1.

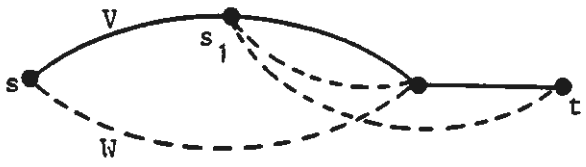
Case 2. $1 \in \mathscr{C}$. $\mathscr{C} \setminus \{1\}$ is a s, t - path W. We shall show that

$$\sum_{k \in V} \pm R_k i_k = \sum_{k \in W} \pm R_k i_k \quad , \tag{3}$$

where the signs are as in (1). Let us start in s and walk along V until we for the first time after s arrive to a vertex s_1 of W. Let V_{s_1} and W_{s_1} denote the s, s_1 – parts of V and W. Here V_{s_1} and W_{s_1} can be paths of length 1, as in the figure to the left, (a).



(a)



(b)

If this is not the case, (b), then V_{s_1} and W_{s_1} taken together is a circuit. Therefore

$$\sum_{k \in V_{s_1}} \pm R_k i_k = \sum_{k \in W_{s_1}} \pm R_k i_k \quad . \tag{4}$$

In case (a), the two sums are the same, in case (b) (4) follows from case 1. We now repeat the process, proceeding from s_1 along V to the first vertex s_2 belonging to the s_1, t - part of W. In this way we can continue, until $s_i = t$. (3) is then obtained by adding the equations of the form (4). From (3) follows, that since N1 is satisfied for the circuit $V \cup \{1\}$, N1 is also satisfied for the circuit $W \cup \{1\}$.

So we have proved the following theorem:

Theorem 12.1. If N consists of resistors and one generator with generator current I, which is not bridge in N, then the network equations have one and only one solution $(\mathbf{i}, \mathbf{v})$. Among all currents in N with value I, $\mathbf{i}$ is (the) one for which the power

$$P(\mathbf{i}) = \sum_{k \in \mathscr{R}}' R_k \mathbf{i}_k^2$$

is smallest.

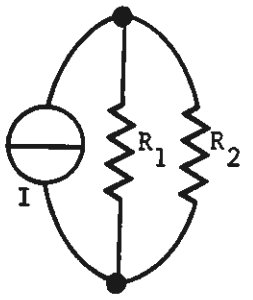
The last statement is called the power theorem.

The decisive step in the above construction was the determination of i_0 such that the power $P(i_0)$ is minimal. If Q is the largest number, which is $\leq P(i)$ for all currents i in N with value I, we have to show, that there exists an i_0 such that $P(i_0) = Q$. This can be seen by an argument similar to the argument page 3.13.

If for example we apply a current I to two resistors R_1 and R_2 in parallel, then the power will be

$$w = R_1 i_1^2 + R_2(I-i_1)^2 = (R_1+R_2)i_1^2 - 2I\, R_2 i_1 + I^2 R_2$$

$$= (R_1+R_2)\left[\left[i_1 - \frac{R_2}{R_1+R_2} I\right]^2 + I^2 \frac{R_2}{(R_1+R_2)^2}\right].$$



where i_1 is the current in R_1. This power is minimal for $i_1 = \frac{R_2}{R_1+R_2} I$. And this is exactly the current through R_1, as the reader may know in advance.

For a moment, let us consider the condition

N1(a). Among all currents with value I, i is one with minimal power.

Form the proceeding results it follows that for a network of resistors with one current generator N1, N2 and N3 have the same solutions as N1(a), N2 and N3. It is interesting,[4], that Kirchhoff's voltage low, which consists of $K - P + S$ linear equations, in this way can be replaced by a minimization of a polynomial of degree 2 in $K - 1$ variables $i_2, i_3, \ldots, i_K$, where the minimum must be found over all currents with value I.

43. KIRCHHOFF'S RULE. In theorem 41.1 we gave a graph theoretic condition for unique solvability of the network equations. And in the preceeding sections we have mentioned two solution methods: The algebraic method and the power minimizing method. The first method is widely used, the other one is difficult. In this section we shall describe a graph theoretic solution method which is due to G. Kirchhoff, [1], 1847. From a computational point of which the method is not good. Its strength is the insight it provides in the connection between the structure of the graph, and the solution.

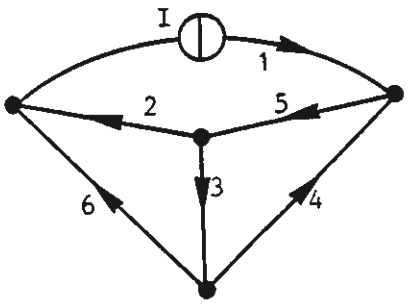
First we consider a regular network N consisting of resistors ($\mathcal{R}$), voltage generators ($\mathcal{V}$) and current generators ($\mathcal{J}$). If $\mathcal{T}$ is a tree in N, we put

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

$\pi(\mathcal{T})$, which is called the cotree product of $\mathcal{T}$, is the product of the resistances not in $\mathcal{T}$. And then we define *the network determinant* $\Delta(N)$ by

$$\Delta N = \sum_{\mathcal{T}} \pi(\mathcal{T}) \ , \tag{1}$$

where the summation is over all those trees in N , each of which contains all voltage generators and no current generator. So the summation is really over all trees $\mathcal{T}$ in $N \div \mathcal{V} - \mathcal{J}$. The number of terms in $\Delta(N)$ grows exponentially with the number of edges in N , so $\Delta(N)$ is not very attractive from a computationally point of view. If, for example, we consider the network



we obtain the following determinant:

$$\Delta(N) = R_2R_3 + R_3R_6 + R_3R_4 + R_3R_5 + R_2R_4 + R_5R_6 + R_2R_5 + R_4R_6 \ . \tag{2}$$

We now turn to the special case, where N consists of one generator G in edge 1 and resistors $R_2, R_3, \cdots, R_K$. We shall derive a formula, (15), which determine the current i_k in R_K . This is of course a complicated matter. Let $k \neq 1$ be an edge in N . A tree $\mathcal{T}$ in N - G is called a k-tree in N if the fundamental circuit $\mathcal{C}(G, \mathcal{T})$ contains k . In particular, then $k \in \mathcal{T}$. A k-tree $\mathcal{T}$ has a *cotreeproduct* $\pi_k(\mathcal{T})$ defined by the equation

(3) $$\pi_k(\mathcal{T}) = \pm \prod_{\substack{k \notin \mathcal{T} \\ k \neq 1}} R_k \ ,$$

where $+$ is used when G and k determines the same direction on $\mathcal{C}(G, \mathcal{T})$, and otherwise $-$. In the first case, $\mathcal{T}$ is called *positive,* in the second *negative.* For example, in the figure above there are 4 trees in N - I which contains $k = 3$. Of these, two are 3–trees, namely

$$\mathcal{T}_1 = \{3, 5, 6\} \quad \text{and} \quad \mathcal{T}_2 = \{2, 3, 4\} \ .$$

The corresponding cotree products are

$$\pi_3(\mathcal{T}_1) = R_2R_4 \quad \text{and} \quad \pi_3(\mathcal{T}_2) = R_5R_6 \ .$$

By means of these concepts we shall now define first a specific current $\mathbf{i}$ in N, and, later, a voltage $\mathbf{v}$. Finally we shall show, that $(\mathbf{i}, \mathbf{v})$ is a solution to the network equations. First we consider the case where G is a current generator, G = I, in edge 1.

Construction of $\mathbf{i}$. For each tree $\mathcal{T}$ in N - I we send a current of size

$$i_{\mathcal{T}} = \frac{\pi(\mathcal{T})}{\Delta(N)} I$$

through the fundamental circuit $\mathcal{C}(I, \mathcal{T})$, in the direction of I. The corresponding current in N is

(4) $$\mathbf{i}_{\mathcal{T}} = \frac{\pi(\mathcal{T})}{\Delta(N)} I \, \underline{\mathcal{C}}(I, \mathcal{T}) \ ,$$

where the orientation of $\mathcal{C}(I, \mathcal{T})$ is determined by I.

If, for example, we consider the tree $\{4, 5, 6\}$ in the example above, it follows from (4) that

$$\mathbf{i}_{\mathcal{T}} = \frac{R_2R_3}{\Delta(N)} I \, (1, 0, 0, -1, 0, 1) \ ,$$

where ΔN is determined by (2).

So, for each tree $\mathcal{T}$ in N - I we consider the current $\mathbf{i}_{\mathcal{T}}$ determined by (4). The current $\mathbf{i}$ is defined as the sum of all these currents,

(6) $$\mathbf{i} = \sum_{\mathcal{T}}' \mathbf{i}_{\mathcal{T}} \ ,$$

where the summation is over all trees in $N - I$.

Let us determine the current i_3 in the example above. First we have to find the expressions of the form (4) (or (5)), for which the 3^{rd} coordinate in the incidence vector $\underline{\mathcal{C}}(I, \mathcal{T})$ is $\neq 0$. There are two trees of this type, namely the 3–trees $\mathcal{T}_1 = \{3, 5, 6\}$ and $\mathcal{T}_2 = \{2, 3, 4\}$. Since the corresponding 3^{rd} coordinates in $\underline{\mathcal{C}}$ is $+1$ respectively -1 , we get

(7) $$i_3 = \frac{R_2R_4 - R_3R_5}{\Delta(N)} I \ ,$$

where $\Delta(N)$ is determined in (2).

Since $\mathbf{i}$ (in (6)) is linear combination of incidence vectors for circuits, it follows from theorem 40.1 that $\mathbf{i}$ is a current in N . We shall find an explicit expression for the current i_k . If k is a resistor, we obtain in (4) a contribution $(\neq 0)$ to i_k from $\mathcal{T}$ precisely when $\mathcal{C}(I, \mathcal{T})$ contains k , that is, if $\mathcal{T}$ is a k–tree. And the contribution from a k–tree $\mathcal{T}$ is exactly $\frac{\pi_k(\mathcal{T})}{\Delta(N)} I$, where the product πR_k itself has its origin in the first factor of (4), while the sign is determined by the k^{te} coordinate in $\underline{\mathcal{C}}(I, \mathcal{T})$. Therefore

(8) $$i_k = \frac{\Sigma \pi_k(\mathcal{T})}{\Delta(N)} I \ , \quad k > 1 \ ,$$

where the summation in the numerator is over all k–trees in N . For i_1 we obtain from (4) the contribution

$$\frac{\pi(\mathcal{T})}{\Delta(N)} I$$

from an arbitrary tree $\mathcal{T}$ in $N - I$, and it follows that

$$\mathbf{i} = \frac{\Sigma \pi(\mathcal{T})}{\Delta(N)} I = \frac{\Delta(N)}{\Delta(N)} I = I \ ,$$

of course a goal we wanted to achieve.

Construction of $\mathbf{v}$. For each tree $\mathcal{T}$ in N, which contains I, we put a voltage of the size

$$v_{\mathcal{T}} = \frac{\pi(\mathcal{T})}{\Delta(N)} I$$

across the edges in the fundamental cutset $\mathcal{D}(I, \mathcal{T})$, in the direction of I. The corresponding voltage in N is

$$v_{\mathcal{T}} = \frac{\pi(\mathcal{T})}{\Delta(N)} I \underline{\mathcal{D}}(I, \mathcal{T}) \quad , \tag{9}$$

where I determines the direction of the fundamental cutset.

For example, if we consider the tree $\mathcal{T} = \{1, 5, 6\}$ in the example above, we get from (9) that

$$v_{\mathcal{T}} = \frac{R_2 R_3 R_4}{\Delta(N)} I \quad (1, -1, -1, 1, 0, 0) \quad . \tag{10}$$

The voltage $\mathbf{v}$ is now defined as the sum of the voltages determined by (9),

$$\mathbf{v} = -\sum_{\mathcal{T}}' v_{\mathcal{T}} \quad , \tag{11}$$

where the summation is over all these trees in N, which contain I.

Let us, in the example above, determine the 3rd coordinate in $\mathbf{v}$. To do this we have to find the expressions of the form (9) (or (10)) for which the 3rd coordinate in $\underline{\mathcal{D}}(I, \mathcal{T})$ is $\neq 0$. There are two trees of this type, namely $\mathcal{T}_1 = \{1, 5, 6\}$ and $\mathcal{T}_2 = \{1, 2, 4\}$. The corresponding 3rd coordinates in $\underline{\mathcal{D}}$ is -1, respectively $+1$. Therefore

$$v_3 = -\frac{-R_2R_3R_4 + R_3R_5R_6}{\Delta(N)} I \quad . \tag{12}$$

Comparing with (7), we observe that $v_3 = R_3 i_3$, again a goal we wanted to achieve.

The vector $\mathbf{v}$ is always a linear combination of incidence vectors of cutsets. According to theorem 40.1, therefore, $\mathbf{v}$ is a voltage in N. We shall find explicit expressions for each coordinate in $\mathbf{v}$. If k is a resistor, then (9) contributes to v_k if $\mathcal{T}$ has the property that $\mathcal{D}(I, \mathcal{T})$ contains k, that is, according to theorem 39.1, if $\mathcal{T}_1 = (\mathcal{T} \setminus I) \cup \{k\}$ is a k–tree. And in this case the contribution is

$$v_{\mathcal{T}} = -R_k \cdot \frac{\pi_k(\mathcal{T}_1)}{\Delta(N)} \quad , \tag{13}$$

where the factor R_k appears because $k \notin \mathcal{T}$, but $k \in \mathcal{T}_1$. $\pi_k(\mathcal{T})$ has its origin in the first factor in (9), and the – sign in (13) comes from theorem 39.1. Therefore, from (11) it follows that

$$v_k = \left[-R_k \sum_{\mathcal{T}}' \frac{\pi_k(\mathcal{T})}{\Delta(N)} \right] I \quad , \quad k > 1 \quad , \tag{14}$$

where the summation is over all k–trees $\mathcal{T}$ in N. By comparing with (8) we observe that $v_k = R_k\, i_k$. So, (i, v) satisfies all the tree network conditions N1, N2 and N3. We have therefore proved the current generator part of

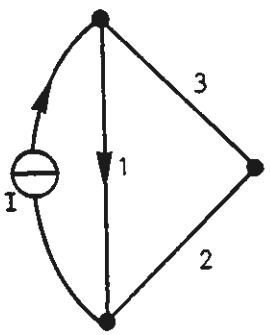
Kirchhoff's Rule. If a network N consists of resistors and one generator with value G, then the current in edge $k(\neq g)$ is

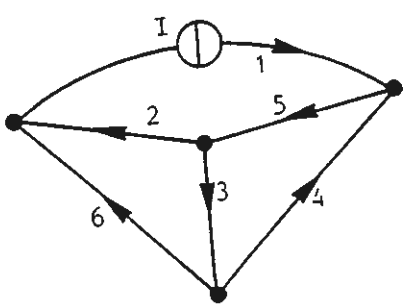
$$i_k = \frac{\pm G}{\Delta(N)} \sum_{\mathcal{T}}' \pi_k(\mathcal{T}) \quad , \tag{15}$$

where the summation is over all k-trees $\mathcal{T}$ in N. + is used when G is a current generator – when G is a voltage generator.

The proof in the case where G is a voltage generator follows page 6.17–6.18. Remember, that $\Delta(N)$ changes value when G is changed from one type of generator to the other.

Example 1. Find the current i_1 in R_1 in the network in the figure to the left. N – I contains 3 trees, {2, 3}, {1, 3}, {1, 2}; so $\Delta(N) = R_1 + R_2 + R_3$. There are two 1-trees, {1, 3} and {1, 2}. Both gives the sign + in (2), and hence $i_1 = \frac{R_2 + R_3}{R_1 + R_2 + R_3} I$.





Example 2. Find the current i_3 in the network in the figure to the right. $\Delta(N)$ is calculated in (2) p(6.10). There are two 3-trees, $\mathcal{T}_1 = \{3, 5, 6\}$ and $\mathcal{T}_2 = \{2, 3, 4\}$. The corresponding resistance products is as mentioned p(6.11, 3rd formula)

$$\pi_3(\mathcal{T}_1) = R_2 R_4 \ , \ \pi_3(\mathcal{T}_2) = - R_5 R_6 \ .$$

Therefore

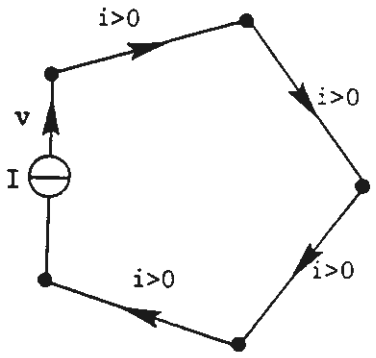
$$i_3 = \frac{R_2R_4 - R_5R_6}{\Delta N} I$$

In the proof of Kirchhoff's rule it was not necessary to find the voltage across the current generator. We now proceeds to the determination of this voltage v_1.

Since the first coordinate in $\underline{\mathcal{D}}(I, \mathcal{T})$ is 1, we obtain from (9) and (11) that

(16) $$v_1 = - \frac{I}{\Delta(N)} \sum_{\mathcal{T}}' \pi(\mathcal{T}) \ ,$$

where the summation is over all trees containing I. (If the reader finds the sign in



(16) surprising, consider a circuit $\mathcal{C}$ in N, as shown in the figure. For this circuit, Kirchhoff's voltage law can only be satisfied if $v < 0$.) Observe, that N reacts on the current generator I as a resistor of size

$$R = \frac{\Sigma \pi(\mathcal{T})}{\Delta(N)} \ .$$

In other words:

Theorem 43.1. Let p and q be two vertices in a network N of resistors and let N_1 be the network obtained from N by connecting a generator G between p and q . The G looks into a resistor of size

$$R = \frac{\Sigma\pi(\mathcal{T}_1)}{\Sigma\pi(\mathcal{T})} \quad ,$$

where the summation in the numerator is over all trees $\mathcal{T}_1$ in N_1 which contains G , while in the denominator the summation is over all trees $\mathcal{T}$ in N .

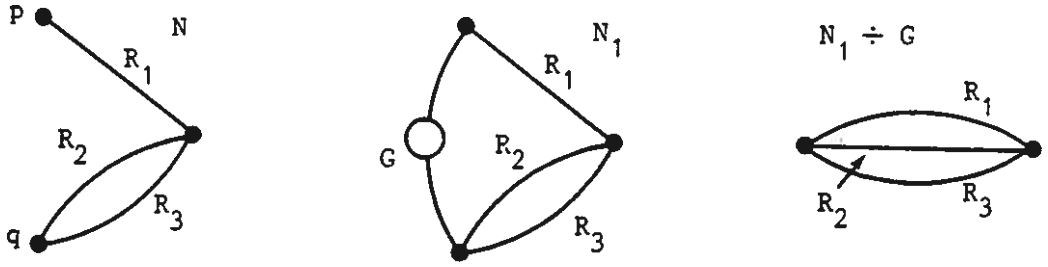
Note, that (16) can also be written

(17) $$I = -\frac{1}{R} V_1 \quad .$$

Therefore, in theorem 43.1 it does not matter whether G is a current generator or a voltage generator. R is called the *driving point resistance* for N between p and q . Since the numerator in the formula in the theorem is $\Delta(N_1 \div G)$, the driving point resistance can be written

(17) $$R = \frac{\Delta(N_1 \div G)}{\Delta(N)} \quad .$$

As an example, let us find the driving point resistance between p and q in the network in the figure below.

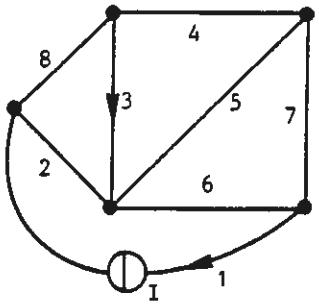


The result is

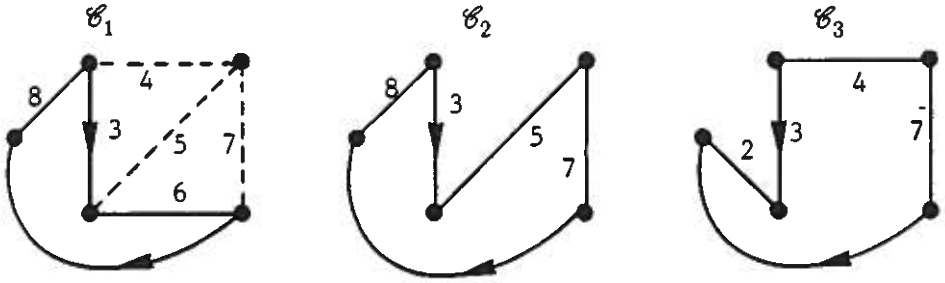
$$R = \frac{R_1R_2+R_1R_3+R_2R_3}{R_2+R_3} = R_1 + \frac{R_2R_3}{R_2+R_3} \, ,$$

the same result which could be obtained by means of the formules for resistors in series and parallel.

By using formula (15), in general it is simplest first to find the fundamental circuits, and after that "grow trees on the circuits, to obtain the correct k–tree products." We illustrate this method by an example:



We want to find the current i_3 in the network in the figure. There are 3 circuits each of which contain both I and 3 . C_1 and C_2 gives positive 3–trees C_3 a negative.



In fact, there are 3 positive 3–trees $\mathscr{T}$ for which $\mathscr{C}_1 = \mathscr{C}(I, \mathscr{T})$, namely $\{8, 3, 6, 4\}$, $\{8, 3, 6, 5\}$ and $\{8, 3, 6, 7\}$. $\mathscr{C}_2 - I = \{8, 3, 5, 7\}$ is a positive 3–tree, while $\mathscr{C}_3 - I = \{2, 3, 4, 7\}$ is a negative 3–tree. Therefore

$$i_3 = \frac{1}{\Delta(N)} (R_2R_5R_7 + R_2R_4R_7 + R_2R_4R_5 + R_2R_4R_6 - R_5R_6R_8) \ .$$

Here $\Delta(N)$ can be determined by (1), but since there will be 21 terms, we shall not write them down here.

We shall now prove Kirchhoff's rule in the case, where G is a voltage generator of size V . Let I be the current in the voltage generator. Then by (15)

$$i_k = \frac{I}{\Delta(N-V)} \sum \pi_k(\mathscr{T}) \ .$$

From (16) we obtain $I = -\frac{1}{R} V$ and from (18)

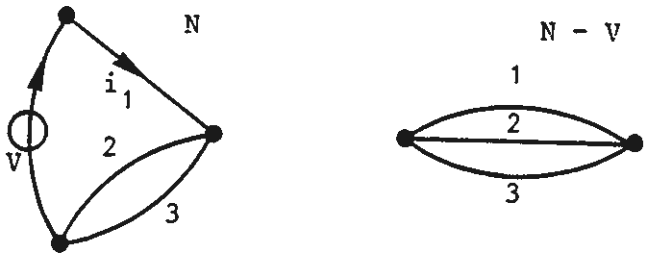
$$R = \frac{\Delta(N)}{\Delta(N-V)} .$$

Inserting these expressions, we get

$$i_k = -\frac{1}{\frac{\Delta(N)}{\Delta(N-V)}} \cdot V \; \frac{I}{\Delta(N-V)} \sum \pi_k(\mathcal{T}) = -\frac{1}{\Delta(N)} \cdot \sum \pi_k(\mathcal{T}) .$$

This proves Kirchhoff's rule in the case where the network is driven by a voltage generator. □

As an example, consider the network in the figure



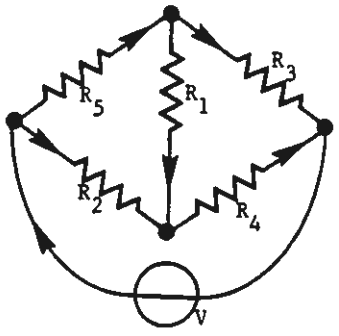
The network determinant is determined by the trees in $N \div V$. The result is

$$\Delta(N) = R_1R_2 + R_1R_3 + R_2R_3 .$$

The two 1–trees $\{1, 2\}$ and $\{1, 3\}$ are both positive. The 1–tree products are R_3 and R_2. Therefore

$$i_1 = -\frac{R_3+R_2}{R_1R_2+R_1R_3+R_2R_3} V .$$

Below we give one further example of the application of Kirchhoff's rule



$$i_1 = \frac{-V\,(+R_2R_3 \qquad -R_4R_5\ )}{R_1R_2R_3 + R_1R_2R_5 + R_1R_3R_4 + R_1R_4R_5 + R_2R_3R_4 + R_2R_3R_5 + R_2R_4R_5 + R_3R_4R_5}$$

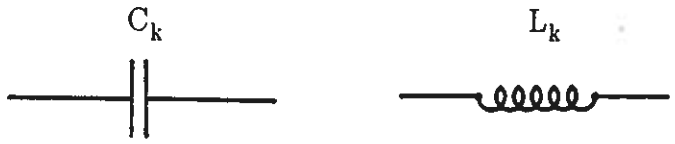
Observe, that $i_1 = 0$ when $R_2 R_3 = R_1 R_4$. This is used in Wheatstones bridge.

44. RCL – NETWORKS. The solution of a regular network N of resistors can be determined by means of a system of linear equations. If N also contains inductors and capacitors, instead we have to use a system of 1st order differential equations. Nevertheless, the solutions can still be found, using graph theoretic methods.

We define a RLC–network N as a directed graph, called the *network graph,* in which the edge set $\{1, 2, \cdots, K\}$ is partitioned into 5 disjoint subsets,

$$\mathscr{K}(N) = \mathscr{V} \cup \mathscr{C} \cup \mathscr{R} \cup \mathscr{L} \cup \mathscr{I} .$$

In the order given here, we call the edges *voltage generators, capacitors, resistors, inductors* and *current generators.* For each $k \in \mathscr{V}$ there is given a real or complex function $V_k = V_k(t)$, which is called the *generator voltage* of k. For each $k \in \mathscr{C}$ there is given a real number $C_k > 0$, called the *capacity* of k. For each $k \in \mathscr{R}$ there is given a real number $R_k > 0$ called the *resistance* of k. For each $k \in \mathscr{L}$ there is given a real number $L_k > 0$ which is called the *inductance* of k. And, finally, for each current generator k there is given a real or complex function $I_k = I_k(t)$, which is called the *generator current* of k. In figures, capacitors and inductors are shown in this way:



Let N be a RLC–network. We shall investigate whether there exist two vector functions

$$\mathbf{v} = \mathbf{v}(t) = (v_1(t), v_2(t), \cdots, v_K(t)) , \quad t \in \mathbb{R}$$

and

$$\mathbf{i} = \mathbf{i}(t) = (i_1(t), i_2(t), \cdots, i_K(t)) , \quad t \in \mathbb{R} ,$$

which satisfies the following three network conditions:

N 1. (Kirchhoff's voltage law.) For every circuit $\mathscr{C}$ in N, $\underline{\mathscr{C}} \cdot \mathbf{v}(t) = 0$ for all $t \in \mathbb{R}$.

N 2. (Kirchhoff's current law.) For every cutset $\mathscr{D}$ in N, $\underline{\mathscr{D}} \cdot \mathbf{i}(t) = 0$ for all $t \in \mathbb{R}$.

N 3. (The element relations):
For $k \in \mathscr{V}$, $v_k(t) = V_k(t)$.
For $k \in \mathscr{C}$, $i_k(t) = C_k\, v_k'(t)$.
For $k \in \mathscr{R}$, $v_k(t) = R_k\, i_k(t)$.
For $k \in \mathscr{L}$, $v_k(t) = L_k\, i_k'(t)$.
For $k \in \mathscr{J}$, $i_k(t) = I_k(t)$.

The network conditions is equivalent to a system of $2K$ equations in the $2K$ unknown coordinates in $\mathbf{i}$ and $\mathbf{v}$. N1 and N2 give K algebraic equations, while N3 contributes with $|\mathscr{V} \cup \mathscr{R} \cup \mathscr{J}|$ algebraic equations and $|\mathscr{C} \cup \mathscr{L}|$ differential equations of order 1. So we have a square system of differential equations, and we must expect infinitely many solutions to the network equations.

There is a clever trick, with a lot of content, which (– to some extend –) reduces RCL–networks to the pure resistive case:

Chose any real or complex number s and assume that all voltages for elements in $\mathscr{V}$ and all currents for elements of $\mathscr{J}$ have the form $c \cdot e^{st}$, $t \in \mathbb{R}$, where we chose a real or complex constant c for each generator. We want to investigate whether the network equations has a solution of the form

$$\text{(1)} \qquad \mathbf{i} = (i_1, i_2, \cdots, i_K)e^{st}\ ,\ \mathbf{v} = (v_1, v_2, \cdots, v_K)e^{st}\ ,$$

where the coefficient vectors are independent of t. The point is now, that if (1) is a solution, then for

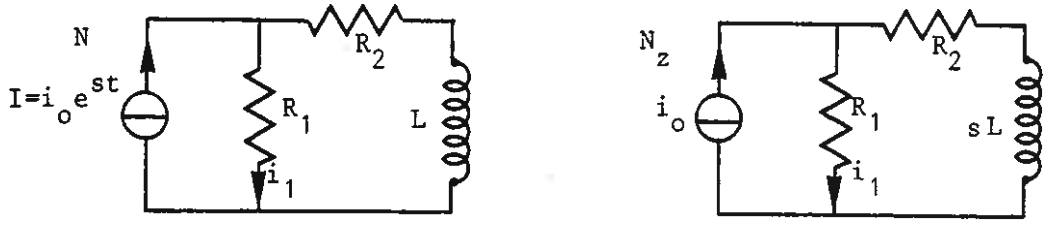
$k \in \mathscr{C}$ we have that $i_k\, e^{st} = C_k \frac{d}{dt}(v_k \cdot e^{st}) = s\, C_k \cdot v_k\, e^{st}$,
$k \in \mathscr{R}$ we have that $v_k \cdot e^{st} = R_k \cdot i_k\, e^{st}$,
$k \in \mathscr{L}$ we have that $v_k\, e^{st} = L_k \frac{d}{dt}(i_k \cdot e^{st}) = s\, L_k \cdot i_k\, e^{st}$,
which (for $s \neq 0$) is equivalent to

$k \in \mathscr{C}$: $\quad v_k = \frac{1}{sC_k}\, i_k$
$k \in \mathscr{R}$: $\quad v_k = R_k\, i_k$
$k \in \mathscr{L}$; $\quad v_k = sL\, i_k$.

Hence (1) is a solution to N if and only if

$$\mathbf{i} = (i_1, i_2, \cdots, i_K) \quad , \quad \mathbf{v} = (v_0, v_1, \cdots, v_K)$$

is a solution to the network N_z of resistors, which is obtained from N by replacing each capacitor C_k with a resistor of size $R_R = \frac{1}{sC_k}$, each inductor L_k with a resistor of size $R_R = sL_k$ and the generator values with the corresponding c–values. Therefore, we can find the solution of the form (1) to N by finding $(i_1, i_2, \cdots, i_K)$ and $(v_1, v_2, \cdots, v_K)$ as the solution to N_z. The numbers $\frac{1}{sC_k}$ and sL_k are called *impedances,* while sC_k is called the *admittance* of capacitor C_k. When N_z is regular, that is, if there exist no circuit of voltage generators and no cutset of current generators, and when s is chosen such that $\Delta(N_z) \neq 0$, then the network equations have one and only one solution of the form (1). This solution is called the *stationary solution* to the input e^{st}, assuming that the c–values are known. The coordinates $(i_k, v_k)e^{st}$ is called the stationary solution for the current and voltage in edge k. The word stationary is chosen to fit the important special case where $s = i\omega$ is purely imaginary and where, therefore, input as well as solution have period $T = \frac{2\pi}{\omega}$. We illustrate the situation with an example:



In the network to the left, $I = i_0 \cdot e^{st}$. Find the stationary solution for the current i_1 through the resistor R_1. If we replace L by an impedance of size sL we obtain the network N_z to the right, which is similar to example 1 p. 6.14. Therefore, in N_z ,

$$i_1 = \frac{R_2+sL}{R_1+R_2+sL}\, i_0$$

and the stationary solution for i_1 in N is

$$i_1(t) = \frac{R_2+sL}{R_1+R_2+sL}\, i_0 \cdot e^{st} \quad .$$

If we put $R_1 = R_2 = 1\Omega$, $L = 1H$, $s = 2s^{-1}$, $i_0 = 1A$, we find, that $i_1(t) = \frac{3}{4} e^{st}$ A .

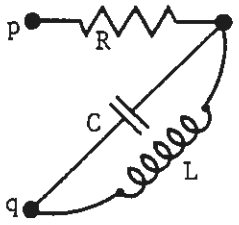
The rational function

$$H(s) = \frac{R_2 + sL}{R_1 + R_2 + sL}$$

is called the *transfer function* for the current i_1 in N .

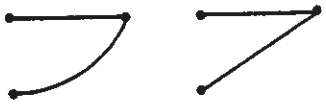
Also, the concept driving point impedance, can be generalized to RCL–networks. Let p and q be two vertices in a RCL–network without generators. Place a generator G with value $c\, e^{st}$ between p and q . In the stationary solution, a capacitor C and inductor L will then behave like resistors of size $\frac{1}{sC}$ and sL , respectively. Therefore G looks into an impedance, the size of which can be determined by means of theorem 43.1.

For example, if we want to determine the driving point impedance Z between p and



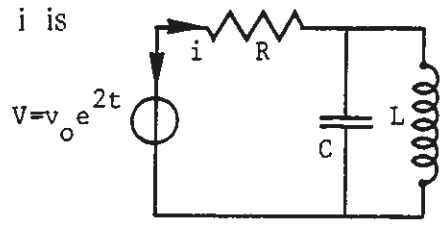
q in the network in the figure, we place a generator of size $c\, e^{st}$ between p and q . Then from theorem 43.1, p. 248, it follows that

$$(2) \qquad z = \frac{\frac{sL}{sC} + \frac{R}{sC} + RsL}{\frac{1}{sC} + sL} = \frac{\frac{sL}{sC} + \frac{R}{sC} + RsL}{\frac{1}{sC} + sL} .$$



If, in particular we assume that G is a voltage generator of size $v_0 \cdot e^{2t}$ V and $R = 1\Omega$, $C = 1F$ and $L = 1H$, then the current i is

$$i = \frac{1}{z} V = \frac{5}{7} v_0\, e^{2t} \text{ A} .$$



Note, that if $s = \pm i \sqrt{\frac{1}{LC}}$, then there is no stationary solution.

If we use Kirchhoff's rule (15) p. 6.24, or (18), theorem 43.1, then the capacitors produce factors of type $\frac{1}{sC}$ in terms in numerator and denominator. It is not only a question of computational technique, it also enlightens the understanding, to multiply in numerator and denominator by

(3) $$\prod_{n \in \mathscr{C}} sC_n$$

to avoid these fractions. In this way, in Kirchhoff's rule $\Delta(N)$ will be replaced by the polynomial

(4) $$P(s) = \sum_{\mathscr{T}} \prod_{\substack{n \in \mathscr{T} \\ n \in \mathscr{C}}} sC_n \prod_{\substack{n \notin \mathscr{T} \\ k \in \mathscr{R}}} R_n \prod_{\substack{n \notin \mathscr{T} \\ n \in \mathscr{L}}} L_n \ ,$$

where the summation is over all trees in $N - \mathscr{J} \div \mathscr{V}$. In the next section we shall consider this polynomial in details.

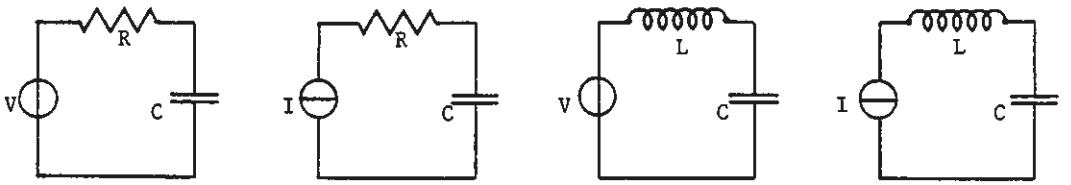
45. THE CHARACTERISTIC POLYNOMIAL . For an arbitrary RCL-network N we call the polynomial

(1) $$P(s) = \sum_{\mathscr{T}} \prod_{\substack{n \in \mathscr{T} \\ n \in \mathscr{C}}} sC_n \prod_{\substack{n \notin \mathscr{T} \\ k \in \mathscr{R}}} R_n \prod_{\substack{n \notin \mathscr{T} \\ n \in \mathscr{L}}} sL_n \ ,$$

the *characteristic polynomial* of N . In (1), the summation is over all trees $\mathscr{T}$ in $N - \mathscr{J} \div \mathscr{V}$, which is the same as those trees in N which contain all voltage generators and no current generator. $P(s)$ is a very important concept in the theory of electrical networks, and it is a point in "network topology" that (1) expresses a very "visual" connection between $P(s)$ and the network structure.

Below, we show four networks and their characteristic polynomials.

N:



N–$\mathcal{J}\div\mathcal{V}$:

$P(s) = RCs + 1$ $\quad$ $P(s) = sC$ $\quad$ $P(s) = LCs^2 + 1$ $\quad$ $P(s) = sC$

We shall describe a method to determine the degree n of P(s) . If there exists a tree which contain all capacitors and no inductors, then we obtain the maximal degree, $n = |\mathcal{C}| + |\mathcal{L}|$. Such a tree exist if and only if $\mathcal{C}$ is circuit free and $\mathcal{L}$ is cutset-free. In general, the degree of P(s) is the largest number n for which the exist a tree $\mathcal{T}$ in $N - \mathcal{J}\div\mathcal{V}$ for which the number of inductors cutside $\mathcal{T}$ plus the number of capacitors in $\mathcal{T}$ is n ,

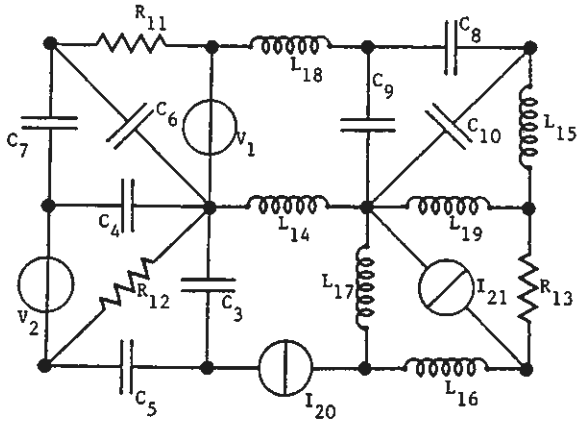
$$n = \max_{\mathcal{T}} \left\{ |\mathcal{C}_1| + |\mathcal{L}_1| \;\middle|\; \mathcal{C}_1 \subseteq \mathcal{C},\ \mathcal{C}_1 \subseteq \mathcal{T},\ \mathcal{L}_1 \subseteq \mathcal{L},\ \mathcal{L}_1 \cap \mathcal{T} = \emptyset \right\} .$$

A tree $\mathcal{T}$ for which this maximum is obtained is called a *normal tree.* A normal tree can be found by means of Kruskal's algorithm in following way: First form $N - \mathcal{J}\div\mathcal{V}$, then give each kapactor weight 1, each resistor weight 0 and each inductor weight –1 . The weight $W(\mathcal{T})$ of a tree $\mathcal{T}$ is

$$w(\mathcal{T}) = |\mathcal{C}_T| - |\mathcal{L}_T| = |\mathcal{C}_T| + |\mathcal{L}_T| - |\mathcal{L}| \ , \tag{2}$$

where $\mathcal{C}_T$ is a number of capacitors in T , $\mathcal{L}_T$ the number of inductors cutside $\mathcal{T}$ and $\mathcal{L}_T$ the number of inductors in $\mathcal{T}$. From the right hand side of (2) we see, that a tree with maximal weight is a tree for which the sum $|\mathcal{C}_T| + |\mathcal{L}_T|$ is maximal.

Instead of deleting $\mathcal{J}$ and contracting $\mathcal{V}$ we can (for a regular network) force $\mathcal{V}$ into the tree by giving each element in $\mathcal{V}$ weight 2 . Similarly, by giving every element in $\mathcal{J}$ weight –2 , we can prevent the element in $\mathcal{J}$ to appear in $\mathcal{T}$. In the figure below we show a network, in which the edges



Kant nr.	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
$\mathcal{T}$	V_1	V_2	C_3	C_4		C_6		C_8	C_9				R_{13}	L_{14}		L_{16}	L_{17}
$\lvert\mathcal{T}\rvert$	1	2	3	4		5		6	7				8	9		10	11

are labelled as done in line 1 in Kruskal's algorithm. Below the figure we show how the construction of a normal tree proceeds. The result is, that

$$\mathcal{T} = \{V_1, V_2, C_3, C_4, C_6, C_8, C_9, R_{13}, L_{14}, L_{16}, L_{17}\}$$

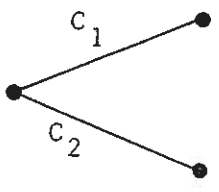
is a normal tree and therefore that the degree of $P(s)$ is 8 .

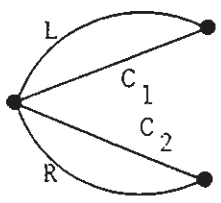
If $\mathcal{T}$ is a normal tree, then the voltages across the capacitors in $\mathcal{T}$ together with the currents in the inductors outside $\mathcal{T}$ can be used as "state variables", as explained in the theory of linear systems.

By means of (1), it is often possible to decide whether a given polynomial is characteristic polynomial for a network. For example, let us investigate whether there exists a network N consisting of a resistor R , an inductor L and two capacitors C_1 and C_2 such that

$$P(s) = R\,C_1\,C_2\,Ls^3 + L\,C_1\,s^2 + R\,C_2\,s + 1 \ .$$

If such a network N exists, from the terms in $P(s)$ we can reconstruct the trees in N : $C_1\,C_2$, $R\,C_1$, $L\,C_2$, RL .





Since the trees have two edges N must have 3 vertices and since $C_1\,C_2$ is a tree, N must contain the subgraph above, left. Since $R\,C_2$ and $L\,C_1$ are not trees, R must be parallel to C_2 and L must be parallel to C_1 , and hence the only possibility is the network N_0 to the right. And it is easy to check that $P(s)$ for N_0 is actually the given polynomial.

If, instead we had asked for the polynomial

$$P(s) = R\,C_1\,C_2\,Ls^3 + L\,C_1\,s^2 + 1$$

corresponding to the trees

$$C_1\,C_2\,,\quad R\,C_1\,,\quad R\,L$$

the above argument is still valid, so again, the network N_0 is the only possibility. Since N_0 has 4 trees, $P(s)$ can not be realized as characteristic polynomial of a network

Without proof we mention that if

$$P(s) = a_n\,s^n + a_{n-1}\,s^{n-1} + \cdots + a_1\,s + a_0$$

is characteristic polynomial for a network N, then *for every homogeneous solution to* N, that is *every solution* (i, v) *to* $N - \mathcal{J} \div \mathcal{V}$ *we have that* $x = i_k$ *as well as* $x = v_k$ *for all* k *is a solution to the differential equation*

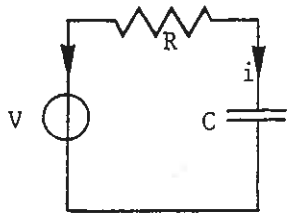
$$a_n \frac{d^n x}{dt^n} + a_{n-1} \frac{d^{n-1} x}{dt} + \cdots + a_1 \frac{dx}{dt} + a_0\,x = 0\ .$$

Conversely, it is not always the case that every solution to the differential equation occurs as coordinate in a solution to the network. But on the other side, no differential equation with degree $< n$ has the property emphasized above.

46. TRANSFER. FUNCTION. DRIVING POINT IMPEDANCE. We now continue where we ended up in section 44. So, we consider the case where the RCL–network N has only one generator G, either a current generator or a voltage generator, we want to determine the current in edge k, which is not the generator. If s is a complex variable, we define the transfer function $H(s)$ for N from G to i_k by the equation

$$(1) \qquad H(s) = \pm \frac{1}{P(s)} \sum_{\mathcal{T}} \pm \prod_{n \in \mathcal{T}} sC_n \prod_{n \notin \mathcal{T}} R_n \prod_{n \notin \mathcal{T}} sL_n\ ,$$

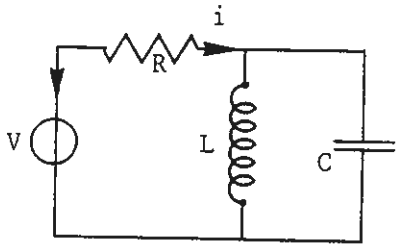
where the summation in the numerator is over all k–trees in N , that is over all those trees in N which do not contain G , and for whivh $\mathscr{C}(G, \mathscr{T})$ contains k . Under the summation + is used for a positive k–tree, – for a negative k–tree. In front of the fraction + is used when G is a current generator and – when G is a voltage generator. Let us take a couple of examples.

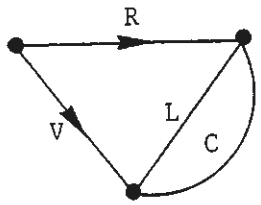


We want to find the transfer function H(s) from V to i in the network in the figure. There is only one C–tree, namely $\mathscr{T}\{R, C\}$. It is negative. P(s) is determined p. 6.25. Therefore

$$H(s) = -\frac{-sC}{RCs+1} = \frac{sC}{RCs+1} \cdot$$

As our next example we will find the transfer function H(s) from V to $i = i_R$ in the network in the figure below to the left. The network graph is shown to the right





and we find

$$P(s) = \quad sL \quad + \quad R \quad + \quad RC\,L\,s^2$$

There are two R–trees in the graph, namely {R, C} and {R, L} . They are both negative, so the transferfunction is

$$H(s) = \frac{LCs^2+1}{RLCs^2+Ls+R} \cdot$$

Note, that the R–tree {R, L} gives a product without factors in the numerator. The value of such a product is 1 .

An important application of the transfer function is expressed in

Theorem 46.1. When N is a RCL–network with one generator $G = c_0\, e^{st}$ then the stationary solution for the current i_k in edge k is determined by

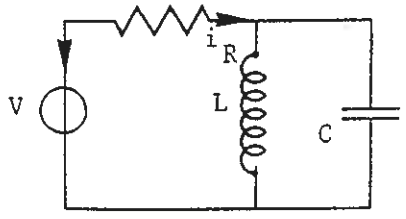
$$i_k = i_k(t) = c_0\, H(s) \cdot e^{st} \quad , \tag{2}$$

where $H(s)$ is the transfer function from G to i_k . (The result is not valid for those values of s for which $H(s) = 0$.)

Proof: As mentioned p 6.21, the stationary solution can be found by formula (15) p 6.24 by putting $G = c_0\, e^{st}$ and replacing kapacitors and inductors by resistors of size $\frac{1}{sC}$ and sL , respectivaly. When this is done, (2) is obtained from (15) by multiplying in numerator and denaminator with $\prod_{k \in \mathscr{C}} sC_k$. □

During the calculations we have to assume $s \neq 0$. However (2) is also valid for $s = 0$ (unless $P(0) = 0$). This can be seen by, from the very beginning, replacing the equation $v_k = \frac{1}{s\, C_k}\, i_k$ by $i_k = s\, C_k\, v_k$.

Formula (2) shows how $H(s)$ can be used to determine the stationary solution. But actually (– and surprisingly) $H(s)$ can be used to determine an arbitrary solution for an arbitrary generator $G = g(t)$. For example, above we found, that the transfer



function $H(s)$ from V to i_R in the network in the figure is

$$H(s) = \frac{LCs^2 + 1}{RLCs^2 + Ls + R} \quad .$$

It can be shown, that for every generator voltage $V(t)$ and for every solution $(\mathbf{i}, \mathbf{v})$ to the network, $i = i_R$ will be a solution to the differential equation

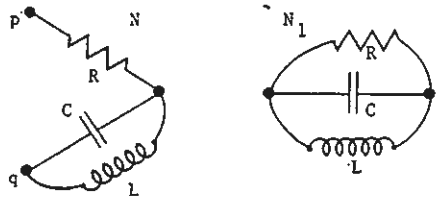
$$RCL\, \frac{d^2 i}{dt^2} + L\, \frac{di}{dt} + R\, i = L\, C\, \frac{d^2 V}{dt^2} + V \quad .$$

The coefficients in the denominator in $H(s)$ are the coeficients on the left hand side of the differential equation, and the numerator coefficients appear on the right hand side. This principle is valid for every transfer function.

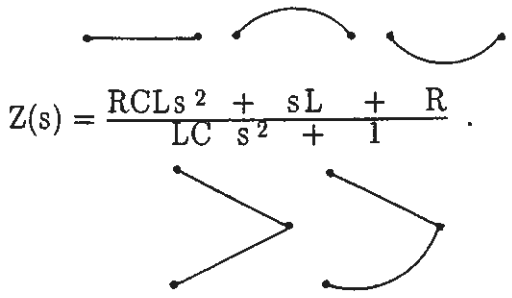
Finally, the driving point impedance $Z(s)$ between two vertices p and q is a RCL-network without generators is defined by the formula

$$Z(s) = \frac{P_1(s)}{P(s)} \ ,$$

where $P(s)$ is the characteristic polynomial for N, and where $P_1(s)$ is the characteristical polynomial for the network N_1 obtained from N by connecting p and q by a voltage generator. For example, the driving point impedance between the



two vertices p and q in the network in the left hand side of the figure is

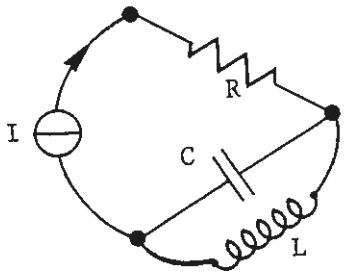


$$Z(s) = \frac{RCLs^2 + sL + R}{LC\ s^2 + 1} \ .$$

Just like we proved in theorem 46.1 by (15) p. 6.24, we can prove the following theorem by means of (18) p. 248.

Theorem 46.2. If p and q are two vertices in a RCL-network without generators and if we place a current generator with size $I = i_0\, e^{st}$ between p and q, then the generator voltage V is

$$V = Z(s)\, I = i_0\, Z(s)\, e^{st} \ .$$

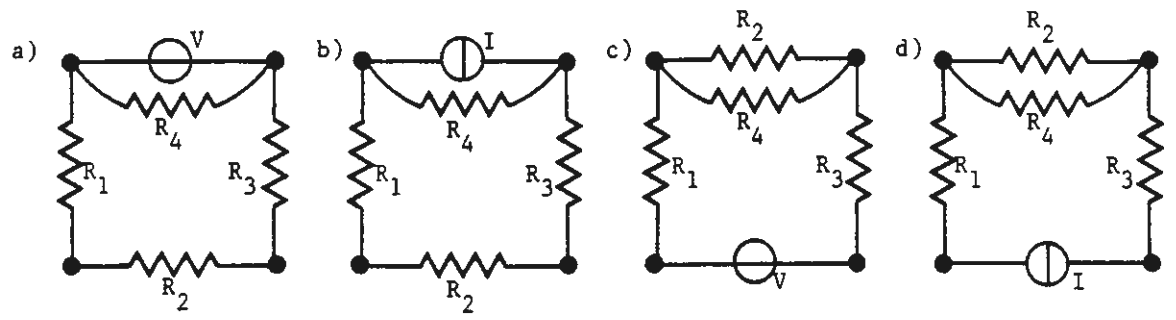


If we, in the figure, put $I = i_0 e^{st}$, then the stationary voltage across the generator is

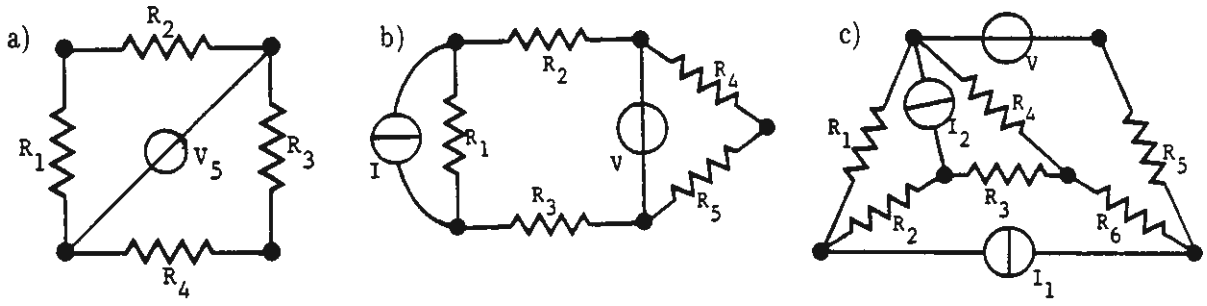
$$V(s) = i_0 Z(s) e^{st} = i_0 \frac{RCL s^2+Ls+R}{LCs^2+1} e^{st} \quad .$$

PROBLEMS TO CHAPTER 6.

6.1. Compute the network determinants for the following networks:



6.2. Compute the network determinants for the following networks:



6.3. Find – if possible – a network of K resistors with determinant:

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

b) $R_1R_2R_5 + R_3R_5 + R_4R_5$, K = 3.

c) $R_1R_2 + R_1R_3 + R_2R_3$, K = 3.

d) $R_1R_2 + R_3R_4 + R_1R_3$, K = 4.

e) $R_1R_2 + R_1R_3 + R_1R_4 + R_2R_3 + R_2R_4 + R_3R_4$, K = 4.

6.4. Given the vectors

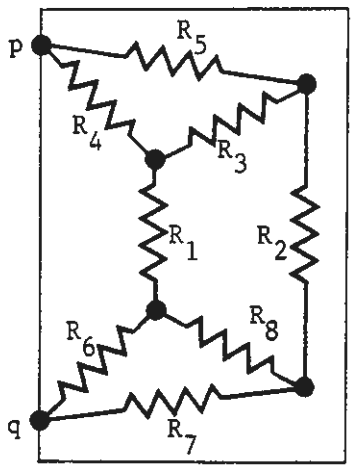
$$\mathbf{x} = (1, 3, 5, 7, 9, 11, 13, 17, -25)\ ,$$

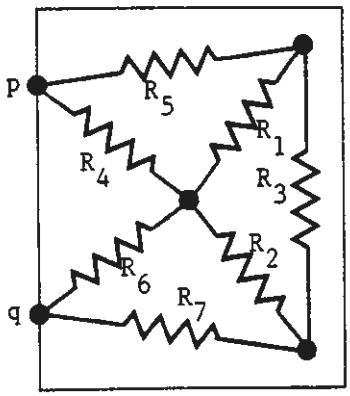
$$\mathbf{y} = (4, 8, 2, 3, 6, 1, 7, 5, 12)\ .$$

a) Find a regular network of resistors, in which the voltage is **x** and the current **y**.

b) Does there exist a regular network of resistors in which the current is **x** and the voltage is **y** ?

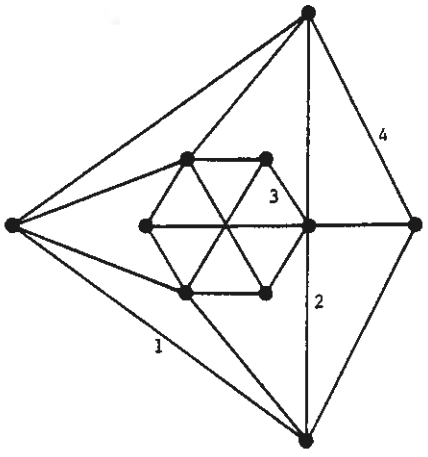
6.5. Find in the network in the figure to the left the driving point resistance between p and q as function of x and y, when $R_1 = x$, $R_2 = y$ and $R_i = 1$ for $i = 3, 4, 5, 6, 7, 8$.





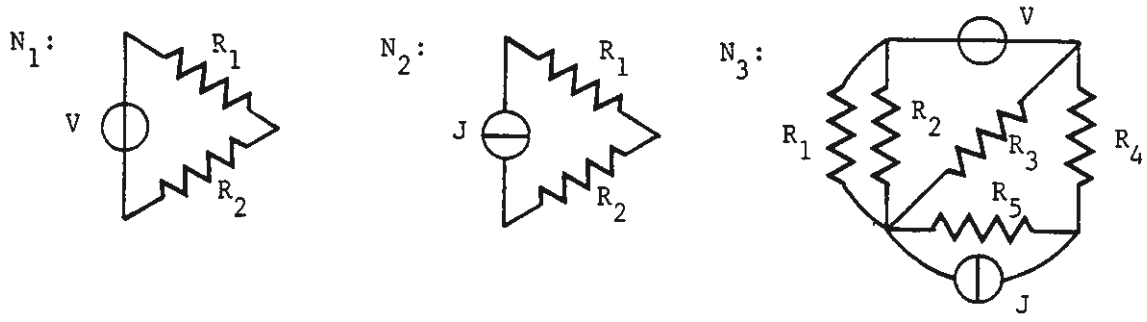
6.6. Find in the network in the figure to the right the driving point resistance between p and q as function of x and y, when $R_1 = x$, $R_2 = y$ and $R_i = 1$ for $i = 3, 4, 5, 6, 7$.

6.7.

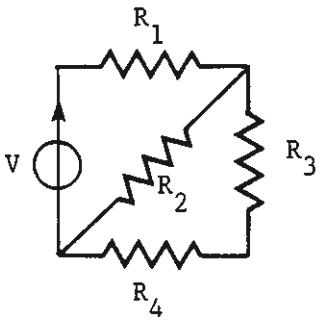


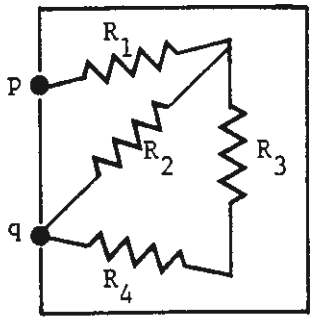
In the network in the figure edge 1 is a voltage generator of size 1, and the other edges are resistors. Answer, for $j = 2, 3$ and 4, the following question: Can the resistors be given values for which the current in edge j is 0 ?

6.8. Find the determinants for the following networks.



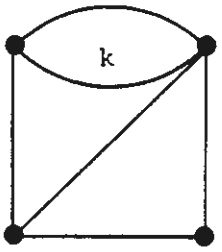
6.9. Find the current through V in the network to the left.

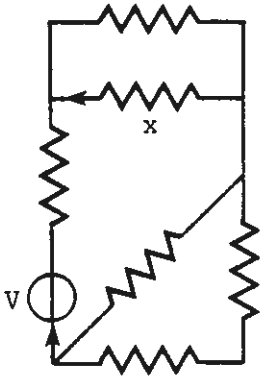




6.10. Find the driving point resistance between p and q in the network to the right.

6.11. a) Calculate, by Feussner's method how many trees containing k there are in the graph to the left. How many trees are there, which does not contain k .

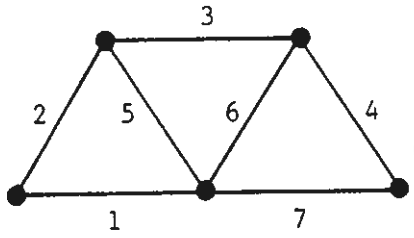


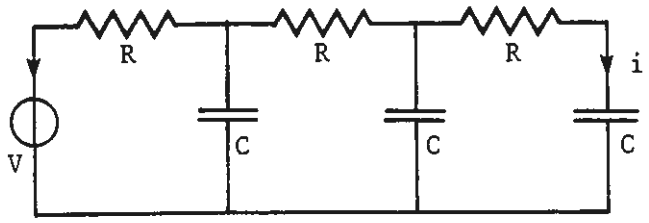


b) Find, in the network to the right, the current through the resistor of size x , if all other resistors have size 1 .

6.12. a) Find the number τ of trees containing edge 1 in the network to the left.

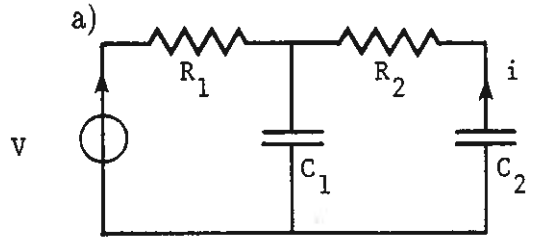
b) For $i = 0, 1, 2, 3$ let τ_i be the number of trees which contains edge 1 and precisely i of the edges 5, 6 and 7. Find the numbers τ_0, τ_1, τ_2 and τ_3.

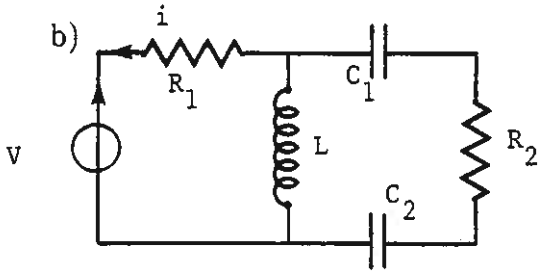




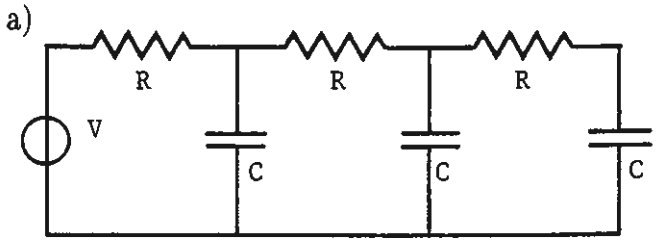
c) Find the transfer function from V to i in the network to the right.

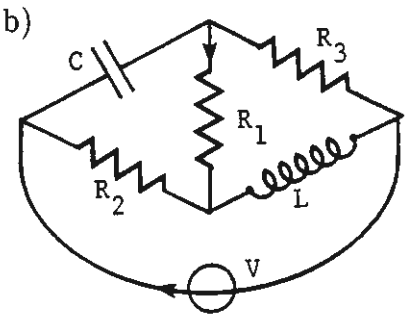
6.13. Find the transfer function from V to i in each of the two networks in the figures below.





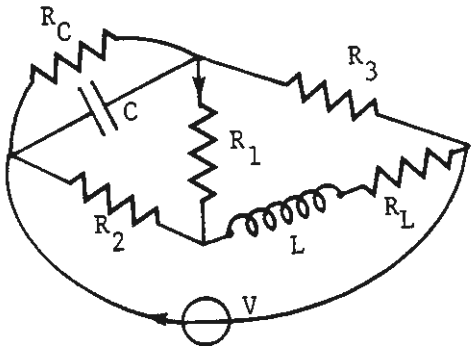
6.14. Find the driving point impedance which V looks into in the figure to the left.





6.15. Find the transfer function from V to i_1 in the network to the right.

6.16. In the network in the figure the characteristic polynomial has the

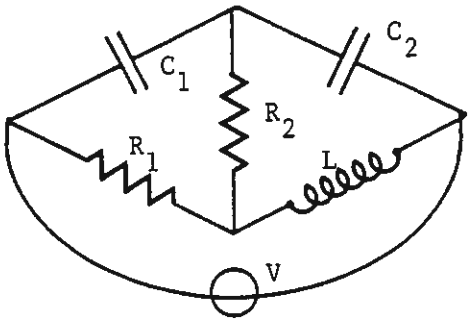


form

$$P(s) = A_1 s^2 + A_2 s + A_3.$$

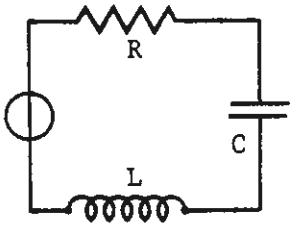
Find the coefficients A_1, A_2 and A_3 by first drawing such graphs that the trees in each graph only contributes to one of the coefficients A_1, A_2 and A_3. Finally, find the transfer function from V to i_1.

6.17. Find the characteristic polynomial for the network in the figure.

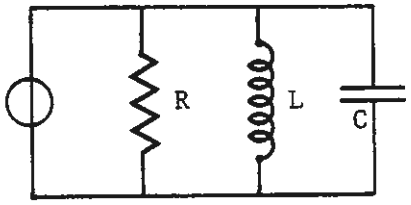


6.18. Find the characteristic polynomials for the networks in the figure.

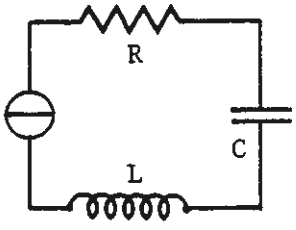
a)



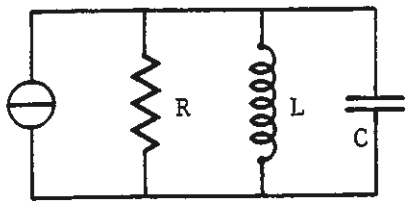
b)



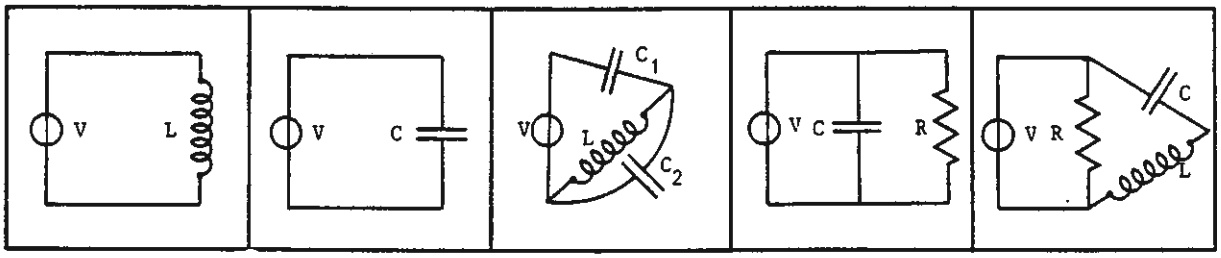
c)



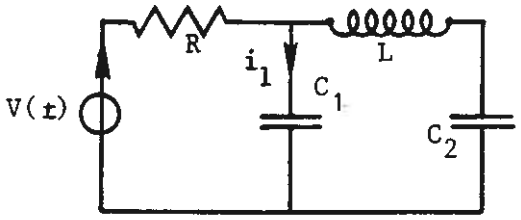
d)



6.19. Find the characteristic polynomials for each of the networks below.



6.20.



In the network in the figure, $V(t) = 2e^{st}$. Find the stationary solution for the current i_1 . Find the transfer function from V to i_1 and write for an arbitrary voltage V(t) a differential equation for $i_1(t)$.

6.21. Construct from the components R_1, R_2, C_1, C_2, L_1, L_2 a network with the characteristic polynomial

$$P(s) = (L_1 + L_2)\, C_1\, C_2\, s^3 + (R_1 + R_2)\, C_1\, C_2\, s^2 + (C_1 + C_2)\, s\ .$$

Is there more than one possibility?

6.22. Can 4 components R_1, R_2, L and C be connected to a network with the characteristic polynomial

$$P(s) = (R_1\, L\, C + R_2\, L\, C)\, s^2 + (L + R_1\, R_2\, C)s + R_1 + R_2\ ?$$

LITERATURE TO CHAPTER 6.

[1] Kirchhoff, G.: Über die Auflösung der Gleichungen, auf welche man bei der Untersuchungen der Linearen Verteilung Galvanisher Ströme geführt wird. Poggendorf Ann. Physik 72 (1847), 497–508. Trans. Inst. Radio Engrs. CT–5 (March 1958), 4–7.

[2] Mayeda, W.: Graph Theory. Wiley 1972.

[3] Seshu, S., M.B. Reed: Linear Graphs and Electrical Networks, Addison–Wesley 1961.

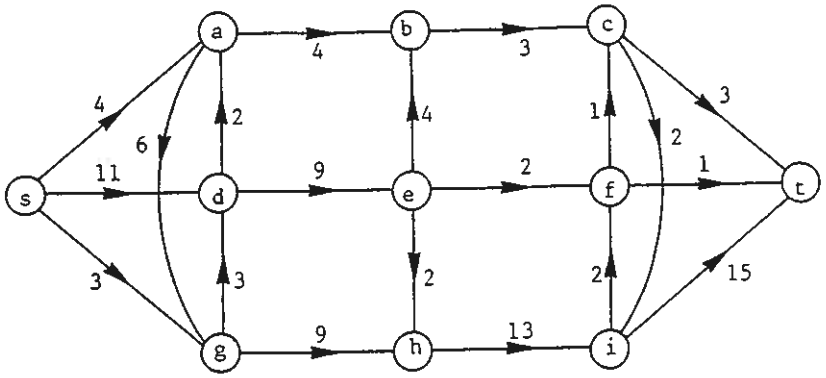
[4] Thomassen, C.: Resistances and Currents in Infinite Electrical Networks. Journal Comb. Theory, Vol. 49, No. 1, June 1990, p. 87–102.

EXSAMINATION PROBLEMS, 1993 AND 1994.

FOUR HOURS WRITTEN EXAMINATION.

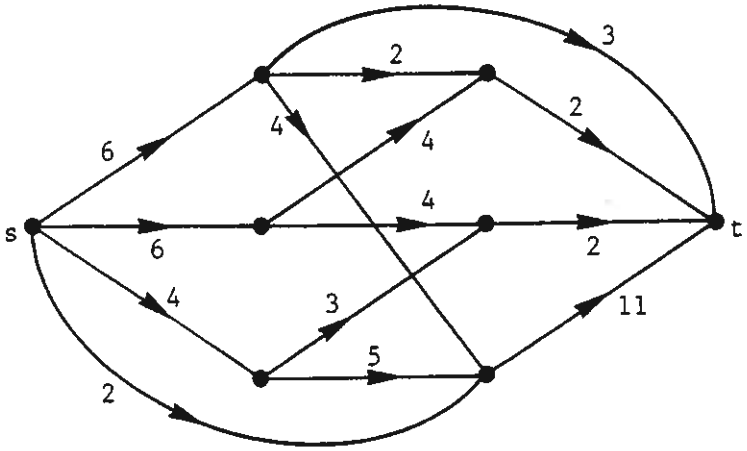
JANUARY 1993.

PROBELM 1.



1) Find a maximal flow and a minimal cut in the transport network in the figure.
2) Find the critical edges.
3) Which of the edges ag , ci and ct have optimal capacity?

PROBLEM 2.

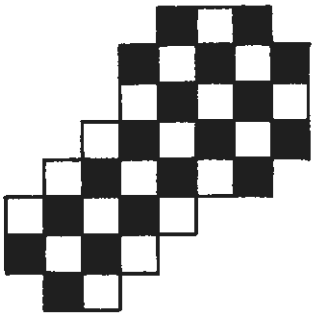


Use Dinic's algorithm to find a maximal flow in the network in the figure.

1) Draw the minimal networks you use and show in each of these the corresponding complete flow.
2) Let F be the value of the flow obtained after two minimal networks have been used. Can the algorithm perform such that

   a) $F = 16$?
   b) $F = 12$?

PROBLEM 3.

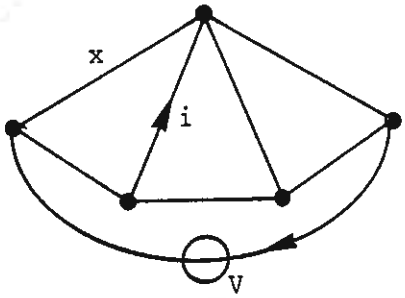


The figure consists 18 black and 18 white squares. A domino is a black and a white square, which are neighbours. Find the maximal number of dominoes, which can be cut out from the figure. Proof in two ways, the number is in fact maximal:

1) By means of a labelling process.

2) By using König's theorem.

PROBLEM 4.



The figure shows an electrical network with a voltage generator V and a resistor of size x. The other 6 edges are resistors of size 1. Find the current i, expressed by V and x.

PROBLEM 5.

Is it possible to connect two resistors R_1 and R_2, an inductor L and a capacitor C to form RCL–networks with the following characteristic polynomials?

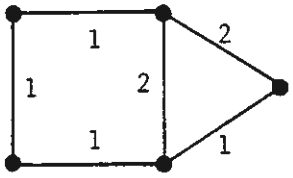
1) $P(s) = (R_1 + R_2)\, L\, C\, s^2 + L\, s + R_2$?

2) $P(s) = R_1\, L\, C\, s^2 + (R_1\, R_2\, C + L)\, s + R_2$?

PROBLEM 6.

Let Γ be a graph in which every edge k has a weight $w(k) \geq 0$. An independence system $\mathcal{U}_0$ on the edge set $\mathcal{K}(\Gamma)$ is defined in the following way:

$$U \in \mathcal{U}_0 \Leftrightarrow U \text{ contains at most 2 circuits.}$$



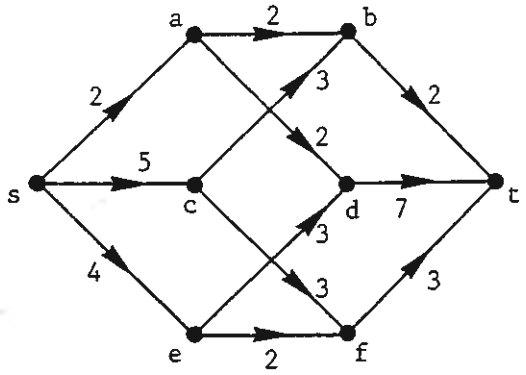
1) The figure shows a graph Γ with 3 circuits. The number next to an edge k is $w(k)$. The greedy algorithm is applied for this graph Γ on $(\mathcal{U}_0, w)$. Which edge sets can be outputs? Does the greedy algorithm work in this case?

2) Is it true that for every graph Γ and for every weight function $w(k)(\geq 0$ for all $k \in \mathcal{K}(\Gamma))$ that the greedy algorithm works on $(\mathcal{U}_0, w)$. Motivate your answer.

———————ooo0ooo———————

JUNE 1993.

PROBLEM 1.

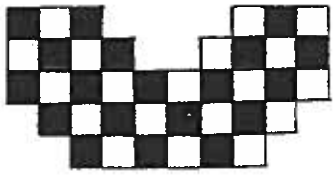


1) Find a maximal flow and a minimal cut in the transport network in the figure.
2) Find the critical edges.
3) Find the edges with optimal capacity.

PROBLEM 2.

Let N be a transport network and k an edge in N. Describe and proof an algorithm which decides whether there exists a maximal flow f such that $f(k) > 0$.

PROBLEM 3.



The figure consists 19 black and 19 white squars. A domino is a black and a white square, which are neighbours. Find the maximal number of dominoes, which can be cut out from the figure. Proof in two ways, the number is in fact maximal:

1) By means of a labelling process.

2) By using König's theorem.

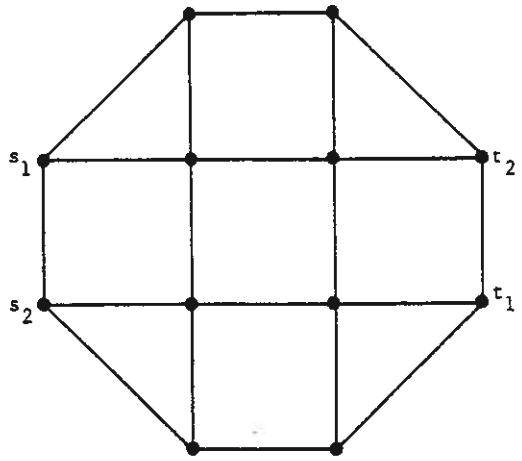
PROBLEM 4.

Is it possible to connect two (respectively three) resistors, one inductor and two capacitors to a RCL–network N with the following characteristic polynomials:

1) $P_N(s) = C_1C_2L(R_1 + R_2)s^3 + C_1(L + C_2R_1R_2)s^2 + (C_2R_1 + C_1R_2 + C_2R_2)s + 1$?
2) $P_N(s) = R_1R_2C_1C_2Ls^3 + (R_1R_2R_3C_1C_2 + R_1C_1L + R_2C_2L)s^2 + (R_1R_3C_1 + R_2R_3C_2 + L)s + R$

PROBLEM 5.

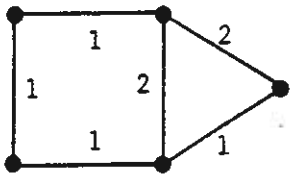
Find a maximal flow in the 2–commodity network in the figure, where all capacities are 1.
Find a minimal cut in the network.



PROBLEM 6.

Let Γ be a graph in which every edge k has a weight $w(k) \geq 0$. An independence system $\mathcal{U}_0$ and the edge set $\mathcal{K}(\Gamma)$ is defined in the following way:

$$U \in \mathcal{U}_0 \Leftrightarrow U \text{ contains at most 1 circuits.}$$



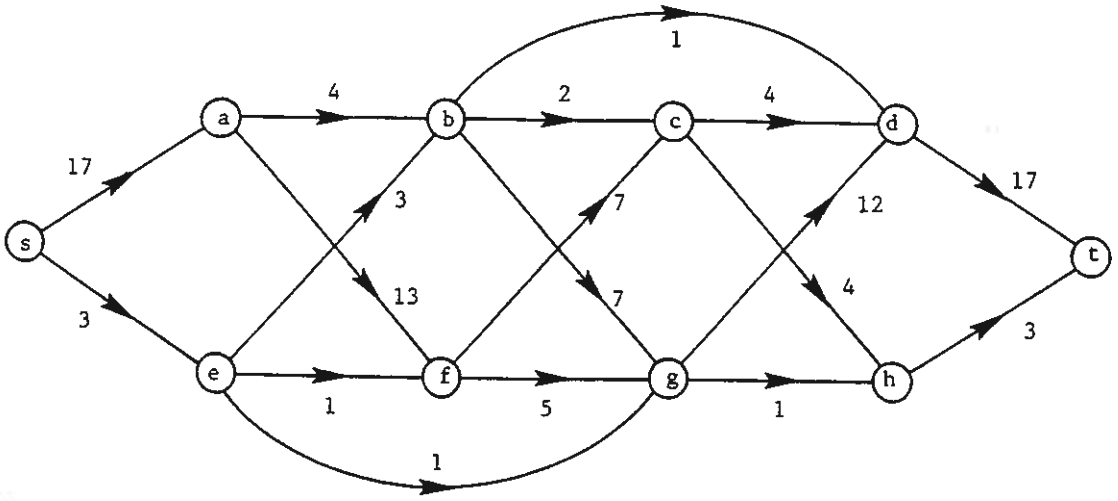
1) The figure shows a graph Γ with 3 circuits. The number next to an edge k is $w(k)$. The greedy algorithm is applied for this graph Γ on $(\mathcal{U}_0, w)$. Which edge sets can be outputs? Does the greedy algorithm work in this case?

2) Is it true that for every graph Γ and for every weight function $w(k)(\geq 0$ for all $k \in \mathcal{K}(\Gamma))$ that the greedy algorithm works on $(\mathcal{U}_0, w)$. Motivate your answer.

———————ooo0ooo———————

JANUARY 1994.

PROBLEM 1.



1) Find a maximal flow and a minimal cut in the transport network in the figure.
2) Find the critical edges.
3) Find the edges with optimal capacity.

PROBLEM 2.

1) Find the normal rank of the matrix $\mathbf{A} = \begin{bmatrix} 1&0&0&1&0&1&0&0&0 \\ 0&1&0&0&0&1&0&1&1 \\ 0&0&1&1&0&0&0&0&0 \\ 0&0&1&1&1&0&0&0&0 \\ 0&0&0&0&1&0&1&1&1 \\ 0&0&1&0&1&0&0&0&0 \\ 0&0&0&1&1&0&0&0&0 \\ 1&1&1&0&1&0&0&0&0 \\ 0&0&1&0&1&0&0&0&0 \end{bmatrix}$.

   Which algorithm do you use?
2) Find a minimal system of rows and columns in $\mathbf{A}$, which together contain all 1's in the matrix. Motivate your answer.
3) Find the complexity of the algorithm you used in 1), when it is used on a $n \times n$ – matrix. The answer should be expressed by n.

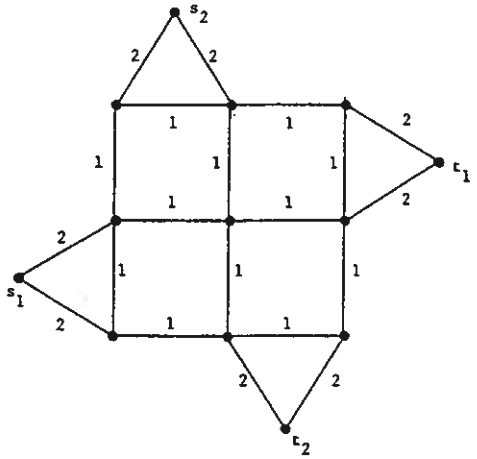
PROBLEM 3.

Edmonds – Korps algorithm is used on transport network with P vertics and K edges in which all capacities are 1 . Find the complexity of the algorithm

1) When the network may contain multiple edges.
2) When the network does not contain multiple edges.

Motivate your answer. The complexity should be expressed by P and K .

PROBLEM 4.



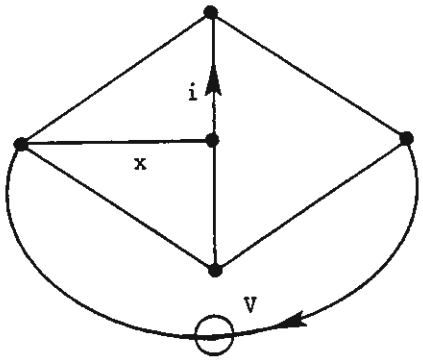
Find a maksimal flow and a minimal cut in the 2–commendity network in the figure.

PROBLEM 5.

Find by means of the assigment algorithm a maximal weight matching for the matrix

$$\begin{bmatrix} 3 & 4 & 7 & 2 & 4 \\ 5 & 6 & 8 & 4 & 4 \\ 3 & 3 & 7 & 2 & 4 \\ 2 & 4 & 6 & 2 & 5 \\ 3 & 4 & 7 & 2 & 5 \end{bmatrix} .$$

PROBLEM 6.



The network in the figure contains a voltage generator and 7 resistors. One of the resistors has size $x\Omega$, the others are 1Ω resistors.

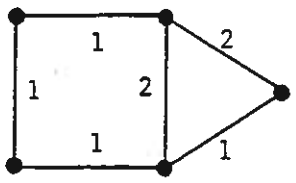
1) Find the network determinant.
2) Find the current i expressed by x and V .

PROBLEM 7.

Let Γ be a connected graph in which every edge k has a weight $w(k) \geq 0$. An independence system $\mathcal{U}_2$ and the edge set $\mathcal{K}(\Gamma)$ is defined in the following way:

$U \in \mathcal{U}_2 \Leftrightarrow$ The subgraph in Γ , which has vertex set $\mathcal{P}(\Gamma)$ and edge set U is circuit free and has at least two connected comparents.

1) The figure shows a graph Γ . The number next to an edge k is $w(k)$. The greedy algorithm is applied for this graph Γ on $(\mathcal{U}_2, w)$. Which edge sets can be outputs? Does the greedy algorithm work in this case?

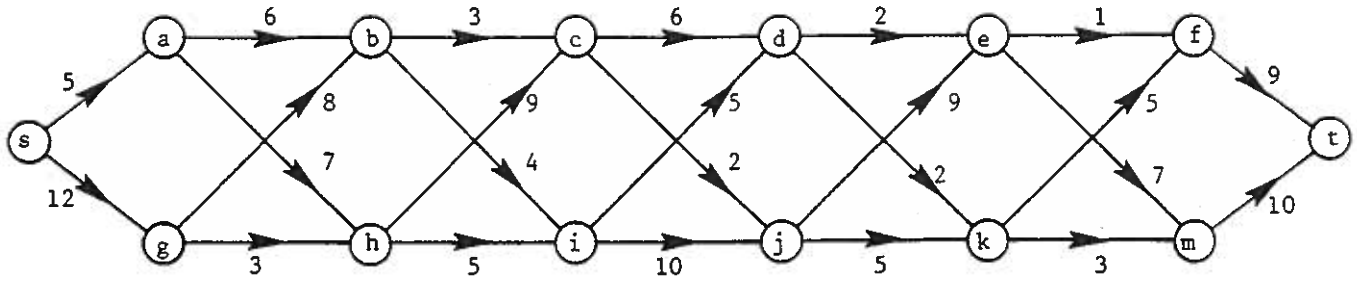


2) Is it true that for every graph Γ and for every weight function $w(k)(\geq 0$ for all $k \in \mathcal{K}(\Gamma))$ that the greedy algorithm works on $(\mathcal{U}_2, w)$. Motivate your answer.

———————ooo0ooo———————

JUNE 1994.

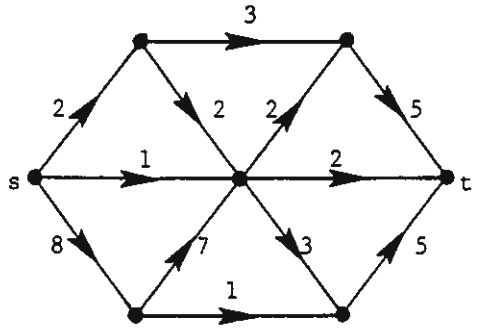
PROBLEM 1.



1) Find a maximal flow and two minimal cuts in the network in the figure.
2) Does there exist a minimal cut in the network, which contains the edge x when

a) x = (de) ; b) x = (kf) ; c) x = (km) ?

PROBLEM 2.



1) Find by means of Dinic's algorithm a maximal flow in the network in the figure. Draw the minimal networks used and show in the figures the augmenting paths used.
2) Is it possible, that Dinic's algorithm performs such that the number of minimal networks used is

a) 2 ; b) 3 ; c) 4 ?

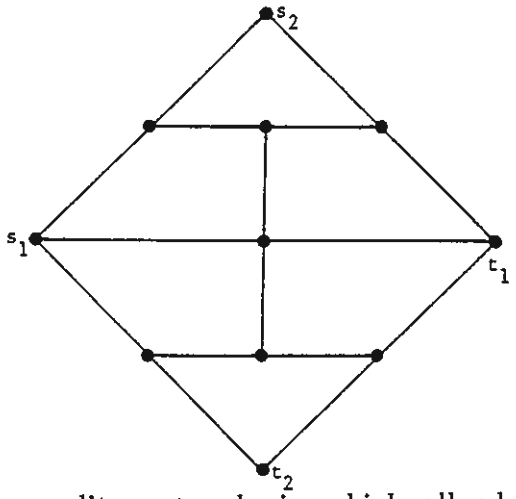
PROBLEM 3.

1) Find the normal rank $\rho(A)$ of the matrix $A = \begin{bmatrix} 0\,1\,0\,0\,1\,1 \\ 1\,1\,1\,1\,0\,1 \\ 0\,1\,0\,0\,1\,1 \\ 0\,1\,0\,0\,1\,0 \\ 1\,1\,1\,0\,1\,1 \\ 0\,1\,0\,0\,1\,1 \end{bmatrix}$.

2) Does A contain a 0 with the property that when it is changed to 1, then $\rho(A)$ is not changed?
3) Does A contain a 1 with the property that when it is changed to 0, then $\rho(A)$ decreases by 1?

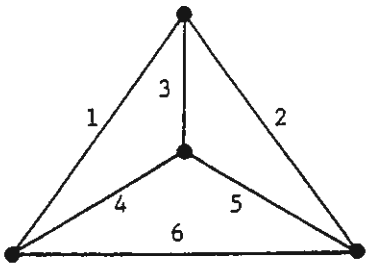
Motivate your answers.

PROBLEM 4.



The figure shows a 2–commordity network, in which all edges has capacity 1. Find a maximal flow and a minimal cut in the network.

PROBLEM 5.



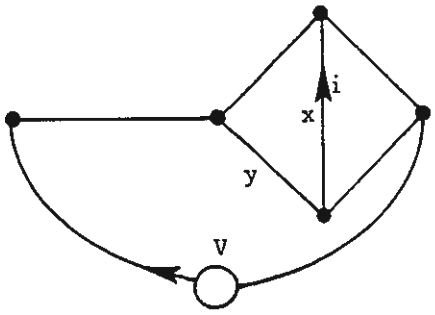
The figure shows a graph Γ with edge set $E = \{1, 2, 3, 4, 5, 6\}$. The set $\mathcal{U}$ of those subsets U of E for which U does not contain the edge set of a triangle in Γ, is an independence system on E.

1) Find the possible outputs from the greedy algorithm when it is applied to $\mathcal{U}$ with the weight function

$$w(1) = w(5) = 2\,,\ w(2) = w(3) = w(4) = w(6) = 1\,.$$

Does GA work in this case?
2) Does GA work on $\mathcal{U}$ for every weight function w?

PROBLEM 6.



The figure shows a network consisting of a voltage generator V, of two resistors of size $x\Omega$ and $y\Omega$, respectively and 4 more resistors, all of size 1Ω.

1) Find the network determinent $\Delta(N)$.
2) Find the current i through the resistor x.

———oooOooo———

INDEX