GRAPH THEORY

Keijo Ruohonen

(Translation by Janne Tamminen, Kung-Chung Lee and Robert Piché)

2006

Contents

Foreword

These lecture notes were translated from the Finnish lecture notes for the TUT course ”Graafiteoria”. The laborious bulk translation was taken care of by the students Janne Tamminen (TUT) and Kung-Chung Lee (visiting from the University of British Columbia). Most of the material was then checked by professor Robert Piché. I want to thank the translation team for their effort.

The notes form the base text for the course ”MAT-41196 Graph Theory”. They contain an introduction to basic concepts and results in graph theory, with a special emphasis put on the network-theoretic circuit-cut dualism. In many ways a model was the elegant and careful presentation of SWAMY & THULASIRAMAN, especially the older (and better) edition. There are of course many modern text-books with similar contents, e.g. the popular GROSS & YELLEN.

One of the usages of graph theory is to give a unified formalism for many very different-looking problems. It then suffices to present algorithms in this common formalism. This has lead to the birth of a special class of algorithms, the so-called *graph algorithms.* Half of the text of these notes deals with graph algorithms, again putting emphasis on network-theoretic methods. Only basic algorithms, applicable to problems of moderate size, are treated here. Special classes of algorithms, such as those dealing with sparse large graphs, ”small-world” graphs, or parallel algorithms will not be treated. In these algorithms, data structure issues have a large role, too (see e.g. SKIENA).

The basis of graph theory is in combinatorics, and the role of ”graphics” is only in visualizing things. Graph-theoretic applications and models usually involve connections to the ”real world” on the one hand—often expressed in vivid graphical terms—and the definitional and computational methods given by the mathematical combinatoric and linear-algebraic machinery on the other. For many, this interplay is what makes graph theory so interesting. There is a part of graph theory which actually deals with graphical drawing and presentation of graphs, briefly touched in Chapter 6, where also simple algorithms are given for planarity testing and drawing. The presentation of the matter is quite superficial, a more profound treatment would require some rather deep results in topology and curve theory. Chapter 7 contains a brief introduction to matroids, a nice generalization and substitute for graphs in many ways.

Proofs of graph-theoretic results and methods are usually not given in a completely rigorous combinatoric form, but rather using the possibilities of visualization given by graphical presentations of graphs. This can lead to situations where the reader may not be completely convinced of the validity of proofs and derivations. One of the goals of a course in graph theory must then

be to provide the student with the correct ”touch” to such seemingly loose methods of proof. This is indeed necessary, as a completely rigoristic mathematical presentation is often almost unreadable, whereas an excessively slack and lacunar presentation is of course useless.

Keijo Ruohonen

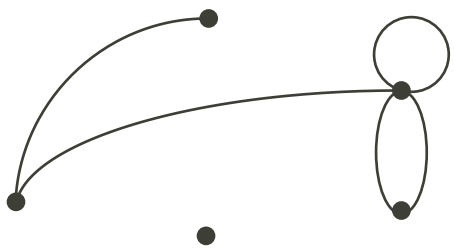
Chapter 1

Definitions and Fundamental Concepts

1.1 Definitions

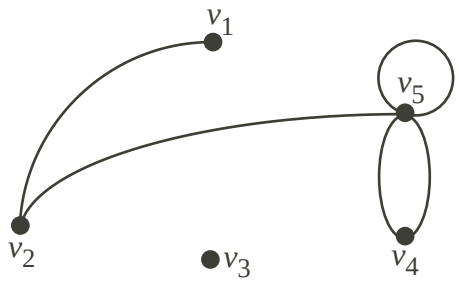
Conceptually, a *graph* is formed by *vertices* and *edges* connecting the vertices.

Example.



Formally, a graph is a pair of sets (V, E), where V is the *set of vertices* and E is the *set of edges*, formed by pairs of vertices. E is a *multiset*, in other words, its elements can occur more than once so that every element has a *multiplicity*. Often, we label the vertices with letters (for example: $a, b, c, \ldots$ or $v_1, v_2, \ldots$) or numbers $1, 2, \ldots$ Throughout this lecture material, we will label the elements of V in this way.

Example. *(Continuing from the previous example) We label the vertices as follows:*

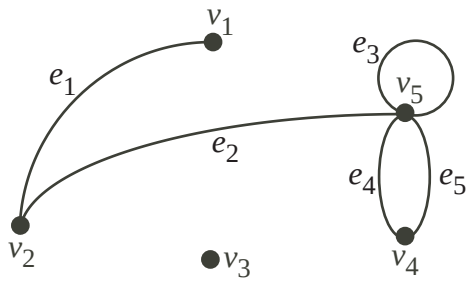


We have $V = \{v_1, \ldots, v_5\}$ *for the vertices and* $E = \{(v_1, v_2), (v_2, v_5), (v_5, v_5), (v_5, v_4), (v_5, v_4)\}$ *for the edges.*

Similarly, we often label the edges with letters (for example: $a, b, c, \ldots$ or $e_1, e_2, \ldots$) or numbers $1, 2, \ldots$ for simplicity.

Remark. *The two edges (u, v) and (v, u) are the same. In other words, the pair is not* ordered.

Example. *(Continuing from the previous example) We label the edges as follows:*



So $E = \{e_1, \ldots, e_5\}$.

We have the following terminologies:

1. The two vertices u and v are *end vertices* of the edge (u, v).
2. Edges that have the same end vertices are *parallel*.
3. An edge of the form (v, v) is a *loop*.
4. A graph is *simple* if it has no parallel edges or loops.
5. A graph with no edges (i.e. E is empty) is *empty*.
6. A graph with no vertices (i.e. V and E are empty) is a *null graph*.
7. A graph with only one vertex is *trivial*.
8. Edges are *adjacent* if they share a common end vertex.
9. Two vertices u and v are *adjacent* if they are connected by an edge, in other words, (u, v) is an edge.
10. The *degree* of the vertex v, written as $d(v)$, is the number of edges with v as an end vertex. By convention, we count a loop twice and parallel edges contribute separately.
11. A *pendant vertex* is a vertex whose degree is 1.
12. An edge that has a pendant vertex as an end vertex is a *pendant edge*.
13. An *isolated vertex* is a vertex whose degree is 0.

Example. *(Continuing from the previous example)*

- *v_4 and v_5 are end vertices of e_5.*
- *e_4 and e_5 are parallel.*
- *e_3 is a loop.*
- *The graph is not simple.*
- *e_1 and e_2 are adjacent.*

- v_1 *and* v_2 *are adjacent.*
- *The degree of* v_1 *is* 1 *so it is a pendant vertex.*
- e_1 *is a pendant edge.*
- *The degree of* v_5 *is* 5.
- *The degree of* v_4 *is* 2.
- *The degree of* v_3 *is* 0 *so it is an isolated vertex.*

In the future, we will label graphs with letters, for example:

$$G = (V, E).$$

The *minimum degree* of the vertices in a graph G is denoted $\delta(G)$ ($= 0$ if there is an isolated vertex in G). Similarly, we write $\Delta(G)$ as the *maximum degree* of vertices in G.

Example. *(Continuing from the previous example)* $\delta(G) = 0$ *and* $\Delta(G) = 5$.

Remark. *In this course, we only consider* finite *graphs, i.e.* V *and* E *are finite sets.*

Since every edge has two end vertices, we get

Theorem 1.1. *The graph* $G = (V, E)$*, where* $V = \{v_1, \ldots, v_n\}$ *and* $E = \{e_1, \ldots, e_m\}$*, satisfies*

$$\sum_{i=1}^{n} d(v_i) = 2m.$$

Corollary. *Every graph has an even number of vertices of odd degree.*

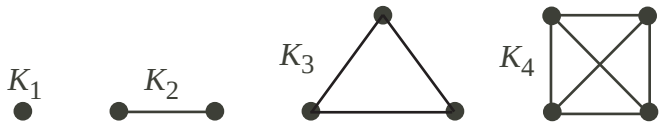
Proof. If the vertices $v_1, \ldots, v_k$ have odd degrees and the vertices $v_{k+1}, \ldots, v_n$ have even degrees, then (Theorem 1.1)

$$d(v_1) + \cdots + d(v_k) = 2m - d(v_{k+1}) - \cdots - d(v_n)$$

is even. Therefore, k is even. □

Example. *(Continuing from the previous example) Now the sum of the degrees is* $1 + 2 + 0 + 2 + 5 = 10 = 2 \cdot 5$*. There are two vertices of odd degree, namely* v_1 *and* v_5.

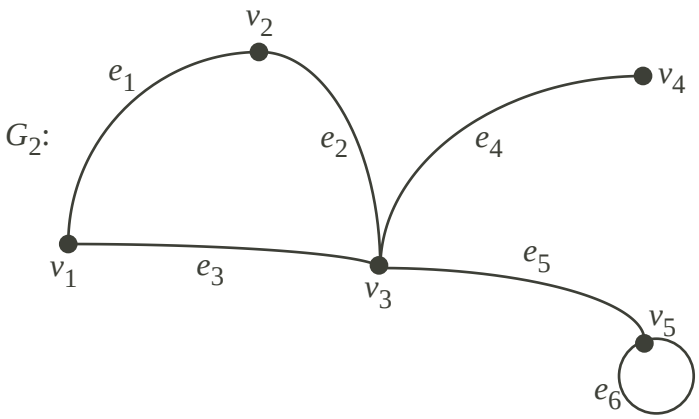
A simple graph that contains every possible edge between all the vertices is called a *complete graph*. A complete graph with n vertices is denoted as K_n. The first four complete graphs are given as examples:



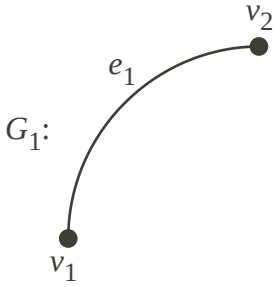
The graph $G_1 = (V_1, E_1)$ is a *subgraph* of $G_2 = (V_2, E_2)$ if

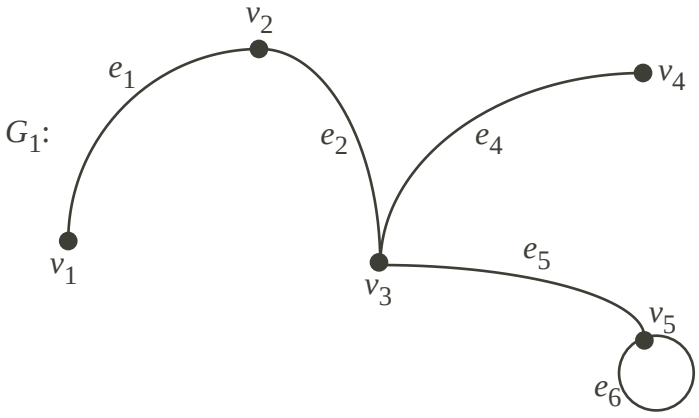
1. $V_1 \subseteq V_2$ and
2. Every edge of G_1 is also an edge of G_2.

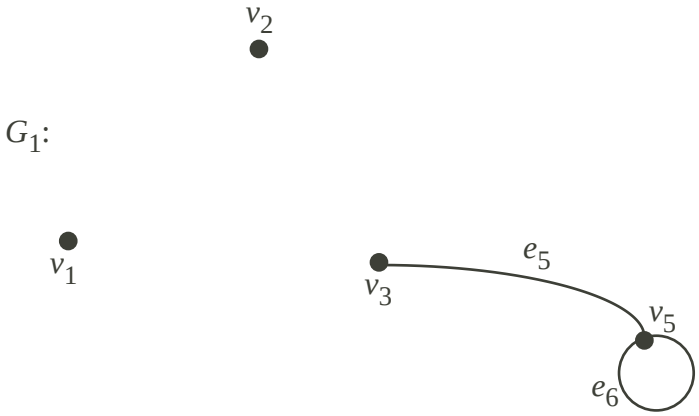
Example. *We have the graph*



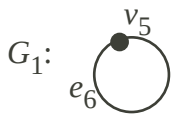
and some of its subgraphs are







and

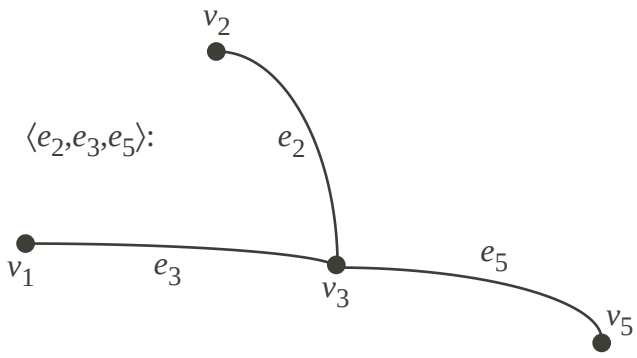


The *subgraph of* $G = (V, E)$ *induced by the edge set* $E_1 \subseteq E$ is:

$$G_1 = (V_1, E_1) =_{\text{def.}} \langle E_1 \rangle,$$

where V_1 consists of every end vertex of the edges in E_1.

Example. *(Continuing from above) From the original graph* G*, the edges* e_2*,* e_3 *and* e_5 *induce the subgraph*

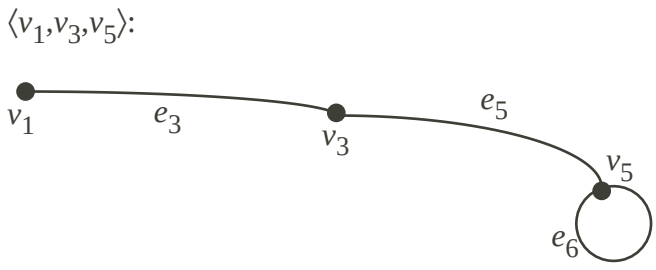


The *subgraph of* $G = (V, E)$ *induced by the vertex set* $V_1 \subseteq V$ is:

$$G_1 = (V_1, E_1) =_{\text{def.}} \langle V_1 \rangle,$$

where E_1 consists of every edge between the vertices in V_1.

Example. *(Continuing from the previous example) From the original graph* G*, the vertices* v_1*,* v_3 *and* v_5 *induce the subgraph*



A complete subgraph of G is called a *clique* of G.

1.2 Walks, Trails, Paths, Circuits, Connectivity, Components

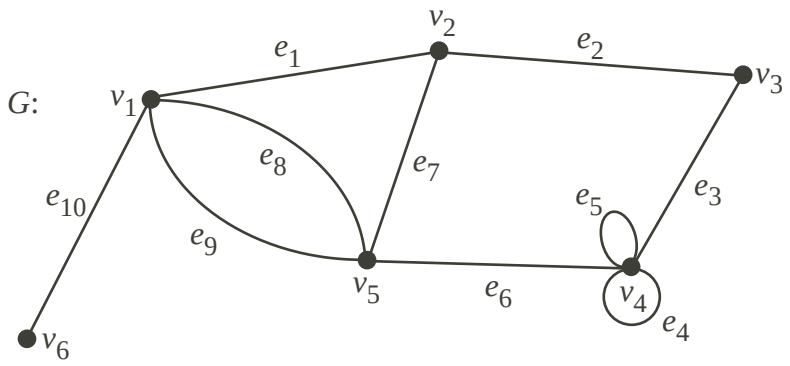
Remark. *There are many different variations of the following terminologies. We will adhere to the definitions given here.*

A *walk* in the graph $G = (V, E)$ is a finite sequence of the form

$$v_{i_0}, e_{j_1}, v_{i_1}, e_{j_2}, \ldots, e_{j_k}, v_{i_k},$$

which consists of alternating vertices and edges of G. The walk starts at a vertex. Vertices $v_{i_{t-1}}$ and v_{i_t} are end vertices of e_{j_t} ($t = 1, \ldots, k$). v_{i_0} is the *initial vertex* and v_{i_k} is the *terminal vertex*. k is the *length* of the walk. A zero length walk is just a single vertex v_{i_0}. It is allowed to visit a vertex or go through an edge more than once. A walk is *open* if $v_{i_0} \neq v_{i_k}$. Otherwise it is *closed*.

Example. *In the graph*



the walk

$$v_2, e_7, v_5, e_8, v_1, e_8, v_5, e_6, v_4, e_5, v_4, e_5, v_4$$

is open. On the other hand, the walk

$$v_4, e_5, v_4, e_3, v_3, e_2, v_2, e_7, v_5, e_6, v_4$$

is closed.

A walk is a *trail* if any edge is traversed at most once. Then, the number of times that the vertex pair u, v can appear as consecutive vertices in a trail is at most the number of parallel edges connecting u and v.

Example. *(Continuing from the previous example) The walk in the graph*

$$v_1, e_8, v_5, e_9, v_1, e_1, v_2, e_7, v_5, e_6, v_4, e_5, v_4, e_4, v_4$$

is a trail.

A trail is a *path* if any vertex is visited at most once except possibly the initial and terminal vertices when they are the same. A closed path is a *circuit*. For simplicity, we will assume in the future that a circuit is not empty, i.e. its length ≥ 1. We identify the paths and circuits with the subgraphs induced by their edges.

Example. *(Continuing from the previous example) The walk*

$$v_2, e_7, v_5, e_6, v_4, e_3, v_3$$

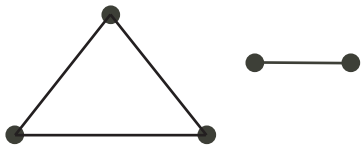
is a path and the walk

$$v_2, e_7, v_5, e_6, v_4, e_3, v_3, e_2, v_2$$

is a circuit.

The walk starting at u and ending at v is called an u–v *walk*. u and v are *connected* if there is a u–v walk in the graph (then there is also a u–v path!). If u and v are connected and v and w are connected, then u and w are also connected, i.e. if there is a u–v walk and a v–w walk, then there is also a u–w walk. A graph is *connected* if all the vertices are connected to each other. (A trivial graph is connected by convention.)

Example. *The graph*



is not connected.

The subgraph G_1 (not a null graph) of the graph G is a *component* of G if

1. G_1 is connected and
2. Either G_1 is trivial (one single isolated vertex of G) or G_1 is not trivial and G_1 is the subgraph induced by those edges of G that have one end vertex in G_1.

Different components of the same graph do not have any common vertices because of the following theorem.

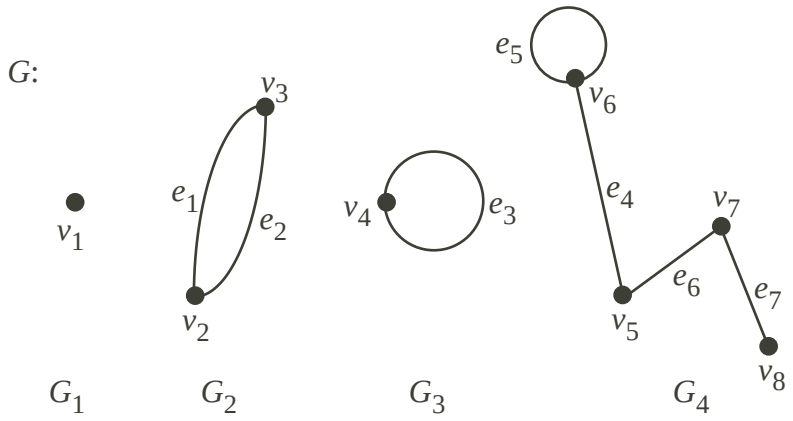
Theorem 1.2. *If the graph G has a vertex v that is connected to a vertex of the component G_1 of G, then v is also a vertex of G_1.*

Proof. If v is connected to vertex v' of G_1, then there is a walk in G

$$v = v_{i_0}, e_{j_1}, v_{i_1}, \ldots, v_{i_{k-1}}, e_{j_k}, v_{i_k} = v'.$$

Since v' is a vertex of G_1, then (condition #2 above) e_{j_k} is an edge of G_1 and $v_{i_{k-1}}$ is a vertex of G_1. We continue this process and see that v is a vertex of G_1. □

Example.



The components of G are G_1, G_2, G_3 and G_4.

Theorem 1.3. *Every vertex of G belongs to exactly one component of G. Similarly, every edge of G belongs to exactly one component of G.*

Proof. We choose a vertex v in G. We do the following as many times as possible starting with $V_1 = \{v\}$:

$(*)$ If v' is a vertex of G such that $v' \notin V_1$ and v' is connected to some vertex of V_1, then $V_1 \leftarrow V_1 \cup \{v'\}$.

Since there is a finite number of vertices in G, the process stops eventually. The last V_1 induces a subgraph G_1 of G that is the component of G containing v. G_1 is connected because its vertices are connected to v so they are also connected to each other. Condition #2 holds because we can not repeat $(*)$. By Theorem 1.2, v does not belong to any other component.

The edges of the graph are incident to the end vertices of the components. □

Theorem 1.3 divides a graph into distinct components. The proof of the theorem gives an algorithm to do that. We have to repeat what we did in the proof as long as we have free vertices that do not belong to any component. Every isolated vertex forms its own component. A connected graph has only one component, namely, itself.

A graph G with n vertices, m edges and k components has the *rank*

$$\rho(G) = n - k.$$

The *nullity* of the graph is

$$\mu(G) = m - n + k.$$

We see that $\rho(G) \geq 0$ and $\rho(G) + \mu(G) = m$. In addition, $\mu(G) \geq 0$ because

Theorem 1.4. $\rho(G) \leq m$

Proof. We will use the second principle of induction (strong induction) for m.

Induction Basis: $m = 0$. The components are trivial and $n = k$.

Induction Hypothesis: The theorem is true for $m < p$. ($p \geq 1$)

Induction Statement: The theorem is true for $m = p$.

Induction Statement Proof: We choose a component G_1 of G which has at least one edge. We label that edge e and the end vertices u and v. We also label G_2 as the subgraph of G and G_1, obtained by removing the edge e from G_1 (but not the vertices u and v). We label G' as the graph obtained by removing the edge e from G (but not the vertices u and v) and let k' be the number of components of G'. We have two cases:

1. G_2 is connected. Then, $k' = k$. We use the Induction Hypothesis on G':

$$n - k = n - k' = \rho(G') \leq m - 1 < m.$$

2. G_2 is not connected. Then there is only one path between u and v:

$$u, e, v$$

and no other path. Thus, there are two components in G_2 and $k' = k + 1$. We use the Induction Hypothesis on G':

$$\rho(G') = n - k' = n - k - 1 \leq m - 1.$$

Hence $n - k \leq m$. □

These kind of combinatorical results have many consequences. For example:

Theorem 1.5. *If G is a connected graph and $k \geq 2$ is the maximum path length, then any two paths in G with length k share at least one common vertex.*

Proof. We only consider the case where the paths are not circuits (Other cases can be proven in a similar way.). Consider two paths of G with length k:

$$v_{i_0}, e_{j_1}, v_{i_1}, e_{j_2}, \ldots, e_{j_k}, v_{i_k} \quad \text{(path } p_1\text{)}$$

and

$$v_{i'_0}, e_{j'_1}, v_{i'_1}, e_{j'_2}, \ldots, e_{j'_k}, v_{i'_k} \quad \text{(path } p_2\text{)}.$$

Let us consider the counter hypothesis: The paths p_1 and p_2 do not share a common vertex. Since G is connected, there exists an v_{i_0}–$v_{i'_k}$ path. We then find the last vertex on this path which is also on p_1 (at least v_{i_0} is on p_1) and we label that vertex v_{i_t}. We find the first vertex of the v_{i_t}–$v_{i'_k}$ path which is also on p_2 (at least $v_{i'_k}$ is on p_2) and we label that vertex $v_{i'_s}$. So we get a v_{i_t}–$v_{i'_s}$ path

$$v_{i_t}, e_{j''_1}, \ldots, e_{j''_\ell}, v_{i'_s}.$$

The situation is as follows:

$$\begin{gathered} v_{i_0}, e_{j_1}, v_{i_1}, \ldots, v_{i_t}, e_{j_{t+1}}, \ldots, e_{j_k}, v_{i_k} \\ e_{j''_1} \\ \vdots \\ e_{j''_\ell} \\ v_{i'_0}, e_{j'_1}, v_{i'_1}, \ldots, v_{i'_s}, e_{j'_{s+1}}, \ldots, e_{j'_k}, v_{i'_k} \end{gathered}$$

From here we get two paths: v_{i_0}–$v_{i'_k}$ path and $v_{i'_0}$–v_{i_k} path. The two cases are:

- $t \geq s$: Now the length of the v_{i_0}–$v_{i'_k}$ path is $\geq k + 1$. $\surd$[1]
- $t < s$: Now the length of the $v_{i'_0}$–v_{i_k} path is $\geq k + 1$. $\surd$ □

A graph is *circuitless* if it does not have any circuit in it.

Theorem 1.6. *A graph is circuitless exactly when there are no loops and there is at most one path between any two given vertices.*

Proof. First let us assume G is circuitless. Then, there are no loops in G. Let us assume the counter hypothesis: There are two different paths between distinct vertices u and v in G:

$$u = v_{i_0}, e_{j_1}, v_{i_1}, e_{j_2}, \ldots, e_{j_k}, v_{i_k} = v \quad \text{(path } p_1\text{)}$$

and

$$u = v_{i'_0}, e_{j'_1}, v_{i'_1}, e_{j'_2}, \ldots, e_{j'_\ell}, v_{i'_\ell} = v \quad \text{(path } p_2\text{)}$$

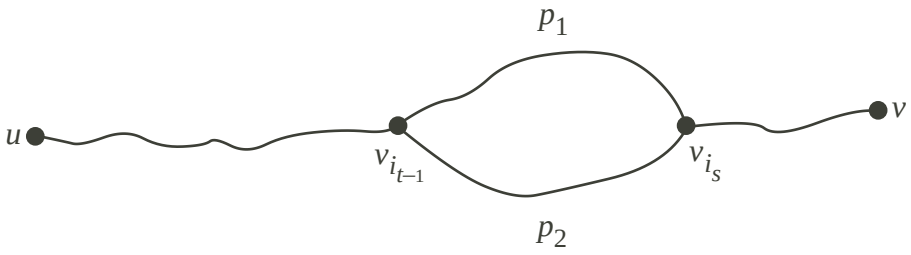
(here we have $i_0 = i'_0$ and $i_k = i'_\ell$), where $k \geq \ell$. We choose the smallest index t such that

$$v_{i_t} \neq v_{i'_t}.$$

There is such a t because otherwise

[1] From now on, the symbol $\surd$ means contradiction. If we get a contradiction by proceeding from the assumptions, the hypothesis must be wrong.

1. $k > \ell$ and $v_{i_k} = v = v_{i'_\ell} = v_{i_\ell}$ ($\surd$) or
2. $k = \ell$ and $v_{i_0} = v_{i'_0}, \ldots, v_{i_\ell} = v_{i'_\ell}$. Then, there would be two parallel edges between two consecutive vertices in the path. That would imply the existence of a circuit between two vertices in G. $\surd$



We choose the smallest index s such that $s \geq t$ and v_{i_s} is in the path p_2 (at least v_{i_k} is in p_2). We choose an index r such that $r \geq t$ and $v_{i'_r} = v_{i_s}$ (it exists because p_1 is a path). Then,

$$v_{i_{t-1}}, e_{j_t}, \ldots, e_{j_s}, v_{i_s}(= v_{i'_r}), e_{j'_r}, \ldots, e_{j'_t}, v_{i'_{t-1}}(= v_{i_{t-1}})$$

is a circuit. $\surd$ (Verify the case $t = s = r$.)

Let us prove the reverse implication. If the graph does not have any loops and no two distinct vertices have two different paths between them, then there is no circuit. For example, if

$$v_{i_0}, e_{j_1}, v_{i_1}, e_{j_2}, \ldots, e_{j_k}, v_{i_k} = v_{i_0}$$

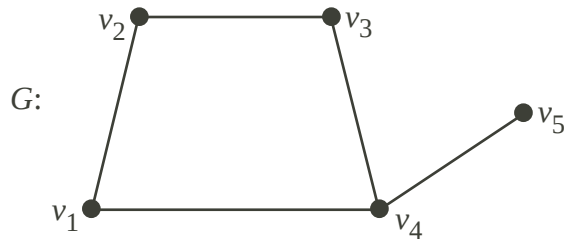
is a circuit, then either $k = 1$ and e_{j_1} is a loop ($\surd$), or $k \geq 2$ and the two vertices v_{i_0} and v_{i_1} are connected by two distinct paths

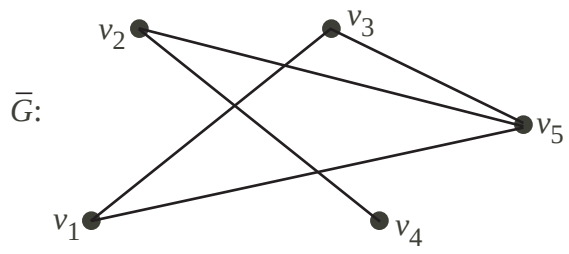
$$v_{i_0}, e_{j_1}, v_{i_1} \quad \text{and} \quad v_{i_1}, e_{j_2}, \ldots, e_{j_k}, v_{i_k} = v_{i_0} \quad (\surd).$$ □

1.3 Graph Operations

The *complement* of the simple graph $G = (V, E)$ is the simple graph $\overline{G} = (V, \overline{E})$, where the edges in $\overline{E}$ are exactly the edges not in G.

Example.

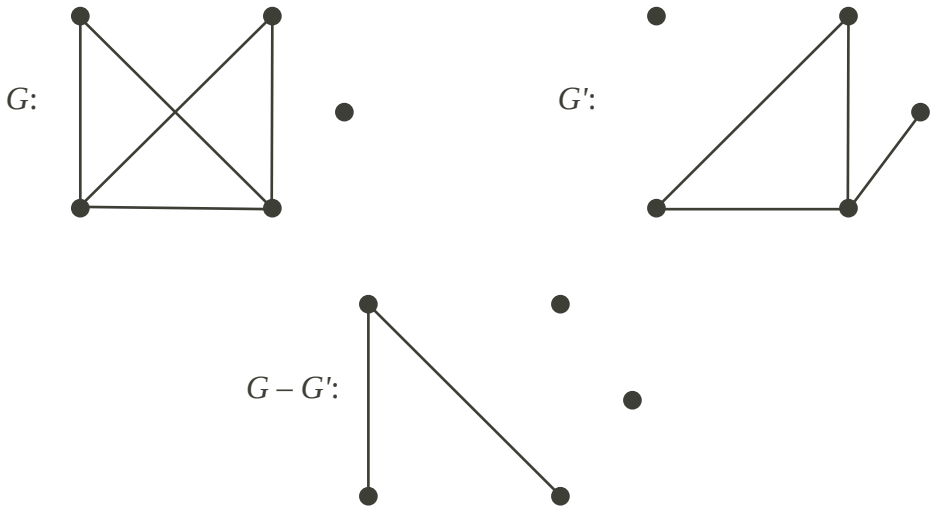




Example. *The complement of the complete graph K_n is the empty graph with n vertices.*

Obviously, $\overline{\overline{G}} = G$. If the graphs $G = (V, E)$ and $G' = (V', E')$ are simple and $V' \subseteq V$, then the *difference* graph is $G - G' = (V, E'')$, where E'' are those edges from G that are not in G' (simple graph).

Example.



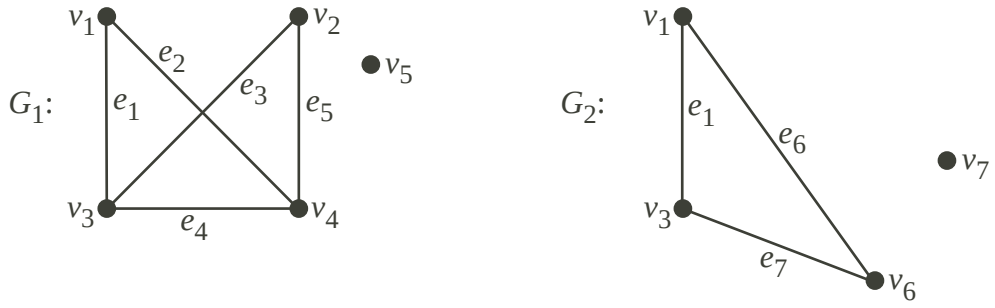
Here are some binary operations between two simple graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$:

- The *union* is $G_1 \cup G_2 = (V_1 \cup V_2, E_1 \cup E_2)$ (simple graph).
- The *intersection* is $G_1 \cap G_2 = (V_1 \cap V_2, E_1 \cap E_2)$ (simple graph).
- The *ring sum* $G_1 \oplus G_2$ is the subgraph of $G_1 \cup G_2$ induced by the edge set $E_1 \oplus E_2$ (simple graph). *Note!* The set operation $\oplus$ is the *symmetric difference*, i.e.

$$E_1 \oplus E_2 = (E_1 - E_2) \cup (E_2 - E_1).$$

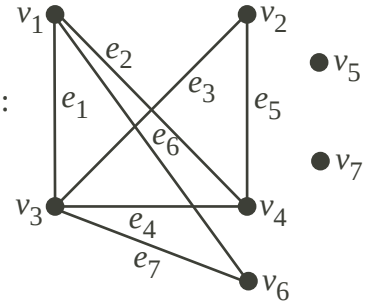
Since the ring sum is a subgraph induced by an edge set, there are no isolated vertices. All three operations are commutative and associative.

Example. *For the graphs*

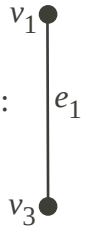


we have

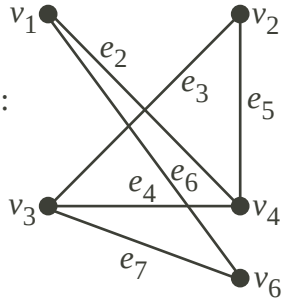
$G_1 \cup G_2$:



$G_1 \cap G_2$:



$G_1 \oplus G_2$:



Remark. *The operations $\cup$, $\cap$ and $\oplus$ can also be defined for more general graphs other than simple graphs. Naturally, we have to "keep track" of the multiplicity of the edges:*

- $\cup$: *The multiplicity of an edge in $G_1 \cup G_2$ is the larger of its multiplicities in G_1 and G_2.*
- $\cap$: *The multiplicity of an edge in $G_1 \cap G_2$ is the smaller of its multiplicities in G_1 and G_2.*
- $\oplus$: *The multiplicity of an edge in $G_1 \oplus G_2$ is $|m_1 - m_2|$, where m_1 is its multiplicity in G_1 and m_2 is its multiplicity in G_2.*

(We assume zero multiplicity for the absence of an edge.) In addition, we can generalize the difference operation for all kinds of graphs if we take account of the multiplicity. The multiplicity of the edge e in the difference $G - G'$ is

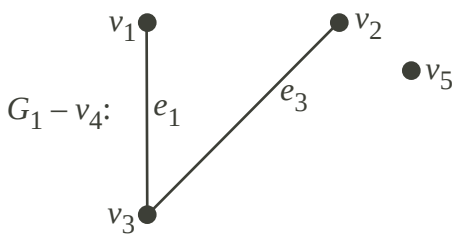
$$m_1 \dot{-} m_2 = \begin{cases} m_1 - m_2, \text{ if } m_1 \geq m_2 \\ 0, \text{ if } m_1 < m_2 \end{cases} \qquad \text{(also known as the proper difference)},$$

where m_1 and m_2 are the multiplicities of e in G_1 and G_2, respectively.

If v is a vertex of the graph $G = (V, E)$, then $G - v$ is the subgraph of G induced by the vertex set $V - \{v\}$. We call this operation the *removal of a vertex*.

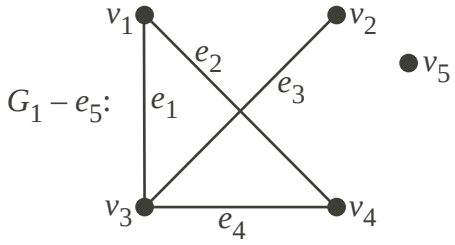
Example. *(Continuing from the previous example)*

$G_1 - v_4$:



Similarly, if e is an edge of the graph $G = (V, E)$, then $G - e$ is graph (V, E'), where E' is obtained by removing e from E. This operation is known as *removal of an edge*. We remark that we are not talking about removing an edge as in Set Theory, because the edge can have nonunit multiplicity and we only remove the edge once.

Example. *(Continuing from the previous example)*



If u and v are two distinct vertices of the graph $G = (V, E)$, then we can *short-circuit* the two vertices u and v and obtain the graph (V', E'), where

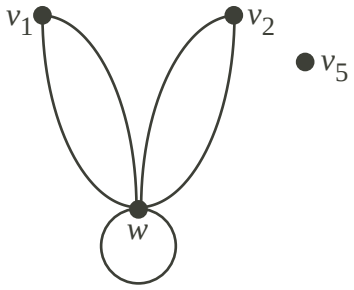
$$V' = (V - \{u, v\}) \cup \{w\} \quad (w \notin V \text{ is the "new" vertex})$$

and

$$E' = (E - \{(v', u), (v', v) \mid v' \in V\}) \cup \{(v', w) \mid (v', u) \in E \text{ or } (v', v) \in E\} \\ \cup \{(w, w) \mid (u, u) \in E \text{ or } (v, v) \in E\}$$

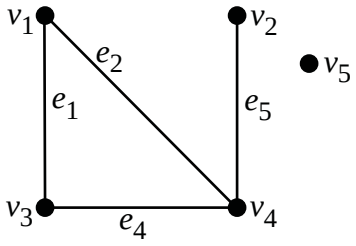
(Recall that the pair of vertices corresponding to an edge is not ordered). *Note!* We have to maintain the multiplicity of the edges. In particular, the edge (u, v) becomes a loop.

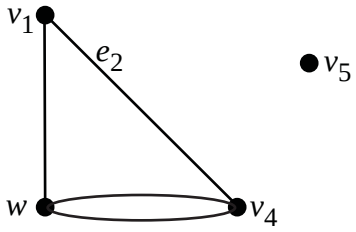
Example. *(Continuing from the previous example) Short-circuit v_3 and v_4 in the graph G_1:*



In the graph $G = (V, E)$, *contracting* the edge $e = (u, v)$ (not a loop) means the operation in which we first remove e and then short-circuit u and v. (Contracting a loop simply removes that loop.)

Example. *(Continuing from the previous example) We contract the edge e_3 in G_1 by first removing e_3 and then short-circuiting v_2 and v_3.*



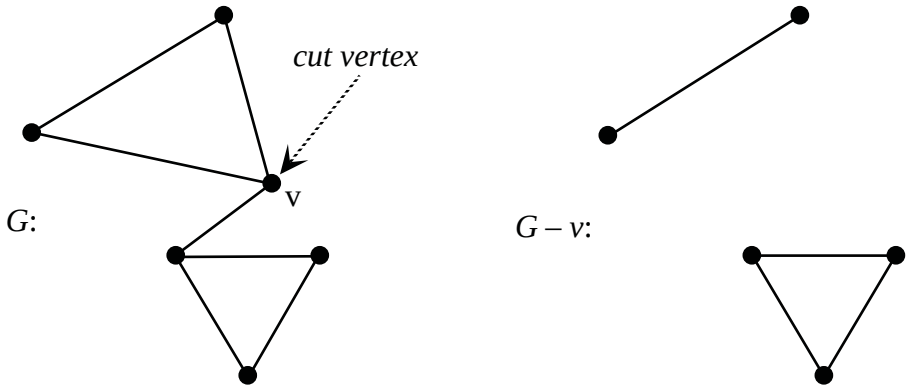


Remark. *If we restrict short-circuiting and contracting to simple graphs, then we remove loops and all but one of the parallel edges between end vertices from the results.*

1.4 Cuts

A vertex v of a graph G is a *cut vertex* or an *articulation vertex* of G if the graph $G - v$ consists of a greater number of components than G.

Example. *v is a cut vertex of the graph below:*

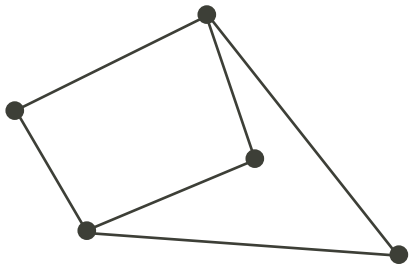


(*Note!* Generally, the only vertex of a trivial graph is not a cut vertex, neither is an isolated vertex.)

A graph is *separable* if it is not connected or if there exists at least one cut vertex in the graph. Otherwise, the graph is *nonseparable*.

Example. *The graph G in the previous example is separable.*

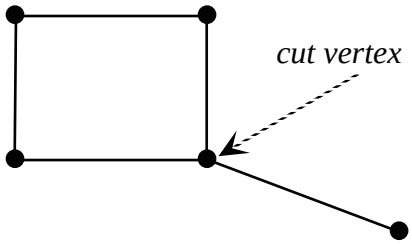
Example. *The graph below is nonseparable.*

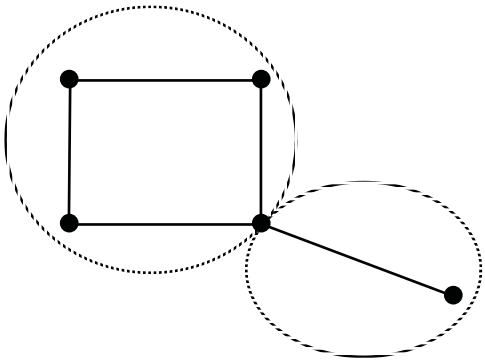


A *block* of the graph G is a subgraph G_1 of G (not a null graph) such that

- G_1 is nonseparable, and
- if G_2 is any other subgraph of G, then $G_1 \cup G_2 = G_1$ or $G_1 \cup G_2$ is separable (think about that!).

Example. *The graphs below are separable:*





Theorem 1.7. *The vertex v is a cut vertex of the connected graph G if and only if there exist two vertices u and w in the graph G such that*

(i) $v \neq u$, $v \neq w$ *and* $u \neq w$, *but*

(ii) v *is on every* u–w *path.*

Proof. First, let us consider the case that v is a cut-vertex of G. Then, $G - v$ is not connected and there are at least two components $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$. We choose $u \in V_1$ and $w \in V_2$. The u–w path is in G because it is connected. If v is not on this path, then the path is also in $G - v$ ($\surd$). The same reasoning can be used for all the u–w paths in G.

If v is in every u–w path, then the vertices u and w are not connected in $G - v$. □

Theorem 1.8. *A nontrivial graph has at least two vertices which are not cut vertices.*

Proof. We will use induction for the graph G with n vertices.

Induction Basis: The case $n = 2$ is obviously true.

Induction Hypothesis: The theorem is true for $n \leq k$. ($k \geq 2$)

Induction Statement: The theorem is true for $n = k + 1$.

Induction Statement Proof: If there are no cut vertices in G, then it is obvious. Otherwise, we consider a cut vertex v of G. Let $G_1, \ldots, G_m$ be the components of $G - v$ (so $m \geq 2$). Every component of G_i falls into one of the two cases:

1. G_i is trivial so the only vertex of G_i is a pendant vertex or an isolated vertex of G but it is not a cut vertex of G.

2. G_i is not trivial. The Induction Hypothesis tells us that there exist two vertices u and w in G_i which are not cut vertices of G_i. If v and u (respectively v and w) are not adjacent in G, then u (respectively w) is not a cut vertex in G. If both v and u as well as u and w are adjacent in G, then u and w can not be cut vertices of G. □

A *cut set* of the connected graph $G = (V, E)$ is an edge set $F \subseteq E$ such that

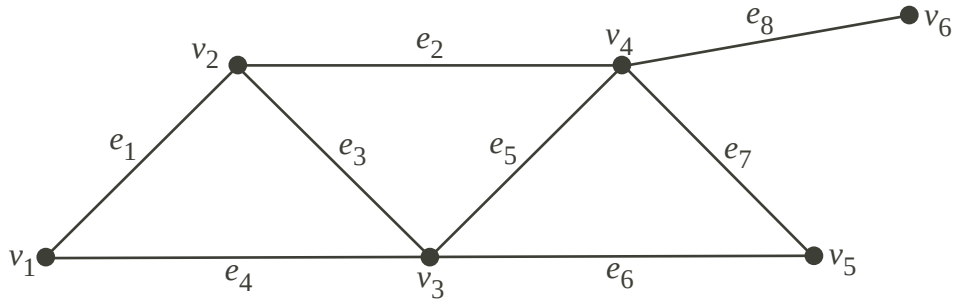
1. $G - F$ (remove the edges of F one by one) is not connected, and

2. $G - H$ is connected whenever $H \subset F$.

Theorem 1.9. *If F is a cut set of the connected graph G, then $G - F$ has two components.*

Proof. Let $F = \{e_1, \ldots, e_k\}$. The graph $G - \{e_1, \ldots, e_{k-1}\}$ is connected (and so is G if $k = 1$) by condition #2. When we remove the edges from the connected graph, we get at most two components. □

Example. *In the graph*



$\{e_1, e_4\}$, $\{e_6, e_7\}$, $\{e_1, e_2, e_3\}$, $\{e_8\}$, $\{e_3, e_4, e_5, e_6\}$, $\{e_2, e_5, e_7\}$, $\{e_2, e_5, e_6\}$ *and* $\{e_2, e_3, e_4\}$ *are cut sets. Are there other cut sets?*

In a graph $G = (V, E)$, a pair of subsets V_1 and V_2 of V satisfying

$$V = V_1 \cup V_2\ , \quad V_1 \cap V_2 = \emptyset\ , \quad V_1 \neq \emptyset\ , \quad V_2 \neq \emptyset,$$

is called a *cut* (or a *partition*) of G, denoted $\langle V_1, V_2 \rangle$. Usually, the cuts $\langle V_1, V_2 \rangle$ and $\langle V_2, V_1 \rangle$ are considered to be the same.

Example. *(Continuing from the previous example)* $\langle\{v_1, v_2, v_3\}, \{v_4, v_5, v_6\}\rangle$ *is a cut.*

We can also think of a cut as an edge set:

$$\text{cut } \langle V_1, V_2\rangle = \{\text{those edges with one end vertex in } V_1 \text{ and the other end vertex in } V_2\}.$$

(*Note!* This edge set does not define V_1 and V_2 uniquely so we can not use this for the definition of a cut.)

Using the previous definitions and concepts, we can easily prove the following:

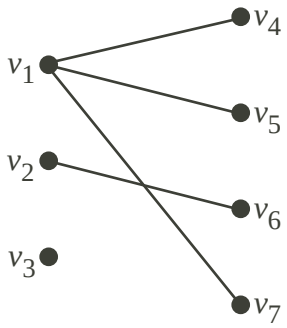
1. The cut $\langle V_1, V_2\rangle$ of a connected graph G (considered as an edge set) is a cut set if and only if the subgraphs induced by V_1 and V_2 are connected, i.e. $G - \langle V_1, V_2\rangle$ has two components.

2. If F is a cut set of the connected graph G and V_1 and V_2 are the vertex sets of the two components of $G - F$, then $\langle V_1, V_2\rangle$ is a cut and $F = \langle V_1, V_2\rangle$.

3. If v is a vertex of a connected (nontrivial) graph $G = (V, E)$, then $\langle\{v\}, V - \{v\}\rangle$ is a cut of G. It follows that the cut is a cut set if the subgraph (i.e. $G - v$) induced by $V - \{v\}$ is connected, i.e. if v is *not* a cut vertex.

If there exists a cut $\langle V_1, V_2\rangle$ for the graph $G = (V, E)$ so that $E = \langle V_1, V_2\rangle$, i.e. the cut (considered as an edge set) includes every edge, then the graph G is *bipartite*.

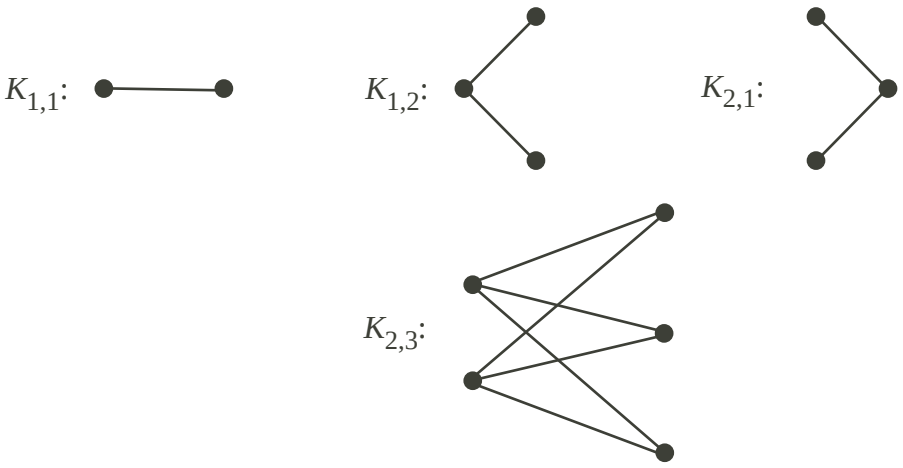
Example. *The graph*



is bipartite. $V_1 = \{v_1, v_2, v_3\}$ *and* $V_2 = \{v_4, v_5, v_6, v_7\}$.

A simple bipartite graph is called a *complete bipartite graph* if we can not possibly add any more edges to the edge set $\langle V_1, V_2\rangle$, i.e. the graph contains exactly all edges that have one end vertex in V_1 and the other end vertex in V_2. If there are n vertices in V_1 and m vertices in V_2, we denote it as $K_{n,m}$ (cf. complete graph).

Example.



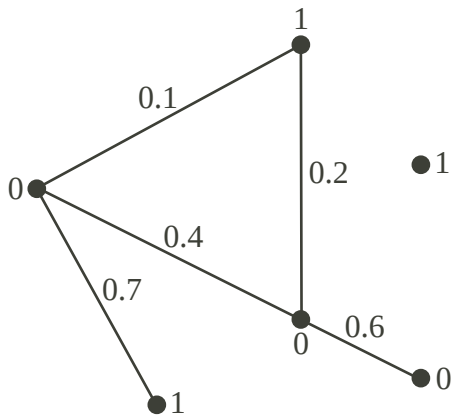
(Usually $K_{n,m}$ and $K_{m,n}$ are considered to be the same.)

1.5 Labeled Graphs and Isomorphism

By a *labeling of the vertices* of the graph $G = (V, E)$, we mean a mapping $\alpha : V \to A$, where A is called the *label* set. Similarly, a *labeling of the edges* is a mapping $\beta : E \to B$, where B is the label set. Often, these labels are numbers. Then, we call them *weights* of vertices and edges. In a weighted graph, the weight of a path is the sum of the weights of the edges traversed.

The labeling of the vertices (respectively edges) is *injective* if distinct vertices (respectively edges) have distinct labels. An injective labeling is *bijective* if there are as many labels in A (respectively in B) as the number of vertices (respectively edges).

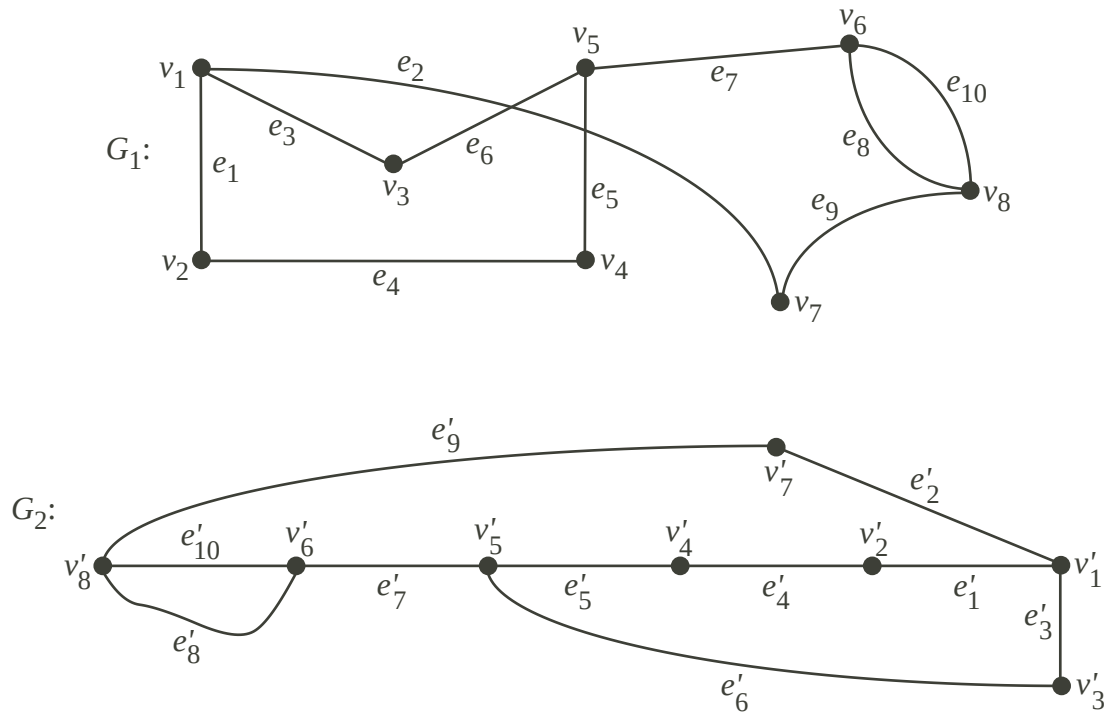
Example. *If $A = \{0, 1\}$ and $B = \mathbb{R}$, then in the graph,*



the labeling of the edges (weights) is injective but not the labeling of the vertices.

The two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are *isomorphic* if labeling the vertices of G_1 bijectively with the elements of V_2 gives G_2. (*Note!* We have to maintain the multiplicity of the edges.)

Example. *The graphs G_1 and G_2 are isomorphic and the vertex labeling $v_i \mapsto v'_i$ and edge labeling $e_j \mapsto e'_j$ define the isomorphism.*



Determining whether or not two graphs are isomorphic is a well researched problem. It differs significantly from other problems in graph theory and network analysis. In addition, it has a lot to do with group theory in algebra. The problem is important in the theory of Computational Complexity. For example, refer to KÖBLER, J. & SCHÖNING, U. & TORÁN, J.: *The Graph Isomorphism Problem. Its Structural Complexity.* Birkhäuser (1993).

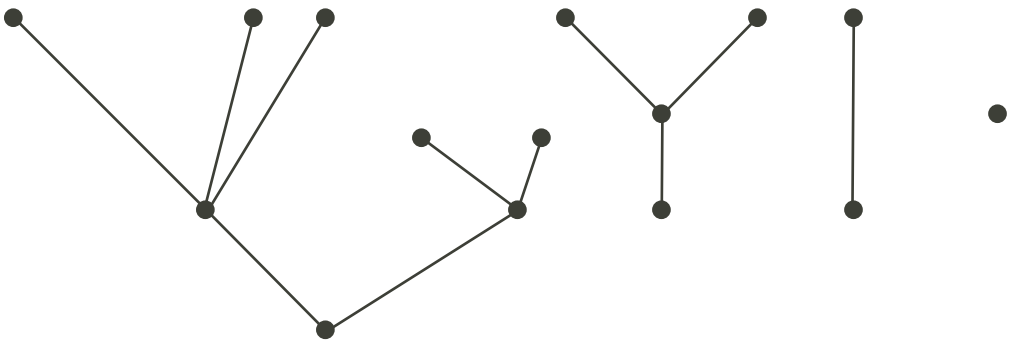
Chapter 2

Trees

2.1 Trees and Forests

A *forest* is a circuitless graph. A *tree* is a connected forest. A *subforest* is a subgraph of a forest. A connected subgraph of a tree is a *subtree*. Generally speaking, a subforest (respectively subtree) of a graph is its subgraph, which is also a forest (respectively tree).

Example. *Four trees which together form a forest:*



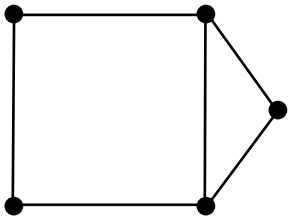
A *spanning tree* of a connected graph is a subtree that includes all the vertices of that graph. If T is a spanning tree of the graph G, then

$$G - T =_{\text{def.}} T^*$$

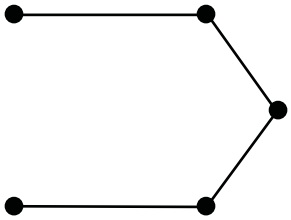
is the *cospanning tree*.

Example.

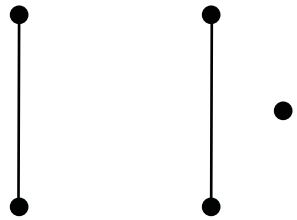
G:



spanning tree:



cospanning tree



The edges of a spanning tree are called *branches* and the edges of the corresponding cospanning tree are called *links* or *chords*.

Theorem 2.1. *If the graph G has n vertices and m edges, then the following statements are equivalent:*

(i) *G is a tree.*

(ii) *There is exactly one path between any two vertices in G and G has no loops.*

(iii) *G is connected and $m = n - 1$.*

(iv) *G is circuitless and $m = n - 1$.*

(v) *G is circuitless and if we add any new edge to G, then we will get one and only one circuit.*

Proof. (i)⇒(ii): If G is a tree, then it is connected and circuitless. Thus, there are no loops in G. There exists a path between any two vertices of G. By Theorem 1.6, we know that there is only one such path.

(ii)⇒(iii): G is connected. Let us use induction on m.

Induction Basis: $m = 0$, G is trivial and the statement is obvious.

Induction Hypothesis: $m = n - 1$ when $m \leq \ell$. ($\ell \geq 0$)

Induction Statement: $m = n - 1$ when $m = \ell + 1$.

Induction Statement Proof: Let e be an edge in G. Then $G - e$ has ℓ edges. If $G - e$ is connected, then there exist two different paths between the end vertices of e so (ii) is false. Therefore, $G - e$ has two components G_1 and G_2. Let there be n_1 vertices and m_1 edges in G_1. Similarly, let there be n_2 vertices and m_2 vertices in G_2. Then,

$$n = n_1 + n_2 \quad \text{and} \quad m = m_1 + m_2 + 1.$$

The Induction Hypothesis states that

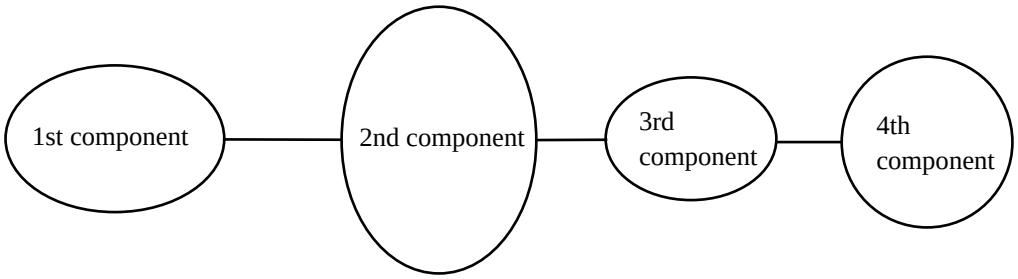
$$m_1 = n_1 - 1 \quad \text{and} \quad m_2 = n_2 - 1,$$

so $m = n_1 + n_2 - 1 = n - 1$.

(iii)⇒(iv): Consider the counter hypothesis: There is a circuit in G. Let e be some edge in that circuit. Thus, there are n vertices and $n - 2$ edges in the connected graph $G - e$. $\surd$ [1]

(iv)⇒(v): If G is circuitless, then there is at most one path between any two vertices (Theorem 1.6). If G has more than one component, then we will not get a circuit when we draw an edge between two different components. By adding edges, we can connect components without creating circuits:

[1] In a connected graph with n vertices, there are at least $n - 1$ edges. (Theorem 1.4)



If we add $k(\geq 1)$ edges, then (because (i)⇒(iii))

$$m + k = n - 1 \quad (\surd \text{ because } m = n - 1).$$

So G is connected. When we add an edge between vertices that are not adjacent, we get only one circuit. Otherwise, we can remove an edge from one circuit so that other circuits will not be affected and the graph stays connected, in contradiction to (iii)⇒(iv). Similarly, if we add a parallel edge or a loop, we get exactly one circuit.

(v)⇒(i): Consider the counter hypothesis: G is not a tree, i.e. it is not connected. When we add edges as we did previously, we do not create any circuits (see figure). $\surd$ □

Since spanning trees are trees, Theorem 2.1 is also true for spanning trees.

Theorem 2.2. *A connected graph has at least one spanning tree.*

Proof. Consider the connected graph G with n vertices and m edges. If $m = n - 1$, then G is a tree. Since G is connected, $m \geq n - 1$ (Theorem 1.4). We still have to consider the case $m \geq n$, where there is a circuit in G. We remove an edge e from that circuit. $G - e$ is now connected. We repeat until there are $n - 1$ edges. Then, we are left with a tree. □

Remark. *We can get a spanning tree of a connected graph by starting from an arbitrary subforest M (as we did previously). Since there is no circuit whose edges are all in M, we can remove those edges from the circuit which are not in M.*

By Theorem 2.1, the subgraph G_1 of G with n vertices is a spanning tree of G (thus G is connected) if any three of the following four conditions hold:

1. G_1 has n vertices.
2. G_1 is connected.
3. G_1 has $n - 1$ edges.
4. G_1 is circuitless.

Actually, conditions #3 and #4 are enough to guarantee that G_1 is a spanning tree. If conditions #3 and #4 hold but G_1 is not connected, then the components of G_1 are trees and the number of edges in G_1 would be

$$\text{number of vertices} - \text{number of components} < n - 1 \quad (\surd).$$

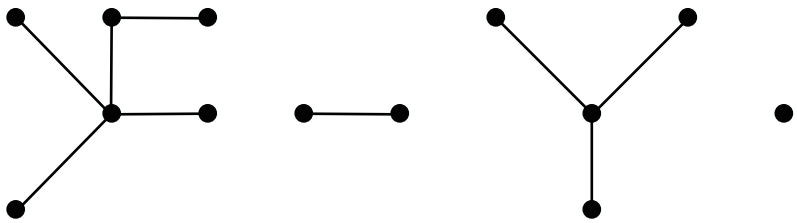
Theorem 2.3. *If a tree is not trivial, then there are at least two pendant vertices.*

Proof. If a tree has $n(\geq 2)$ vertices, then the sum of the degrees is $2(n-1)$. If every vertex has a degree ≥ 2, then the sum will be $\geq 2n$ ($\surd$). On the other hand, if all but one vertex have degree ≥ 2, then the sum would be $\geq 1 + 2(n-1) = 2n - 1$ ($\surd$). (This also follows from Theorem 1.8 because a cut vertex of a tree is not a pendant vertex!) □

A forest with k components is sometimes called a k-*tree*. (So a 1-tree is a tree.)

Example.

4-tree:

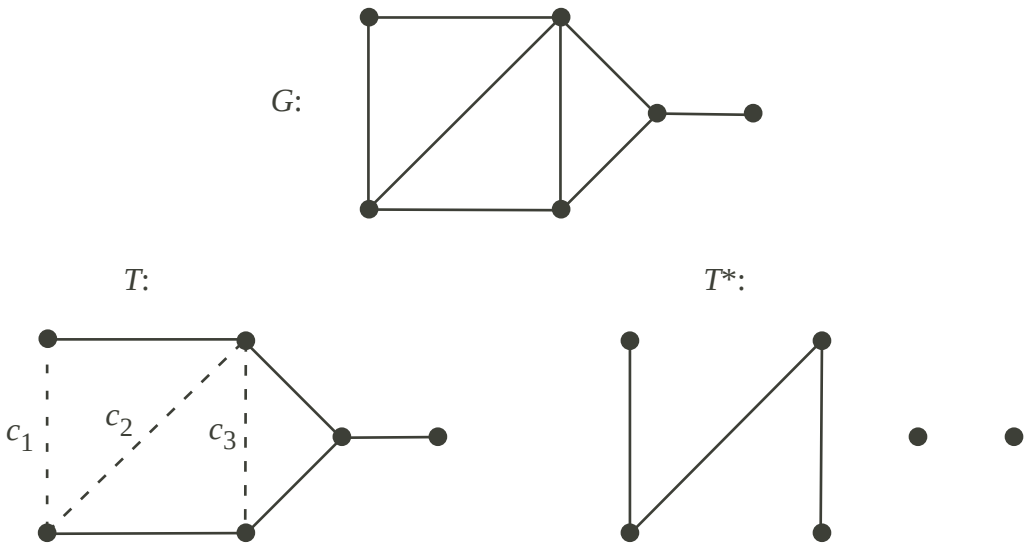


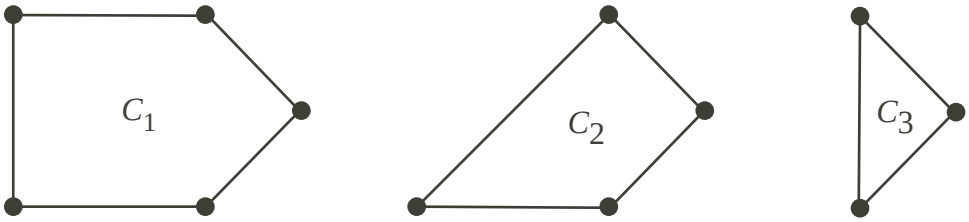
We use Theorem 2.1 to see that a graph with k components has a *spanning* k*-tree*, also known as a *spanning forest*, which has k components.

2.2 (Fundamental) Circuits and (Fundamental) Cut Sets

If the branches of the spanning tree T of a connected graph G are $b_1, \ldots, b_{n-1}$ and the corresponding links of the cospanning tree T^* are $c_1, \ldots, c_{m-n+1}$, then there exists one and only one circuit C_i in $T + c_i$ (which is the subgraph of G induced by the branches of T and c_i) (Theorem 2.1). We call this circuit a *fundamental circuit*. Every spanning tree defines $m - n + 1$ fundamental circuits $C_1, \ldots, C_{m-n+1}$, which together form a *fundamental set of circuits*. Every fundamental circuit has exactly one link which is not in any other fundamental circuit in the fundamental set of circuits. Therefore, we can not write any fundamental circuit as a ring sum of other fundamental circuits in the same set. In other words, the fundamental set of circuits is linearly independent under the ring sum operation.

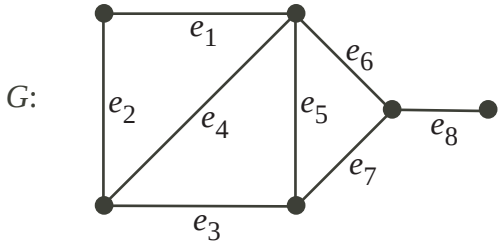
Example.



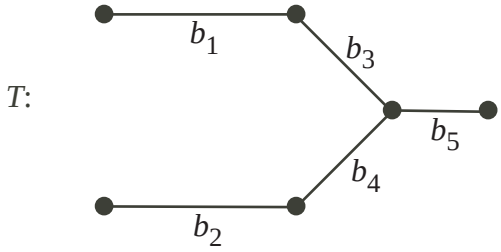


The graph $T - b_i$ has two components T_1 and T_2. The corresponding vertex sets are V_1 and V_2. Then, $\langle V_1, V_2 \rangle$ is a cut of G. It is also a cut set of G if we treat it as an edge set because $G - \langle V_1, V_2 \rangle$ has two components (result #1 p. 17). Thus, every branch b_i of T has a corresponding cut set I_i. The cut sets $I_1, \ldots, I_{n-1}$ are also known as *fundamental cut sets* and they form a *fundamental set of cut sets*. Every fundamental cut set includes exactly one branch of T and every branch of T belongs to exactly one fundamental cut set. Therefore, every spanning tree defines a unique fundamental set of cut sets for G.

Example. *(Continuing from the previous example) The graph*



has the spanning tree



that defines these fundamental cut sets:

$$b_1 : \{e_1, e_2\} \qquad b_2 : \{e_2, e_3, e_4\} \quad b_3 : \{e_2, e_4, e_5, e_6\}$$
$$b_4 : \{e_2, e_4, e_5, e_7\} \quad b_5 : \{e_8\}$$

Next, we consider some properties of circuits and cut sets:

(a) The cut set of a connected graph G includes at least one branch from every spanning tree of G. (Counter hypothesis: Some cut set F of G does not include any branches of a spanning tree T. Then, T is a subgraph of $G - F$ and $G - F$ is connected. $\surd$)

(b) Every circuit of the connected graph G includes at least one link from every cospanning tree of G. (Counter hypothesis: Some circuit C of G does not include any link of a cospanning tree T^*. Then, $T = G - T^*$ has a circuit and T is not a tree. $\surd$)

Theorem 2.4. *The edge set F of the connected graph G is a cut set of G if and only if*

(i) *F includes at least one branch from every spanning tree of G, and*

(ii) *if $H \subset F$, then there is a spanning tree none of whose branches is in H.*

Proof. Let us first consider the case where F is a cut set. Then, (i) is true (previous proposition (a)). If $H \subset F$ then $G - H$ is connected and has a spanning tree T. This T is also a spanning tree of G. Hence, (ii) is true.

Let us next consider the case where both (i) and (ii) are true. Then $G - F$ is disconnected. If $H \subset F$ there is a spanning tree T none of whose branches is in H. Thus T is a subgraph of $G - H$ and $G - H$ is connected. Hence, F is a cut set. □

Similarly:

Theorem 2.5. *The subgraph C of the connected graph G is a circuit if and only if*

(i) *C includes at least one link from every cospanning tree of G, and*

(ii) *if D is a subgraph of C and $D \neq C$, then there exists a cospanning tree none of whose links is in D.*

Proof. Let us first consider the case where C is a circuit. Then, C includes at least one link from every cospanning tree (property (b) above) so (i) is true. If D is a proper subgraph of C, it obviously does not contain circuits, i.e. it is a forest. We can then supplement D so that it is a spanning tree of G (see remark on p.22), i.e. some spanning tree T of G includes D and D does not include any link of T^*. Thus, (ii) is true.

Now we consider the case where (i) and (ii) are both true. Then, there has to be at least one circuit in C because C is otherwise a forest and we can supplement it so that it is a spanning tree of G (see remark on p.22). We take a circuit C' in C. Since (ii) is true, $C' \neq C$ is not true, because C' is a circuit and it includes a link from every cospanning tree (see property (b) above). Therefore, $C = C'$ is a circuit. □

Theorem 2.6. *A circuit and a cut set of a connected graph have an even number of common edges.*

Proof. We choose a circuit C and a cut set F of the connected graph G. $G - F$ has two components $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$. If C is a subgraph of G_1 or G_2, then the theorem is obvious because they have no common edges. Let us assume that C and F have common edges. We traverse around a circuit by starting at some vertex v of G_1. Since we come back to v, there has to be an even number of edges of the cut $\langle V_1, V_2 \rangle$ in C. □

The reader is advised to read the following several times:

Theorem 2.7. *A fundamental circuit corresponding to link c of the cospanning tree T^* of a connected graph is formed exactly by those branches of T whose corresponding fundamental cut set includes c.*

Proof. There exists a fundamental circuit C that corresponds to link c of T^*. The other edges $b_1, \ldots, b_k$ of C are branches of T. We denote I_i as the fundamental cut set that corresponds to branch b_i. Then, b_i is the only branch of T which is in both C and I_i. On the other hand, c is the only link of T^* in C. By Theorem 2.6, we know that the common edges of C and I_i are b_i and c, in other words, c is an edge of I_i. Then, we show that there is no c in the fundamental cut sets $I_{k+1}, \ldots, I_{n-1}$ that correspond to the branches $b_{k+1}, \ldots, b_{n-1}$ of T. For instance, if c were in I_{k+1}, then the fundamental cut set I_{k+1} and the circuit C would have exactly one common edge. ($\surd$). So c is only in the fundamental cut sets $I_1, \ldots, I_k$. □

The following is the corresponding theorem for fundamental cut sets:

Theorem 2.8. *The fundamental cut set corresponding to branch b of the spanning tree T of a connected graph consists exactly of those links of T^* whose corresponding fundamental circuit includes b.*

Proof. Let I be a fundamental cut set that corresponds to the branch b of T. Other edges $c_1, \ldots, c_k$ of I are links of T^*. Let C_i denote the fundamental circuit that corresponds to c_i. Then, c_i is the only link of T^* in both I and C_i. On the other hand, b is the only branch of T in I. By Theorem 2.6, the common edges of I and C_i are b and c_i, in other words, b is an edge of C_i. Then, we show that the fundamental circuits $C_{k+1}, \ldots, C_{m-n+1}$ corresponding to the links $c_{k+1}, \ldots, c_{m-n+1}$ do not include b. For example, if b were in C_{k+1}, then the fundamental circuit C_{k+1} and the cut set I would have exactly one common edge ($\surd$). Hence, the branch b is only in fundamental circuits $C_1, \ldots, C_k$. □

From the results, we can see the duality between cut sets and circuits of a graph: The theorems for cut sets can generally be converted to dual theorems for circuits and vice versa. Usually, we just need to change some of the key terminologies to their duals in the theorems and proofs. In particular, we take advantage of this dualism for dealing with matroids (see Chapter 7).

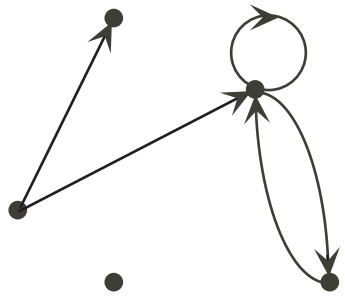
Chapter 3

Directed Graphs

3.1 Definition

Intuitively, a *directed graph* or *digraph* is formed by vertices connected by *directed edges* or *arcs*.[1]

Example.



Formally, a digraph is a pair (V, E), where V is the vertex set and E is the set of vertex pairs as in "usual" graphs. The difference is that now the elements of E are ordered pairs: the arc from vertex u to vertex v is written as (u, v) and the other pair (v, u) is the opposite direction arc. We also have to keep track of the multiplicity of the arc (direction of a loop is irrelevant). We can pretty much use the same notions and results for digraphs from Chapter 1. However:

1. Vertex u is the *initial vertex* and vertex v is the *terminal vertex* of the arc (u, v). We also say that the arc is *incident out* of u and *incident into* v.

2. The *out-degree* of the vertex v is the number of arcs out of it (denoted $d^+(v)$) and the *in-degree* of v is the number of arcs going into it (denoted $d^-(v)$).

3. In the *directed walk* (*trail, path or circuit*),

$$v_{i_0}, e_{j_1}, v_{i_1}, e_{j_2}, \ldots, e_{j_k}, v_{i_k}$$

   v_{i_ℓ} is the initial vertex and $v_{i_{\ell-1}}$ is the terminal vertex of the arc e_{j_ℓ}. .

4. When we treat the graph (V, E) as a usual undirected graph, it is the *underlying undirected graph* of the digraph $G = (V, E)$, denoted G_{u}.

[1] This not a standard terminology. We will however call directed edges arcs in the sequel.

5. Digraph G is *connected* if G_{u} is connected. The *components* of G are the directed subgraphs of G that correspond to the components of G_{u}. The vertices of G are connected if they are connected in G_{u}. Other notions for undirected graphs can be used for digraphs as well by dealing with the underlying undirected graph.

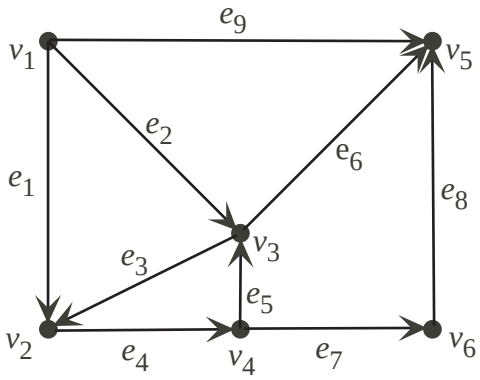
6. Vertices u and v are *strongly connected* if there is a directed u–v path and also a directed v–u path in G.

7. Digraph G is *strongly connected* if every pair of vertices is strongly connected. By convention, the trivial graph is strongly connected.

8. A *strongly connected component* H of the digraph G is a directed subgraph of G (not a null graph) such that H is strongly connected, but if we add any vertices or arcs to it, then it is not strongly connected anymore.

Every vertex of the digraph G belongs to one strongly connected component of G (compare to Theorem 1.3). However, an arc does not necessarily belong to any strongly connected component of G.

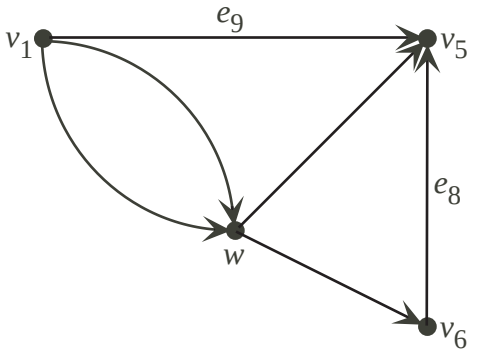
Example. *For the digraph G*



the strongly connected components are $(\{v_1\}, \emptyset)$, $(\{v_2, v_3, v_4\}, \{e_3, e_4, e_5\})$, $(\{v_5\}, \emptyset)$ *and* $(\{v_6\}, \emptyset)$.

The *condensed graph* G_{c} of the digraph G is obtained by contracting all the arcs in every strongly connected component.

Example. *(Continuing from the previous example) The condensed graph is*



3.2 Directed Trees

A directed graph is *quasi-strongly connected* if one of the following conditions holds for every pair of vertices u and v:

(i) $u = v$ or

(ii) there is a directed u–v path in the digraph or

(iii) there is a directed v–u path in the digraph or

(iv) there is a vertex w so that there is a directed w–u path and a directd w–v path.

Example. *(Continuing from the previous example) The digraph G is quasi-strongly connected.*

Quasi-strongly connected digraphs are connected but not necessarily strongly connected.

The vertex v of the digraph G is a *root* if there is a directed path from v to every other vertex of G.

Example. *(Continuing from the previous example) The digraph G only has one root, v_1.*

Theorem 3.1. *A digraph has at least one root if and only if it is quasi-strongly connected.*

Proof. If there is a root in the digraph, it follows from the definition that the digraph is quasi-strongly connected.

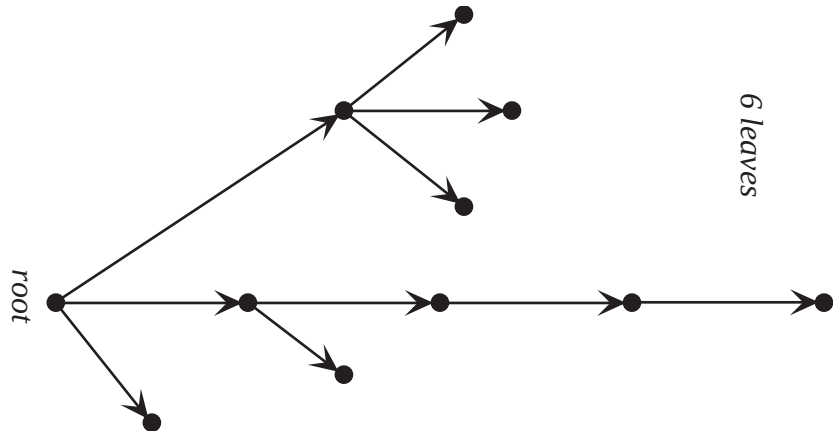
Let us consider a quasi-strongly connected digraph G and show that it must have at least one root. If G is trivial, then it is obvious. Otherwise, consider the vertex set $V = \{v_1, \ldots, v_n\}$ of G where $n \geq 2$. The following process shows that there must be a root:

1. Set $P \leftarrow V$.
2. If there is a directed u–v path between two distinct vertices u and v in P, then we remove v from P. Equivalently, we set $P \leftarrow P - \{v\}$. We repeat this step as many times as possible.
3. If there is only one vertex left in P, then it is the root. For other cases, there are at least two distinct vertices u and v in P and there is no directed path between them in either direction. Since G is quasi-strongly connected, from condition (iv) it follows that there is a vertex w and a w–u path as well as a w–v path. Since u is in P, w can not be in P. We remove u and v from P and add w, i.e. we set $P \leftarrow P - \{u, v\}$ and $P \leftarrow P \cup \{w\}$. Go back to step #2.
4. Repeat as many times as possible.

Every time we do this, there are fewer and fewer vertices in P. Eventually, we will get a root because there is a directed path from some vertex in P to every vertex we removed from P. □

The digraph G is a *tree* if G_{u} is a tree. It is a *directed tree* if G_{u} is a tree and G is quasi-strongly connected, i.e. it has a root. A *leaf* of a directed tree is a vertex whose out-degree is zero.

Example.



Theorem 3.2. *For the digraph G with $n > 1$ vertices, the following are equivalent:*

(i) *G is a directed tree.*

(ii) *G is a tree with a vertex from which there is exactly one directed path to every other vertex of G.*

(iii) *G is quasi-strongly connected but $G - e$ is not quasi-strongly connected for any arc e in G.*

(iv) *G is quasi-strongly connected and every vertex of G has an in-degree of 1 except one vertex whose in-degree is zero.*

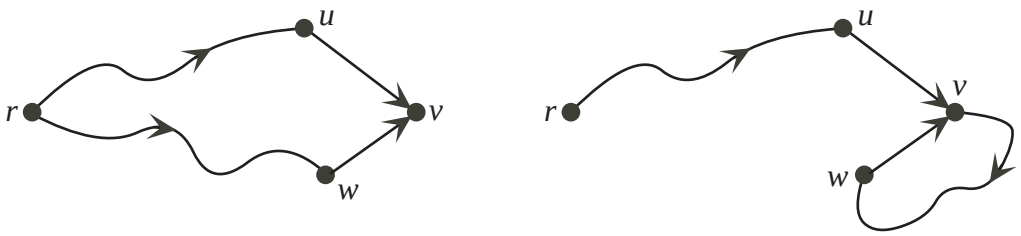
(v) *There are no circuits in G (i.e. not in G_{u}) and every vertex of G has an in-degree of 1 except one vertex whose in-degree is zero.*

(vi) *G is quasi-strongly connected and there are no circuits in G (i.e. not in G_{u}).*

Proof. (i)⇒(ii): If G is a directed tree, then there is a root. This implies that there is a directed path from the root to every other vertex in G (but not more than one path since G_{u} is a tree).

(ii)⇒(iii): If (ii) is true, then G obviously is quasi-strongly connected. We will prove by contradiction by considering the counter hypothesis: There is an arc e in G such that $G - e$ is quasi-strongly connected. The arc e is not a loop because G is a directed tree. Let u and v be the two different end vertices of e. There does not exist a directed u–v path or a directed v–u path in $G - e$ (otherwise G_{u} would have a circuit). Therefore, there is a vertex w and a directed w–u path as well as a directed w–v path. However, this leads to the existence of two directed w–u paths or two directed w–v paths in G depending on the direction of the arc e. Then, there is a circuit in the tree G_{u}. ($\surd$ Theorem 1.6).

(iii)⇒(iv): If G quasi-strongly connected, then it has a root r (Theorem 3.1) so that the in-degrees of other vertices are ≥ 1. We start by considering the counter hypothesis: There exists a vertex $v \neq r$ and $d^-(v) > 1$. Then, v is the terminal vertex of two distinct arcs (u, v) and (w, v). If there were a loop e in G, then $G - e$ would be quasi-strongly connected ($\surd$). Thus, $u \neq v$ with $w \neq v$. Now, there are two distinct directed trails from r to v. The first one includes (u, v) and the second one includes (w, v). We have two possible cases:



In the digraph on the left, the paths r–u and r–w do not include the arcs (u, v) and (w, v). Both $G - (u, v)$ and $G - (w, v)$ are quasi-strongly connected. In the digraph on the right, the r–u path includes the arc (w, v) or (as in the figure) the r–w path includes the arc (u, v). In either case, only one of $G - (u, v)$ and $G - (w, v)$ is quasi-strongly connected because the root is r (Theorem 3.1). ($\surd$) We still have to show that $d^-(r) = 0$. Let us consider the counter hypothesis: $d^-(r) \geq 1$. Then, r is the terminal vertex of some arc e. However, the tree $G - e$ is then quasi-strongly connected since r is its root (Theorem 3.1). ($\surd$)

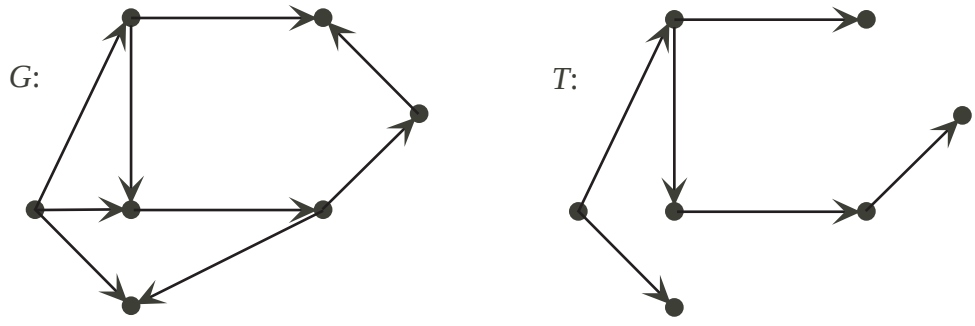
(iv)⇒(v): If (iv) is true, then it is enough to show that there are no circuits in G_{u}. The sum of in-degrees of all the vertices in G is $n - 1$ and the sum of out-degrees of all the vertices in G is also $n - 1$, i.e. there are $n - 1$ arcs in G. Since G is quasi-strongly connected, it is connected and it is a tree (Theorem 2.1). Therefore, there are no circuits in G_{u}.

(v)⇒(vi): If we assume that (v) is true, then there are $n - 1$ arcs in G (compare to the previous proof). By Theorem 2.1, G is a tree. We denote by r the vertex satisfying condition (v). By Theorem 2.1, we see that there is exactly one path to any other vertex of G from r. These paths are also directed. Otherwise, $d^-(r) \geq 1$ or the in-degree of some vertex on that path is > 1 or the in-degree of some other vertex other than r on that path is zero. Hence, r is a root and G is quasi-strongly connected (Theorem 3.1).

(vi)⇒(i): If G is quasi-strongly connected, then it has a root (Theorem 3.1). Since G is connected and there are no circuits in G, it is a tree. □

A directed subgraph T of the digraph G is a *directed spanning tree* if T is a directed tree and T includes every vertex of G.

Example.



Theorem 3.3. *A digraph has a directed spanning tree if and only if it is quasi-strongly connected.*

Proof. If the digraph G has a directed spanning tree T, then the root of T is also a root for G and it is quasi-strongly connected (Theorem 3.1).

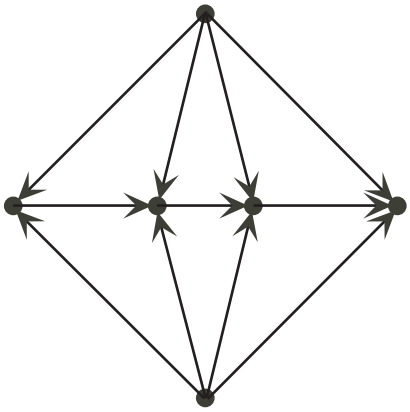
We now assume that G is quasi-strongly connected and show that it has a directed spanning tree. If G is a directed tree, then it is obvious. Otherwise, from Theorem 3.2, we know that there

is an arc e in G so that if we remove e, G remains quasi-strongly connected. We systematically remove these kind of arcs until we get a directed tree. (Compare to the proof for Theorem 2.2) □

3.3 Acyclic Directed Graphs

A directed graph with at least one circuit is said to be *cyclic*. A directed graph is *acylic* otherwise. Obviously, directed trees are acyclic but the reverse implication is not true.

Example. *The digraph*



is acyclic but it is not a directed tree.

Theorem 3.4. *In an acyclic digraph, there exist at least one* source *(a vertex whose in-degree is zero) and at least one* sink *(a vertex whose out-degree is zero).*

Proof. Let G be an acyclic digraph. If G has no arcs, then it is obvious. Otherwise, let us consider the directed path

$$v_{i_0}, e_{j_1}, v_{i_1}, e_{j_2}, \ldots, e_{j_k}, v_{i_k},$$

which has the maximum path length k. Since G is acyclic, $v_{i_0} \neq v_{i_k}$. If (v, v_{i_0}) is an arc, then one of the following is true:

- $v \neq v_{i_t}$ for every value of $t = 0, \ldots, k$. Then,

  $$v, (v, v_{i_0}), v_{i_0}, e_{j_1}, v_{i_1}, e_{j_2}, \ldots, e_{j_k}, v_{i_k}$$

  is a directed path with length $k + 1$. $\surd$

- $v = v_{i_t}$ for some value of t. We choose the smallest such t. Then, $t > 0$ because there are no loops in G and

  $$v_{i_0}, e_{j_1}, v_{i_1}, e_{j_2}, \ldots, e_{j_t}, v_{i_t}, (v, v_{i_0}), v_{i_0}$$

  is a directed circuit. $\surd$

Hence, $d^-(v_{i_0}) = 0$. Using a similar technique, we can show that $d^+(v_{i_k}) = 0$ as well. □

If $G = (V, E)$ is a digraph with n vertices, then a labeling of the vertices with an injective function $\alpha : V \to \{1, \ldots, n\}$ which satisfies the condition $\alpha(u) < \alpha(v)$ whenever (u, v) is an arc in G is known as *topological sorting*.

Theorem 3.5. *We can sort the vertices of a digraph topologically if and only if the graph is acyclic.*

Proof. If the digraph is cyclic, then obviously we can not sort the vertices topologically.

If the digraph G is acyclic, then we can sort the vertices in the following manner: [2]

1. We choose a vertex v which is a sink. It exists by Theorem 3.4. We set $\alpha(v) \leftarrow n$, $G \leftarrow G - v$ and $n \leftarrow n - 1$.

2. If there is just one vertex v in G, set $\alpha(v) \leftarrow 1$. Otherwise, go back to step #1. □

[2] This is known as *Marimont's Algorithm*. The algorithm itself contains other items, too. The original reference is MARIMONT, R.B.: A New Method of Checking the Consistency of Precedence Matrices. *Journal of the Association for Computing Machinery* **6** (1959), 164–171.

Chapter 4

Matrices and Vector Spaces of Graphs

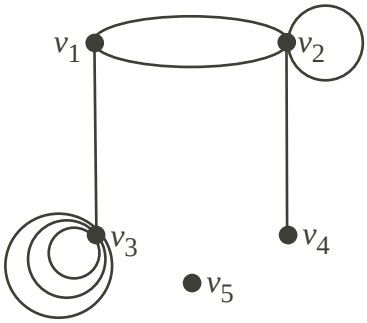
4.1 Matrix Representation of Graphs

The *adjacency matrix* of the graph $G = (V, E)$ is an $n \times n$ matrix $\mathbf{D} = (d_{ij})$, where n is the number of vertices in G, $V = \{v_1, \ldots, v_n\}$ and

$$d_{ij} = \text{number of edges between } v_i \text{ and } v_j.$$

In particular, $d_{ij} = 0$ if (v_i, v_j) is not an edge in G. The matrix $\mathbf{D}$ is symmetric, i.e. $\mathbf{D}^{\mathrm{T}} = \mathbf{D}$.

Example.



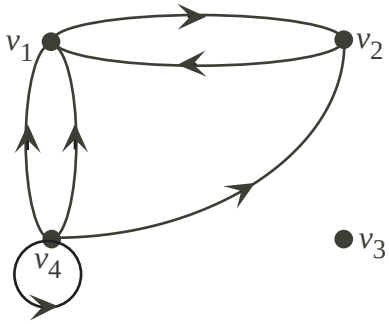
$$\mathbf{D} = \begin{pmatrix} 0 & 2 & 1 & 0 & 0 \\ 2 & 1 & 0 & 1 & 0 \\ 1 & 0 & 3 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Obviously, an adjacency matrix defines a graph completely up to an isomorphism.

The adjacency matrix of a directed graph G is $\mathbf{D} = (d_{ij})$, where

$$d_{ij} = \text{number of arcs that come out of vertex } v_i \text{ and go into vertex } v_j.$$

Example.

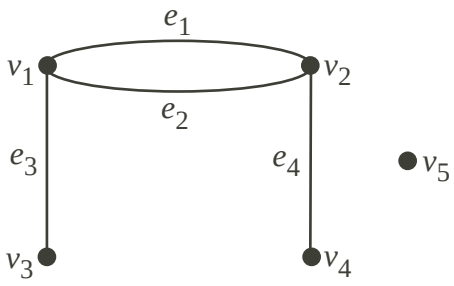


$$\mathbf{D} = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 2 & 1 & 0 & 1 \end{pmatrix}$$

The *all-vertex incidence matrix* of a non-empty and loopless graph $G = (V, E)$ is an $n \times m$ matrix $\mathbf{A} = (a_{ij})$, where n is the number of vertices in G, m is the number of edges in G and

$$a_{ij} = \begin{cases} 1 \text{ if } v_i \text{ is an end vertex of } e_j \\ 0 \text{ otherwise.} \end{cases}$$

Example.

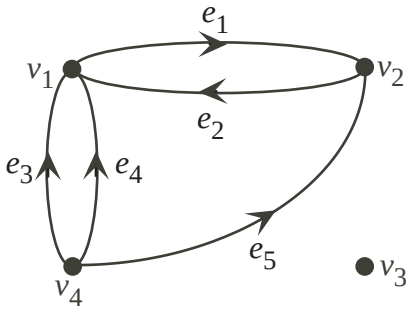


$$\mathbf{A} = \begin{array}{c} \begin{matrix} e_1 & e_2 & e_3 & e_4 \end{matrix} \\ \begin{pmatrix} 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix} \end{array} \begin{matrix} v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \end{matrix}$$

The *all-vertex incidence matrix* of a non-empty and loopless directed graph G is $\mathbf{A} = (a_{ij})$, where

$$a_{ij} = \begin{cases} 1 \text{ if } v_i \text{ is the initial vertex of } e_j \\ -1 \text{ if } v_i \text{ is the terminal vertex of } e_j \\ 0 \text{ otherwise.} \end{cases}$$

Example.



$$\mathbf{A} = \begin{array}{c} \begin{matrix} e_1 & e_2 & e_3 & e_4 & e_5 \end{matrix} \\ \begin{pmatrix} 1 & -1 & -1 & -1 & 0 \\ -1 & 1 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 \end{pmatrix} \end{array} \begin{matrix} v_1 \\ v_2 \\ v_3 \\ v_4 \end{matrix}$$

Since every column of an all-vertex incidence matrix contains exactly two non-zero numbers, two ones, we can remove a row and still have enough information to define the graph. The *incidence matrix* of a graph is obtained by removing a row from the all-vertex incidence matrix. It is not unique because there are n possible rows to remove. The vertex corresponding to the row removed is called the *reference vertex*.

Similarly, every column in the all-vertex incidence matrix of a digraph contains exactly two non-zero numbers, $+1$ and -1. We can remove a row from the all-vertex incidence matrix and obtain the *incidence matrix*. Notice that the rows of an all-vertex incidence matrix are linearly dependent because the sum of rows is a zero vector.

Theorem 4.1. *The determinant of an incidence matrix of a nontrivial tree is* ± 1*, regardless of whether the tree is directed or not.*

Proof. We use induction on n, the number of vertices in the tree.

Induction Basis: $n = 2$ and it is obvious.

Induction Hypothesis: The theorem is true for $n \leq k$. ($k \geq 2$)

Induction Statement: The theorem is true for $n = k + 1$.

Induction Statement Proof: Let T be a tree which has $k + 1$ vertices and let $\mathbf{A}$ be an (arbitrary) incidence matrix of T. T has at least two leaves (Theorem 2.3). We choose a leaf v_i which is not the reference vertex of $\mathbf{A}$ and an edge e_t which is incident on v_i. Then,

$$a_{it} = (\pm)1 \quad \text{and} \quad a_{ij} = 0, \text{ when } j \neq t.$$

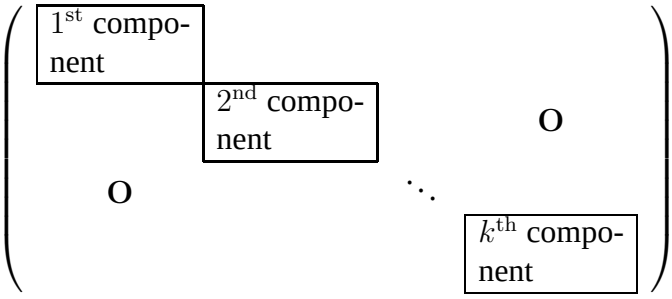
We expand the determinant of $|\mathbf{A}|$ by the i^{th} row:

$$|\mathbf{A}| = (\pm)(-1)^{i+t}|\mathbf{A}'|,$$

where $\mathbf{A}'$ is the minor corresponding to a_{it}. We write $T' = T - v_i$ which is also a tree (v_i is a leaf). We use the induction hypothesis to get $|\mathbf{A}'| = \pm 1$ because $\mathbf{A}'$ is obviously an incidence matrix of T'. □

Corollary. *If the digraph G has no loops, then the rank of its all-vertex incidence matrix is $\rho(G)$.*

Proof. If we rearrange the rows or columns of the all-vertex incidence matrix, the rank of the matrix will not change. Let us rearrange the vertices and arcs to group them by components. Then, the all-vertex incidence matrix is a block diagonal matrix in which each block is an all-vertex incidence matrix of a component.



We denote n_{i} as the number of vertices in the i^{th} component. Every component has a spanning tree whose incidence matrix has a nonzero determinant by Theorem 4.1, i.e. the matrix is not singular. The all-vertex incidence matrix of the i^{th} component is obtained by adding rows to an incidence matrix of the corresponding spanning tree. The row added is linearly dependent of other rows so that the rank of this matrix is the same as the rank of the incidence matrix ($= n_i - 1$). Notice that in the special case when a component is trivial, the rank is zero $= 1 - 1$. Therefore,

$$\begin{aligned} \text{rank of } \mathbf{A} &= \text{sum of the ranks of the components} \\ &= (n_1 - 1) + \cdots + (n_k - 1) \\ &= \underbrace{n_1 + \cdots + n_k}_{= n} - k = \rho(G). \end{aligned}$$

□

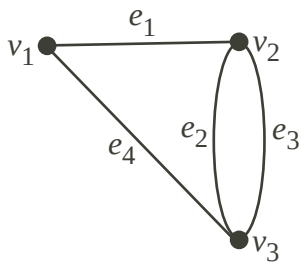
Remark. *From this proof, we can also get a basis for the row space and the column space of the all-vertex incidence matrix. The columns corresponding to the branches of the spanning tree of G are a basis of the column space. We can get a basis of the row space by removing one row out of each component block.*

4.2 Cut Matrix

If all the cuts of a nontrivial and loopless graph $G = (V, E)$ are $I_1, \ldots, I_t$, then the *cut matrix* of G is a $t \times m$ matrix $\mathbf{Q} = (q_{ij})$, where m is the number of edges in G and

$$q_{ij} = \begin{cases} 1 \text{ if } e_j \in I_i \text{ (the cut is interpreted as an edge set)} \\ 0 \text{ otherwise.} \end{cases}$$

Example. *For the graph*



the cuts are $I_1 = \{e_1, e_4\}$, $I_2 = \{e_2, e_3, e_4\}$ *and* $I_3 = \{e_1, e_2, e_3\}$. *The cut matrix is*

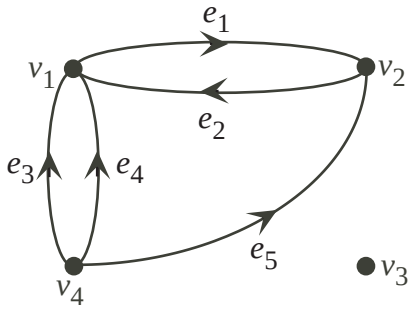
$$\mathbf{Q} = \begin{array}{c} \begin{matrix} e_1 & e_2 & e_3 & e_4 \end{matrix} \\ \begin{pmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix} \end{array} \begin{matrix} I_1 \\ I_2 \\ I_3 \end{matrix}$$

Remark. *If the graph has* n *vertices, then it has* $\frac{1}{2}(2^n - 2) = 2^{n-1} - 1$ *cuts. Usually, there are not this many distinct edge sets. For the cut matrix, we only take one cut corresponding to an edge set so that there would not be repeated rows. Even so, there are usually too many rows.*

If G is a nontrivial and loopless digraph, then we assign an arbitrary direction to every cut $\langle V_1, V_2 \rangle$: the *orientation* of $\langle V_1, V_2 \rangle$ is from V_1 to V_2. In other words, we consider *oriented cuts* and we pick only one direction from the two possibilities. Then, the *cut matrix* $\mathbf{Q} = (q_{ij})$ is

$$q_{ij} = \begin{cases} 1 \text{ if } e_j \in I_i \text{ and they are in the same direction} \\ -1 \text{ if } e_j \in I_i \text{ and they are in opposite directions} \\ 0 \text{ otherwise.} \end{cases}$$

Example. *For the digraph*



the different cuts (interpreted as edge sets) are $I_1 = \{e_1, e_2, e_3, e_4\}$ *(in the direction of* e_1*),* $I_2 = \{e_3, e_4, e_5\}$ *(in the direction of* e_3*),* $I_3 = \{e_1, e_2, e_5\}$ *(in the direction of* e_1*) and* $I_4 = \emptyset$. *The cut matrix is*

$$\mathbf{Q} = \begin{array}{c} \begin{matrix} e_1 & e_2 & e_3 & e_4 & e_5 \end{matrix} \\ \begin{pmatrix} 1 & -1 & -1 & -1 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 1 & -1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix} \end{array} \begin{matrix} I_1 \\ I_2 \\ I_3 \\ I_4 \end{matrix}$$

Since $\langle\{v\}, V - \{v\}\rangle$ is a cut for every vertex v, rows of the all-vertex incidence matrix are rows of $\mathbf{Q}$. If we are dealing with directed graphs, then these rows may have to be multiplied by -1.

Theorem 4.2. *Every row of the cut matrix of a digraph can be expressed in two different ways as a linear combination of the rows of the all-vertex incidence matrix. The non-zero coefficients are either all* $= +1$ *or all* $= -1$.

Proof. Let $\mathbf{Q}$ be the cut matrix of a digraph $G = (V, E)$ and let $\mathbf{A}$ be the all-vertex incidence matrix. Let $\langle V_1, V_2\rangle$ (note that it is oriented) be the cut corresponding to the i^{th} row of $\mathbf{Q}$. Reindexing if needed, we can assume that

$$V_1 = \{v_1, \ldots, v_r\} \quad \text{and} \quad V_2 = \{v_{r+1}, \ldots, v_n\}.$$

We write

$$\mathbf{q}_i = i^{\text{th}} \text{ row of } \mathbf{Q} \quad \text{and} \quad \mathbf{a}_t = t^{\text{th}} \text{ row of } \mathbf{A}.$$

We show that

$$\mathbf{q}_i = \sum_{t=1}^{r} \mathbf{a}_t = -\sum_{t=r+1}^{n} \mathbf{a}_t,$$

which proves the theorem. Let $(v_p, v_q) = e_k$ be the k^{th} arc of G. Then,

$$a_{pk} = k^{\text{th}} \text{ element of the vector } \mathbf{a}_p = 1,$$
$$a_{qk} = k^{\text{th}} \text{ element of the vector } \mathbf{a}_q = -1$$

and

$$a_{jk} = 0 \quad \text{if} \quad j \neq p, q.$$

We get four cases:

- $v_p \in V_1$ and $v_q \in V_2$: Now $p \leq r$ and $q \geq r + 1$ so $q_{ik} = 1$ and

$$q_{ik} = \sum_{t=1}^{r} a_{tk} = -\sum_{t=r+1}^{n} a_{tk}.$$

- $v_p \in V_2$ and $v_q \in V_1$: Now $p \geq r + 1$ and $q \leq r$ so $q_{ik} = -1$ and

$$q_{ik} = \sum_{t=1}^{r} a_{tk} = -\sum_{t=r+1}^{n} a_{tk}.$$

- $v_p \in V_1$ and $v_q \in V_1$: Now $p \leq r$ and $q \leq r$ so $q_{ik} = 0$ and

$$q_{ik} = \sum_{t=1}^{r} a_{tk} = -\underbrace{a_{r+1,k}}_{=0} - \cdots - \underbrace{a_{nk}}_{=0}.$$

- $v_p \in V_2$ and $v_q \in V_2$: Now $p \geq r + 1$ and $q \geq r + 1$ so $q_{ik} = 0$ and

$$q_{ik} = \underbrace{a_{1k}}_{=0} + \cdots + \underbrace{a_{rk}}_{=0} = -\sum_{t=r+1}^{n} a_{tk}.$$

The statements above are valid for every k. □

Example. *(Continuing from the previous example) The corresponding row of I_1 is*

$$(1,-1,-1,-1,0) = (1,-1,-1,-1,0) = -(-1,1,0,0,-1) - (0,0,0,0,0) - (0,0,1,1,1).$$

Corollary. *The rank of the cut matrix of a digraph G is $\rho(G)$.*

Proof. The all-vertex incidence matrix $\mathbf{A}$ of G is also a submatrix of the cut matrix $\mathbf{Q}$ of G. Then, (by Corollary of Theorem 4.1)

$$\text{rank}(\mathbf{Q}) \geq \text{rank}(\mathbf{A}) = \rho(G).$$

On the other hand, by Theorem 4.2, every row of $\mathbf{Q}$ can be expressed as a linear combination of the rows of $\mathbf{A}$. Therefore,

$$\text{rank}(\mathbf{Q}) = \text{rank}(\mathbf{A}) = \rho(G).$$

□

Another consequence is that the cut matrix $\mathbf{Q}$ can be expressed as

$$\mathbf{Q} = \mathbf{A}_1\mathbf{A},$$

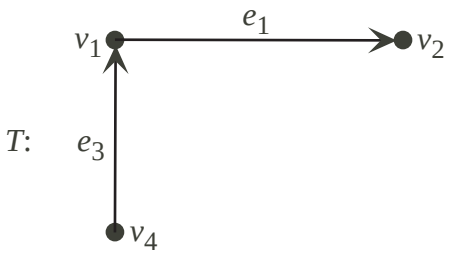
where the elements of $\mathbf{A}_1$ are 0 or ± 1. In addition, the matrix $\mathbf{A}_1$ can be constructed from the process in the proof of Theorem 4.2.

If the graph G is connected, then it has a spanning tree T and an associated fundamental cut set. The fundamental cut sets are also cuts (when cuts are interpreted as edge sets). Therefore, the cut matrix $\mathbf{Q}$ of G has a submatrix $\mathbf{Q}_\text{f}$ that corresponds to these fundamental cut sets. This matrix is called the *fundamental cut set matrix*. Similarly, the connected digraph G has a fundamental cut set matrix: if we interpret a fundamental cut set as a set, then the direction of the cut is chosen to be the same as the direction of the corresponding branch of T. If we rearrange the edges of G so that we have the branches first and sort the fundamental cut sets in the same order, then we get the fundamental cut set matrix in the form

$$\mathbf{Q}_\text{f} = \left(\begin{array}{c|c} \mathbf{I}_{n-1} & \mathbf{Q}_\text{fc} \end{array} \right),$$

where $\mathbf{I}_{n-1}$ is the identity matrix with $n-1$ rows. The rank of $\mathbf{Q}_\text{f}$ is thus $n-1 = \rho(G)$.

Example. *(Continuing from the previous example) We left out vertex v_3 so we get a connected digraph. We choose the spanning tree*



The fundamental cut sets are $I_2 = \{e_3, e_4, e_5\}$ (in the direction of e_3) and $I_3 = \{e_1, e_2, e_5\}$ (in the direction of e_1). Then,

$$\mathbf{Q}_{\mathrm{f}} = \begin{array}{c} \begin{array}{cc|ccc} e_1 & e_3 & e_2 & e_4 & e_5 \end{array} \\ \left(\begin{array}{cc|ccc} 1 & 0 & -1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 \end{array}\right) \begin{array}{c} I_3 \\ I_2 \end{array} \end{array}$$

and

$$\mathbf{Q} = \begin{array}{c} \begin{array}{ccccc} e_1 & e_2 & e_3 & e_4 & e_5 \end{array} \\ \left(\begin{array}{ccccc} 1 & -1 & -1 & -1 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 1 & -1 & 0 & 0 & 1 \end{array}\right) \begin{array}{c} \leftarrow \\ \leftarrow \end{array} \end{array}$$

4.3 Circuit Matrix

We consider a graph $G = (V, E)$ which contains circuits. We enumerate the circuits of G: $C_1, \ldots, C_\ell$. The *circuit matrix* of G is an $\ell \times m$ matrix $\mathbf{B} = (b_{ij})$ where

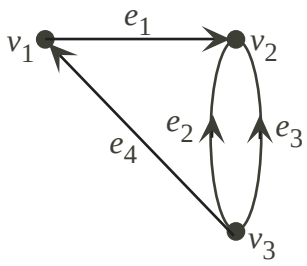
$$b_{ij} = \begin{cases} 1 \text{ if the arc } e_j \text{ is in the circuit } C_i \\ 0 \text{ otherwise} \end{cases}$$

(as usual, $E = \{e_1, \ldots, e_m\}$).

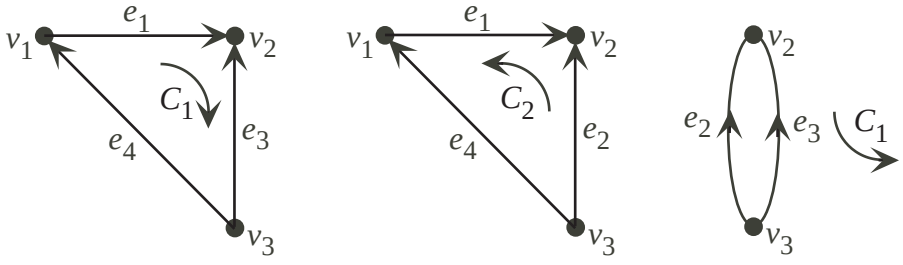
The circuits in the digraph G are *oriented*, i.e. every circuit is given an arbitrary *direction* for the sake of defining the circuit matrix. After choosing the orientations, the circuit matrix of G is $\mathbf{B} = (b_{ij})$ where

$$b_{ij} = \begin{cases} 1 \text{ if the arc } e_j \text{ is in the circuit } C_i \text{ and they in the same direction} \\ -1 \text{ if the arc } e_j \text{ is in the circuit } C_i \text{ and they are in the opposite direction} \\ 0 \text{ otherwise.} \end{cases}$$

Example. *For the directed graph*



the circuits are



and the circuit matrix is

$$\mathbf{B} = \begin{array}{c} \begin{matrix} e_1 & e_2 & e_3 & e_4 \end{matrix} \\ \begin{pmatrix} 1 & 0 & -1 & 1 \\ -1 & 1 & 0 & -1 \\ 0 & -1 & 1 & 0 \end{pmatrix} \end{array} \begin{matrix} C_1 \\ C_2 \\ C_3 \end{matrix}$$

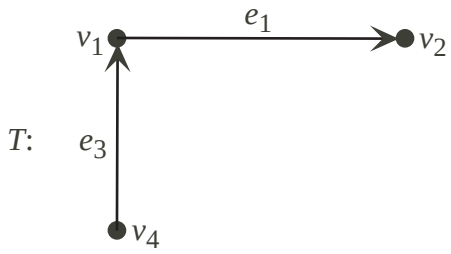
If the graph G is connected and contains at least one circuit, then it has a cospanning tree T^* and the corresponding fundamental circuits. By choosing the corresponding rows of the circuit matrix $\mathbf{B}$, we get an $(m-n+1)\times m$ matrix $\mathbf{B}_\mathrm{f}$, called the *fundamental circuit matrix*. Similarly, a connected digraph G with at least one circuit has a fundamental circuit matrix: the direction of a fundamental circuit is the same as the direction of the corresponding link in T^*.

When we rearrange the edges of G so that the links of T^* come last and sort the fundamental circuits in the same order, the fundamental circuit matrix takes the form

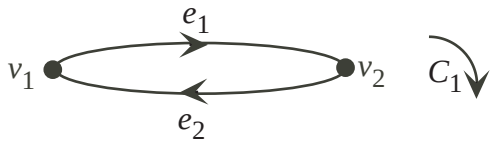
$$\mathbf{B}_\mathrm{f} = \left(\begin{array}{c|c} \mathbf{B}_\mathrm{ft} & \mathbf{I}_{m-n+1} \end{array} \right),$$

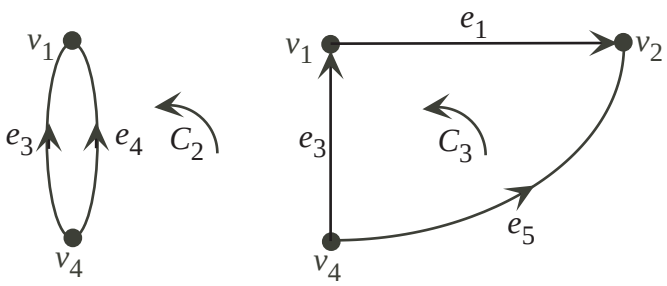
where $\mathbf{I}_{m-n+1}$ is the identity matrix with $m-n+1$ rows. The rank of $\mathbf{B}_\mathrm{f}$ is thus $m-n+1=\mu(G)$ and the rank of $\mathbf{B}$ is $\geq m-n+1$.

Example. *(Continuing from the previous example) We left out vertex v_3 so we get a connected digraph (see p.34) and we chose the spanning tree*



The fundamental circuits are





and

$$\mathbf{B}_\mathrm{f} = \begin{array}{c} \begin{matrix} e_1 & e_3 & e_2 & e_4 & e_5 \end{matrix} \\ \left(\begin{array}{cc|ccc} 1 & 0 & 1 & 0 & 0 \\ 0 & -1 & 0 & 1 & 0 \\ -1 & -1 & 0 & 0 & 1 \end{array}\right) \end{array} \begin{matrix} C_1 \\ C_2 \\ C_3 \end{matrix}$$

Theorem 4.3. *A directed cut and a directed circuit of a digraph have an even number of common arcs. Half of these arcs have the same orientation in the cut and in the circuit, and the remaining arcs have opposite orientations in the cut and in the circuit.*

Proof. Compare to the proof of Theorem 2.6. □

Theorem 4.4. *For a digraph,* $\mathbf{B}\mathbf{Q}^{\mathrm{T}} = \mathbf{O}$ *(zero matrix).*

Proof. By the previous theorem, half of the nonzero numbers in the dot product corresponding to each element of $\mathbf{B}\mathbf{Q}^{\mathrm{T}}$ are $+1$. The remaining nonzero numbers are -1. Therefore, the dot product is $= 0$. □

Theorem 4.5. *If the digraph* G *contains at least one circuit, then the rank of its circuit matrix* $\mathbf{B}$ *is* $\mu(G)$*. Furthermore, if* G *is connected, then the circuit matrix* $\mathbf{B}$ *can be expressed as* $\mathbf{B} = \mathbf{B}_2\mathbf{B}_{\mathrm{f}}$*, where the matrix* $\mathbf{B}_2$ *consists of* 0*'s and* ± 1*'s, and the cut matrix* $\mathbf{Q}$ *can be expressed as* $\mathbf{Q} = \mathbf{Q}_1\mathbf{Q}_{\mathrm{f}}$*, where the matrix* $\mathbf{Q}_1$ *consists of* 0*'s and* ± 1*'s.*

Proof. First we consider the case when G is connected. We choose a spanning tree T of G and rearrange the m edges of G so that the branches of T come first and the links of T^* come last. We sort the fundamental cut sets in the same order as the branches and links. Then,

$$\mathbf{Q}_{\mathrm{f}} = \left(\begin{array}{c|c} \mathbf{I}_{n-1} & \mathbf{Q}_{\mathrm{fc}} \end{array}\right) \quad \text{and} \quad \mathbf{B}_{\mathrm{f}} = \left(\begin{array}{c|c} \mathbf{B}_{\mathrm{ft}} & \mathbf{I}_{m-n+1} \end{array}\right).$$

The blocks of $\mathbf{B}$ can be constructed in a similar way:

$$\mathbf{B} = \left(\begin{array}{c|c} \mathbf{B}_1 & \mathbf{B}_2 \end{array}\right).$$

Since $\mathbf{Q}_{\mathrm{f}}$ is a submatrix of $\mathbf{Q}$ and $\mathbf{B}_{\mathrm{f}}$ is a submatrix of $\mathbf{B}$, it follows from Theorem 4.4 that

$$\begin{aligned}\mathbf{O} = \mathbf{B}_{\mathrm{f}}\mathbf{Q}_{\mathrm{f}}^{\mathrm{T}} &= \left(\begin{array}{c|c} \mathbf{B}_{\mathrm{ft}} & \mathbf{I}_{m-n+1} \end{array}\right)\left(\begin{array}{c|c} \mathbf{I}_{n-1} & \mathbf{Q}_{\mathrm{fc}} \end{array}\right)^{\mathrm{T}} = \left(\begin{array}{c|c} \mathbf{B}_{\mathrm{ft}} & \mathbf{I}_{m-n+1} \end{array}\right)\begin{pmatrix}\mathbf{I}_{n-1} \\ \hline \mathbf{Q}_{\mathrm{fc}}^{\mathrm{T}}\end{pmatrix} \\ &= \mathbf{B}_{\mathrm{ft}}\mathbf{I}_{n-1} + \mathbf{I}_{m-n+1}\mathbf{Q}_{\mathrm{fc}}^{\mathrm{T}} = \mathbf{B}_{\mathrm{ft}} + \mathbf{Q}_{\mathrm{fc}}^{\mathrm{T}}.\end{aligned}$$

Hence

$$\mathbf{B}_{\mathrm{ft}} = -\mathbf{Q}_{\mathrm{fc}}^{\mathrm{T}}.$$

Furthermore, since $\mathbf{Q}_{\mathrm{f}}$ is a submatrix of $\mathbf{Q}$, we can use the same theorem to get

$$\begin{aligned}\mathbf{O} = \mathbf{B}\mathbf{Q}_{\mathrm{f}}^{\mathrm{T}} &= \left(\begin{array}{c|c} \mathbf{B}_1 & \mathbf{B}_2 \end{array}\right)\left(\begin{array}{c|c} \mathbf{I}_{n-1} & \mathbf{Q}_{\mathrm{fc}} \end{array}\right)^{\mathrm{T}} = \left(\begin{array}{c|c} \mathbf{B}_1 & \mathbf{B}_2 \end{array}\right)\begin{pmatrix}\mathbf{I}_{n-1} \\ \hline \mathbf{Q}_{\mathrm{fc}}^{\mathrm{T}}\end{pmatrix} \\ &= \mathbf{B}_1\mathbf{I}_{n-1} + \mathbf{B}_2\mathbf{Q}_{\mathrm{fc}}^{\mathrm{T}} = \mathbf{B}_1 - \mathbf{B}_2\mathbf{B}_{\mathrm{ft}}.\end{aligned}$$

Hence

$$\mathbf{B} = \left(\begin{array}{c|c} \mathbf{B}_2\mathbf{B}_{\mathrm{ft}} & \mathbf{B}_2 \end{array}\right) = \mathbf{B}_2\left(\begin{array}{c|c} \mathbf{B}_{\mathrm{ft}} & \mathbf{I}_{m-n+1} \end{array}\right) = \mathbf{B}_2\mathbf{B}_{\mathrm{f}},$$

as claimed. In the same way, $\mathbf{Q}$ can be expressed as $\mathbf{Q} = \mathbf{Q}_1\mathbf{Q}_{\mathrm{f}}$, as claimed, which is clear anyway since the rank of $\mathbf{Q}$ is $n-1$ and its elements are 0's and ± 1's.

Every row of $\mathbf{B}$ is a linear combination of the rows corresponding to the fundamental circuits and the rank of $\mathbf{B}$ is at most equal to the rank of $\mathbf{B}_{\mathrm{f}} = m - n + 1$. On the other hand, as we pointed out earlier, the rank of $\mathbf{B}$ is $\geq m - n + 1$. Thus, $\mathrm{rank}(\mathbf{B}) = m - n + 1$ $(= \mu(G))$ for a connected digraph.

In the case of a disconnected digraph G (which contains at least one circuit), it is divided into components ($k \geq 2$ components) and the circuit matrix $\mathbf{B}$ is divided into blocks corresponding to the components (compare to the proof of the corollary of Theorem 4.1), in which case

$$\mathrm{rank}(\mathbf{B}) = \sum_{i=1}^{k}(m_i - n_i + 1) = m - n + k = \mu(G).$$

□

Notice that the proof also gives the formula, $\mathbf{B}_{\mathrm{ft}} = -\mathbf{Q}_{\mathrm{fc}}^{\mathrm{T}}$, which connects the cut matrix and the fundamental circuit matrix.

4.4 An Application: Stationary Linear Networks

A *stationary linear network* is a directed graph G that satisfies the following conditions:

1. G is connected.
2. Every arc of G belongs to some circuit and there are no loops in G.
3. Every arc e_j in G is associated with a number i_j called the *through-quantity* or *flow*. If there are m arcs in G, then we write

$$\mathbf{i} = \begin{pmatrix} i_1 \\ \vdots \\ i_m \end{pmatrix}$$

   (*through-vector*).
4. Every vertex v_i in G is associated with a number p_i called the *potential*. Furthermore, the *across-quantity* or *potential difference* of the arc $e_j = (v_{i_1}, v_{i_2})$ is

$$u_j = p_{i_2} - p_{i_1}.$$

   If there are n vertices and m arcs in G, then we write

$$\mathbf{p} = \begin{pmatrix} p_1 \\ \vdots \\ p_n \end{pmatrix} \quad \text{and} \quad \mathbf{u} = \begin{pmatrix} u_1 \\ \vdots \\ u_m \end{pmatrix}$$

   (*potential vector* and *across-vector*). (Potentials are rarely needed.)
5. Every arc e_j is one of the following:
   (a) a *component*[1], for which there is an associated number r_j. r_j is constant ($\neq 0$) (stationarity) and the following equation links the quantities:

$$u_j = i_j r_j \quad \text{(linearity)}.$$

   (b) a *through-source*, for which the through-quantity i_j is fixed.
   (c) an *across-source*, for which the across-quantity u_j is fixed.
6. (*Kirchhoff's Through-Quantity Law*) The sum of the through-quantities of a directed cut of G is zero when the cut is interpreted as an edge set and the sign of a through-quantity is changed if the directions of a cut and an arc are different.
7. (*Kirchhoff's Across-Quantity Law*) The sum of the across-quantities of a directed circuit of G is zero when the sign of an across-quantity is changed if the directions of a circuit and an arc are different.

[1] Not to be confused with a component of a graph!

Example. *A typical stationary linear network is an electrical circuit with linear resistors, constant current sources and constant voltage sources. The components are resistors and r_j are the resistances. Equation 5.(a) is Ohm's Law.*

We take a spanning tree T of a stationary linear network G, its fundamental cut matrix $\mathbf{Q}_\mathrm{f}$ and its fundamental circuit matrix $\mathbf{B}_\mathrm{f}$. Let us rearrange the arcs in these matrices and vectors $\mathbf{i}$ and $\mathbf{u}$ like we did before. That is, the branches of T will come first followed by the links of T^*. Kirchhoff's Laws can then be written as

$$\mathbf{Q}\mathbf{i} = \mathbf{0} \quad \text{and} \quad \mathbf{B}\mathbf{u} = \mathbf{0}.$$

On the other hand, the rows of the fundamental cut matrix $\mathbf{Q}_\mathrm{f}$ span all the rows of $\mathbf{Q}$, and similarly rows of the fundamental circuit matrix $\mathbf{B}_\mathrm{f}$ span the rows of $\mathbf{B}$. Then, Kirchhoff's Laws can also be written as

$$\mathbf{Q}_\mathrm{f}\mathbf{i} = \mathbf{0}_{n-1} \quad \text{and} \quad \mathbf{B}_\mathrm{f}\mathbf{u} = \mathbf{0}_{m-n+1}.$$

Let us form the diagonal matrices $\mathbf{K} = \lceil k_1, \ldots, k_m \rfloor$ and $\mathbf{L} = \lceil \ell_1, \ldots, \ell_m \rfloor$, where

$$k_j = \begin{cases} -r_j \text{ if } e_j \text{ is a component} \\ 1 \text{ if } e_j \text{ is a through-source} \\ 0 \text{ if } e_j \text{ is an across-source} \end{cases} \quad \text{and} \quad \ell_j = \begin{cases} 1 \text{ if } e_j \text{ is a component} \\ 0 \text{ if } e_j \text{ is a through-source} \\ 1 \text{ if } e_j \text{ is an across-source,} \end{cases}$$

and the m-vector $\mathbf{s} = (s_1, \ldots, s_m)^\mathrm{T}$, where

$$s_j = \begin{cases} 0 \text{ if } e_j \text{ is a component} \\ i_j \text{ if } e_j \text{ is a through-source} \\ u_j \text{ if } e_j \text{ is an across-source.} \end{cases}$$

Then, all the information can be expressed as a system of linear equations

$$\left(\begin{array}{c|c} \mathbf{K} & \mathbf{L} \\ \hline \mathbf{Q}_\mathrm{f} & \mathbf{O}_{(n-1)\times m} \\ \hline \mathbf{O}_{(m-n+1)\times m} & \mathbf{B}_\mathrm{f} \end{array}\right) \left(\begin{array}{c} \mathbf{i} \\ \hline \mathbf{u} \end{array}\right) = \left(\begin{array}{c} \mathbf{s} \\ \hline \mathbf{0}_{n-1} \\ \hline \mathbf{0}_{m-n+1} \end{array}\right),$$

known as the *fundamental equations*. The through and across quantities can be solved (ideally) if r_j and the sources are given.

Remark. *The same procedure can be applied to form state (differential) equations for* dynamic *networks, which have nonstationary components.*

The matrix of this system of linear equations does not have to be nonsingular and the system does not even have to have a unique solution at all. For example, in the matrix above, we can easily see that it is singular if some circuit only consists of across-sources or if some cut only consists of through-sources. As a matter of fact, this is the only case when the sources are not defined uniquely if the constants r_j are real numbers with the same sign (and often otherwise too).

We choose a specific spanning tree T to explore these concepts more carefully:

Lemma. *If no cut of G consists of only through-sources and no circuit of G consists of only across-sources, then G has a spanning tree T such that every across-source is a branch of T and every through-source is a link of T^*.*

Proof. If G satisfies the hypothesis, then we first choose a digraph M which has every vertex and across-source (arc) of G. There are no circuits in this digraph. Then we add components to M one by one and try to come up with a spanning tree. If this fails at some point, then G has a cut with only through-sources, which is impossible. □

Now let us assume that no cut of G consists of only through-sources and no circuit of G consists of only across-sources. We use the spanning tree T mentioned in the lemma. We rearrange the arcs of G so that (as before) the branches of T come first. Within these branches, the across-sources come first followed by components. Similarly, the links are rearranged so that the components come first and the through-sources come last.

The system of $2m$ equations can then be written as

$$\begin{pmatrix} \mathbf{O} & \mathbf{O} & \mathbf{O} & \mathbf{O} & \mathbf{I} & \mathbf{O} & \mathbf{O} & \mathbf{O} \\ \mathbf{O} & -\mathbf{R}_1 & \mathbf{O} & \mathbf{O} & \mathbf{O} & \mathbf{I} & \mathbf{O} & \mathbf{O} \\ \mathbf{O} & \mathbf{O} & -\mathbf{R}_2 & \mathbf{O} & \mathbf{O} & \mathbf{O} & \mathbf{I} & \mathbf{O} \\ \mathbf{O} & \mathbf{O} & \mathbf{O} & \mathbf{I} & \mathbf{O} & \mathbf{O} & \mathbf{O} & \mathbf{O} \\ \mathbf{I} & \mathbf{O} & \mathbf{Q}_{11} & \mathbf{Q}_{12} & \mathbf{O} & \mathbf{O} & \mathbf{O} & \mathbf{O} \\ \mathbf{O} & \mathbf{I} & \mathbf{Q}_{21} & \mathbf{Q}_{22} & \mathbf{O} & \mathbf{O} & \mathbf{O} & \mathbf{O} \\ \mathbf{O} & \mathbf{O} & \mathbf{O} & \mathbf{O} & \mathbf{B}_{11} & \mathbf{B}_{12} & \mathbf{I} & \mathbf{O} \\ \mathbf{O} & \mathbf{O} & \mathbf{O} & \mathbf{O} & \mathbf{B}_{21} & \mathbf{B}_{22} & \mathbf{O} & \mathbf{I} \end{pmatrix} \begin{pmatrix} \mathbf{i}_1 \\ \mathbf{i}_2 \\ \mathbf{i}_3 \\ \mathbf{i}_4 \\ \mathbf{u}_1 \\ \mathbf{u}_2 \\ \mathbf{u}_3 \\ \mathbf{u}_4 \end{pmatrix}$$

$$= \begin{pmatrix} \mathbf{s}_1 \\ \mathbf{0} \\ \mathbf{0} \\ \mathbf{s}_2 \\ \mathbf{0} \\ \mathbf{0} \\ \mathbf{0} \\ \mathbf{0} \end{pmatrix} \begin{matrix} \leftarrow \text{across-sources (in branches)} \\ \leftarrow \text{components (in branches)} \\ \leftarrow \text{components (in links)} \\ \leftarrow \text{through-sources (in links)} \\ \leftarrow \text{fundamental cut sets (across-sources)} \\ \leftarrow \text{fundamental cut sets (components)} \\ \leftarrow \text{fundamental circuits (components)} \\ \leftarrow \text{fundamental circuits (through-sources)} \end{matrix}$$

where the $\mathbf{I}$'s are identity matrices of the right dimensions, the $\mathbf{O}$'s are zero matrices of the right dimensions and the $\mathbf{0}$'s are zero vectors of the right dimensions.

Remark. *Here we assume that G has all these four types of arcs (across-source branch, component branch, through-source link and component link). In other cases (for example, when there are no through-sources), we leave the corresponding rows, columns and elements out of the system of equations. Other cases are treated in a similar way.*

Solving the equations, we get

$$\mathbf{u}_1 = \mathbf{s}_1 \quad , \quad \mathbf{u}_2 = \mathbf{R}_1\mathbf{i}_2 \quad , \quad \mathbf{u}_3 = \mathbf{R}_2\mathbf{i}_3 \quad , \quad \mathbf{i}_4 = \mathbf{s}_2$$

which leaves this system of equations:

$$\begin{pmatrix} \mathbf{I} & \mathbf{O} & \mathbf{Q}_{11} & \mathbf{O} \\ \mathbf{O} & \mathbf{I} & \mathbf{Q}_{21} & \mathbf{O} \\ \mathbf{O} & \mathbf{B}_{12}\mathbf{R}_1 & \mathbf{R}_2 & \mathbf{O} \\ \mathbf{O} & \mathbf{B}_{22}\mathbf{R}_1 & \mathbf{O} & \mathbf{I} \end{pmatrix} \begin{pmatrix} \mathbf{i}_1 \\ \mathbf{i}_2 \\ \mathbf{i}_3 \\ \mathbf{u}_4 \end{pmatrix} = - \begin{pmatrix} \mathbf{Q}_{12}\mathbf{s}_2 \\ \mathbf{Q}_{22}\mathbf{s}_2 \\ \mathbf{B}_{11}\mathbf{s}_1 \\ \mathbf{B}_{21}\mathbf{s}_1 \end{pmatrix}.$$

Thus,

$$\mathbf{i}_1 = -\mathbf{Q}_{11}\mathbf{i}_3 - \mathbf{Q}_{12}\mathbf{s}_2 \quad , \quad \mathbf{i}_2 = -\mathbf{Q}_{21}\mathbf{i}_3 - \mathbf{Q}_{22}\mathbf{s}_2 \quad , \quad \mathbf{u}_4 = -\mathbf{B}_{22}\mathbf{R}_1\mathbf{i}_2 - \mathbf{B}_{21}\mathbf{s}_1.$$

From the results in the previous section, we get

$$\mathbf{B}_{\mathrm{ft}} = \left(\begin{array}{c|c} \mathbf{B}_{11} & \mathbf{B}_{12} \\ \hline \mathbf{B}_{21} & \mathbf{B}_{22} \end{array}\right) = -\mathbf{Q}_{\mathrm{fc}}^{\mathrm{T}} = -\left(\begin{array}{c|c} \mathbf{Q}_{11} & \mathbf{Q}_{12} \\ \hline \mathbf{Q}_{21} & \mathbf{Q}_{22} \end{array}\right)^{\mathrm{T}} = \left(\begin{array}{c|c} -\mathbf{Q}_{11}^{\mathrm{T}} & -\mathbf{Q}_{21}^{\mathrm{T}} \\ \hline -\mathbf{Q}_{12}^{\mathrm{T}} & -\mathbf{Q}_{22}^{\mathrm{T}} \end{array}\right).$$

Therefore, $\mathbf{B}_{11} = -\mathbf{Q}_{11}^{\mathrm{T}}$ and $\mathbf{B}_{12} = -\mathbf{Q}_{21}^{\mathrm{T}}$. Finally, there is only one set of equations for $\mathbf{i}_3$:

$$(\mathbf{Q}_{21}^{\mathrm{T}}\mathbf{R}_1\mathbf{Q}_{21} + \mathbf{R}_2)\mathbf{i}_3 = \mathbf{Q}_{11}^{\mathrm{T}}\mathbf{s}_1 - \mathbf{Q}_{21}^{\mathrm{T}}\mathbf{R}_1\mathbf{Q}_{22}\mathbf{s}_2.$$

The matrix[2] of this system of equations can written as

$$\mathbf{Q}_{21}^{\mathrm{T}}\mathbf{R}_1\mathbf{Q}_{21} + \mathbf{R}_2 = \left(\begin{array}{c|c} \mathbf{Q}_{21}^{\mathrm{T}} & \mathbf{I} \end{array}\right) \left(\begin{array}{c|c} \mathbf{R}_1 & \mathbf{O} \\ \hline \mathbf{O} & \mathbf{R}_2 \end{array}\right) \left(\begin{array}{c} \mathbf{Q}_{21} \\ \hline \mathbf{I} \end{array}\right).$$

We can see that it is not singular if the diagonal elements of $\mathbf{R}_1$ and $\mathbf{R}_2$ are all positive or all negative.

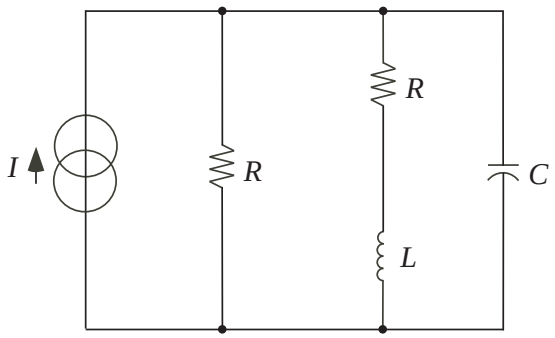
Therefore, we get

Theorem 4.6. *If the constants r_j are real numbers with the same sign, then the fundamental equations of the stationary linear network have a unique solution exactly when no cut of the network consists of only through-sources and no circuit of the network consists of only across-sources.*

From the theorem above, we notice that the number of equations we have to solve (numerically) is considerably fewer than $2m$.

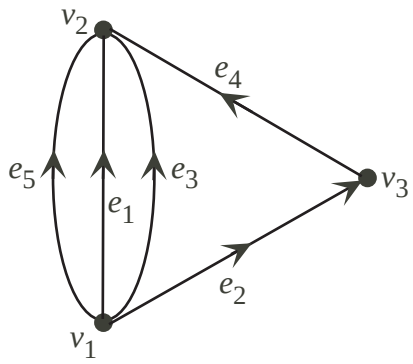
Example. *A mono-frequency AC circuit with passive elements (resistors, capacitors and inductors) can also be modelled as a stationary linear network (Theorem 4.6 does not apply). In the circuit below, the component values are $R = 10\ \Omega$, $C = 100\ \mu\mathrm{F}$, $L = 10\ \mathrm{mH}$ and the current source is*

$$I = 10\cos(1000t)\ \mathrm{A}.$$



The complex current of the source is $10e^{j1000t}$, where j is the imaginary unit. The (angular) frequency is $\omega = 1000\ \mathrm{rad/s}$. There are no voltage sources. The corresponding digraph is

[2]This matrix is called the *admittance matrix*. Similarly, the *impedance matrix* can be constructed from the blocks of the fundamental circuit matrix



The voltages and currents written as complex exponentials are $i_k = I_k e^{j1000t}$ *and* $u_k = U_k e^{j1000t}$. *In particular, the current source is* $i_5 = s_5 = 10e^{j1000t}$. *We get* r_k *from the familiar formulae from electrical circuit analysis:*

$$r_1 = r_4 = R = 10 \quad , \quad r_3 = \frac{1}{j\omega C} = -10j \quad , \quad r_2 = j\omega L = 10j.$$

We choose the arcs e_1 *and* e_2 *as the branches of the spanning tree* T. *Because of the linearity of the system of equations, the exponential factors* e^{j1000t} *cancel out and we get*

$$\left(\begin{array}{cc|cc|c|ccc|cc} -10 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & -10j & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ \hline 0 & 0 & 10j & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & -10 & 0 & 0 & 0 & 0 & 1 & 0 \\ \hline 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ \hline 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & -1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 & 1 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 1 \end{array}\right) \left(\begin{array}{c} I_1 \\ I_2 \\ \hline I_3 \\ I_4 \\ \hline I_5 \\ \hline U_1 \\ U_2 \\ \hline U_3 \\ U_4 \\ \hline U_5 \end{array}\right) = \left(\begin{array}{c} 0 \\ 0 \\ \hline 0 \\ 0 \\ \hline 10 \\ \hline 0 \\ 0 \\ \hline 0 \\ 0 \\ \hline 0 \end{array}\right).$$

Notice that we have left out the across-sources because there are none. This system is easily solved using computer programs, e.g. MATLAB:

```
»H=[-10      0     0    0  0   1  0  0  0  0 ;
      0  -10*j     0    0  0   0  1  0  0  0 ;
      0      0  10*j    0  0   0  0  1  0  0 ;
      0      0     0  -10  0   0  0  0  1  0 ;
      0      0     0    0  1   0  0  0  0  0 ;
      1      0     1    1  1   0  0  0  0  0 ;
      0      1     0   -1  0   0  0  0  0  0 ;
      0      0     0    0  0  -1  0  1  0  0 ;
      0      0     0    0  0  -1  1  0  1  0 ;
      0      0     0    0  0  -1  0  0  0  1];
»s=[0 0 0 0 10 0 0 0 0 0]';
»UV=inv(H)*s;
»[UV angle(UV) abs(UV)]

ans =

  -6.0000 + 2.0000i   2.8198            6.3246
  -2.0000 + 4.0000i   2.0344            4.4721
  -2.0000 - 6.0000i  -1.8925            6.3246
  -2.0000 + 4.0000i   2.0344            4.4721
  10.0000                  0           10.0000
 -60.0000 +20.0000i   2.8198           63.2456
```

```
-40.0000 -20.0000i  -2.6779           44.7214
-60.0000 +20.0000i   2.8198           63.2456
-20.0000 +40.0000i   2.0344           44.7214
-60.0000 +20.0000i   2.8198           63.2456
```

Thus, for example, the complex voltage across the current source is

$$u_5 = U_5 e^{j1000t} = 63.25 e^{j(1000t+2.82)}$$

and the real voltage is $63.25\cos(1000t + 2.82)$ V.

Kirchhoff's Through-Quantity Law can also be written in the form

$$\mathbf{A}\mathbf{i} = \mathbf{0}_n,$$

where $\mathbf{A}$ is the all-vertex incidence matrix of G. Furthermore,

$$\mathbf{A}^{\mathrm{T}}\mathbf{p} = -\mathbf{u}.$$

Hence

$$\mathbf{u} \bullet \mathbf{i} = \mathbf{u}^{\mathrm{T}}\mathbf{i} = -\mathbf{p}^{\mathrm{T}}\mathbf{A}\mathbf{i} = 0.$$

This result only depends on the structure of the digraph G (through the all-vertex incidence matrix). Now we get the famous theorem:

Theorem 4.7. (Tellegen) *If two stationary linear networks have the same digraph with corresponding through-vectors* $\mathbf{i}_1$ *and* $\mathbf{i}_2$ *as well as corresponding across-vectors* $\mathbf{u}_1$ *and* $\mathbf{u}_2$*, then*

$$\mathbf{u}_1 \bullet \mathbf{i}_2 = 0 \quad \textit{and} \quad \mathbf{u}_2 \bullet \mathbf{i}_1 = 0.$$

If we apply this to the case when the two networks are exactly the same ($= G$), then we get

$$\mathbf{u} \bullet \mathbf{i} = 0,$$

known as the *Law of Conservation of Energy*.

Remark. *More details on this subject can be found e.g. in* SWAMY & THULASIRAMAN *or* VÁGÓ*, as well as* DOLAN & ALDOUS.

4.5 Matrices over $\mathrm{GF}(2)$ and Vector Spaces of Graphs

The set $\{0, 1\}$ is called a *field* (i.e. it follows the same arithmetic rules as real numbers) if addition and multiplication are defined as follows:

$+$	0	1
0	0	1
1	1	0

$\times$	0	1
0	0	0
1	0	1

In this case $-1 = 1$ and $1^{-1} = 1$. This is the field $\mathrm{GF}(2)$.

If we think of the elements 0 and 1 of the all-vertex incidence, cut, fundamental cut, circuit and fundamental circuit matrices of a ("undirected") graph as elements of the field $\mathrm{GF}(2)$, then Theorems 4.1, 4.2, 4.4, 4.5 and their corollaries also apply to "undirected graphs". (Keep in mind that $-1 = 1$ in the field $\mathrm{GF}(2)$.) The proofs are the same.

For ”undirected” graphs, the vector spaces are over the field $\mathrm{GF}(2)$. For directed graphs, the vector spaces are real (i.e. over the field $\mathbb{R}$). The row space of the cut matrix of a (di)graph is the *cut space*. Similarly, the row space of the circuit matrix is the *circuit space*. The dimension of the cut space is the rank of the (di)graph and the dimension of the circuit space is the nullity of the (di)graph. Furthermore, the cut space and the circuit space are orthogonal complements. (All of these statements follow directly from the results above.)

Often, we deal with the above mentioned spaces through subgraphs, i.e. we identify a vector with the subgraph generated by the corresponding arcs. In the case of ”undirected” graphs, the addition of $\mathrm{GF}(2)$ vectors corresponds to the ring sum operation.

Chapter 5

Graph Algorithms

5.1 Computational Complexity of Algorithms

The *complexity* of a problem is related to the resources required to compute a solution as a function of the size of the problem. The size of a problem is measured by the size of the input N, and the resources required are usually measured by time (number of steps) and space (maximum amount of memory measured appropriately). *Decision problems* or *yes-or-no questions* are very common. Read HOPCROFT & ULLMAN for classical complexity theory.

To make computational complexities comparable, we need to agree on some specific mathematical models for algorithms. For example, consider computing with Turing Machines and refer to courses in Formal Languages, Theory of Automata and Mathematical Logic. We have *deterministic* and *nondeterministic* version of algorithm models. In the deterministic version, there are no choices to be made. In the nondeterministic version, there is a choice to be made somewhere on the way. For a nondeterministic algorithm, we have to make the following assumptions so that we can actually solve problems:

1. The algorithm terminates at some point no matter how we choose the steps.
2. The algorithm can terminate without yielding a solution.
3. When the algorithm terminates and yields a solution, the solution is correct (it is possible to have more than one solution).
4. For decision problems, if the algorithm fails to give a positive answer (yes), then the answer is interpreted to be negative (no).
5. If the problem is to compute a value, then the nondeterministic algorithm has to have a solution for every input (value of the function).

Nondeterministic algorithms are best treated as *verification procedures* for problems rather than procedures for producing answers.

Computational complexity is considered *asymptotically*, that is for large problems, time or space complexities that differ by constant coefficients are not distinguished because linear acceleration and compression of space are easy to perform in any kind of algorithm model. Although the choice of an algorithm model has a clear impact on the complexity, it is not an essential characteristic, i.e. it does not change the complexity class. Often, we use the *big-*O *notation* for complexities. $\mathrm{O}(f(N))$ refers to the class of functions $g(N)$ such that if $N \geq N_0$ holds, then $|g(N)| \leq Cf(N)$ holds, where C is a constant.

Without exploring algorithm models any further, we define a couple of important complexity classes. The time complexity class $\mathcal{P}$ (*deterministic polynomial time problems*) consists of problems of (input) size N where it takes at most $p(N)$ steps to solve the problem using deterministic algorithms. $p(N)$ is some problem dependent polynomial of N. The time complexity class $\mathcal{NP}$ (*nondeterministic polynomial time problems*) consists of problems of size N where it takes at most $p(N)$ steps to solve the problem using nondeterministic algorithms. Once again, $p(N)$ is some problem dependent polynomial of N.

Time complexity class $\mathrm{co}{-}\mathcal{NP}$ (*complements of nondeterministic polynomial time problems*) consists of decision problems whose complements are in $\mathcal{NP}$. (The *complement* of a problem is obtained by swapping the positive and the negative answer.)

Obviously, $\mathcal{P} \subseteq \mathcal{NP}$ and (for decision problems) $\mathcal{P} \subseteq \mathrm{co}{-}\mathcal{NP}$. Whether or not the inclusion is proper is an open problem, actually quite a famous problem. It is widely believed that both of the inclusions are proper. It is not known if the following holds for decision problems:

$$\mathcal{NP} = \mathrm{co}{-}\mathcal{NP} \quad \text{or} \quad \mathcal{P} = \mathcal{NP} \cap \mathrm{co}{-}\mathcal{NP}$$

Most researchers believe that they do not hold.

The space complexity class $\mathcal{PSPACE}$ (*deterministic polynomial space problems*) consists of problems of (input) size N where it takes at most $p(N)$ memory units to solve the problem using deterministic algorithms. $p(N)$ is some problem dependent polynomial of N. The space complexity class $\mathcal{NPSPACE}$ (*nondeterministic polynomial space problems*) consists of problems of size N where it takes at most $p(N)$ memory units to solve the problem using nondeterministic algorithms. Once again, $p(N)$ is some problem dependent polynomial of N. It is known that

$$\mathcal{NP} \subseteq \mathcal{PSPACE} = \mathcal{NPSPACE},$$

but it is not known whether the inclusion is proper or not.

An algorithm may include some ideally generated random numbers. The algorithm is then called *probabilistic* or *stochastic*. The corresponding polynomial time complexity class is $\mathcal{BPP}$ (*random polynomial time problems* or *bounded-error probabilistic polynomial time problems*). Some stochastic algorithms may fail occasionally, that is, they produce no results and terminate prematurely. These algorithms are called *Las Vegas algorithms*. Some stochastic algorithms may also produce wrong answers (ideally with a small probability). These kind of algorithms are called *Monte Carlo algorithms*. Some stochastic algorithms seldom yield exact solutions. Nevertheless, they give accurate approximate solutions with great probablity. These kind of algorithms are called *approximation algorithms*.

The task of an algorithm may be to convert a problem to another. This is known as *reduction*. If problem A can be reduced to another problem B by using a (deterministic) polynomial time algorithm, then we can get a polynomial time algorithm for problem A from a polynomial time algorithm B. A problem is $\mathcal{NP}$*-hard* if every problem in $\mathcal{NP}$ can be reduced to it by a polynomial time algorithm. $\mathcal{NP}$-hard problems are $\mathcal{NP}$*-complete* if they are actually in $\mathcal{NP}$. $\mathcal{NP}$-complete problems are the "worst kind". If any problem in $\mathcal{NP}$ could be shown to be deterministic polynomial time, then every problem in $\mathcal{NP}$ would be in $\mathcal{P}$ and $\mathcal{P} = \mathcal{NP}$. Over one thousand $\mathcal{NP}$-complete problems are known currently.

The old division of problems into *tractable* and *intractable* mean that $\mathcal{P}$ problems are tractable and others are not. Because we believe that $\mathcal{P} \neq \mathcal{NP}$ in general, $\mathcal{NP}$-complete problems are intractable. In the following, graph algorithms are either in $\mathcal{P}$ or they are approximations of some more demanding problems. The size of an input can be for example the number of nonzero elements in an incidence matrix, the number of vertices n or the number of edges m or some combination of n and m.

5.2 Reachability: Warshall's Algorithm

We only deal with directed graphs in this section. The results also hold for "undirected" graphs if we interpret an edge as a pair of arcs in opposite directions.

Problem. *We are given an adjacency matrix of the digraph $G = (V, E)$. We are to construct the* reachability matrix $\mathbf{R} = (r_{ij})$ *of G, where*

$$r_{ij} = \begin{cases} 1 \text{ if } G \text{ has a directed } v_i\text{–}v_j \text{ path} \\ 0 \text{ otherwise.} \end{cases}$$

(Note that $V = \{v_1, \ldots, v_n\}$.) In particular, we should note that if $r_{ii} = 1$, then v_i is in a directed circuit.

Warshall's Algorithm constructs a series of $n \times n$ matrices $\mathbf{E}_1, \ldots, \mathbf{E}_n$ where

1. elements of $\mathbf{E}_i$ are either zero or one.
2. $\mathbf{E}_i \leq \mathbf{E}_{i+1}$ ($i = 0, \ldots, n-1$) (comparison is done element by element).
3. $\mathbf{E}_0$ is obtained from the adjacency matrix $\mathbf{D}$ by replacing the positive elements with ones.
4. $\mathbf{E}_n = \mathbf{R}$.

The algorithm is presented as a pseudocode:

```
procedure Warshall
begin
   E := E_0
   for i := 1 to n do
      for j := 1 to n do
         if (E)_ji = 1 then for k := 1 to n do
            (E)_jk := max((E)_jk, (E)_ik)
         fi
      od
   od
end
```

In this case, the maximizing operation is sometimes called the *Boolean sum*:

max	0	1
0	0	1
1	1	1

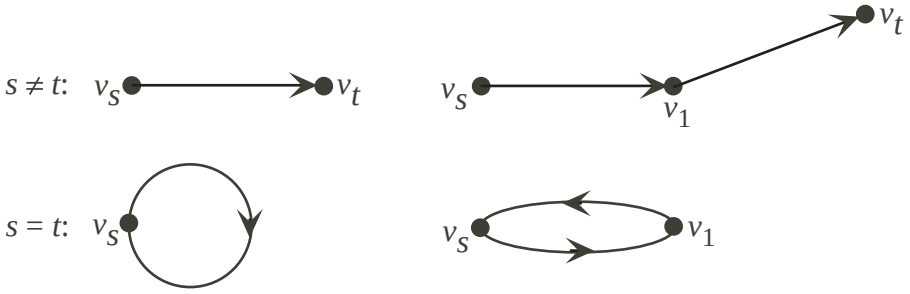
Let us show that Warshall's Algorithm gives us the desired results. Let $\mathbf{E}_i$ denote the value of $\mathbf{E}$ after i steps.

Statement. (i) *If there is a directed path from v_s to v_t such that apart from v_s and v_t, the path only includes vertices in the set $\{v_1, \ldots, v_i\}$, then $(\mathbf{E}_i)_{st} = 1$.*

(ii) *If vertex v_s belongs to a directed circuit whose other vertices are in the set $\{v_1, \ldots, v_i\}$, then $(\mathbf{E}_i)_{ss} = 1$.*

Proof. We will use induction on i.

Induction Basis: $i = 1$. $(\mathbf{E}_1)_{st} = 1$ if $(\mathbf{E}_0)_{st} = 1$, or $(\mathbf{E}_0)_{s1} = 1$ and $(\mathbf{E}_0)_{1t} = 1$. We have one of the following cases:



Induction Hypothesis: The statement is true for $i < \ell$. ($\ell \geq 2$)

Induction Statement: The statement is true for $i = \ell$.

Induction Statement Proof: Let us handle both statements together. The proof for (ii) is given in square brackets. We have two cases:

- v_ℓ belongs to the directed path [resp. directed circuit] but $\ell \neq s, t$ [resp. $\ell \neq s$]. Then, we use the Induction Hypothesis:

  $$(\mathbf{E}_{\ell-1})_{s\ell} = 1 \quad \text{and} \quad (\mathbf{E}_{\ell-1})_{\ell t} = 1 \quad \text{[resp.} \quad (\mathbf{E}_{\ell-1})_{s\ell} = 1 \quad \text{and} \quad (\mathbf{E}_{\ell-1})_{\ell s} = 1\text{]},$$

  so $(\mathbf{E}_\ell)_{st} = 1$ [resp. $(\mathbf{E}_\ell)_{ss} = 1$].

- v_ℓ is either v_s or v_t [resp. v_ℓ is v_s] or it does not belong to the directed path [resp. directed circuit] at all. Then, by the Induction Hypothesis

  $$(\mathbf{E}_{\ell-1})_{st} = 1 \quad \text{[resp.} \quad (\mathbf{E}_{\ell-1})_{ss} = 1\text{]},$$

  so $(\mathbf{E}_\ell)_{st} = 1$ [resp. $(\mathbf{E}_\ell)_{ss} = 1$]. □

In Warshall's Algorithm, the maximizing operation is performed at most n^3 times.

5.3 Depth-First and Breadth-First Searches

Problem. *We have to traverse through a (di)graph to find some kind of vertices or edges.*

We assume that the (di)graph is connected and loopless. For disconnected graphs, we have to go through the components separately. We ignore loops if the (di)graph has any.

Depth-First Search, DFS, has many uses. The procedure is a bit different for undirected graphs and directed graphs. Therefore they will be treated separately.

Undirected Graphs

We choose a starting vertex r (*root*) to start the search. Then, we traverse an edge $e = (r, v)$ to go to the vertex v. At the same time, we *direct* e from r to v. Now, we say that the edge e is *examined* and we call it a *tree edge*. The vertex r is called the *father* of v and we denote it $r = \mathrm{FATHER}(v)$.

We continue the search. At a vertex x, there are two cases:

(1) If every edge connected to x has been examined, return to the father of x and continue the process from $\mathrm{FATHER}(x)$. The vertex x is said to be *completely scanned*.

(2) If there exist some unexamined edges connected to x, then we choose one such edge $e = (x, y)$ and direct it from x to y. This edge is now said to be *examined*. We have two subcases now:

(2.1) If y has not been visisted before, then we traverse the edge (x, y), visit y and continue the search from y. In this case, e is a *tree edge* and $\mathrm{FATHER}(y) = x$.

(2.2) If y has been visited before, then we select some other unexamined edge connected to x. In this case, the edge e is called a *back edge*.

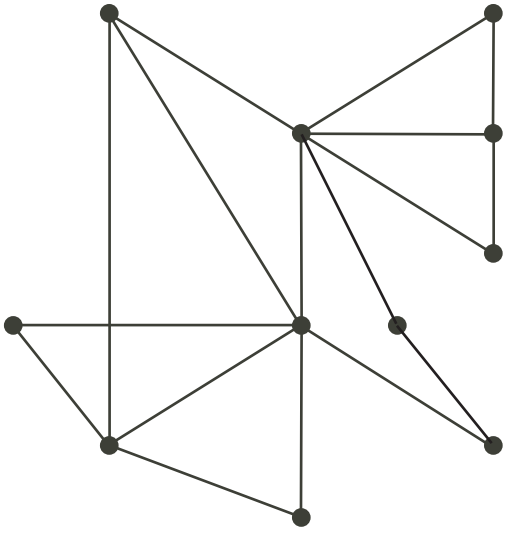
Every time we come to a new vertex which has never been visited before, we give it a distinct number. The number of the root is 1. We write

$$\mathrm{DFN}(x) = \text{running number of vertex } x.$$

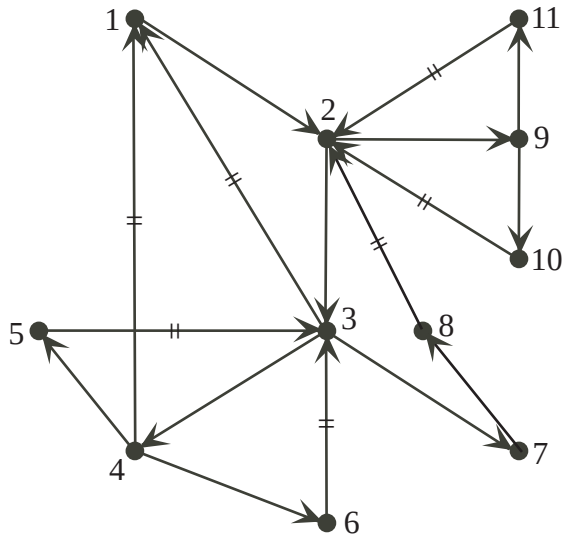
A complete DFS ends when we traverse back to the root and we have visited every vertex or when we have found the desired edge/vertex.

DFS divides the edges of G into tree edges and back edges. Obviously, the tree edges form a spanning tree of G, also known as a *DFS tree*. If we include the directions of the tree edges, we get a *directed DFS tree*. DFS gives a direction to every edge in G. When we use these directions, we get a digraph whose underlying subgraph is G. It has the DFS tree as a directed spanning tree.

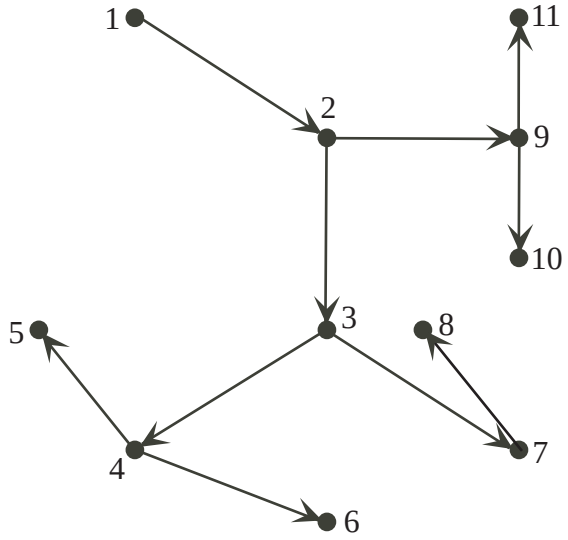
Example. *For the graph*



we start the DFS from a root in the upper left corner. The back edges are marked with two lines.



The corresponding DFS tree is



In the following, we denote,

$$K(x) = \begin{cases} 0 \text{ if vertex } x \text{ has not been visisted} \\ 1 \text{ if vertex } x \text{ has been visited} \end{cases}$$

and TREE and BACK are set variables containing the directed tree edges and back edges.

Depth-First Search for Graphs:

1. Set $\mathrm{TREE} \leftarrow \emptyset$, $\mathrm{BACK} \leftarrow \emptyset$ and $i \leftarrow 1$. For every vertex x of G, set $\mathrm{FATHER}(x) \leftarrow 0$ and $K(x) \leftarrow 0$.

2. Choose a vertex r for which $K(r) = 0$ (this condition is needed only for disconnected graphs, see step #6). Set $\mathrm{DFN}(r) \leftarrow i$, $K(r) \leftarrow 1$ and $u \leftarrow r$.

3. If every edge connected to u has been examined, go to step #5. Otherwise, choose an edge $e = (u, v)$ that has not been examined.

4. We direct edge e from u to v and label it examined.

   4.1 If $K(v) = 0$, then set $i \leftarrow i + 1$, $\mathrm{DFN}(v) \leftarrow i$, $\mathrm{TREE} \leftarrow \mathrm{TREE} \cup \{e\}$, $K(v) \leftarrow 1$, $\mathrm{FATHER}(v) \leftarrow u$ and $u \leftarrow v$. Go back to step #3.

   4.2 If $K(v) = 1$, then set $\mathrm{BACK} \leftarrow \mathrm{BACK} \cup \{e\}$ and go back to step #3.

5. If $\mathrm{FATHER}(u) \neq 0$, then set $u \leftarrow \mathrm{FATHER}(u)$ and go back to step #3.

6. (Only for disconnected graphs so that we can jump from one component to another.) If there is a vertex r such that $K(r) = 0$, then set $i \leftarrow i + 1$ and go back to step #2.

7. Stop.

We denote T as the DFS tree and $\vec{G}$ as the directed graph obtained from the algorithm. T is a directed spanning tree of $\vec{G}$. If there is a directed path from u to v in T, then we call u an *ancestor* of v and v a *descendent* of u. Vertices u and v are *related* if one of them is an ancestor of the other. In particular, if (u, v) is an edge of T, then u is the father of v and v is a *son* of u. An edge (u, v) of G, where u and v are unrelated, is called a *cross edge*. However,

Statement. *Cross edges do not exist.*

Proof. Let u and v be two distinct vertices which are unrelated. Then, (by quasi-strong connectivity) there are two vertices u' and v' such that

- $\mathrm{FATHER}(u') = \mathrm{FATHER}(v')$,
- $u' = u$ or u' is an ancestor of u and
- $v' = v$ or v' is an ancestor of v.

We examine the case where $\mathrm{DFN}(u') < \mathrm{DFN}(v')$ (the other case is obviously symmetrical). We label T_1 as the directed subtree of T whose root is u' and T_2 as the directed subtree of T whose root is v'. Obviously, DFS goes through the vertices of T_2 only after u' is completely scanned. Furthermore, u' is completely scanned only after all the vertices of T_1 are completely scanned. Hence, it is impossible to have an edge (u, v). □

Directed Graphs

Depth-first search in a (connected and loopless) digraph G is similar to the case for undirected graphs. The algorithm divides the arcs in G into four different classes. If the search proceeds to an unexamined arc $e = (x, y)$, then the four possible classes are:

(1) If y has not been visited, then e is a *tree edge*.

(2) If y has been visited, then there are three cases:

   (2.1) y is a descendent of x in the subgraph induced by existing tree edges. Then, e is a *forward edge* and $\mathrm{DFN}(y) > \mathrm{DFN}(x)$.

   (2.2) x is a descendent of y in the subgraph induced by the existing tree edges. Then, e is a *back edge* and $\mathrm{DFN}(y) < \mathrm{DFN}(x)$.

   (2.3) x and y are not related by any of the existing tree edges. Then, e is a *cross edge* and $\mathrm{DFN}(y) < \mathrm{DFN}(x)$. (*Note!* It is impossible that $\mathrm{DFN}(y) > \mathrm{DFN}(x)$. This is proven in the same way as we did previously.)

The directed subgraph of G induced by tree edges is called the *DFS forest* (directed forest).

If $\mathrm{DFN}(y) > \mathrm{DFN}(x)$ holds for the arc (x, y), then (x, y) is a tree edge or a forward edge. During the search, it is easy to distinguish the two because (x, y) is a tree edge if y has not been visited and it is a forward edge otherwise. If $\mathrm{DFN}(y) < \mathrm{DFN}(x)$, then (x, y) is a back edge or a cross edge. During the search, it is easy to distinguish the two because (x, y) is a cross edge if y is completely scanned and it is a back edge otherwise.

In the following, K, FATHER, TREE and BACK are defined as previously. We also have two new variables FORWARD and CROSS (their meanings are obvious) and

$$L(x) = \begin{cases} 1 \text{ if } x \text{ is completely scanned} \\ 0 \text{ otherwise.} \end{cases}$$

Depth-First Search for Digraphs:

1. Set $\mathrm{TREE} \leftarrow \emptyset$, $\mathrm{FORWARD} \leftarrow \emptyset$, $\mathrm{BACK} \leftarrow \emptyset$, $\mathrm{CROSS} \leftarrow \emptyset$ and $i \leftarrow 1$. For every vertex x in G, set $\mathrm{FATHER}(x) \leftarrow 0$, $K(x) \leftarrow 0$ and $L(x) \leftarrow 0$.

2. Choose a vertex r such that $K(r) = 0$ and set $\mathrm{DFN}(r) \leftarrow i$, $K(r) \leftarrow 1$ and $u \leftarrow r$.

3. If every arc coming out of u has already been examined, then set $L(u) \leftarrow 1$ and go to step #5. Otherwise, choose an unexamined arc $e = (u, v)$.

4. Label the arc e examined.

   4.1 If $K(v) = 0$, then set $i \leftarrow i + 1$, $\mathrm{DFN}(v) \leftarrow i$, $\mathrm{TREE} \leftarrow \mathrm{TREE} \cup \{e\}$, $K(v) \leftarrow 1$, $\mathrm{FATHER}(v) \leftarrow u$ and $u \leftarrow v$. Go to step #3.

   4.2 If $K(v) = 1$ and $\mathrm{DFN}(v) > \mathrm{DFN}(u)$, then set $\mathrm{FORWARD} \leftarrow \mathrm{FORWARD} \cup \{e\}$ and go to step #3.

   4.3 If $K(v) = 1$ and $\mathrm{DFN}(v) < \mathrm{DFN}(u)$ and $L(v) = 0$, then set $\mathrm{BACK} \leftarrow \mathrm{BACK} \cup \{e\}$ and go to step #3.

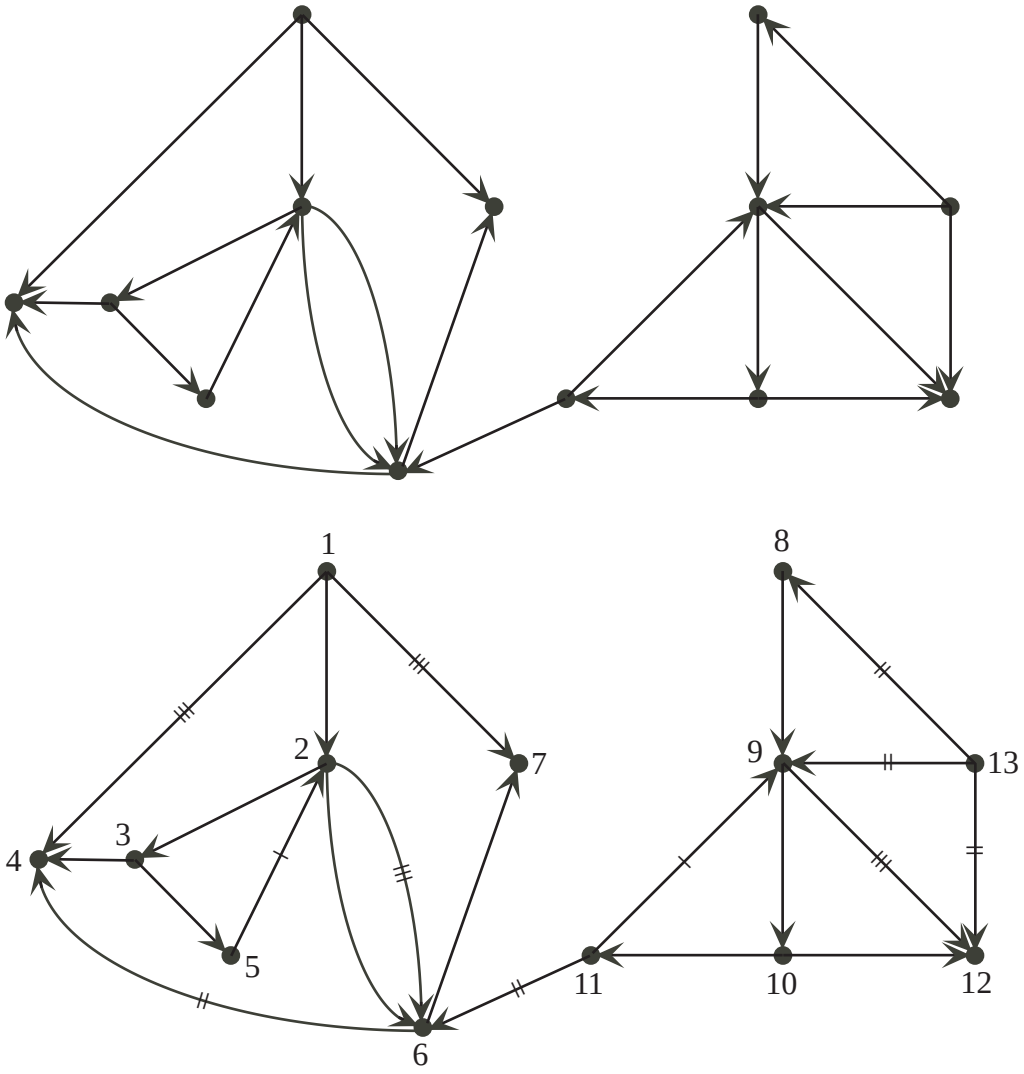
   4.4 If $K(v) = 1$ and $\mathrm{DFN}(v) < \mathrm{DFN}(u)$ and $L(v) = 1$, then set $\mathrm{CROSS} \leftarrow \mathrm{CROSS} \cup \{e\}$ and go to step #3.

5. If $\mathrm{FATHER}(u) \neq 0$, then set $u \leftarrow \mathrm{FATHER}(u)$ and to go step #3.

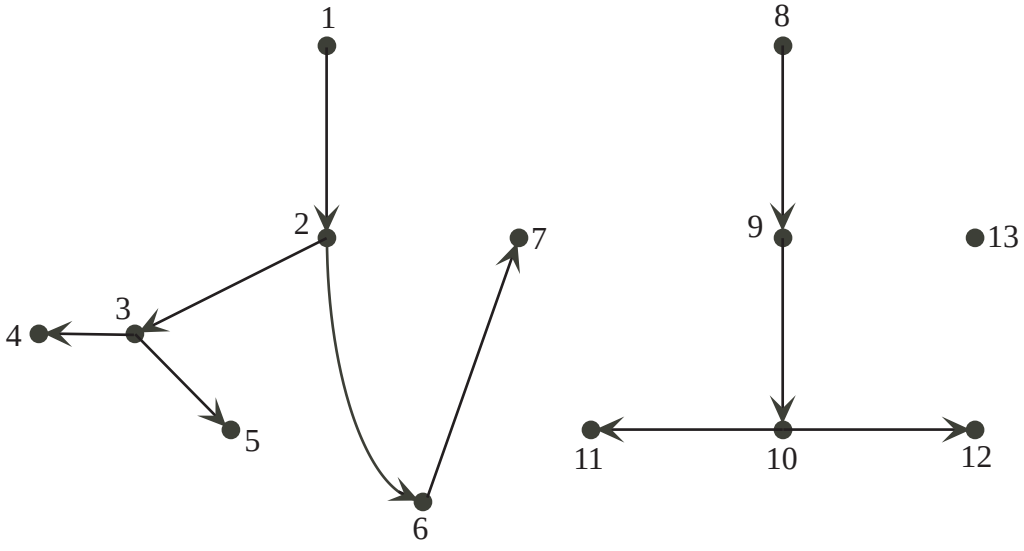
6. If there is a vertex r such that $K(r) = 0$, then set $i \leftarrow i + 1$ and go to step #2.

7. Stop.

Example. *DFS in the following digraph starts from a root in the upper left corner and proceeds like this (back edges are marked with one line, cross edges are marked with two lines and forward edges are marked with three lines):*



The corresponding DFS forest is



Theorem 5.1. *If a depth-first search in a quasi-strongly connected digraph starts from one of its roots, then the DFS forest is a directed tree. In particular, the DFS forest of a strongly connected digraph is a directed tree no matter where the search starts from.*

Proof. Let us prove by contradiction and consider the counter hypothesis: The DFS forest T resulted from a DFS in a quasi-strongly connected digraph G that began from root r is not a directed tree.

Since T is a directed forest, the component T_1 of T which has the root r does not contain some vertex v of G. On the other hand, there is a directed path from r to v. We choose the last vertex u on this path which is in T_1 and an arc $e = (u, w)$. Since the vertex w is not in T_1, the edge e is not a tree edge, a back edge nor a forward edge. Then, it must be a cross edge. Because the search began at r, the vertex w has to be in T_1 ($\surd$).

Strongly connected digraphs are also quasi-strongly connected and any vertex can be chosen as a root. □

Breadth-first search, BFS, is related to DFS. Let us consider a connected graph G.

Breadth-First Search for Graphs:

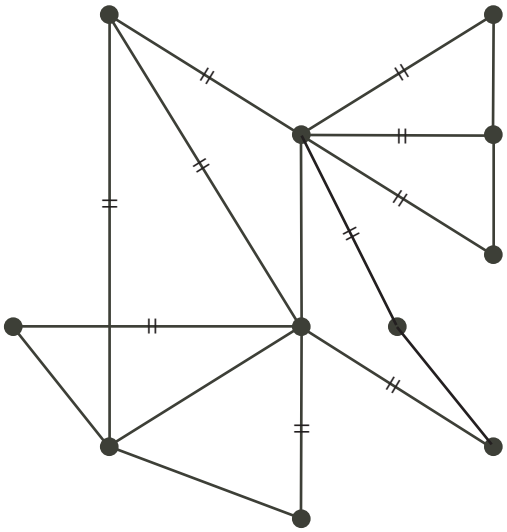
1. In the beginning, no vertex is labeled. Set $i \leftarrow 0$.
2. Choose a (unlabeled) starting vertex r (root) and label it with i.
3. Search the set J of vertices that are not labeled and are adjacent to some vertex labeled with i.
4. If $J \neq \emptyset$, then set $i \leftarrow i + 1$. Label the vertices in J with i and go to step #3.
5. (Only for disconnected graphs so we can jump from one component to another.) If a vertex is unlabeled, then set $i \leftarrow 0$ and go to step #2.
6. Stop.

BFS also produces a spanning tree, called the *BFS tree*, when we take the edges

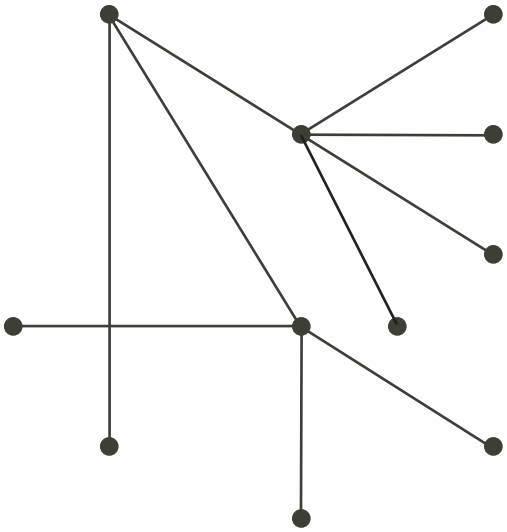
$$(\text{vertex labeled with } i, \text{unlabeled vertex})$$

while forming J. One such *tree edge* exists for each vertex in J. We obtain the *directed BFS tree* by orienting the edges away from the labeled vertex to the unlabeled vertex. BFS as presented above does not however orient every edge in the graph. Obviously, the label of a vertex is the length of the shortest path from the root to it, in other words, the *distance* from the root.

Example. *BFS in the graph we had in the previous example starts at a root in the upper left corner and proceeds as follows. (Tree edges are marked with two cross lines.)*



The corresponding BFS tree is



We obtain the directed BFS tree by orienting the branches away from the root.

BFS in a digraph G is very similar to what we just did.

Breadth-First Search for Digraphs:

1. In the beginning, no vertex is labeled. Set $i \leftarrow 0$.
2. Choose an unlabeled starting vertex r (root) and label it with i.
3. Search the set J of terminal vertices of arcs whose initial vertices have been labeled with i but whose terminal vertices have not been labeled.
4. If $J \neq \emptyset$, then set $i \leftarrow i + 1$. Label the vertices in J with i and go to step #3.
5. If not all vertices have been labeled, then set $i \leftarrow 0$ and go to step #2.
6. Stop.

BFS in a digraph produces a BFS forest (directed forest) when we take the examined arcs

$$(\text{vertices labeled with} i, \text{unlabeled vertices})$$

while forming J. One such *tree edge* exists for each vertex in J.

Remark. *In addition, BFS can be modified to sort the arcs like DFS.*

5.4 The Lightest Path: Dijkstra's Algorithm

Problem. *The edges of a (di)graph are given non-negative weights. The weight of a path is the sum of the weights of the path traversed. We are to find the lightest (directed) path in the (di)graph from vertex u to vertex v ($\neq u$) if the path exists (sometimes also called the shortest path). We should state if such path does not exist.*

Obviously, we can assume that we do not have any loops or parallel edges. Otherwise, we simply remove the loops and choose the edge with the lowest weight out of the parallel edges. From now on, we only consider directed graphs. Undirected graphs can be treated in the same way by replacing an edge with two arcs in opposite directions with the same weight.

We denote $\alpha(r, s)$ as the weight of the arc (r, s). *Dijkstra's Algorithm* marks the vertices as *permanent* or *temporary* vertices. The label of a vertex r is denoted $\beta(r)$ and we define

$$\gamma(r) = \begin{cases} 1 \text{ if the label is permanent} \\ 0 \text{ if the label is temporary.} \end{cases}$$

A permanent label $\beta(r)$ expresses the weight of the lightest directed u–r path. A temporary label $\beta(r)$ gives an upper limit to this weight (can be ∞). Furthermore, we denote:

$$\pi(r) = \begin{cases} \text{the predecessor of vertex } r \text{ on the lightest directed } u\text{–}r \text{ path if such a path exists} \\ 0 \text{ otherwise,} \end{cases}$$

so we can construct the directed path with the lowest weight.

Dijkstra's Algorithm:

1. Set $\beta(u) \leftarrow 0$ and $\gamma(u) \leftarrow 1$. For all other vertices r, set $\beta(r) \leftarrow \infty$ and $\gamma(r) \leftarrow 0$. For all vertices r, we set $\pi(r) \leftarrow 0$. Furthermore, set $w \leftarrow u$.

2. For every arc (w, r), where $\gamma(r) = 0$ and $\beta(r) > \beta(w) + \alpha(w, r)$, set

$$\beta(r) \leftarrow \beta(w) + \alpha(w, r) \quad \text{and} \quad \pi(r) \leftarrow w.$$

3. Find a vertex r^* for which $\gamma(r^*) = 0$, $\beta(r^*) < \infty$ and

$$\beta(r^*) = \min_{\gamma(r)=0} \{\beta(r)\}.$$

   Set

$$\gamma(r^*) \leftarrow 1 \quad \text{and} \quad w \leftarrow r^*.$$

   If there is no such vertex r^*, a directed u–v path does not exist and we stop.

4. If $w \neq v$, then go to step #2.

5. Stop.

We see that the algorithm is correct as follows. We denote (for every step):

$$V_1 = \{\text{permanently labeled vertices}\}$$
$$V_2 = \{\text{temporarily labeled vertices}\}.$$

($\langle V_1, V_2 \rangle$ is a cut with the completely scanned vertices on one side and other vertices on the other side.)

Statement. *The label $\beta(r)$ of the vertex r in V_1 is the weight of the lightest directed u–r path and $\pi(r)$ is the predecessor of r on such a path.*

Proof. After step #2, the temporary labels of r are always the weight of the directed u–r path with the lowest weight whose vertices are in V_1 except r ($= \infty$ if there is no such path), and $\pi(r)$ is a predecessor of r on this path (or $= 0$). This is because (two cases):

- Before step #2, $\beta(r) = \infty$. The only "new" vertex in V_1 is now w so every possible directed u–r path has to visit w. If there is no such path, then the case is obvious ($\beta(r)$ stays at ∞ and $\pi(r)$ stays at zero). Let us assume that we have the (lightest) directed u–r path that contains only vertices of V_1 and r as well. In particular, w is included. The subpath from u to w has of course the lowest weight. We consider the vertex s ($\in V_1$) which is the predecessor of r on the directed u–r path. If $s = w$, then the case is clear. If $s \neq w$, then s has been a w before, in which case $\beta(r)$ can not be $= \infty$ (step #2) ($\surd$).

- Before step #2, $\beta(r) < \infty$. Then, $\beta(r)$ is the weight of the lightest directed u–r path whose vertices are in $V_1 - \{w\}$ except for r. The only "new" vertex in V_1 is w so every possible lighter directed u–r path has to visit w. If there is no such path, then the case is obvious ($\beta(r)$ and $\pi(r)$ remain unchanged). Let us assume that we have a (lighter) directed u–r path that contains only vertices of V_1 and r as well. In particular, w is included. The subpath from u to w has of course the lowest weight. We consider the vertex s ($\in V_1$) which is a predecessor of r on the directed u–r path. If $s = w$, then the case is clear. If $s \neq w$, then s is in $V_1 - \{w\}$. Since s has been a w before, there is a lightest directed u–s path that does not contain w (otherwise, we should have chosen r^* in step #3 to be some predecessor of s on the directed u–w–s path). Then, we get a directed u–r path with a lower weight that contains r and only vertices in $V_1 - \{w\}$ ($\surd$).

The permanent label is the weight we seek because of the minimization in step #3 and $\pi(r)$ gives a predecessor of r as we claimed. □

At the end of the algorithm, vertex v gets a permanent label or the process stops at step #3 (which means a directed u–v path does not exist). The directed path with the lowest weight can be obtained by starting from the vertex v and finding the predecessors by using the label π.

If we replace step #4 by

4'. Go to step #2.

and continue the process until it stops at step #3, we get

$$\beta(w) = \begin{cases} 0 \text{ if } w = u \\ \text{the weight of the lightest directed } u\text{–}w \text{ path if there is one} \\ \infty \text{ otherwise} \end{cases}$$

and

$$\pi(w) = \begin{cases} \text{the predecessor of } w \text{ on the lightest directed } u\text{–}w \text{ path if there is one and } w \neq u \\ 0 \text{ otherwise.} \end{cases}$$

Remark. *Dijkstra's algorithm may fail if there are negative weights. These cases are investigated in the next section.*

5.5 The Lightest Path: Floyd's Algorithm

Problem. *We are to find the lighest path from vertex u to vertex v ($\neq u$) in a digraph or to show that there is no such path when the arcs of the digraph have been assigned arbitrary weights. Note that the weight of a directed path is the sum of the weights of the arcs traversed.*

Obviously, we can assume there are no loops or parallel arcs. Otherwise, we simply remove the loops and choose the arc with the lowest weight out of the parallel arcs. Floyd's Algorithm only works for digraphs. We write the weight of (x, y) as $\alpha(x, y)$ and construct the *weight matrix* $\mathbf{W} = (w_{ij})$ where

$$w_{ij} = \begin{cases} \alpha(v_i, v_j) \text{ if there is an arc } (v_i, v_j) \\ \infty \text{ otherwise.} \end{cases}$$

(Once again, $V = \{v_1, \ldots, v_n\}$ is the vertex set of the digraph.) Floyd's Algorithm is similar to Warshall's Algorithm. It only works if the digraph has no negative cycles, i.e. no cycle in the digraph has a negative weight. In this case, the lightest directed path is the lightest directed walk.

Floyd's Algorithm constructs a sequence of matrices $\mathbf{W}_0, \mathbf{W}_1, \ldots, \mathbf{W}_n$ where $\mathbf{W}_0 = \mathbf{W}$ and

$$(\mathbf{W}_k)_{ij} = \text{ weight of the lightest directed } v_i\text{–}v_j \text{ path, where there are only vertices } v_1, \ldots, v_k \text{ on the path besides } v_i \text{ and } v_j$$

($= \infty$ if there is no such path).

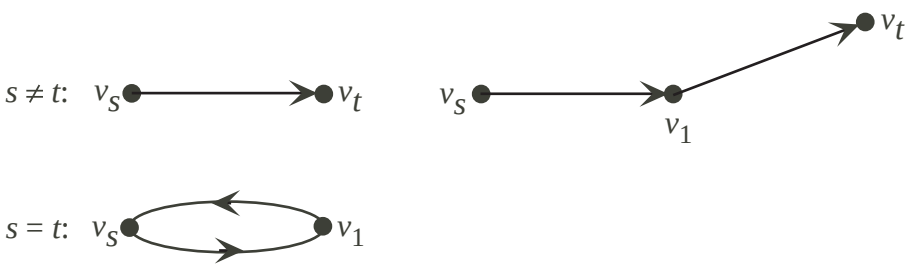
Statement. *When $\mathbf{W}_k$ is computed from $\mathbf{W}_{k-1}$ by the formula*

$$(\mathbf{W}_k)_{st} = \min\{(\mathbf{W}_{k-1})_{st}, (\mathbf{W}_{k-1})_{sk} + (\mathbf{W}_{k-1})_{kt}\},$$

then we get the previously mentioned sequence of weight matrices. If the digraph has negative cycles, then the sequence is correct up to the point when one of the diagonal elements turns negative for the first time.

Proof. We use induction on k.

Induction Basis: $k = 1$. Since the digraph is loopless, the diagonal elements of $\mathbf{W}_0$ can only be ∞ and the lightest directed path (if there is one) is one of the following, and the statement is obvious:



Induction Hypothesis: The statement is true for $k < \ell$. ($\ell \geq 2$)
Induction Statement: The statement is true for $k = \ell$.
Induction Statement Proof: The diagonal elements of $\mathbf{W}_{\ell-1}$ have to be nonnegative (∞ is permitted) for us to get this k. Let us consider the case where $s \neq t$. (The case $s = t$ is analogous.) We have five cases:

- Vertex v_ℓ is on the lightest directed path but it is not v_s or v_t, i.e. $\ell \neq s, t$. Let us consider the directed subpath from v_s to v_ℓ whose vertices other than v_s and v_ℓ are in $\{v_1, \ldots, v_{\ell-1}\}$. Suppose the lightest directed v_s–v_ℓ path of this kind has common vertices with the directed subpath from v_ℓ to v_t other than v_ℓ itself, e.g. v_p. The directed v_s–v_p–v_ℓ–v_t walk we get would be lighter than the original directed v_s–v_t path. By removing cycles, we would get a directed v_s–v_t path that would be lighter and would only contain v_s as well as v_t and the vertices $v_1, \ldots, v_\ell$ ($\surd$). (We have to remember that weights of cycles are not negative!) Therefore, the directed subpath from v_s to v_ℓ is the lightest directed v_s–v_ℓ path which contains the vertices $v_1, \ldots, v_{\ell-1}$ as well as v_s and v_ℓ. Similarly, the directed subpath from v_ℓ to v_t is the lightest directed v_ℓ–v_t path which contains the vertices $v_1, \ldots, v_{\ell-1}$ as well as v_t and v_ℓ. Now, we use the Induction Hypothesis:

  $$(\mathbf{W}_\ell)_{st} < (\mathbf{W}_{\ell-1})_{st}$$

  (check the special case $(\mathbf{W}_{\ell-1})_{st} = \infty$) and

  $$(\mathbf{W}_\ell)_{st} = (\mathbf{W}_{\ell-1})_{s\ell} + (\mathbf{W}_{\ell-1})_{\ell t}.$$

- The directed v_s–v_t path with the lowest weight exists and $v_\ell = v_s$. By the Induction Hypothesis, $(\mathbf{W}_\ell)_{st} = (\mathbf{W}_{\ell-1})_{st}$ and

  $$(\mathbf{W}_{\ell-1})_{s\ell} + (\mathbf{W}_{\ell-1})_{\ell t} = (\mathbf{W}_{\ell-1})_{\ell\ell} + (\mathbf{W}_{\ell-1})_{\ell t} \geq (\mathbf{W}_{\ell-1})_{\ell t} = (\mathbf{W}_{\ell-1})_{st},$$

  so $(\mathbf{W}_{\ell-1})_{\ell\ell} \geq 0$ (possibly $= \infty$).

- The directed v_s–v_t path exists and $v_\ell = v_t$. By the Induction Hypothesis, $(\mathbf{W}_\ell)_{st} = (\mathbf{W}_{\ell-1})_{st}$ and

  $$(\mathbf{W}_{\ell-1})_{s\ell} + (\mathbf{W}_{\ell-1})_{\ell t} = (\mathbf{W}_{\ell-1})_{s\ell} + (\mathbf{W}_{\ell-1})_{\ell\ell} \geq (\mathbf{W}_{\ell-1})_{s\ell} = (\mathbf{W}_{\ell-1})_{st},$$

  so $(\mathbf{W}_{\ell-1})_{\ell\ell} \geq 0$ (possibly $= \infty$).

- The lightest directed v_s–v_t exists but v_ℓ is not on the path. Now, we construct the lightest directed v_s–v_ℓ path and the lightest v_ℓ–v_t path which, in addition to the end vertices, contain only vertices $v_1, \ldots, v_{\ell-1}$, if it is possible. By combining these two paths, we get a directed v_s–v_t walk. By removing possible cycles from this walk, we get an as light or even lighter v_s–v_t path, which only contains vertices $v_1, \ldots, v_\ell$ as well as v_s and v_t. (We have to remember that weights of cycles are not negative!) Therefore, this is the case

  $$(\mathbf{W}_{\ell-1})_{s\ell} + (\mathbf{W}_{\ell-1})_{\ell t} \geq (\mathbf{W}_{\ell-1})_{st}$$

  and the equation in the statement gives the right result. If there is not a directed v_s–v_ℓ path or v_ℓ–v_t path, then it is obvious.

- The lightest directed v_s–v_t path does not exist. Then, $(\mathbf{W}_\ell)_{st} = \infty$ and $(\mathbf{W}_{\ell-1})_{st} = \infty$. On the other hand, at least one of the elements $(\mathbf{W}_{\ell-1})_{s\ell}$ or $(\mathbf{W}_{\ell-1})_{\ell t}$ is $= \infty$ because otherwise we would get a directed v_s–v_t path by combining the v_s–v_ℓ path with the v_ℓ–v_t path as well as removing all possible cycles ($\surd$). □

Floyd's Algorithm also constructs another sequence of matrices $\mathbf{Z}_0, \ldots, \mathbf{Z}_n$ in which we store the lightest directed paths in the following form

$$(\mathbf{Z}_k)_{ij} = \begin{cases} \ell \text{ where } v_\ell \text{ is the vertex following } v_i \text{ on the lightest directed} \\ \quad v_i\text{–}v_j \text{ path containing only vertices } v_i \text{ and } v_j \text{ as well as } v_1, \ldots, v_k \\ \quad \text{(if such a path exists)} \\ 0 \text{ otherwise.} \end{cases}$$

Obviously,

$$(\mathbf{Z}_0)_{ij} = \begin{cases} j \text{ if } (\mathbf{W})_{ij} \neq \infty \\ 0 \text{ otherwise.} \end{cases}$$

The matrix $\mathbf{Z}_k$ ($k \geq 1$) of the sequence can be obtained from the matrix $\mathbf{Z}_{k-1}$ by

$$(\mathbf{Z}_k)_{ij} = \begin{cases} (\mathbf{Z}_{k-1})_{ik} \text{ if } (\mathbf{W}_{k-1})_{ik} + (\mathbf{W}_{k-1})_{kj} < (\mathbf{W}_{k-1})_{ij} \\ (\mathbf{Z}_{k-1})_{ij} \text{ otherwise,} \end{cases}$$

so the sequence can be constructed with the sequence $\mathbf{W}_0, \mathbf{W}_1, \ldots, \mathbf{W}_n$ at the same time. Finally, Floyd's Algorithm is presented in the following pseudocode. We have added a part to test if there are negative elements on the diagonal and the construction of the $\mathbf{Z}_0, \ldots, \mathbf{Z}_n$ sequence of matrices.

```
procedure Floyd
begin
   W := W_0
   k := 0
   for i := 1 to n do
      for j := 1 to n do
         if (W)_ij = ∞ then
            (Z)_ij := 0
         else
            (Z)_ij := j
         fi
      od
   od
   while k < n and Test(W) do
      Iteration(W, Z, k)
   od
end

subprocedure Test(W)
begin
   for i := 1 to n do
      if (W)_ii < 0 then
         return FALSE
      fi
   od
   return TRUE
end
```

```
subprocedure Iteration(W, Z, k)
begin
   k := k + 1
   for i := 1 to n do
      for j := 1 to n do
         if (W)_ik + (W)_kj < (W)_ij then
            (W)_ij := (W)_ik + (W)_kj
            (Z)_ij := (Z)_ik
         fi
      od
   od
end
```

5.6 The Lightest Spanning Tree: Kruskal's and Prim's Algorithms

Problem. *We have to find the spanning tree with the lowest weight of a connected graph if the edges of the graph have been weighted arbitrarily and the weight of a tree is the sum of all the weights of the branches.*

Obviously, we can assume that the graph $G = (V, E)$ is nontrivial and simple. Otherwise, we simply remove the loops and choose the edge with the lowest weight out of the parallel edges. We denote the weight of the edge e as $\alpha(e)$ and the weight of the spanning tree T as $\gamma(T)$. As usual, we write the number of vertices as n, number of edges as m, $V = \{v_1, \ldots, v_n\}$ and $E = \{e_1, \ldots, e_m\}$.

The *distance* between two spanning trees T_1 and T_2 of G is

$$n - 1 - \#(T_1 \cap T_2) =_{\text{def.}} d(T_1, T_2),$$

where $\#(T_1 \cap T_2)$ is the number of edges in the intersection of T_1 and T_2. Obviously, $d(T_1, T_2) = 0$ if and only if $T_1 = T_2$. If $d(T_1, T_2) = 1$, then T_1 and T_2 are *neighboring trees*.

The spanning tree T of G is *cut minimal* if the weights of the edges of the fundamental cut set determined by the branch b are $\geq \alpha(b)$ for every branch b. Similarly, the spanning tree T is *circuit minimal* if the edges of the fundamental circuits are $\leq \alpha(c)$ for every link c in the cospanning tree T^*. The spanning tree T is *locally minimal* if $\gamma(T) \leq \gamma(T')$ for every neighboring tree T' of T.

Lemma. *The following three conditions are equivalent for the spanning tree T:*

(i) *T is cut minimal.*

(ii) *T is circuit minimal.*

(iii) *T is locally minimal.*

Proof. (i)⇒(ii): Let us assume T is cut minimal and let us consider a fundamental circuit C of G corresponding to the link c of the cospanning tree T^*. Other than c, the branches in C are branches of T. Every such branch b defines a fundamental cut set of T, which also contains c (Theorem 2.7). Hence $\alpha(b) \leq \alpha(c)$.

(ii)⇒(iii): Let us assume that T is circuit minimal and let us consider a neighboring tree T' of T. T' has (exactly one) branch e' which is not in T, i.e. e' is a link of T^*. We examine the

fundamental circuit C defined by e'. Not all edges of C are in T'. We choose an edge e in C that is not in T'. Then, e is a branch of T (actually the only branch of T that is not in T'). Now, we remove e out of T and add e' to T. The result has to be T'. Because of circuit minimality, $\alpha(e') \geq \alpha(e)$, i.e. $\gamma(T') \geq \gamma(T)$.

(iii)⇒(i): We consider the locally minimal spanning tree T. We take an arbitrary branch b from T corresponding to a fundamental cut set I and an arbitrary link $c \neq b$ in I. Then, b belongs to the fundamental circuit of T defined by c (Theorem 2.8). By removing the branch b from T and adding the edge c to T, we get the neighboring tree T' of T. Because of local minimality, $\gamma(T) \leq \gamma(T')$, i.e. $\alpha(c) \geq \alpha(b)$. □

The spanning tree T is *minimal* if it has the lowest possible weight.

Theorem 5.2. *The following three conditions are equivalent for the spanning tree T:*

(i) *T is cut minimal.*

(ii) *T is circuit minimal.*

(iii) *T is minimal.*

Proof. By the lemma above, (i) and (ii) are equivalent. A minimal spanning tree is obviously locally minimal. Thus, it suffices to prove that a cut minimal spanning tree is also minimal. We will prove by contradiction and consider the counter hypothesis: There is a cut minimal spanning tree T which is not minimal. Let us consider the minimal spanning tree T' and choose T and T' so that the distance $d(T, T')$ is as small as possible. By the lemma, $d(T, T') > 1$.

T has a branch e which is not in T', i.e. it is a link of $(T')^*$. We label the fundamental cut set of T defined by e as I and the fundamental circuit of T' defined by e as C. In the intersection $I \cap C$, there are also other edges besides e (Theorem 2.6). We choose such an edge e'. Then, e' is a link of T^* and a branch of T'. Since T is cut minimal, $\alpha(e') \geq \alpha(e)$. Since T' is (circuit) minimal, $\alpha(e') \leq \alpha(e)$. Therefore, $\alpha(e') = \alpha(e)$. By removing e' from T' and adding e to T', we get a minimal spanning tree T'' which has the same weight as T'. However, $d(T, T'') < d(T, T')$. ✓ □

In *Kruskal's Algorithm*, the edges of the graph G (and their weights) are listed as $e_1, \ldots, e_m$. The algorithm constructs a circuit minimal spanning tree by going through the list to take some edges to form the tree. This is especially effective if the edges are sorted in ascending order by weight.

In the *dual form of Kruskal's Algorithm*, we construct a cut minimal spanning tree by going through the list of edges to take some edges to form the cospanning tree. Once again, this is especially effective if the edges are sorted in descending order by weight.

In all, we get four different versions of Kruskal's Algorithm. (We have to remember that the subgraph induced by the edge set A is written as $\langle A \rangle$.)

Kruskal's Algorithm No. 1

Here we assume that the edges are given in *ascending order* by weight.

1. Set $k \leftarrow 1$ and $A \leftarrow \emptyset$.
2. If e_k does not form a circuit with the edges in A, then set $A \leftarrow A \cup \{e_k\}$ as well as $k \leftarrow k + 1$ and go to step #4.
3. If e_k forms a circuit with the edges in A, then set $k \leftarrow k + 1$ and go to step #4.

4. If (V, A) is not a tree, then go to step #2. Otherwise stop and output the spanning tree $T = \langle A \rangle$.

Whenever we leave out an edge from A (step #3), its end vertices are already connected in A. Thus, the vertices of G are connected in T as they are in G. Since T is obviously circuitless (step #3), it is also a spanning tree of G. At each stage, the branches of the fundamental circuit defined by the link belonging to T^* (step #3) are predecessors of that link in the list. Hence, T is circuit minimal and thus minimal.

Remark. *In every step, the branches and links are permanent. We do not have to know the edges beforehand as long as we process them one by one in ascending order. The rank of the graph (number of branches in a spanning tree) is then required beforehand so we know when to stop.*

Kruskal's Algorithm No. 2

Here we assume the edges are given in an *arbitrary order*.

1. Set $k \leftarrow 1$ and $A \leftarrow \emptyset$.

2. If $\langle A \cup \{e_k\} \rangle$ contains no circuits, then set $A \leftarrow A \cup \{e_k\}$ as well as $k \leftarrow k + 1$ and go to step #4.

3. If $\langle A \cup \{e_k\} \rangle$ contains a circuit C, then choose the edge with the largest weight e in C (if there are more than one, take any), set $A \leftarrow (A \cup \{e_k\}) - \{e\}$ as well as $k \leftarrow k + 1$ and go to step #4.

4. If $k \leq m$, then go to step #2. Otherwise, stop and output the spanning tree $T = \langle A \rangle$.

Whenever we leave out an edge from A (step #3), its end vertices are already connected in A. Thus, the vertices of G are connected in T as they are in G. Since T is obviously circuitless (step #3), it is a spanning tree of G.

We see that T is circuit minimal (and minimal) by the following logic. During the whole process, $\langle A \rangle$ is a forest by step #4. In addition, if u and w are connected in $\langle A \rangle$ at some point, then they are also connected afterwards. The u–w path in $\langle A \rangle$ is unique but it can change to another path later in step #3. Nevertheless, whenever this change occurs, the maximum value of the weights of the edges of the path can not increase anymore. Every link c of T^* has been removed from A in step #3. Then, the weight of c is at least as large as the weights of the other edges in C, After we have gone through step #3, the only connected end vertices of c in $\langle A \rangle$ have to go through the remaining edges of C. The final connection between the end vertices of c in T goes through the edges of the fundamental circuit defined by c. Therefore, the weights of the edges of this fundamental circuit are $\leq \alpha(c)$.

Remark. *In each step, the links (e in step #3) are permanent and the branches are not. We do not have to know the edges beforehand as long as we process them one by one. However, we need to know the nullity of the graph (number of links in a cospanning tree) so that we know when to stop. The algorithm can also be used to update a minimal spanning tree if we add edges to the graph or decrease their weight.*

Kruskal's Algorithm No. 3

Here we assume the edges are given in *descending order* by weight.

1. Set $A \leftarrow E$ and $k \leftarrow 1$.
2. If $(V, A - \{e_k\})$ is connected, then set $A \leftarrow A - \{e_k\}$ as well as $k \leftarrow k + 1$ and go to step #4.
3. If $(V, A - \{e_k\})$ is disconnected, then set $k \leftarrow k + 1$ and go to step #4.
4. If (V, A) is not a tree, then we go to step #2. Otherwise we stop and output the spanning tree $T = (V, A)$.

T is obviously connected because (V, A) is connected everytime we go to step #4. On the other hand, T is circuitless because if the circuit C is in T and the edge c is in the circuit, then c is removed from A in step #2 when $e_k = c$ ($\surd$). Thus, T is a spanning tree of G. In each step, the links of the fundamental cut set defined by the branch belonging to T (step #3) are predecessors of that branch in the list. Hence, T is circuit minimal and it is thus minimal.

Remark. *In each step, the branches and links are permanent. We have to know the edges beforehand. On the other hand, we do not have to know their weights as long as we get them one by one in descending order.*

Kruskal's Algorithm No. 4

Here we assume the edges are given in an *arbitrary order*.

1. Set $A \leftarrow E$ and $k \leftarrow 1$.
2. If $(V, A - \{e_k\})$ is connected, then set $A \leftarrow A - \{e_k\}$ as well as $k \leftarrow k + 1$ and go to step #4.
3. If $(V, A - \{e_k\})$ is disconnected, then it has two components. The corresponding vertex sets form a cut $\langle V_1, V_2 \rangle$. We interpret it as an edge set and choose the edge e with the lowest weight in $\langle V_1, V_2 \rangle$ (if there are more than one, take any). Set $A \leftarrow (A - \{e_k\}) \cup \{e\}$ as well as $k \leftarrow k + 1$ and go to step #4.
4. If $k \leq m$, then go to step #2. Otherwise stop and output the spanning tree $T = (V, A)$.

T is obviously connected because (V, A) is connected everytime we go to step #4. (Take note that the connectivity is preserved everytime we go through step #3.) On the other hand, T is circuitless. If a circuit C of G ends up in T and c is the edge of C, which is first in the list, then c must be removed from A in step #2 when $e_k = c$. (Note that the edge removed from the circuit first can not be removed in step #3.) If c comes back later (in step #3), then it forms a cut set of (V, A) by itself in which case some other edge of C has been removed. By continuing this process, we see that all the edges of C can not be in A in the end ($\surd$). Therefore, T is a spanning tree of G.

In addition, T is cut minimal and minimal because every branch b of T comes in in step #3. The links of the fundamental cut set defined by b are either edges of the cut $\langle V_1, V_2 \rangle$ which is examined at that point or they are links of the cuts we examined later in step #3. Whenever an edge of this kind gets removed later in step #3, it is always compensated by edges that are heavier in weight than b. Those heavier edges are in the cut $\langle V_1, V_2 \rangle$ which is examined at that time. Therefore, the weights of the fundamental cut set defined by b are $\geq \alpha(b)$.

Remark. *In each step, the branches (e in step #3) are permanent and the links are not. We have to know the edges beforehand. We do not have to know the weights beforehand as long as we process them one by one. This algorithm can also be used for updating a minimal spanning tree if we remove edges from a graph or if we increase the weights of edges.*

Prim's Algorithm

In *Prim's Algorithm* (also known as *Jarnik's Algorithm*), we use the all-vertex incidence matrix of G. If we label the set of edges incident on vertex v as $\Omega(v)$, then we can get a list $\Omega(v_1), \ldots, \Omega(v_n)$, i.e. the cuts defined by the vertices (interpreted as edge sets). In addition, we assign weights to the vertices.

The algorithm works in the same way as Dijkstra's Algorithm by constructing the spanning tree branch by branch. The variables are A (set of branches of the spanning tree we have at the time), B (set of vertices of the spanning tree we have at the time) and I (the cut interpreted as an edge set from which we choose the next branch).

Prim's Algorithm (First Version):

1. Choose a starting vertex r and set $A \leftarrow \emptyset$, $B \leftarrow \{r\}$ as well as $I \leftarrow \Omega(r)$.
2. Choose the lightest edge e from I (if there are more than one, choose any). Take the end vertex v of e that is not in B. Set $A \leftarrow A \cup \{e\}$, $B \leftarrow B \cup \{v\}$ as well as $I \leftarrow I \oplus \Omega(v)$ and go to step #3. (Remember that $\oplus$ denotes the symmetric difference operation between two sets, see page 12.)
3. If $B \neq V$, then go to step #2. Otherwise, stop and output the spanning tree $T = (B, A) = \langle A \rangle$.

Since the edge e was chosen from a cut, T is circuitless. On the other hand, because there is a path from r to every other vertex, T has every vertex of G and it is connected. T is thus a spanning tree. It is also minimal because

Statement. *During the whole process, (B, A) is a subtree of some minimal spanning tree of G.*

Proof. We use induction on ℓ, the number of vertices in B.

Induction Basis: $\ell = 1$. The case is obvious because (B, A) is trivial.

Induction Hypothesis: The statement is true for $\ell = k - 1$. ($k \geq 2$)

Induction Statement: The statement is true for $\ell = k$.

Induction Statement Proof: In step #2, we can write $A = A' \cup \{e\}$, where $e \in I'$ and $B = B' \cup \{v\}$. (B', A') is a subtree of some minimal spanning tree $T_{\min}$ from the induction hypothesis. If e belongs to $T_{\min}$, then the case is clear. Otherwise, there is a fundamental circuit C in $T_{\min} + e$ and there is another edge e' of I' in C (Theorem 2.6). Then, $\alpha(e') \geq \alpha(e)$ and $(T_{\min} + e) - e'$ is also a minimal spanning tree and (B, A) is its subtree (because $T_{\min}$ is circuit minimal and $\alpha(e') \leq \alpha(e)$). □

Often, we use one or two additional labels for the vertices to make Prim's Algorithm easier. In the next version of the algorithm, we will use two labels $\pi(v)$ and $\beta(v)$, which are used to perform step #2 more effectively. The values of π are weights (up to ∞) and the values of β are edges (or $= 0$). Otherwise, the algorithms works in the same way as before.

Prim's Algorithm (Second Version):

1. Choose a starting vertex r and set $\pi(r) \leftarrow 0$. For every other vertex v, set $\pi(v) \leftarrow \infty$. For every vertex v, set $\beta(v) \leftarrow 0$ as well as $A \leftarrow \emptyset$ and $B \leftarrow \emptyset$.

2. Choose a vertex $u \notin B$ for which

$$\pi(u) = \min_{v \notin B}\{\pi(v)\}.$$

   Set $B \leftarrow B \cup \{u\}$. If $\beta(u) \neq 0$, then set $A \leftarrow A \cup \{\beta(u)\}$.

3. Go through all the edges $e = (u, v)$ where $v \notin B$. If $\alpha(e) < \pi(v)$, then set $\pi(v) \leftarrow \alpha(e)$ and $\beta(v) \leftarrow e$.

4. If $B \neq V$, then go to step #2. Otherwise, stop and output the spanning tree $T = (B, A) = \langle A \rangle$.

5.7 The Lightest Hamiltonian Circuit (Travelling Salesman's Problem): The Annealing Algorithm and the Karp–Held Heuristics

Problem. *If it is possible, we are to find the* Hamiltonian circuit *with the lowest weight. A Hamiltonian circuit visits all the vertices of a graph. As usual, the weights of the edges have been assigned and the weight of a (directed) circuit is the sum of the weights of the edges traversed.*

Obviously, we can assume that the graph is nontrivial, connected (otherwise it would not be possible to get a Hamiltonian circuit) and simple. If not, then we simply remove all the loops and choose the edge with the lowest weight out of the parallel edges. As usual, we denote n as the number of vertices, m as the number of edges, $V = \{v_1, \ldots, v_n\}$ and $E = \{e_1, \ldots, e_m\}$. We label the weight of an edge $e = (v_i, v_j)$ as $\alpha(e) = \alpha(v_i, v_j)$ and the weight of a Hamiltonian circuit H as $\gamma(H)$. We agree that the "first" vertex of a Hamiltonian circuit is v_1.

The same problem exists for directed graphs in which case we are looking for the directed Hamiltonian circuit with the lowest weight (known as the *Unsymmetric Travelling Salesman's Problem*).

The Travelling Salesman's Problem (TSP)[1] is an $\mathcal{NP}$-complete problem, read e.g. MEHLHORN for more information. Actually, even determining the existence of a Hamiltonian circuit is an $\mathcal{NP}$-complete problem. Solving a small TSP takes a lot of time and larger problems take so much time that it is almost impossible to obtain accurate solutions. Therefore, many stochastic and approximation methods are used in practice. Then, we have to accept the possibility of inaccurate outcomes or even the lack of results.

[1]The name "Travelling Salesman's Problem" comes from an interpretation where the vertices of a graph are cities and the weights of the edges between the cities are travelling times. The salesman needs to visit every city in the shortest amount of time.

The Annealing Algorithm

The *annealing algorithms* or *thermodynamic algorithms* have the following common features:

(A) The system in question is always in some *state* s. The set of all states S is finite and known. In the TSP, a state is a Hamiltonian circuit.

(B) Each state s has a *response* $f(s)$, which can be calculated in a timely fashion from the state. Our goal is to find a state whose response is near the minimum/maximum value. The response of a state of a TSP is the weight of a Hamiltonian circuit.

(C) There is a procedure A_k which is used to move from state s to state $A_k(s)$. k is a parameter of the procedure which belongs to the set K. K can change during the procedure. The purpose is to move to certain states "near" the state s which are defined by the parameter k. By repeating the procedure with proper values of k, we should be able to move from any state to any other state. (In some cases, we can omit this last part.)

(D) Every time we move from one state to another, we should be able to choose the parameter k quickly and randomly from K. In particular, the set K itself should be easily computable.

(E) We should be able to quickly perform the procedure A_k given a value of k.

(F) We should be able to find a starting state s_0. For the TSP, the starting state is a Hamiltonian circuit.

The algorithm is as follows:

The Annealing Algorithm:

1. Choose the starting state s_0, the *initial temperature* T_0 and set $s \leftarrow s_0$ as well as $T \leftarrow T_0$.
2. When we are in the state s, we randomly choose a parameter $k \in K$ and compute $s' = A_k(s)$.
3. If $f(s') \leq f(s)$, then set $s \leftarrow s'$ and go to step #5.
4. If $f(s') > f(s)$, then generate a random number r in the interval $[0, 1)$. If $r \leq e^{(f(s)-f(s'))/T}$, then set $s \leftarrow s'$. Thus, we accept a "worse" state with probability $e^{(f(s)-f(s'))/T}$. Note that the greater the temperature T, the greater the probability that we go "uphill".
5. If we have gone through a maximum total number of iterations, then we stop and output s. Otherwise, if we have gone through sufficiently many iterations of the procedure using temperature T, then we lower T by some rule and go to step #2.

Remark. *The distribution of the probability* $p_{s'} = e^{\frac{f(s)-f(s')}{T}}$ *used in step #4 is (apart from normalizing) is a so-called* maximum entropy distribution *with the following condition on the expected value:*

$$\sum_{\substack{s'=A_k(s) \\ k\in K \\ f(s')>f(s)}} p_{s'} f(s') = \mu$$

where μ *depends on* T *and* s. *The distribution is also called a* Boltzman distribution *and it is analogous to the distribution of the same name in Statistical Mechanics. Refer to courses in Physics and Information Theory for more information.*

At first, we wait until the fluctuation in the states settles to a certain equilibrium (using the response $f(s)$). After that, we lower the value of T a bit and wait again for the equilibrium. Then, we lower T again and so on. We continue this until the change in values of $f(s)$ is sufficiently small or if we have ran out of time.

The operation A_k and the set K of the neighboring states depend on the problem. The state structure and the response function also depend on the problem. For the TSP, we still need to assign A_k and K for every situation. For this purpose, we take another parameter j and set $j \leftarrow 2$. In step #2, we update j in the following way:

$$j \leftarrow \begin{cases} j+1 \text{ if } j < n \\ 2 \text{ otherwise.} \end{cases}$$

(Another way of choosing j in step #2 would be to choose it randomly out of $\{2, \ldots, n\}$.) Furthermore, we choose

$$K = \{2, \ldots, n\} - \{j\}.$$

A_k is defined by the following operation (known as the *reversal*):

- If $k > j$, then we reverse the order of the vertices $v_{i_j}, \ldots, v_{i_k}$ on the corresponding subpath in the current Hamiltonian circuit

$$s : v_1, v_{i_2}, \ldots, v_{i_n}, v_1.$$

- If $k < j$, then we reverse the order of the vertices $v_{i_k}, \ldots, v_{i_j}$ on the corresponding subpath in the current Hamiltonian circuit

$$s : v_1, v_{i_2}, \ldots, v_{i_n}, v_1.$$

We add the missing edges to the graph with very large weights so that we get a complete graph and we will not have to worry about the existence of a Hamiltonian circuit in the first place. If we still do not get a Hamiltonian circuit in the end without those added edges, then there is not a Hamiltonian circuit.

The starting temperature T_0 should be much larger than the values of $|f(s') - f(s)|$ which guarantees that we can in principle move to any state ("annealing") in the earlier stages of the algorithm. After that, we lower the temperature applying some rule, for example a 10% change.

The annealing algorithm also works for the unsymmetric TSP with obvious changes.

Karp–Held Heuristics

In the *Karp–Held Heuristics*, we do not directly look for a Hamiltonian circuit but look for a similar subgraph, known as a *spanning 1-tree*[2]. The process does not work for the unsymmetric TSP. The spanning 1-tree S_v corresponding to the vertex v (known as the *reference vertex*) is a subgraph of G that satisfies the following conditions:

(a) S_v is connected and contains every vertex of G.

(b) S_v contains exactly one circuit C and the vertex v belongs to C.

(c) S_v has exactly two edges incident on v.

[2]Not to be confused with the 1-tree on p.23!

Clearly, a Hamiltonian circuit is a spanning 1-tree corresponding to any of the vertices. The *weight* of the spanning 1-tree S_v is the sum of of the weights of all its edges, denoted $\gamma(S_v)$. S_v is *minimal* if it has the lowest possible weight.

Statement. *S_v is minimal if and only if*

(i) *$S_v - v$ is a minimal spanning tree of $G - v$, and*

(ii) *the two edges of S_v incident on v are the two lightest edges of G out of all the edges incident on v.*

Proof. Let S_v be a minimal spanning 1-tree. Let e and e' be the two edges in S_v incident on v. Then, $S_v - v$ is a spanning tree of $G - v$ because removing v destroys the circuit but the connections remain unsevered. If $S_v - v$ is not a minimal spanning tree of $G - v$ then there is a lighter spanning tree T of $G - v$. By adding the vertex v and the edges e and e' to T, we get a spanning 1-tree corresponding to vertex v which is lighter than S_v ($\checkmark$). Therefore, (i) is true. Obviously, (ii) is true (because otherwise we would get a lighter spanning 1-tree by replacing e and e' with the two lightest edges in G incident on v).

Let us assume that (i) and (ii) are true. If S_v is not minimal, then there is a lighter minimal spanning 1-tree S'_v corresponding to v. Because S'_v also satisfies (ii), the two edges incident on v are the same (or at least they have the same weight) in S_v and S'_v. Thus, $S'_v - v$ is lighter than $S_v - v$ ($\checkmark$). □

It follows from the statement that any algorithm that finds the minimum spanning tree also works for finding the minimum spanning 1-tree with minor modifications. Especially, Kruskal's and Prim's Algorithms are applicable.

In the Karp–Held Heuristics, we also use weights of vertices, denoted $\beta(v)$. With these, we can define the *virtual weight* of an edge as

$$\alpha'(v_i, v_j) = \alpha(v_i, v_j) + \beta(v_i) + \beta(v_j).$$

With the concept of virtual weights, we get the virtual weight of a spanning 1-tree S_v (we label the edge set of S_v as A):

$$\begin{aligned}\gamma'(S_v) &= \sum_{(v_i,v_j)\in A} \alpha'(v_i, v_j) = \sum_{(v_i,v_j)\in A} \alpha(v_i, v_j) + \sum_{(v_i,v_j)\in A} (\beta(v_i) + \beta(v_j)) \\ &= \gamma(S_v) + \sum_{(v_i,v_j)\in A} (\beta(v_i) + \beta(v_j)).\end{aligned}$$

Now we denote the degree of the vertex u in S_v as $d_{S_v}(u)$. Then,

$$\sum_{(v_i,v_j)\in A} (\beta(v_i) + \beta(v_j)) = \sum_{i=1}^{n} \beta(v_i) d_{S_v}(v_i)$$

and

$$\gamma'(S_v) = \gamma(S_v) + \sum_{i=1}^{n} \beta(v_i) d_{S_v}(v_i).$$

In particular, if we have a Hamiltonian circuit H (a special spanning 1-tree), then

$$d_H(v_1) = \cdots = d_H(v_n) = 2$$

and

$$\gamma'(H) = \gamma(H) + \underbrace{2\sum_{i=1}^{n}\beta(v_i)}_{\text{Does not depend on } H!}.$$

Minimization of the Hamiltonian circuits using virtual weights yields the same minimal circuit than obtained by using real weights. In general though, if we use virtual weights to search for spanning 1-trees, then we get results different from the spanning 1-trees obtained by using real weights.

From now on, we only consider the spanning 1-tree corresponding to vertex v_1 and we leave out the subscript. This is not a limitation of any kind on the Hamiltonian circuits although it might be a good idea to change the reference vertex every now and then. We assume that $H_{\min}$ is a minimal Hamiltonian circuit and S' is the minimal spanning 1-tree obtained from using virtual weights (which of course corresponds to v_1). Then,

$$\gamma'(H_{\min}) \geq \gamma'(S').$$

In addition,

$$\gamma'(H_{\min}) = \gamma(H_{\min}) + 2\sum_{i=1}^{n}\beta(v_i)$$

and

$$\gamma'(S') = \gamma(S') + \sum_{i=1}^{n}\beta(v_i)d_{S'}(v_i).$$

Thus,

$$\begin{aligned}\gamma(H_{\min}) &= \gamma'(H_{\min}) - 2\sum_{i=1}^{n}\beta(v_i) \geq \gamma'(S') - 2\sum_{i=1}^{n}\beta(v_i)\\ &= \gamma(S') + \sum_{i=1}^{n}\beta(v_i)(d_{S'}(v_i) - 2),\end{aligned}$$

from which we get a *lower limit* on $\gamma(H_{\min})$.

The idea of the Karp–Held Heuristics is to guide the degrees of the vertices in S' to the value 2 by changing the weights of the vertices. If we succeed, then we get a minimal Hamiltonian circuit. In all cases, we get a lower limit on the weights $\gamma(H)$ of the (possible) Hamiltonian circuits by using the calculation above. (Note that $d_{S'}(v_1)$ is always $= 2$ if S' is the spanning 1-tree corresponding to v_1.)

The Karp–Held Heuristics:

1. Set $\beta(v) \leftarrow 0$ for every vertex v.
2. Set $\alpha'(u, v) \leftarrow \alpha(u, v) + \beta(u) + \beta(v)$ for each edge (u, v).
3. Find the minimal spanning 1-tree S' using virtual weights $\alpha'(u, v)$. If we fail to find this kind of spanning 1-tree, then there is no Hamiltonian circuit and we can stop.
4. If S' is a circuit, then output the minimal Hamiltonian circuit $H = S'$ and stop.
5. If S' is not a circuit and the lower limit calculated from S' increased during the last K iterations, then set $\beta(v) \leftarrow \beta(v) + d_{S'}(v) - 2$ for every vertex v and go to step #2. (K is a fixed upper bound on the number of iterations.)

6. If the lower limit calculated from S' has not increased during the last K iterations, then output that lower limit and stop.

This procedure does not always produce a minimal Hamiltonian circuit even if there exists one. In practice, it often produces either a minimal Hamiltonian circuit or a good lower limit on the weight of it. Getting a number for the lower limit does not, however, guarantee the existence of a Hamiltonian circuit in the graph!

Karp–Held Heuristics has many steps where we have to choose between different options (such as the reference vertex and the spanning 1-tree). We can not go through every possibility so we must choose randomly. Then, we have a Las Vegas algorithm or stochastic algorithm.

5.8 Maximum Matching in Bipartite Graphs: The Hungarian Algorithm

A *matching* in the graph $G = (V, E)$ is a set of edges $S \subseteq E$ none of which are adjacent to each other. A matching is a *maximum matching* if it has the greatest possible number of edges. The end vertex of an edge in a matching is *matched*.

Problem. *We want to find the maximum matching in a bipartite graph.*

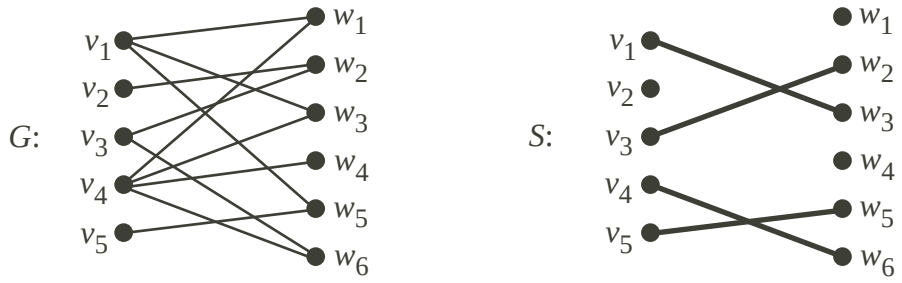
An *alternating path* of a matching S is a path that satisfies the following conditions:

(1) The first vertex on the path is not matched, and

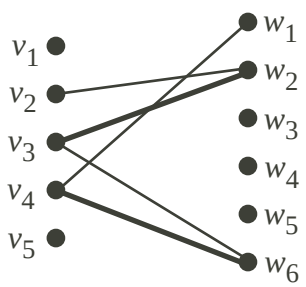
(2) every second edge is in the matching and the remaining edges are not in the matching.

Note that the first edge in an alternating path is not in the matching. In addition, if the last vertex of an alternating path is not matched, then this path is an *augmenting path* of S. A matching without augmenting paths is called a *maximal matching*.

Example. *For the bipartite graph*



one augmenting path of the matching $S = \{(v_1, w_3), (v_3, w_2), (v_4, w_6), (v_5, w_5)\}$ *is the path where the vertices are* $v_2, w_2, v_3, w_6, v_4, w_1$.

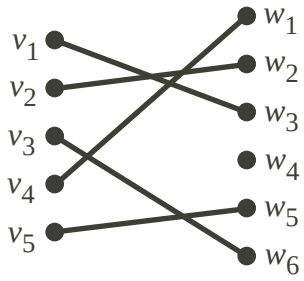


We can *augment* a matching S using its augmenting path p as follows:

1. We remove the edges of S in p, and
2. We add the edges in p which are not in S.

The new edge set is obviously a matching. Note that the number of edges in S on an augmenting path is one fewer than the number of the remaining edges. Therefore, the number of edges in a matching increases by one after the augmenting operation. It is not possible to augment a maximal matching.

Example. *(Continuing from the previous example) By using the given augmenting path from the matching S, we get a new maximal matching $S_1 = \{(v_1, w_3), (v_2, w_2), (v_3, w_6), (v_4, w_1), (v_5, w_5)\}$.*



In the *Hungarian Algorithm*, we systematically search for augmenting paths until we get a maximal matching. After that, it suffices to prove that a maximal matching is a maximum matching. From now on, we only consider *bipartite graphs* because the algorithm is then much simpler. We search for augmenting paths by constructing an *alternating tree* of a matching S which is a subtree of G such that

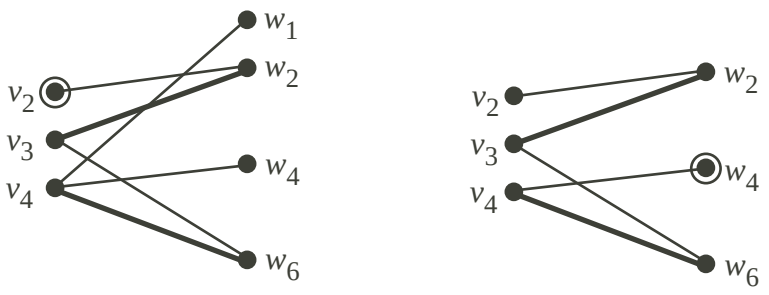
(1) a vertex r (the *root* of the tree) is unmatched,

(2) every second edge on the path out from r is in S and the remaining edges are not in S, and

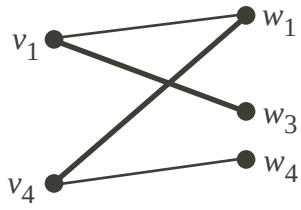
(3) either there is an augmenting path out from r or we can not add any more edges to S.

An alternating tree is an *augmenting tree* if it has an augmenting path. Otherwise, it is a *Hungarian tree*. Every augmenting path is obviously an augmenting tree by itself. Note that the only unmatched vertex of a Hungarian tree is the root.

Example. *(Continuing from the previous example) Two alternating trees of the matching S are (the root is circled):*



Both of them are augmenting trees. An alternating tree of the matching S_1 (the root is w_4) is the Hungarian tree



Augmenting and Hungarian trees are not unique. We can have many different trees depending on the order we take the edges for constructing the trees even though the roots are the same. On the other hand,

Statement. *If a matching in a bipartite graph G has a Hungarian tree, then it does not have an augmenting tree with the same root.*

Proof. Let us prove by contradiction and consider the counter hypothesis: A matching S has a Hungarian tree U and an augmenting tree T with the same root r. We get an augmenting path in the augmenting tree

$$p : r = v_0, e_1, v_1, e_2, \ldots, e_k, v_k.$$

We choose the last vertex v_i which is in U from the path p (at least $r = v_0$ is in U). Since v_k is not in U, $i < k$. Furthermore, the edge e_{i+1} is not in the matching nor in U (otherwise v_{i+1} would also be in U). On the other hand, since e_{i+1} is not in U, v_{i+1} has to be an end vertex of another edge in U ($\surd$) because the only reason why the edge e_{i+1} is not put into U while constructing U is that the other end vertex v_{i+1} of e_{i+1} is already in U. Note how the bipartiteness of the G comes in: If the cut in G that results in the bipartition is $\langle V_1, V_2 \rangle$, then the vertices of U and p alternate between V_1 and V_2. Therefore, the length of the r–v_i path is even in p and U. □

Constructing an alternating tree from a root always leads to a Hungarian tree or an augmenting tree but not both. The order of the edges taken does not matter. (This is not the case for general graphs.)

For the bipartite graph $G = (V, E)$, the Hungarian Algorithm is as follows. The cut that yields the bipartition is $\langle V_1, V_2 \rangle$.

The Hungarian Algorithm:

1. Set $S \leftarrow \emptyset$. (We can also use some other initial matching.)
2. If every vertex in V_1 or in V_2 is matched in S, then S is a maximum matching and we stop.
3. If there are unmatched vertices in S of V_1, then go through them in some order constructing alternating trees (the method of construction is not important as we claimed). If there is an augmenting tree, then augmenting the matching S by using the augmenting path we have another matching S_1. Set $S \leftarrow S_1$ and go to #2.
4. If all the alternating trees that have unmatched vertices in V_1 as roots are Hungarian, S is maximal and we stop.

Theorem 5.3. *A maximal matching in a bipartite graph is a maximum matching.*

Proof. Let us prove by contradiction and consider the counter hypothesis: A maximal matching S in the bipartite graph $G = (V, E)$ is not a maximum matching. Then, there are more edges in the maximum matching $S_{\max}$ than in S and in V_1 there are more vertices matched in $S_{\max}$ than in S. We choose an arbitrary vertex $v \in V_1$, which is matched in $S_{\max}$ but not in S. Then, we have a path

$$p : v = v_0, e_1, v_1, e_2, \ldots, e_k, v_k = w,$$

whose edges are alternating between $S_{\max}$ and S, i.e. e_1 is in $S_{\max}$ and e_2 is in S and so on. We choose the longest such path p. Because p is obviously an alternating path of S, it has even length, i.e. e_k is an edge of S. (Otherwise, p would be an augmenting path of S which is impossible because S is maximal.) Thus, w is matched in S but not matched in $S_{\max}$ (because the path p can not be continued).

Hence, every vertex $v \in V_1$ which is matched in $S_{\max}$ but not in S corresponds to a vertex $w \in V_1$, which is matched in S but not in $S_{\max}$. Now, every path that ends at w must start from the vertex v if the starting vertex is matched in $S_{\max}$ but not in S. The last edge of such a path has to be e_k (the only edge in S incident on w) and the second to the last vertex has to be v_{k-1}. Furthermore, the second to the last edge of this path has to be e_{k-1} (the only edge of $S_{\max}$ incident on v_{k-1}) and the third to the last vertex has to be v_{k-2}, and so on.

However, there are then in V_1 at least as many vertices w that are matched in S but not in $S_{\max}$ as there are vertices v that are matched in $S_{\max}$ but not in S ($\surd$). □

Corollary. *The Hungarian algorithm produces a maximum matching in a bipartite graph.*

A matching is *perfect* if it matches every vertex of a graph. Thus, a graph with an odd number of vertices can not have a percet matching. Let us consider the graph $G = (V, E)$ and denote $\nu(v) = \{\text{adjacent vertices of } v\}$ as well as $\nu(A) = \bigcup_{v \in A} \nu(v)$ for the vertex set $A \subseteq V$. Let us denote by $\#(X)$ the number of elements in the set X (the cardinality of the set). With these notions, we can present the following famous characterization:

Theorem 5.4. (Hall's Theorem or "Marriage Theorem") *A bipartite graph G whose bipartition cut is $\langle V_1, V_2 \rangle$ has a perfect matching if and only if every vertex set $A \subseteq V_1$ and $B \subseteq V_2$ satisfies the conditions $\#(A) \leq \#(\nu(A))$ and $\#(B) \leq \#(\nu(B))$.*

Proof. If a perfect matching exists, then obviously $\#(A) \leq \#(\nu(A))$ and $\#(B) \leq \#(\nu(B))$ hold for all sets of vertices $A \subseteq V_1$ and $B \subseteq V_2$. (Otherwise, we can not find a pair for every vertex in A or B in the matching.)

Let us assume that for all sets of vertices $A \subseteq V_1$ and $B \subseteq V_2$, $\#(A) \leq \#(\nu(A))$ and $\#(B) \leq \#(\nu(B))$. Let S be a maximum matching in G. We will prove by contradiction and consider the counter hypothesis: S is not perfect. We choose a vertex v which is not matched in S. Let us examine the case where $v \in V_1$ (the other case where $v \in V_2$ is obviously symmetrical). The contradiction is apparent if v is an isolated vertex so we can move to the case where v is an end vertex of an edge. The alternating tree with the root v is then nontrivial and since the matching is also maximal, this tree is Hungarian. We choose such a Hungarian tree U. We label the set of vertices of V_1 (resp. V_2) in U by A (resp. by B). Because of the construction of U, $B = \nu(A)$. On the other hand, the vertices of A and B in U are pairwise matched by the edges of S, except for the root r. Hence, $\#(A) = \#(B) + 1 > \#(B)$ ($\surd$). □

5.9 Maximum Flow in a Transport Network: The Ford–Fulkerson Algorithm

A *transport network* is a directed graph $G = (V, E)$ with weighted arcs that satisfies the following:

(1) G is connected and loopless,

(2) G has only one source s,

(3) G has only one sink t, and

(4) the weight $c(e)$ of the arc e is called the *capacity* and it is a nonnegative real number, i.e. we have a mapping $c : E \to \mathbb{R}_0$.

(Compare to stationary linear networks in Section 4.4.) Actually, we could assume that G has every possible arc except loops and it can even have multiple parallel arcs. If this is not the case, then we simply add the missing arcs with capacity zero. Naturally, we can also assume that G is nontrivial.

A *flow* f of a transport network is a weight mapping $E \to \mathbb{R}_0$, which satisfies:

(i) For each arc e, we have the *capacity constraint* $f(e) \leq c(e)$, and

(ii) each vertex $v \neq s, t$ satisfies the *conservation condition* (also called *Kirchhoff's Flow Law*, compare to Section 4.4)

$$\sum_{\substack{\text{initial vertex} \\ \text{of } e \text{ is } v}} f(e) = \sum_{\substack{\text{terminal vertex} \\ \text{of } e \text{ is } v}} f(e).$$

$f(e)$ is called the *flow* of e. The flow of the arc (u, v) is also denoted as $f(u, v)$. The *value* of the flow f is

$$|f| = \sum_{\substack{\text{initial vertex} \\ \text{of } e \text{ is } s}} f(e).$$

A flow f^* is a *maximum flow* if its value is the largest possible, i.e. $|f^*| \geq |f|$ for every other flow f.

An *s–t cut* of a transport network S is a (directed) cut $I = \langle V_1, V_2 \rangle$ such that s is in V_1 and t is in V_2. The *capacity* of such a cut is

$$c(I) = \sum_{\substack{u \in V_1 \\ v \in V_2}} c(u, v).$$

(Note that the arcs in the direction opposite to the cut do not affect the capacity.) The capacity of the cut $\langle V_1, V_2 \rangle$ is also denoted as $c(V_1, V_2)$. Furthermore, we define the *flux* of the cut $I = \langle V_1, V_2 \rangle$ as

$$f^+(I) = \sum_{\substack{u \in V_1 \\ v \in V_2}} f(u, v)$$

and the *counter-flux* as

$$f^-(I) = \sum_{\substack{u \in V_2 \\ v \in V_1}} f(u, v).$$

The value of a flow can now be obtained from the fluxes of any s–t cut:

Theorem 5.5. *If f is a flow of a transport network and I is an s–t cut, then*

$$|f| = f^+(I) - f^-(I).$$

Proof. Obviously,

$$\sum_{\substack{\text{initial vertex}\\ \text{of } e \text{ is } v}} f(e) - \sum_{\substack{\text{terminal vertex}\\ \text{of } e \text{ is } v}} f(e) = \begin{cases} |f| \text{ if } v = s \\ 0 \text{ if } v \neq s, t. \end{cases}$$

We denote $I = \langle V_1, V_2 \rangle$. By going through the vertices v in V_1 and by adding up the equations we get

$$\sum_{v \in V_1} \sum_{\substack{\text{initial vertex}\\ \text{of } e \text{ is } v}} f(e) - \sum_{v \in V_1} \sum_{\substack{\text{terminal vertex}\\ \text{of } e \text{ is } v}} f(e) = |f|.$$

For each arc e whose end vertices are both in V_1, $f(e)$ and $-f(e)$ are added exactly once and thus they cancel out. Therefore,

$$\sum_{\substack{u \in V_1 \\ v \in V_2}} f(u, v) - \sum_{\substack{u \in V_2 \\ v \in V_1}} f(u, v) = |f|.$$

□

Corollary. *If f is a flow of a transport network and I is an s–t cut, then $|f| \leq c(I)$.*

Proof. $|f| = f^+(I) - f^-(I) \leq f^+(I) \leq c(I)$. □

An arc e of a transport network is *saturated* if $f(e) = c(e)$. Otherwise, it is *unsaturated*. Now, we point out that $|f| = c(V_1, V_2)$ if and only if

(i) the arc (u, v) is saturated whenever $u \in V_1$ and $v \in V_2$, and

(ii) $f(u, v) = 0$ whenever $u \in V_2$ and $v \in V_1$.

An s–t cut I^* of a transport network is called a *minimum cut* if $c(I^*) \leq c(I)$ for every other s–t cut I.

Corollary. *If f is a flow of a transport network, I is an s–t cut and $|f| = c(I)$, then f is a maximum flow and I is a minimum cut.*

Proof. If f^* is a maximum flow and I^* is a minimum cut, then $|f^*| \leq c(I^*)$ by the corollary above. Thus,

$$|f| \leq |f^*| \leq c(I^*) \leq c(I)$$

and f is indeed a maximum flow and I is indeed a minimum cut. □

Actually, the value of the maximum flow is the capacity of the minimum cut. The show this, we examine a path from vertex s to vertex v (not necessarily a directed path):

$$s = v_0, e_1, v_1, e_2, \ldots, e_k, v_k = v \quad \text{(path } p\text{)}.$$

If $e_i = (v_{i-1}, v_i)$, then the arc e_i is a *forward arc*. If $e_i = (v_i, v_{i-1})$, then the arc e_i is a *back arc*. The arc e_i of p is now weighted by the following formula:

$$\epsilon(e_i) = \begin{cases} c(e_i) - f(e_i) \text{ if } e_i \text{ is a forward arc} \\ f(e_i) \text{ if } e_i \text{ is a back arc} \end{cases}$$

and the path p is weighted by the following formula:

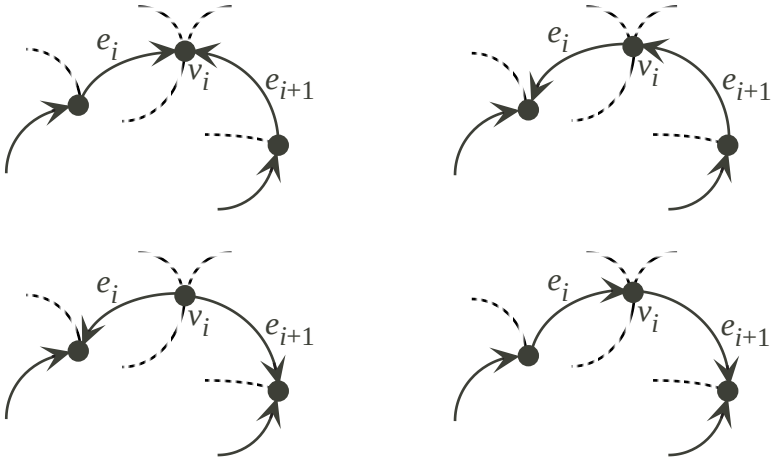
$$\epsilon(p) = \min_{i=1}^{k}\{\epsilon(e_i)\}.$$

The path p is *unsaturated* if $\epsilon(p) > 0$, i.e. all of the forward arcs of p are unsaturated and $f(e_i) > 0$ for all the back arcs e_i of p.

In particular, an s–t path can be unsaturated in which case it is called an *augmenting path*[3]. All of these definitions are of course attached to a certain flow f. By starting from an s–t path p (and a flow f), we can define a new flow:

$$\overline{f} = \begin{cases} f(e) + \epsilon(p) \text{ if } e \text{ is a forward arc of } p \\ f(e) - \epsilon(p) \text{ if } e \text{ is a back arc of } p \\ f(e) \text{ otherwise.} \end{cases}$$

$\overline{f}$ is really a flow. Changes in f can only occur at the arcs and vertices of p. Every arc of p satisfies the capacity constraint because of how $\epsilon(p)$ and $\overline{f}(e)$ are defined. A vertex v_i of p satisfies the conservation condition which we can verify. We have four cases:



Obviously (think about the source s)

$$|\overline{f}| = |f| + \epsilon(p),$$

so f is not a maximum flow if it has an augmenting path. Moreover, the converse is true as well. Hence,

Theorem 5.6. *A flow is a maximum flow if and only if it does not have any augmenting path.*

Proof. As we claimed, a maximum flow can not have an augmenting path. Let us assume that a flow f does not have an augmenting path. We denote the set of vertices which we can reach from the source s along unsaturated paths by V_1. Then, trivially, $s \in V_1$ and $t \notin V_1$ (because there are no augmenting paths). Thus, the cut $I = \langle V_1, V_2 \rangle$ is an s–t cut. We proceed to prove that $|f| = c(I)$. By the previous corollary, f is then a maximum flow.

Let us consider the arc (u, v), where $u \in V_1$ and $v \in V_2$. Then, there exists an unsaturated s–u path p. The edge (u, v) is saturated because there would be an unsaturated s–v path otherwise. Similarly, we conclude that $f(u, v) = 0$ for every arc (u, v), where $u \in V_2$ and $v \in V_1$. Therefore, the flux $f^+(I)$ is $c(I)$ and the counter-flux $f^-(I)$ is zero. By Theorem 5.5, $|f| = c(I)$. □

[3]Not to be confused with the augmenting path in the previous section!

We have also proven the celebrated

Theorem 5.7. (Max-Flow Min-Cut Theorem) *The value of a maximum flow in a transport network is the same as the capacity of a minimum cut.*

If the capacities of the arcs are rational numbers, then a maximum flow can be found by using Theorem 5.6. The algorithm tries to find an augmenting path for f. If it can not be found, then we have a maximum flow. If we find an augmenting path, then we use it to create a greater flow $\overline{f}$. In the algorithm, we use a label α for the vertices in the following way:

$$\alpha(v) = (u, \text{direction}, \Delta),$$

where u is a vertex in the transport network (or $-$ if it is not defined), "direction" is either forward ($\rightarrow$) or back ($\leftarrow$) (or $-$ if it is not defined) and Δ is a nonnegative real number (or ∞). The point is, whenever a vertex v is labeled, there is an s–v path p which contains the ("directed") arc (u, v) and $\Delta = \epsilon(p)$. A direction is forward if an arc is in the direction of the path and back otherwise. We can label a vertex v when the vertex u has been labeled and either (u, v) or (v, u) is an arc. We have two cases:

(1) (*Forward Label*) If $e = (u, v)$ is an arc and $\alpha(u) = (\cdot, \cdot, \Delta_u)$ as well as $c(e) > f(e)$, then we can write $\alpha(v) = (u, \rightarrow, \Delta_v)$, where

$$\Delta_v = \min\{\Delta_u, c(e) - f(e)\}.$$

(2) (*Back Label*) If $e = (v, u)$ is an arc and $\alpha(u) = (\cdot, \cdot, \Delta_u)$ as well as $f(e) > 0$, then we can write $\alpha(v) = (u, \leftarrow, \Delta_v)$, where

$$\Delta_v = \min\{\Delta_u, f(e)\}.$$

There are two phases in the algorithm. In the first phase, we label the vertices as presented above and each vertex is labeled at most once. The phase ends when the sink t gets labeled as $\alpha(t) = (\cdot, \rightarrow, \Delta_t)$, or when we can not label any more vertices. In the second case, there are no augmenting paths and the flow we obtain is a maximum flow so we stop. In the first case, the flow we obtain is not a maximum flow and we have an augmenting path p for which $\epsilon(p) = \Delta_t$. The algorithm moves on to the second phase. In the second phase, we construct a new greater flow $\overline{f}$ by using the labels of the vertices of p obtained previously. After this, we go back to the first phase with this greater new flow.

The Ford–Fulkerson Algorithm:

1. Choose an initial flow f_0. If we do not have a specific flow in mind, we may use $f_0(e) = 0$. Label the source s by $\alpha(s) \leftarrow (-, -, \infty)$. Set $f \leftarrow f_0$.

2. If we have a unlabeled vertex v, which can be labeled either forward by $(w, \rightarrow, \Delta_v)$ or backward by $(w, \leftarrow, \Delta_v)$, then we choose one such vertex and label it. (There can be many ways of doing this and all of them are permitted.) If such a vertex v does not exist, output the maximum flow f and stop.

3. If t has not been labeled, go to step #2. Otherwise, set $u \leftarrow t$.

4. If $\alpha(u) = (w, \rightarrow, \Delta_u)$, then set

$$f(w,u) \leftarrow f(w,u) + \Delta_t \quad \text{and} \quad u \leftarrow w.$$

   If $\alpha(u) = (w, \leftarrow, \Delta_u)$, then set

$$f(u,w) \leftarrow f(u,w) - \Delta_t \quad \text{and} \quad u \leftarrow w.$$

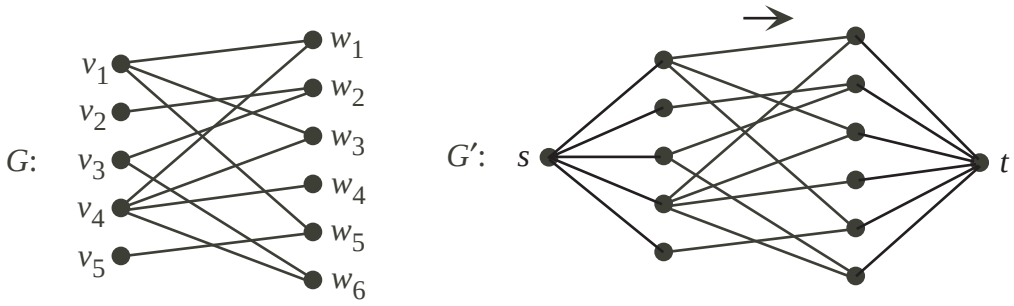
5. If $u = s$, then remove all the labels α but not the label of the source and go to step #2. If $u \neq s$, then go to step #4.

If $f_0(e)$ and $c(e)$ are rational numbers, then the algorithm stops and produces a maximum flow.[4] In this case, we can assume that these weights and capacities are nonnegative integers. Thus, the value of a flow increases by a positive integer every time we move from the second phase to the first phase and the value reaches a maximum eventually. On the other hand, the number of steps can be as large as the value of the maximum flow. The performance time of the algorithm does not only depend on the number of vertices but also the capacities.

The algorithm can be modified[5] so that it does not depend on the capacities. Thus, it will work for irrational capacities. In this case, our purpose during the labeling phase is to find the shortest augmenting path. We get this by always choosing the vertex v in step #2 in such a way that in $\alpha(v) = (w, \cdot, \Delta_v)$, w received its label as early as possible.

The Ford–Fulkerson Algorithm also works for finding a maximum matching in a bipartite graph. Let us do an example:

Example. *Using the bipartite graph G from an example in the previous section, we get a transport network G':*



Every edge of G' is directed from left to right and given a capacity of 1. The initial flow is a zero flow (or a greater flow we obtain from some other initial flow). During the whole process, the flows of the edges are integers 0 or 1. We take into the matching those edges in G whose corresponding edges e in G' receive a flow $f(e) = 1$ and a maximum flow gives a maximum matching. Note that an augmenting path can be of length larger than three in this case. (We can also claim now that the augmenting paths here and the augmenting paths obtained from the Hungarian Algorithm do have something in common after all!)

[4] If there are irrational capacities or flows $f_0(e)$, then the algorithm may not stop at all and it may not produce a maximum flow even if the process repeats endlessly. Of course, we do not have to use irrational flows. In practice, we will not use irrational capacities.

[5] This is known as the *Edmonds–Karp Modification* (refer e.g. to SWAMY & THULASIRAMAN).

Chapter 6

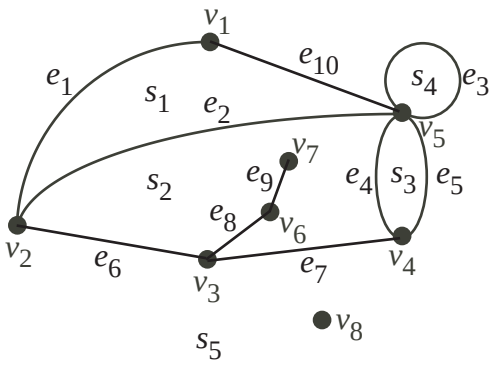
Drawing Graphs

6.1 Planarity and Planar Embedding

We have not treated graphs as geometric objects so far in the course. In practice, we *draw* graphs, i.e. we treat vertices as geometric points and edges as continuous curves. If a graph G can be drawn on a plane (or a sphere) so that the edges only intersect at vertices, then it is *planar*. Such a drawing of a planar graph is a *planar embedding* of the graph.

A connected part of a plane which does not contain any vertices and is surrounded by edges is called a *region* of a planar embedding. In addition, the part outside the embedding is considered as a region, known as the *exterior region* (when we draw a planar graph on a plane or on a sphere, it is just like any other region). The vertices surrounding a region s are called *boundary vertices* and the edges surrounding s are called *boundary edges*. Two regions are *adjacent* if they share a boundary edge. Note that a region can be adjacent to itself.

Example. *In the following planar embedding*



the regions are s_1, s_2, s_3, s_4 and s_5 (the exterior region) and their boundary vertices and edges as well as their adjacent regions are given in the table below:

region	boundary vertices	boundary edges	adjacent regions
s_1	v_1, v_5, v_2	e_1, e_{10}, e_2	s_2, s_5
s_2	$v_2, v_5, v_4, v_3, v_6, v_7$	$e_2, e_4, e_7, e_9, e_8, e_6$	s_1, s_2, s_3, s_5
s_3	v_4, v_5	e_4, e_5	s_2, s_5
s_4	v_5	e_3	s_5
s_5	$v_1, v_5, v_4, v_3, v_2, v_8$	$e_{10}, e_3, e_5, e_7, e_6, e_1$	s_1, s_2, s_3, s_4

In the following, we investigate some fundamental properties of planar embeddings of graphs.

Theorem 6.1. (Euler's Polyhedron Formula[1]) *If a planar embedding of a connected graph G has n vertices, m edges and f regions, then*

$$f + n = m + 2.$$

Proof. Let us use induction on m.

Induction Basis: $m = 0$. Planar embedding of G has only one vertex and one region (the exterior region) so the claim is true.

Induction Hypothesis: The theorem is true for $m \leq \ell$. ($\ell \geq 0$)

Induction Statement: The theorem is true for $m = \ell + 1$.

Induction Statement Proof: We choose an edge e of G and examine the graph $G' = G - e$. If e is in a circuit, then G' is connected and by the Induction Hypothesis, we get

$$f' + n = (m - 1) + 2,$$

where f' is the number of regions in G'. However, closing the circuit with e increases the number of regions by one so $f' = f - 1$ and the theorem is true. If $G - e$ is disconnected, then it has two planar components, G_1 and G_2 whose number of vertices, edges and regions are n_1, n_2, m_1, m_2, f_1 and f_2, respectively. By the Induction Hypothesis,

$$f_1 + n_1 = m_1 + 2 \quad \text{and} \quad f_2 + n_2 = m_2 + 2.$$

While adding e, the number of regions becomes $f_1 + f_2 - 1$ (G_1 and G_2 share the same exterior region or one exterior region is drawn to be a region of the other component), the number of vertices becomes $n_1 + n_2$ and the number of edges becomes $m_1 + m_2 + 1$. Hence, the claim is true. □

Example. *(Continuing from the previous example) We remove the vertex v_8 to get a connected planar embedding. Now, we have 7 vertices, 10 edges, 5 regions and $5 + 7 = 10 + 2$.*

Theorem 6.2. (The Linear Bound) *If a simple connected planar graph G has $n \geq 3$ vertices and m edges, then*

$$m \leq 3n - 6.$$

Proof. If the regions of a planar embedding of G are $s_1, \ldots, s_f$, then we denote the number of boundary edges of s_i by r_i ($i = 1, \ldots, f$). The case $f = 1$ is obvious because G is then a tree and $m = n - 1 \leq 3n - 6$. Thus, we assume that $f \geq 2$. Since G is simple, every region has at least 3 boundary edges and thus

$$\sum_{i=1}^{f} r_i \geq 3f.$$

Every edge is a boundary edge of one or two regions in the planar embedding, so

$$\sum_{i=1}^{f} r_i \leq 2m.$$

The result now follows directly from Euler's Polyhedron Formula. □

[1]The name comes from a polyhedron with n vertices, m edges, f faces and no holes.

Theorem 6.3. (The Minimum Degree Bound) *For a simple planar graph G, $\delta(G) \leq 5$.*

Proof. Let us prove by contradiction and consider the counter hypothesis: G is a simple planar graph and $\delta(G) \geq 6$. Then, (by Theorem 1.1) $m \geq 3n$, where n is the number of vertices and m is the number of edges in G. ($\surd$ Theorem 6.2) □

A characterization of planar graphs is obtained by examining certain forbidden subgraphs.

Theorem 6.4. (Kuratowski's Theorem) *A graph is planar if and only if none of its subgraphs can be transformed to K_5 or $K_{3,3}$ by contracting edges.*

The proof is quite complicated (but elegant!), refer e.g. to SWAMY & THULASIRAMAN for more information. K_5 and $K_{3,3}$ are not planar, which can be verified easily.

There are many fast but complicated algorithms for testing planarity and drawing planar embeddings. For example, the *Hopcroft–Tarjan Algorithm*[2] is one. We present a slower classical polynomial time algorithm, the *Demoucron–Malgrange–Pertuiset Algorithm*[3] (usually just called *Demoucron's Algorithm*). The idea of the algorithm is to try to draw a graph on a plane piece by piece. If this fails, then the graph is not planar.

If G is a graph and R is a planar embedding of a planar subgraph S of G, then an *R-piece* P of G is

- either an edge of $G - S$ whose end vertices are in S, or
- a component of $G - S$ which contains the edges (if any) that connect S to the component, known as *pending edges*, and their end vertices.

Those vertices of an R-piece of G that are end vertices of pending edges connecting them to S are called *contact vertices*. We say that a planar embedding R of the planar subgraph S is *planar extendable* to G if R can be extended to a planar embedding of the whole G by drawing more vertices and/or edges. Such an extended embedding is called a *planar extension* of R to G. We say further that an R-piece P of G is *drawable* in a region s of R if there is a planar extension of R to G where P is inside s. Obviously all contact vertices of P must then be boundary vertices of s, but this is of course not sufficient to guarantee planar extendability of R to G. Therefore we say that a P is *potentially drawable* in s if its contact vertices are boundary vertices of s. In particular, a piece with no contact vertices is potentially drawable in any region of R.

Demoucron's Algorithm:

1. We first check whether or not G is a forest. If it is a forest, then it clearly is planar and can be planar embedded. (There are fast algorithms for this purpose.) We can then stop.
2. If G is not a forest then it must contain at least one circuit. We choose a circuit C, embed it to get the planar embedding D, and set $R \leftarrow D$. (A circuit is obviously planar and is easily planar embedded.)
3. If R is a planar embedding of G, then we output it and stop.

[2]The original reference is HOPCROFT, J.E. & TARJAN, R.E.: Efficient Planarity Testing. *Journal of the ACM* **21** (1974), 549–568.

[3]The original reference is DEMOUCRON, G. & MALGRANGE, Y. & PERTUISET, R.: Graphes planaires: reconnaissance et construction des représentations planaires topologiques. *Revue Française Recherche Opérationnelle* **8** (1964), 33–47.

4. We construct the set $\mathcal{P}$ of all R-pieces of G. For each piece $P \in \mathcal{P}$ we denote by $\mathcal{S}(P)$ the set of all those regions of R which P is potentially drawable in.

5. If, for an R-piece $P \in \mathcal{P}$, the set $\mathcal{S}(P)$ is empty then G is not planar. We can then output this information and stop.

6. Choose an R-piece P, starting from those potentially drawable only in one region.

7. Depending on the number of contact vertices of P, we planar extend R:

   7.1 If P has no contact vertices, we call Demoucron's Algorithm recursively with input P. If it turns out that P is not planar, then G is not planar, and we output this information and stop. Otherwise we extend R to a planar embedding U by drawing P in one of its regions, set $R \leftarrow U$, and return to step #3.

   7.2 If P has exactly one contact vertex v, with the corresponding pendant edge e, we call Demoucron's Algorithm recursively with input P. If it turns out that P is not planar, then G is not planar, and we output this information and stop. Otherwise we extend R to a planar embedding U by drawing P in a region with boundary vertex v, set $R \leftarrow U$, and return to step #3. (This region of R will then be an exterior region of the planar embedding of P.)

   7.3 If P has (at least) two contact vertices v_1 and v_2, they are connected by a path p in P. We then extend R to a planar embedding U by drawing p in a region of R with boundary vertices v_1 and v_2 where P is potentially drawable, set $R \leftarrow U$, and return to step #3.

Clearly, if G is not planar, Demoucron's Algorithm will output this information. On the other hand, the algorithm will not get stuck without drawing the planar embedding if the input is planar, because

Statement. *If G is planar, then at each step of the algorithm R is planar extendable to G.*

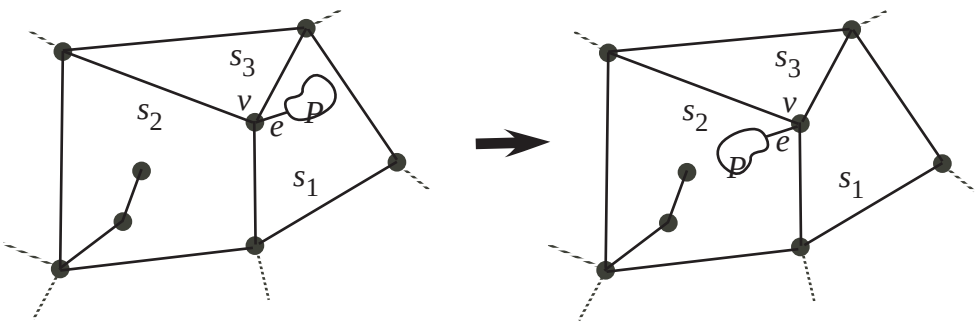
Proof. We use induction on the number of times ℓ the algorithm visits step #7.

Induction Basis: $\ell = 0$. Now either G is a forest (and R is not needed) or R is a circuit of G. Obviously the planar embedding of this circuit can be planar extended to G.

Induction Hypothesis: The statement is true for $\ell \leq r$. ($r \geq 0$)

Induction Statement: The statement is true for $\ell = r + 1$.

Incuction Statement Proof: For step #7.1 the matter is clear. If, in step #7.2, P is potentially drawable in the region s of R, it can always be drawn in this region without endangering subsequent steps. In other words, any possible region can be chosen. This is because the region can be exchanged for another at all times by "reflection" with respect to the vertex v (and possibly rescaling):



Similarly, if in step #7.3, P is drawable in a region of R, then it can be drawn in this region without endangering subsequent steps. If P is drawable in both region s_1 and region s_2, its contact vertices are boundary vertices of both s_1 and s_2. At any time, a drawn P (or part of it) can be moved from region s_1 to s_2, or vice versa, simply by reflection with respect to the common boundary (and possibly rescaling to fit into the region). □

Remark. *Nonplanar graphs may be embedded on closed continuous surfaces with holes. For instance, a torus is closed surface with exactly one hole. On a torus we can embed the nonplanar graphs K_5, K_6 and K_7, and also $K_{3,3}$. K_8 is more complex and its embedding requires a closed surface with two holes. The smallest number of holes in a closed surface required for embedding the graph G on it is called the* genus *of G. On the other hand the smallest number of crossings of edges in a drawing of G on plane is called the* crossing number *of G. Computation of genus and crossing number are both $\mathcal{NP}$-complete problems.*

A coloring of a graph is a labeling of vertices where adjacent vertices never share a label. The labels are then often called *colors*. We say that a graph is *k-colorable* if it can be colored using (at most) k colors. If a graph is colorable then it obviously can not have loops. Equally obviously, parallel edges can be reduced to one, so we may assume our graphs here to be simple. The smallest number k for which the graph G is k-colorable, is called the *chromatic number* of G, denoted by $\chi(G)$.

K_4 is an example of a planar simple graph which is not 3-colorable. On the other hand there is the celebrated

Theorem 6.5. (The Four-Color Theorem) *Every simple planar graph is 4-colorable.*

Proof. The only known proofs require extensive computer runs. The first such proof was obtained by Kenneth Appel ja Wolfgang Haken in 1976. It takes a whole book to present the proof: APPEL, K. & HAKEN, W.: *Every Planar Map is Four Colorable.* American Mathematical Society (1989). □

If we require a bit less, i.e. 5-colorability, then there is much more easily provable result, and an algorithm.

Theorem 6.6. (Heawood's Theorem or The Five-Color Theorem) *Every simple planar graph is 5-colorable.*

Proof. We may think of G as a planar embedding. We use induction on the number n of vertices of G.

Induction Basis: $n = 1$. Our graph is now 1-colorable since there are no edges.

Induction Hypothesis: The theorem is true for $n \leq \ell$. ($\ell \geq 1$)

Induction Statement: The theorem is true for $n = \ell + 1$.

Induction Statement Proof: According to the Minimum Degree Bound, there is a vertex v in G of degree at most 5. On the other hand, according to the Induction Hypothesis the graph $G - v$ is 5-colorable. If, in this coloring, the vertices adjacent to v are colored using at most four colors, then clearly we can 5-color G.

So we are left with the case where the vertices v_1, v_2, v_3, v_4, v_5 adjacent to v are colored using different colors. We may assume that the indexing of the vertices proceeds clockwise, and we label the colors with the numbers $1, 2, 3, 4, 5$ (in this order). We show that the coloring of $G - v$ can be changed so that (at most) four colors suffice for coloring v_1, v_2, v_3, v_4, v_5.

We denote by $H_{i,j}$ the subgraph of $G - v$ induced by the vertices colored with i and j. We have two cases:

- v_1 and v_3 are in different components H_1 and H_3 of $H_{1,3}$. We then interchange the colors 1 and 3 in the vertices of H_3 leaving the other colors untouched. In the resulting 5-coloring of $G - v$ the vertices v_1 and v_3 both have the color 1. We can then give the color 3 to v.

- v_1 and v_3 are connected in $H_{1,3}$. Then there is a v_1–v_3 path in $H_{1,3}$. Including the vertex v we get from this path a circuit C. Now, since we indexed the vertices v_1, v_2, v_3, v_4, v_5 clockwise, exactly one of the vertices v_2 and v_4 is inside C. We deduce that v_2 and v_4 are in different components of $H_{2,4}$, and we have a case similar to the previous one. □

The proof gives a simple (recursive) algorithm for 5-coloring a planar graph, the so-called *Heawood's Algorithm*.

6.2 The Davidson–Harel Algorithm

For the actual drawing of a graph we need to define the drawing area (the "window"), i.e. a rectangular area with sides parallel to the coordinate axes, the drawing curve of the edges (here edges are drawn as line segments), and certain "criteria of beauty", so that the resulting drawing is pleasant to the eye, balanced, and as clear as possible. Such "beauty criteria" are of course context-dependent and even matters of individual taste. In the sequel we restrict ourselves to simple graphs, given by, say, an adjacency matrix or an all-vertex incidence matrix.

We will now present the so-called *Davidson–Harel Algorithm*[4] which, applying an annealing algorithm, aims at better and better drawings of a graph using a certain *ugliness function* (cf. Section 5.7). An ugliness function R computes a numerical *ugliness value* obtained from a drawing P of a graph G. This value is a sum of various contributing factors. We denote, as usual, the sets of vertices and edges of G by $\{v_1, \ldots, v_n\}$ and $\{e_1, \ldots, e_m\}$, respectively. We also denote by $\mathbf{v}_i$ the vector (or geometric point) corresponding to the vertex v_i, and by $\mathbf{e}_j$ the line segment corresponding to the edge e_j. Further, we denote

$$
\begin{aligned}
d_{ij} &= \|\mathbf{v}_i - \mathbf{v}_j\|, \\
r_i &= \text{distance of } \mathbf{v}_i \text{ from the right border of the window,} \\
l_i &= \text{distance of } \mathbf{v}_i \text{ from the left border of the window,} \\
u_i &= \text{distance of } \mathbf{v}_i \text{ from the upper border of the window,} \\
b_i &= \text{distance of } \mathbf{v}_i \text{ from the lower border of the window,} \\
c_j &= \text{length of the line segment } \mathbf{e}_j, \\
f_{ij} &= \begin{cases} 1, \text{ if the line segments } \mathbf{e}_i \text{ and } \mathbf{e}_j \text{ intersect without } e_i \text{ and } e_j \text{ being adjacent} \\ 0 \text{ otherwise,} \end{cases} \\
g_{ij} &= \begin{cases} \text{distance of } \mathbf{v}_i \text{ from the line segment } \mathbf{e}_j \text{ if it exceeds } \gamma \text{ and } v_i \text{ is not an end vertex of } e_j \\ \gamma \text{ otherwise.} \end{cases}
\end{aligned}
$$

γ is a parameter of the algorithm telling how close to vertices edges can be. The ugliness function is then given by

[4]The original reference is DAVIDSON, R. & HAREL, D.: Drawing Graphs Nicely Using Simulated Annealing. *ACM Transactions on Graphics* **15** (1996), 301–331.

$$R(P) = \lambda_1 \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{1}{d_{ij}^2} + \lambda_2 \sum_{i=1}^{n} \left(\frac{1}{r_i^2} + \frac{1}{l_i^2} + \frac{1}{u_i^2} + \frac{1}{b_i^2} \right) + \lambda_3 \sum_{j=1}^{m} c_j^2 + \lambda_4 \sum_{i=1}^{m-1} \sum_{j=i+1}^{m} f_{ij} + \lambda_5 \sum_{i=1}^{n} \sum_{j=1}^{m} \frac{1}{g_{ij}^2},$$

where $\lambda_1, \ldots, \lambda_5$ are nonnegative-valued parameters weighting the contributions of the various factors. (One could actually use negative values as well, whatever the interpretation then might be.)

We can compute $d_{ij}, \ldots, g_{ij}$ quite easily using some basic formulae of vector geometry. We must, however, think about the speed of the computation as well. One way to speed up the computation is to use complex arithmetic. $d_{ij}, \ldots, g_{ij}$ are then equally easily computable.[5] It may also be of advantage to force the vertices into a lattice of geometric points. This can be achieved for instance by rounding the coordinates (or complex numbers) to a fixed accuracy and abandoning drawings where the ugliness function has the value ∞ (this happens e.g. when vertices occupy the same point).

In the annealing process the state is P and the response is $R(P)$. An initial state can be obtained by choosing the points $\mathbf{v}_1, \ldots, \mathbf{v}_n$ in the window randomly, and then drawing the edges accordingly. The state transition process $P \leftarrow A_\rho(P)$ is the following:

- Choose a random vertex $\mathbf{v}_i$. (Alternatively the vertices may be circulated cyclically.)
- Draw a circle of radius ρ centered on $\mathbf{v}_i$. The radius ρ is a parameter, which is initially large and gradually reduced later in some systematic fashion.
- Chose a random point $\mathbf{u}$ on this circle.
- If $\mathbf{u}$ is outside the drawing window, the state remains the same. Otherwise set $\mathbf{v}_i \leftarrow \mathbf{u}$ and change the edges accordingly in the drawing.

The remaing parts of the algorithm are very similar to the annealing algorithm for the TSP in Section 5.7.

Remark. *This method has numerous variants. The window could be a circle and the edges concentric arcs or radii. Or the window could be a sphere and edges drawn as arcs of great circles. The window could also be unbounded, for instance, the whole of $\mathbb{R}^2$. We could "draw" graphs three-dimensionally. Etc. We could also use a metric other than the Euclidean one when computing distances, e.g. the Manhattan metric ("1-norm") or the max-metric ("∞-norm"), geodetic distances on a sphere, etc. Needless to say, the resulting drawings are rather different using these variants of the algorithm.*

It may be noted that using nearly any effective criteria, finding the optimally pleasing drawing of a simple graph is an $\mathcal{NP}$-hard problem.

[5] Note that if $z_1 = x_1 + jy_1$ and $z_2 = x_2 + jy_2$, where j is the imaginary unit, then the real part of $\overline{z_1} z_2$ equals the dot product $(x_1, y_1) \bullet (x_2, y_2)$ and the imaginary part equals the determinant $\begin{vmatrix} x_1 & x_2 \\ y_1 & y_2 \end{vmatrix}$.

Chapter 7

MATROIDS

Many concepts in the preceding chapters do not so much deal with graphs themselves as their structural properties. Examples are various dualities (cut set vs. circuit), principles behind certain algorithms (e.g. Kruskal's Algorithms), and various extremality properties (many structures are the "smallest of their kind", one cannot e.g. remove an edge of a cut set without it losing this property).

Exactly corresponding structures were found in many other areas of mathematics, and they were called matroids.[1]

7.1 Hereditary Systems

A *hereditary family of sets* is a family of sets such that whenever a set F is in the family then so are all subsets of F (and in particular the empty set $\emptyset$). A *hereditary system* M of a set E is a nonempty hereditary family of subsets of E. Included there are also the various ways of specifying $\mathcal{I}_M$, called *aspects.* It will be assumed in what follows that E is a finite set. The following nomenclature is traditional:

- Sets in the family $\mathcal{I}_M$ are called *independent sets* of M.
- The family of subsets of E other than those in $\mathcal{I}_M$ is denoted by $\mathcal{D}_M$ and called the family of *dependent sets* of M.
- An independent set is *maximal* if it is not a proper subset of another independent set. A maximal independent set is called a *basis.* The family of all bases is denoted by $\mathcal{B}_M$. Note that an independent set is always contained in a basis.
- A dependent set is *minimal* if no dependent set is its proper subset. A minimal dependent set is called a *circuit.*[2] (Recall that the empty set is always in $\mathcal{I}_M$.) The family of all circuits is denoted by $\mathcal{C}_M$. Note that a dependent set always contains a circuit.
- A circuit consisting of only one element is a so-called *loop.* Elements of a circuit with two elements are called *parallel.* A hereditary system is *simple* if it has no loops and no parallel elements.

[1] The remarkable thing is that many of these structures were found independently at the same time around the year 1935: Hassler Whitney investigated planarity of graphs, Saunders MacLane geometric lattices of points, and Bartel van der Waerden's topic was independence in vector spaces.

[2] This or any other "familiar sounding" concept should not be confused with the corresponding concept for graphs, even though there is a certain connection, as will be seen!

- The *rank* of a subset F of E is the largest size of an independent set contained in F. (Recall that E is assumed to be finite.) Note that the empty set is always an independent set contained in F. The rank of F is denoted by $\rho_M(F)$, and ρ_M is called the *rank function* of M.

A notation similar to one used for graphs will be adopted in the sequel concerning adding an element e to the set F (denoted by $F + e$) or removing it from F (denoted by $F - e$). Two easy properties of the rank function are the following

Theorem 7.1. *If M is a hereditary system of the set E then*

(i) $\rho_M(\emptyset) = 0$, *and*

(ii) *for any subset F of E and any element e,*

$$\rho_M(F) \leq \rho_M(F + e) \leq \rho_M(F) + 1.$$

Proof. Item (i) is clear, so let us move to item (ii).

Since $F + e$ contains those independent sets that are contained in F, we have $\rho_M(F + e) \geq \rho_M(F)$. On the other hand, possible independent subsets of $F + e$ not contained in F may only consist of an independent subset of F and e, so $\rho_M(F + e) \leq \rho_M(F) + 1$. □

A hereditary system M may of course be specified by giving its independent sets, that is by giving $\mathcal{I}_M$. It can be specified as well by giving its bases, i.e. $\mathcal{B}_M$, independent sets will then be exactly all subsets of bases. On the other hand, M can be specified by giving its circuits, i.e. $\mathcal{C}_M$, independent sets are then the sets not containg circuits. Finally, M can be defined by giving the rank function ρ_M, since a set F is independent exactly when $\rho_M(F) = \#(F)$. (As before, we denote cardinality of a set F by $\#(F)$.) Thus an aspect may involve any of $\mathcal{I}_M$, $\mathcal{B}_M$, $\mathcal{C}_M$ and ρ_M.

It might be mentioned that a hereditary system is a far too general concept to be of much use. This means that well chosen aspects are needed to restrict the concept to a more useful one (that is, a matroid). Let us have a look at certain proper aspects in connection with a matroid well familiar from the preceding chapters.

7.2 The Circuit Matroid of a Graph

The *circuit matroid* $M(G)$ of a graph $G = (V, E)$ is a hereditary system of the edge set E whose circuits are the circuits of G, considered as edge sets. (It is naturally assumed that G is not empty.) The bases of $M(G)$ are the maximal independent edge sets, i.e. spanning forests of G, and the independent sets of $M(G)$ are the subforests, both considered as edge sets. Let us denote $G_F = (V, F)$ for a subset F of E. The number of vertices of G is denoted by n, as usual.

Remark. *A hereditary system that is not directly a circuit matroid of any graph but has a structure identical to one is called a* graphic matroid.

Let us then take a look at different aspects of the circuit matroid.

Basis Exchange Property

Let us consider two bases (i.e. spanning forests) B_1 and B_2. If e is an edge in B_1, its removal divides some component G' of the graph G into two disjoint subgraphs. Now certain edges of B_1 will be the branches of a spanning tree T_1 of G', and similarly, certain edges in B_2 will be the branches of a spanning tree T_2 of G'. The removed edge e is either a branch of T_2 or then a link of T_2^*. In the latter case e will be in the fundamental cut set determined by a branch f of T_2 (cf. Theorem 2.7). Then $T_1 - e + f$ is also a spanning tree of G' and we can replace e by f and get again a spanning forest of G, that is, a basis.

Hence we have

Basis Exchange Property: *If B_1 and B_2 are different bases and $e \in B_1 - B_2$ then there is an element $f \in B_2 - B_1$ such that $B_1 - e + f$ is a basis.*

In general, a hereditary system with the basis exchange property will be a matroid. In other words, the basis exchange property is a proper aspect. Using basis exchange one can move from one basis to another. All bases are thus of the same size.

Uniformity. Absorptivity

For a subset F of E let us denote by n_F the number of vertices in the subgraph $\langle F \rangle$ of G induced by F, and by k_F the number of its components. Then there are $n_F - k_F$ edges in a spanning forest of $\langle F \rangle$. Let us denote further by K_F the number of components of the subgraph G_F of G. Clearly then

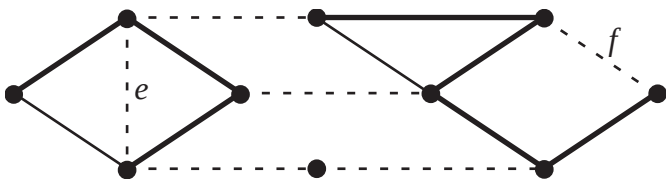
$$\rho_{M(G)}(F) = n_F - k_F = n - K_F,$$

and all such spanning forests are of the same size. Hence

Uniformity: *For a subset F of E all maximal independent subsets of F are of the same size. (Maximality of a set means here that there are no independent sets J such that $H \subset J \subseteq F$.)*

In general, a hereditary system with the uniformity property will be a matroid, and uniformity is a proper aspect.

In the figure below continuous lines are the edges of F, with the thick ones being the branches of a spanning forest. Dashed lines indicate the remaining edges in E.



If e is an edge of G and $\rho_{M(G)}(F + e) = \rho_{M(G)}(F)$ then e does not connect two components of G_F. Suppose f is another edge with the same property, that is, $\rho_{M(G)}(F + f) = \rho_{M(G)}(F)$. Clearly then

$$\rho_{M(G)}(F + e + f) = \rho_{M(G)}(F).$$

Thus we get

Weak Absorptivity: *If $e, f \in E$ and $F \subseteq E$ and*

$$\rho_M(F) = \rho_M(F + e) = \rho_M(F + f)$$

then also

$$\rho_M(F + e + f) = \rho_M(F).$$

In general a weakly absorptive hereditary system is a matroid, and thus weak absorptivity is another proper aspect.

By repeating the above argument sufficiently many times we see that if F and F' are sets of edges of G, and for each edge e in F' we have $\rho_{M(G)}(F + e) = \rho_{M(G)}(F)$, then

$$\rho_{M(G)}(F \cup F') = \rho_{M(G)}(F).$$

Hence also

Strong Absorptivity: *If $F, F' \subseteq E$ and $\rho_M(F + e) = \rho_M(F)$ for each element e in F' then*

$$\rho_M(F \cup F') = \rho_M(F).$$

We conclude that strong absorptivity is a proper aspect.

Augmentation

Suppose I_1 and I_2 are independent sets of the circuit matroid $M(G)$ (edge sets of subforests of G) and $\#(I_1) < \#(I_2)$. The subgraph G_{I_1} then has $n - \#(I_1)$ components, and the subgraph G_{I_2} has $n - \#(I_2)$ components, so strictly less than G_{I_1}. Adding an edge does not reduce the number of components exactly in the case where the edge is added in some component. Thus, if adding any edge in $I_2 - I_1$ to G_{I_1} preserves the number of components then it must be that the edge is added in some component of G_{I_1}, and G_{I_2} cannot have fewer components than G_{I_1}. But as noted, this is not the case if $\#(I_1) < \#(I_2)$, and so

Augmentation: *If I_1 and I_2 are independent sets of the hereditary system M and $\#(I_1) < \#(I_2)$ then there exists an element $e \in I_2 - I_1$ such that $I_1 + e$ is in $\mathcal{I}_M$.*

In general, a hereditary system with the augmentation property is a matroid. Thus augmentation is a proper aspect, too.

Elimination

The circuits of the circuit matroid $M(G)$ are the edge sets of the circuits of G. The degree of a vertex in a circuit is two. If C_1 and C_2 are different circuits of $M(G)$ then the degree of a vertex of the ring sum $\langle C_1 \rangle \oplus \langle C_2 \rangle$ is also even, see Section 1.3. Hence $\langle C_1 \rangle \oplus \langle C_2 \rangle$ must contain at least one circuit as a subgraph, since a ring sum does not have isolated vertices and a nonempty forest has at least one pending vertex (Theorem 2.3). Recalling the definition of ring sum in Section 1.3 it is noticed that such a circuit does not contain edges in the intersection $C_1 \cap C_2$, at least not with as high multiplicity as in $C_1 \cup C_2$. Thus

Elimination Property: *If C_1 and C_2 are different circuits of the hereditary system M and $e \in C_1 \cap C_2$ then there is a circuit $C \in \mathcal{C}_M$ such that $C \subseteq C_1 \cup C_2 - e$.*

Again, elimination property is a proper aspect, and a hereditary system with the elimination property is a matroid.

Induced Circuits

If I is an independent set of the circuit matroid $M(G)$ (edge set of a subforest) then adding one edge either closes exactly one circuit in a component of G_I (Theorem 2.3), or then it connects two components of G_I and does not create a circuit. We have then

Property of Induced Circuits: *If I is an independent set of a hereditary system M and $e \in E$ then $I + e$ contains at most one circuit.*

The property of induced circuits is a proper aspect, and a hereditary system having this property will be a matroid.

7.3 Other Basic Matroids

Vectorial Matroid

Let E be a finite set of vectors of a vector space (say $\mathbb{R}^n$) and the independent sets of a hereditary system M of E be exactly all linearly independent subsets of E (including the empty set). M is then a so-called *vectorial matroid.* Here E is usually allowed to be a multiset, i.e. its elements have multiplicities—cf. parallel edges of graphs. It is then agreed that a subset of E is linearly dependent when one its elements has a multiplicity higher than one. A hereditary system that is not directly vectorial but is structurally identical to a vectorial matroid M' is called a *linear matroid*, and the matroid M' is called its *representation.*

A circuit of a vectorial matroid is a linearly dependent set C of vectors such that removing any of its elements leaves a linearly independent set—keeping in mind possible multiple elements. An aspect typical to vectorial matroids is the elimination property. If $C_1 = \{\mathbf{r}, \mathbf{r}_1, \ldots, \mathbf{r}_k\}$ and $C_2 = \{\mathbf{r}, \mathbf{r}'_1, \ldots, \mathbf{r}'_l\}$ are different circuits sharing (at least) the vector $\mathbf{r}$ then $\mathbf{r}$ can be represented as linear combinations of other vectors in both C_1 and C_2, and in such a way that all coefficients in the combinations are nonzero. We get thus an equality

$$\sum_{i=1}^{k} c_i \mathbf{r}_i - \sum_{j=1}^{l} c'_j \mathbf{r}'_j = \mathbf{0}.$$

Combining (possible) repetitive vectors on the left hand side, and noticing that this does not make it empty, we see that $C_1 \cup C_2 - \mathbf{r}$ contains a circuit. (Note especially the case where either $C_1 = \{\mathbf{r}, \mathbf{r}\}$ or $C_2 = \{\mathbf{r}, \mathbf{r}\}$.)

In the special case where E consists of columns (or rows) of a matrix $\mathbf{A}$, a vectorial matroid of E is called a *matrix matroid* and denoted by $M(\mathbf{A})$. For example, the circuit matroid $M(G)$ of a graph G is a linear matroid whose representation is obtained using the rows of the circuit matrix of G in the binary field $\mathrm{GF}(2)$ (see Section 4.5).[3] Of course, if desired, any vectorial matroid of E may be considered as a matrix matroid simply by taking the vectors of E as columns (or rows) of a matrix.[4]

[3] Hereditary systems with a representation in the binary field $\mathrm{GF}(2)$ are called *binary matroids.* The circuit matroid of a graph is thus always binary.

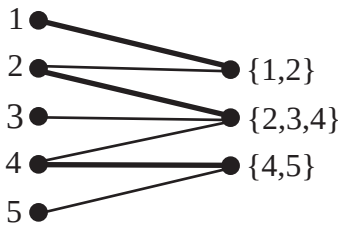
[4] This actually is the origin of the name "matroid". A matroid is a generalization of a linear matroid and a linear matroid may be thought of as a matrix. Indeed, not all matroids are linear. The name "matroid" was strongly opposed at one time. Even today there are people who prefer to use names like "geometry" or "combinatorial geometry".

Transversal Matroid

Let $\mathcal{A} = \{A_1, \ldots, A_k\}$ be a family of nonempty finite sets. The *transversal matroid* $M(\mathcal{A})$ is a hereditary system of the set $E = A_1 \cup \cdots \cup A_k$ whose independent sets are exactly all subsets of E containing at most one element of each of the sets A_i (including the empty set). Here it is customary to allow the family $\mathcal{A}$ to be a multiset, that is, a set A_i may appear several times as its element, thus allowing more than one element of A_i in an independent set.

A natural aspect of transversal matroids is augmentation, and it is connected with augmentings of matchings of bipartite graphs! (See Section 5.8.) Let us define a bipartite graph $G = (V, E)$ as follows: The vertex set is $V = E \cup \mathcal{A}$, and the vertices e and A_j are connected by an edge exactly when $e \in A_j$. (Note how the vertex set V is naturally divided into the two parts of the cut, E and $\mathcal{A}$.) An independent set of $M(\mathcal{A})$ is then a set of matched vertices of G in E, and vice versa.

Example. *In the figure below is the bipartite graph corresponding to the transversal matroid of the family* $\{\{1,2\},\{2,3,4\},\{4,5\}\}$*, and its independent set* $\{1,2,4\}$ *(thick line).*



Very much in the same way as in the proof of Theorem 5.3 one may show that if I_1 and I_2 are independent sets (vertex sets of the matchings S_1 and S_2) and $\#(I_1) < \#(I_2)$ then there is an augmenting path of the matching S_1 such that the new matched vertex is in I_2. Thus $M(\mathcal{A})$ indeed has the augmentation property.

Remark. *For matchings of bipartite graphs the situation is completely general. That is, matchings of bipartite graphs can always be thought of as independent sets of transversal matroids. In fact this remains true for matchings of general graphs, too, leading to the so-called* matching matroids, *see e.g.* SWAMY & THULASIRAMAN.

If the sets of the family $\mathcal{A}$ are disjoint—i.e. they form a partition of E—then the transversal matroid is also called *partition matroid.* For a partition matroid augmentation is obvious.

Uniform Matroid

For all finite sets E one can define the so-called *uniform matroids.* The uniform matroid of E of rank k, denoted $U_k(E)$, is a hereditary system whose independent sets are exactly all subsets of E containing at most k elements. The bases of $U_k(E)$ are those subsets containing exactly k elements, and the circuits are the subsets containing exactly $k+1$ elements. In particular, all subsets of E form a uniform matroid of E of rank $\#(E)$, this is often called the *free matroid* of E. Quite obviously $U_k(E)$ has the basis exchange property and the augmentation property.

Uniform matroids are not very interesting as such. They can be used as "building blocks" of much more complicated matroids, however. It may also be noted that uniform matroids are transversal matroids (can you see why?).

7.4 Greedy Algorithm

Many problems of combinatorial optimization[5] may be thought of as finding a heaviest or a lightest independent set of a hereditary system M of E, when each element of E is given a weight. The weighting function is $\alpha : E \to \mathbb{R}$ and the weight of a set $F \subseteq E$ is

$$\sum_{e \in F} \alpha(e).$$

The two optimization modes are interchanged when the signs of the weights are reversed.

One may also find the heaviest or the lightest bases. Again reversing the signs of the weights interchanges maximization and minimization. If all bases are of the same size—as will be the case for matroids—they can be restricted to the case where there weights are positive. Indeed, if A is the smallest weight of an element of E then changing the weight function to

$$\beta : \beta(e) = 1 + \alpha(e) - A$$

one gets an equivalent optimization problem with positive weights. On the other hand, maximization and minimization are interchanged when the weighting function is changed to

$$\beta : \beta(e) = 1 + B - \alpha(e)$$

where B is the largest weight of an element of E.

Example. *(A bit generalized) Kruskal's Algorithm (see Section 5.6) finds a lightest spanning forest of an edge-weighted graph G, i.e. a lightest basis of the circuit matroid of G. As was seen, this can be done quite fast—and even faster if the edges are given in the order of increasing weight when one can always consider the "best" remaining edge to be included in the forest. Kruskal's Algorithm No. 1 is an example of a so-called greedy algorithm that always proceeds in the "best" available direction. Such a greedy algorithm is fast, indeed, it only needs to find this "best" element to be added in the set already constructed.*

It might be mentioned that Kruskal's Algorithm No. 3 is also a greedy algorithm, it finds a heaviest cospanning forest in the dual matroid of the circuit matroid, the so-called bond matroid of G (see Section 7.6).

Even though greedy algorithms produce the correct result for circuit matroids they do not always do so.

Example. *Finding a lightest Hamiltonian circuit of an edge-weighted graph G may also be thought of as finding the lightest basis of a hereditary system—assuming of course that there are Hamitonian circuits. The set E is again taken to be the edge set of G but now the bases are the Hamiltonian circuits of G (considered as edge sets). A lightest basis is then a lightest Hamiltonian circuit. As was noticed in Section 5.7, finding a lightest Hamiltonian circuit is a well-known $\mathcal{NP}$-complete problem and no greedy algorithm can thus always produce a (correct) result—at least if $\mathcal{P} \neq \mathcal{NP}$. The hereditary system thus obtained is in general a matroid, however (e.g. it does not generally have the basis exchange property).*

It would thus appear that—at least for matroids—greedy algorithms are favorable methods for finding heaviest/lightest bases (or independent sets). Indeed, matroids are precisely those hereditary systems for which this holds true. To be able to proceed further we define the *greedy algorithm* formally. We consider first maximization of independent sets, minimization is given in brackets. The input is a hereditary system M of the set E, and a weighting function α.

[5] These problems are dealt with more extensively in the course Optimization Theory 2.

Greedy Algorithm for Independent Sets:

1. Sort the elements $e_1, \ldots, e_m$ of E according to decreasing [increasing] weight: $e_{(1)}, \ldots, e_{(m)}$.
2. Set $F \leftarrow \emptyset$ and $k \leftarrow 1$.
3. If $\alpha(e_{(k)}) \leq 0$ [$\alpha(e_{(k)}) \geq 0$], return F and quit.
4. If $\alpha(e_{(k)}) > 0$ [$\alpha(e_{(k)}) < 0$] and $F \cup \{e_{(k)}\}$ is independent, set $F \leftarrow F \cup \{e_{(k)}\}$.
5. If $k = m$, return F and quit. Else set $k \leftarrow k + 1$ and go to #3.

For bases the algorithm is even simpler:

Greedy Algorithm for Bases:

1. Sort the elements $e_1, \ldots, e_m$ of E according to decreasing [increasing] weight: $e_{(1)}, \ldots, e_{(m)}$.
2. Set $F \leftarrow \emptyset$ and $k \leftarrow 1$.
3. If $F \cup \{e_{(k)}\}$ is independent, set $F \leftarrow F \cup \{e_{(k)}\}$.
4. If $k = m$, return F and quit. Else set $k \leftarrow k + 1$ and go to #3.

The main result that links working of greedy algorithms and matroids is

Theorem 7.2. (Matroid Greediness Theorem) *The greedy algorithm produces a correct heaviest independent set of a hereditary system* for all weight functions *if and only if the system is a matroid. (This is the so-called* greediness property.*) The corresponding result holds true for bases, and also for finding lightest independent sets and bases. Furthermore, in both cases it suffices to consider positive weights.*

Proof. The first sentence of the theorem is proved as part of the proof of Theorem 7.3 in the next section.

As noted above, greediness is equivalent for maximization and minimization, for both independent sets and bases. It was also noted that finding a heaviest basis may be restricted to the case of positive weights. Since for positive weights a heaviest independent set is automatically a basis, greediness for bases follows from greediness for independent sets.

On the other hand, if greediness holds for bases, it holds for independent sets as well. Maximization of independent sets using the weight function α then corresponds to maximization of bases for the positive weight function

$$\beta : \beta(e) = 1 + \max(0, \alpha(e)),$$

the greedy algorithms behave exactly similarly, item #3 is not activated for independent sets. Elements of weight 1 should be removed from the output. □

Remark. *Greediness is thus also a proper aspect for matroids. For hereditary families of sets it is equivalent to usefulness of the greedy algorithm. Certain other similar but more general families of sets have their own "greediness theorems". Examples are the so-called* greedoids *and* matroid embeddings.

7.5 The General Matroid

Any one of the several aspects above makes a hereditary system a matroid. After proving that they are all equivalent, we may define a *matroid* as a hereditary system that has (any) one of these aspects.

Before that we add one aspect to the list, which is a bit more difficult to prove directly for circuits matroids of graphs:

Submodularity: *If M is a hereditary system of the set E and $F, F' \subseteq E$ then*

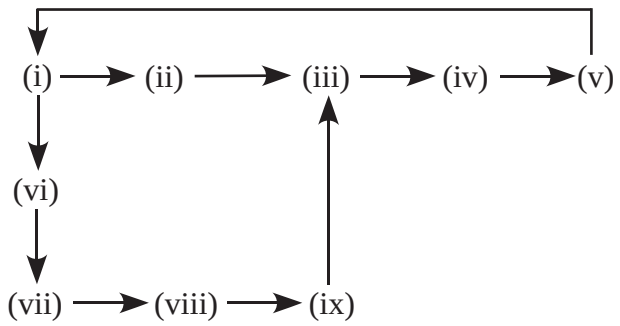
$$\rho_M(F \cap F') + \rho_M(F \cup F') \leq \rho_M(F) + \rho_M(F').$$

Let us then prove the equivalences, including submodularity.

Theorem 7.3. *If a hereditary system has (any) one of the nine aspects below then it has them all (and is a matroid).*

(i) *Uniformity*
(ii) *Basis exchange property*
(iii) *Augmentation property*
(iv) *Weak absorptivity*
(v) *Strong absorptivity*
(vi) *Submodularity*
(vii) *Elimination property*
(viii) *Property of induced circuits*
(ix) *Greediness*

Proof. The implications are proved following the strongly connected digraph below:



All nine aspects are then connected by implication chains in both directions, and are thus logically equivalent. Let us consider a general hereditary system M of the set E.

(i)⇒(ii): As a consequence of uniformity, all bases of M are of the same size. If $B_1, B_2 \in \mathcal{B}_M$ and $e \in B_1 - B_2$, we may apply uniformity to the set $F = (B_1 - e) \cup B_2$. All maximal independent sets included in F are then of the same size as B_2 (and B_1). Now $B_1 - e$ is not one of these maximal sets having too few elements. On the other hand, by adding one element f to $B_1 - e$ we get such an independent set H. The element f must then be in the set difference $B_2 - B_1$, so $H = B_1 - e + f$. Moreover, H has as many elements as B_1, and so it is a basis.

(ii)⇒(iii): If $I_1, I_2 \in \mathcal{I}_M$ and $\#(I_1) < \#(I_2)$, we choose bases B_1 and B_2 such that $I_1 \subseteq B_1$ and $I_2 \subseteq B_2$. Applying basis exchange (repeatedly) we replace those elements of $B_1 - I_1$ that are not in B_2 by elements of B_2. After this operation we may assume that $B_1 - I_1 \subseteq B_2$. As a consequence of the basis exchange property all bases are of the same size. Thus

$$\#(B_1 - I_1) = \#(B_1) - \#(I_1) > \#(B_2) - \#(I_2) = \#(B_2 - I_2),$$

and $B_1 - I_1$ cannot be included in $B_2 - I_2$. Therefore there is an element e of $B_1 - I_1$ in I_2 and $I_1 + e$ is an independent set.

(iii)⇒(iv): Let us consider a situation where

$$\rho_M(F) = \rho_M(F + e) = \rho_M(F + f).$$

If now $\rho_M(F+e+f) > \rho_M(F)$, we take a maximal independent subset I_1 of F and a maximal inpendent subset I_2 of $F + e + f$. Then $\#(I_2) > \#(I_1)$ and by the augmentation property I_1 can be augmented by an element of I_2. This element cannot be in F (why not?), so it must be either e or f. But then $\rho_M(F) < \rho_M(F + e)$ or $\rho_M(F) < \rho_M(F + f)$ ($\surd$).

(iv)⇒(v): Let us assume weak absorptivity and consider subsets F and F' of E such that $\rho_M(F + e) = \rho_M(F)$ for each element e of F'. We use induction on $k = \#(F' - F)$ and show that $\rho_M(F) = \rho_M(F \cup F')$ (strong absorptivity).

Induction Basis: Now $k = 0$ or $k = 1$ and the matter is clear.

Induction Hypothesis: The claimed result holds true when $k \leq \ell$. ($\ell \geq 1$)

Induction Statement: The claimed result holds true when $k = \ell + 1$.

Induction Statement Proof: Choose distinct elements $e, f \in F' - F$ and denote $F'' = F' - e - f$. The Induction Hypothesis implies that

$$\rho_M(F) = \rho_M(F \cup F'') = \rho_M(F \cup F'' + e) = \rho_M(F \cup F'' + f).$$

Applying weak absorptivity to this it is seen that

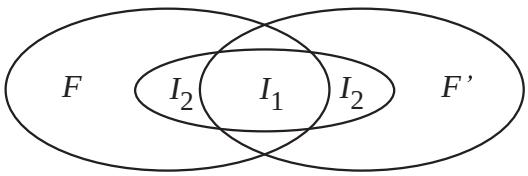
$$\rho_M(F) = \rho_M(F \cup F'' + e + f) = \rho_M(F \cup F').$$

(v)⇒(i): If I is a maximal independent subset of F then $\rho_M(I+e) = \rho_M(I)$ for elements e in the set difference $F - I$ (if any). Strong absorptivity implies then that $\rho_M(F) = \rho_M(I) = \#(I)$, i.e. all these independent sets are of the same size and uniformity holds true.

(i)⇒(vi): Let us consider sets $F, F' \subseteq E$ and denote by I_1 a maximal independent subset of the intersection $F \cap F'$ and by I_2 a maximal inpendent subset of the union $F \cup F'$. Uniformity implies augmentation, so we may assume that I_2 is obtained from I_1 by adding elements, that is $I_1 \subseteq I_2$. Now $I_2 \cap F$ is an independent subset of F and $I_2 \cap F'$ is an independent subset of F', and both of them include I_1. So

$$\begin{aligned}\rho_M(F \cap F') + \rho_M(F \cup F') &= \#(I_1) + \#(I_2) \\ &\stackrel{*}{=} \#(I_2 \cap F) + \#(I_2 \cap F') \leq \rho_M(F) + \rho_M(F').\end{aligned}$$

The equality marked by an asterisk is a set-theoretical one, see the figure below.



(vi)⇒(vii):Let us consider distinct circuits $C_1, C_2 \in \mathcal{C}_M$ and an element $e \in C_1 \cap C_2$. Then $\rho_M(C_1) = \#(C_1) - 1$ and $\rho_M(C_2) = \#(C_2) - 1$, and $\rho_M(C_1 \cap C_2) = \#(C_1 \cap C_2)$. (Remember that every proper subset of a circuit is independent.) If now $C_1 \cup C_2 - e$ does not contain a circuit, it is independent and $\rho_M(C_1 \cup C_2 - e) = \#(C_1 \cup C_2) - 1$, whence $\rho_M(C_1 \cup C_2) \geq \#(C_1 \cup C_2) - 1$. Submodularity however implies that

$$\rho_M(C_1 \cap C_2) + \rho_M(C_1 \cup C_2) \leq \rho_M(C_1) + \rho_M(C_2),$$

and further that (check!)

$$\#(C_1 \cap C_2) + \#(C_1 \cup C_2) \leq \#(C_1) + \#(C_2) - 1.$$

This is a set-theoretical impossibility, and thus $C_1 \cup C_2 - e$ does contain a circuit.

(vii)⇒(viii): If I is an independent set and $I + e$ contains two distinct circuits C_1 and C_2 then obviously both C_1 and C_2 contain the element e. The elimination property implies that $C_1 \cup C_2 - e$ contains a circuit. Since $C_1 \cup C_2 - e$ is however contained in I, it is independent ($\surd$). So $I + e$ contains at most one circuit.

(viii)⇒(ix): Let us denote by I the output of the greedy algorithm for the weighting function α. (The problem is finding a heaviest independent set.) If I is a heaviest independent set, then the matter is clear. Otherwise we take a heaviest independent set having the largest intersection with I. Let us denote this heaviest independent set by I'. I cannot be a subset of I', because the greedy algorithm would then find an even heavier independent set. Let us further denote by e the first element of the set difference $I - I'$ that the greedy algorithm chooses. $I' + e$ is a dependent set and contains thus exactly one circuit C (remember the property of induced circuits). This circuit of course is not included in I, so there is an element $f \in C - I$. Since $I' + e$ contains only one circuit, $I' + e - f$ is an independent set. I' is maximal, so that $\alpha(f) \geq \alpha(e)$. On the other hand, f and those elements of I that the greedy algorithm chose before choosing e are all in I', whence adding f to the elements does not create a circuit. This means that f was available for the greedy algorithm when it chose e, and so $\alpha(f) \leq \alpha(e)$. We conclude that $\alpha(f) = \alpha(e)$ and the sets $I' + e - f$ and I' have equal weight. This however is contrary to the choice of I' because $\#((I' + e - f) \cap I) > \#(I' \cap I)$. (The reader may notice a similarity to the proof of Theorem 5.2. Indeed, this gives another proof for Kruskal's Algorithm No. 1.)

(ix)⇒(iii): Let us consider independent sets I_1 and I_2 such that $\#(I_1) < \#(I_2)$. For brevity we denote $k = \#(I_1)$. Consider then the weighting function

$$\alpha : \alpha(e) = \begin{cases} k + 2, \text{ if } e \in I_1 \\ k + 1, \text{ if } e \in I_2 - I_1 \\ 0 \text{ otherwise.} \end{cases}$$

The weight of I_2 is then

$$\sum_{e \in I_2} \alpha(e) \geq (k+1)^2 > k(k+2) = \sum_{e \in I_1} \alpha(e).$$

It is thus larger than the weight of I_1, so I_1 is not a heaviest independent set. On the other hand, when finding a heaviest independent set the greedy algorithm will choose all elements of I_1 before it ever chooses an element of $I_2 - I_1$. Since it is now assumed to produce a heaviest independent set, it must choose at least one element e of $I_2 - I_1$ and $I_1 + e$ is thus an independent set. This shows that the augmentation property holds true. □

The most popular aspect defining a matroid is probably the augmentation property.

7.6 Operations on Matroids

In the preceding chapters, in connection with fundamental cut sets and fundamental circuits, mutual duality was mentioned. Duality is a property that is very natural for hereditary systems and matroids.

The *dual* (*system*) M^* of a hereditary system M of the set E is a hereditary system of E whose bases are the complements of the bases of M (against E). Often the bases of M^* are called *cobases* of M, circuits of M^* are called *cocircuits* of M, and so on. It is easily checked that M^* really is a hereditary system of E: If $\overline{B}_1$ and $\overline{B}_2$ are distinct bases of M^* then B_1 and B_2 are distinct bases of M. Thus, if $\overline{B}_1 \subseteq \overline{B}_2$ then $B_2 \subseteq B_1$ ($\surd$). Note also that $(M^*)^* = M$.

Theorem 7.4. (Whitney's Theorem) *The dual M^* of a matroid M is a matroid, the so-called* dual matroid, *and*

$$\rho_{M^*}(F) = \#(F) - \rho_M(E) + \rho_M(\overline{F}).$$

(Note that $\rho_M(E)$ is the size of a basis of M.)

Proof. Let us show that M^* has the basis exchange property, which makes it a matroid according to Theorem 7.3. If $\overline{B}_1$ and $\overline{B}_2$ are distinct bases of M^* and $e \in \overline{B}_1 - \overline{B}_2$ then B_1 and B_2 are distinct bases of M and $e \in B_2 - B_1$. Since B_1 is a basis of M, $B_1 + e$ contains exactly one circuit C of M (the property of induced circuits) and this circuit must have an element $f \in B_2 - B_1$. Then however $B_1 + e - f$ does not contain a circuit of M, i.e. it is an independent set of M, and has the same size as B_1. All bases have the same size, so $B_1 + e - f$ is a basis of M and its complement $\overline{B}_1 - e + f$ is a basis of M^*.

To compute the rank $\rho_{M^*}(F)$ we take a maximal independent set H of M^* included in F. Then

$$\rho_{M^*}(F) = \rho_{M^*}(H) = \#(H).$$

Then $\overline{H}$ is a minimal set containing the set $\overline{F}$ and a basis of M. (This is simply the same statement in other words. Note that H is included in some basis of M^*.) But such a set is obtained starting from $\overline{F}$, taking a maximal independent set of M contained in $\overline{F}$—which has $\rho_M(\overline{F})$ elements—and extending it to a basis—which has $\rho_M(E)$ elements. So

$$\#(\overline{H}) - \#(\overline{F}) = \rho_M(E) - \rho_M(\overline{F}).$$

Set theory tells us that

$$\#(\overline{H}) + \#(H) = \#(E) = \#(\overline{F}) + \#(F).$$

Combining these we get the claimed formula for $\rho_{M^*}(F)$ (check!). □

Dualism gives a connection between bases of a matroid M and circuits of its dual matroid M^* (i.e. cocircuits of M):

Theorem 7.5. (i) *Circuits of the dual matroid of a matroid M are the minimal sets that intersect every basis of M.*

(ii) *Bases of a matroid M are the minimal sets that intersect every circuit of the dual matroid M^*.*

Proof. (i) The circuits of M^* are the minimal sets that are not contained in any complement of a basis of M. Thus they must intersect every basis of M.

(ii) Bases of M^* are the maximal sets that do not contain any circuit of M^*. The same in other words: Bases of M are the minimal sets that intersect every circuit of M^*. □

Example. *Bases of the circuit matroid $M(G)$ of a connected graph G are the spanning trees. Bases of the dual matroid $M^*(G)$ are the complements of these, i.e. the cospanning trees. By the theorem, circuits of the dual matroid are the cut sets of G. (Cf. Theorems 2.4 and 2.5.) Because according to Whitney's Theorem $M^*(G)$ is a matroid, it has the greediness property, that is, the*

greedy algorithm finds a heaviest/lightest basis. Working of Kruskal's Algorithm No. 3 is based on this. The algorithm finds the heaviest cospanning tree.

Analogous concepts can naturally be defined for a general, possibly disconnected, graph G. Bases of $M^*(G)$ are then the cospanning forests of G. The dual matroid $M^*(G)$ is called the bond matroid *or the* cut matroid *or the* cocircuit matroid *of G. So, when is the bond matroid $M^*(G)$ graphic, i.e. the circuit matroid of a graph? The so-called* Whitney Planarity Theorem *tells us that this happens exactly when G is a planar graph! (See e.g.* WEST.*)*

If M_i is a hereditary system of the set E_i for $i = 1, \ldots, k$ then the *direct sum* $M = M_1 \oplus \cdots \oplus M_k$ of the systems $M_1, \ldots, M_k$ is the hereditary system of the set $E = E_1 \cup \cdots \cup E_k$ whose independent sets are exactly all sets $I_1 \cup \cdots \cup I_k$ where $I_i \in \mathcal{I}_{M_i}$ ($i = 1, \ldots, k$). In particular, if $E_1 = \cdots = E_k = E$ then the direct sum M is called the *union* of the systems $M_1, \ldots, M_k$, denoted by $M = M_1 \cup \cdots \cup M_k$. Note that each hereditary system M_i could also be thought of as a hereditary system of the set E simply by adding elements of $E - E_i$ as circuits (loops, that is).

It is not exactly difficult to see that if $M_1, \ldots, M_k$ are matroids and the sets $E_1, \ldots, E_k$ are pairwise disjoint then $M = M_1 \oplus \cdots \oplus M_k$ is a matroid, say, by demonstrating the augmentation property (try it!). But actually a more general result holds true:

Theorem 7.6. (Matroid Union Theorem[6]) *If $M_1 \ldots, M_k$ are matroids of the set E then the union $M = M_1 \cup \cdots \cup M_k$ is also a matroid of E and*

$$\rho_M : \rho_M(F) = \min_{F' \subseteq F} \Big(\#(F - F') + \sum_{i=1}^{k} \rho_{M_i}(F') \Big).$$

Proof. The proof is rather long and difficult, and is not given here (see e.g. WEST or OXLEY.) It might be mentioned, though, that the rank formula is not valid for hereditary systems in general. □

The theorem has many fundamental corollaries, e.g.

Corollary. (Matroid Covering Theorem[7]) *If M is a loopless matroid of the set E then the smallest number of independent sets whose union equals E is*

$$\max_{F \subseteq E} \left\lceil \frac{\#(F)}{\rho_M(F)} \right\rceil .$$

Proof. Note first that since M is loopless, each element of E is in itself an independent set. The set E thus can be covered as stated. Take now k copies of M as the matroids $M_1, \ldots, M_k$ in the union theorem. Then E is a union of k independent sets of M exactly when it is an independent set of the union matroid $M' = M_1 \cup \cdots \cup M_k$. The covering property we are interested in can then be expressed in the form $\rho_{M'}(E) = \#(E)$ or, by the union theorem,

$$\#(E) = \min_{F \subseteq E} \Big(\#(E - F) + \sum_{i=1}^{k} \rho_{M_i}(F) \Big)$$

i.e.

$$\min_{F \subseteq E} (k\rho_M(F) - \#(F)) = 0.$$

Since the difference to be minimized is $= 0$ when F is the empty set, k will be the smallest number such that $k \geq \#(F)/\rho_M(F)$ for all nonempty subsets $F \subseteq E$. □

[6] Also known by the names *Edmonds–Fulkerson Theorem* and *Matroid Sum Theorem.*

[7] Also known as *Edmonds' Covering Theorem.*

Example. *For the circuit matroid $M(G)$ of a loopless graph G independent sets are the subforests of G, and we are interested in the minimum number of subforests needed to contain all edges of G. Let us denote this number by $A(G)$, it is called the* arboricity *of G.*

To analyze the maximization in the covering theorem we divide the subgraph $\langle F\rangle$ induced by the edges in F into its components. Numbers of vertices and edges of these components are denoted by $n_1,\dots,n_{k_F}$ and $m_1,\dots,m_{k_F}$, respectively. We use an indexing such that

$$\frac{m_{k_F}}{n_{k_F}-1}\geq\frac{m_{k_F-1}}{n_{k_F-1}-1}\geq\cdots\geq\frac{m_1}{n_1-1}.$$

Now, in general if $\frac{x_2}{y_2}\geq\frac{x_1}{y_1}$ then $\frac{x_2}{y_2}\geq\frac{x_1+x_2}{y_1+y_2}$. Thus

$$\frac{m_2}{n_2-1}\geq\frac{m_1+m_2}{n_1+n_2-2},$$

and continuing inductively, also

$$\frac{m_i}{n_i-1}\geq\frac{m_1+\cdots+m_i}{n_1+\cdots+n_i-i}\quad(i=1,\dots,k_F).$$

In particular then

$$\frac{m_{k_F}}{n_{k_F}-1}\geq\frac{m_1+\cdots+m_{k_F}}{n_1+\cdots+n_{k_F}-k_F}=\frac{\#(F)}{\rho_{M(G)}(F)}.$$

Maximization can thus be restricted to edge-sets F such that $\langle F\rangle$ is connected and $\rho_{M(G)}(F)=n_F-1$ where n_F is the number of vertices of F. (It might be further restricted to edge-sets F such that $\langle F\rangle$ also equals the subgraph induced by its vertices, since connecting two vertices by an edge increases the numerator of the fraction to be maximized, the denominator remaining the same.) Thus we get the celebrated Nash-Williams Formula *for arboricity:*

$$A(G)=\max_{F\subseteq E}\left\lceil\frac{\#(F)}{n_F-1}\right\rceil.$$

It might be noted that since for a simple planar graph $\#(F)\leq 3n_F-6$ (Linear Bound applied to $\langle F\rangle$), $A(G)$ is then at most 3.

The *restriction* of a hereditary system M of the set E into the set $F\subseteq E$ is a hereditary system $M|F$ whose independent sets are exactly those subsets of F that are independent sets of M. The *contraction* of M into the set F is the hereditary system $(M^*|F)^*$, often denoted by $M.F$. Clearly the augmentation property of M is directly transferred to $M|F$, so (cf. Whitney's Theorem)

Theorem 7.7. *If M is a matroid of the set E and $F\subseteq E$ then $M|F$ and $M.F$ are bot matroids, too.*

The *minors* of a matroid M are all those matroids that can be obtained from M by consecutive restrictions and contractions.

References

1. ANDRÁSFAI, B.: *Introductory Graph Theory.* The Institute of Physics (1978)
2. ANDRÁSFAI, B.: *Graph Theory: Flows, Matrices.* The Institute of Physics (1991)
3. BANG-JENSEN, J. & GUTIN, G.: *Digraphs: Theory, Algorithms and Applications.* Springer–Verlag (2002)
4. BOLLOBÁS, B.: *Modern Graph Theory.* Springer–Verlag (2002)
5. CHRISTOFIDES, N.: *Graph Theory. An Algorithmic Approach.* Academic Press (1975)
6. DIESTEL, R.: *Graph Theory.* Springer–Verlag (2005)
7. DOLAN, A. & ALDOUS, J.: *Networks and Algorithms. An Introductory Approach.* Wiley (1999)
8. GIBBONS, A.: *Algorithmic Graph Theory.* Cambridge University Press (1987)
9. GIBBONS, A. & RYTTER, W.: *Efficient Parallel Algorithms.* Cambridge University Press (1990)
10. GONDRAN, M. & MINOUX, M.: *Graphs and Algorithms.* Wiley (1986)
11. GRIMALDI, R.P.: *Discrete and Combinatorial Mathematics.* Addison–Wesley (2003)
12. GROSS, J. & YELLEN, J.: *Graph Theory and Its Applications.* CRC Press (2006)
13. GROSS, J. & YELLEN, J.: *Handbook of Graph Theory.* CRC Press (2003)
14. HOPCROFT, J.E. & ULLMAN, J.D.: *Introduction to Automata Theory, Languages, and Computation.* Addison–Wesley (1979)
15. JUNGNICKEL, D.: *Graphs, Networks and Algorithms.* Springer–Verlag (2004)
16. MCELIECE, R.J. & ASH, R.B. & ASH, C.: *Introduction to Discrete Mathematics.* McGraw–Hill (1990)
17. MCHUGH, J.A.: *Algorithmic Graph Theory.* Prentice–Hall (1990)
18. MEHLHORN, K.: *Graph Algorithms and NP-Completeness.* Springer–Verlag (1984)
19. NOVAK, L. & GIBBONS, A.: *Hybrid Graph Theory and Network Analysis.* Cambridge University Press (1999)
20. OXLEY, J.G.: *Matroid Theory.* Oxford University Press (2006)

21. READ, R.C. & WILSON, R.J.: *An Atlas of Graphs.* Oxford University Press (2004)

22. SKIENA, S.S.: *The Algorithm Design Manual.* Springer–Verlag (1998)

23. SWAMY, M.N.S. & THULASIRAMAN, K.: *Graphs, Networks, and Algorithms.* Wiley (1981)

24. SWAMY, M.N.S. & THULASIRAMAN, K.: *Graphs: Theory and Algorithms.* Wiley (1992)

25. VÁGÓ, I.: *Graph Theory. Application to the Calculation of Electrical Networks.* Elsevier (1985)

26. WALTHER, H.: *Ten Applications of Graph Theory.* Kluwer (1985)

27. WEST, D.B.: *Introduction to Graph Theory.* Prentice–Hall (1996)

Index