BABEŞ-BOLYAI UNIVERSITY
FACULTY OF MATHEMATICS AND INFORMATICS

Zoltán Kása

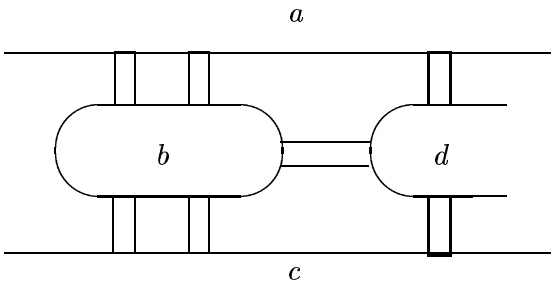
Graph algorithms

2003

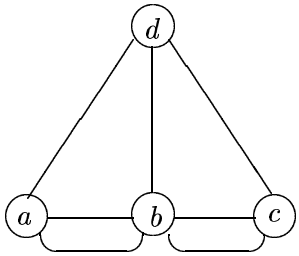
Contents

1 Problems that led to graphs

1) The Königsberg bridge problem (The Euler problem)

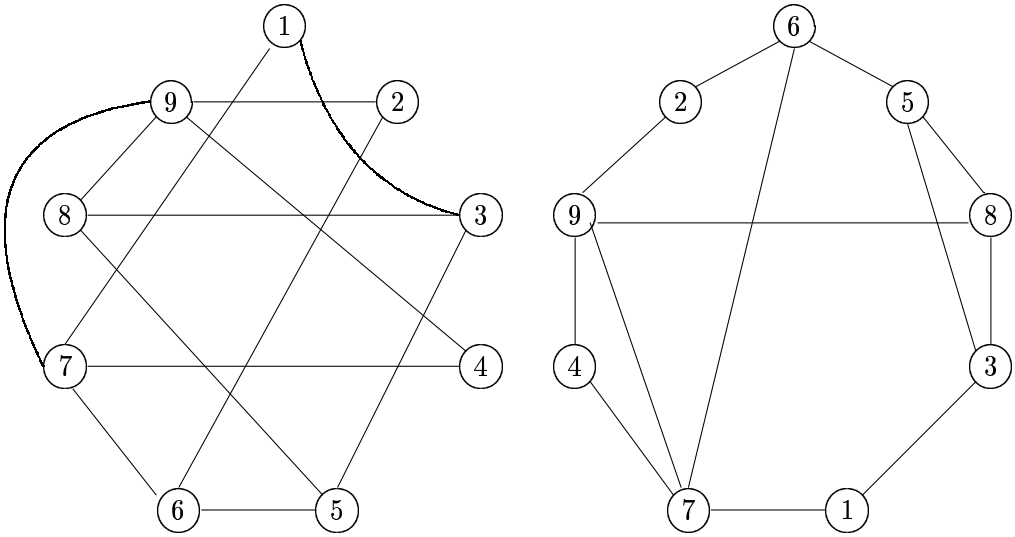


The corresponding graph



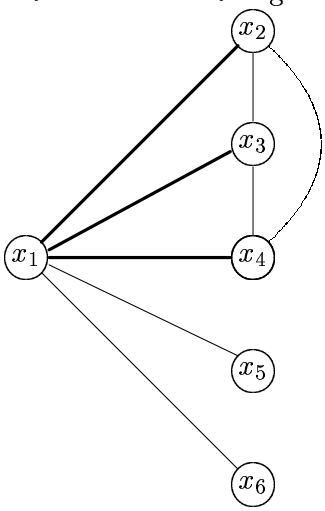
2) A diverting problem (The Hamilton problem)

Put the numbers 1,2,3, ..., 9 on a circle, such that the sum of every two neighbours is not divisible by 3, 5, and 7.



3) A diverting problem 2

In a group of at least six people there are always three mutual acquaintances or three mutual strangers.



Let color the edge between x_i and x_j in red if x_i and x_j are acquainted,

and in blue if they are strangers. The problem can be reformulated: if in a graph with at least six vertices any two vertices are linked by a red or a blue edge, then there exists a red or a blue triangle (three vertices linked pairwise by only red edges or by only blue edges).

x_1 must be linked with at least three vertices by red (or maybe blue) edges. Let these be: x_2, x_3, x_4. If the edge between x_2 and x_3, or between x_2 and x_4, or between x_3 and x_4 is a red one, we have a red triangle (x_1, x_2, x_3 or x_1, x_2, x_4 or x_1, x_3, x_4). If not, then all the edges between three vertices (x_2, x_3, x_4) must be colored din blue, so a blue triangle exists.

2 Basic definitions

$A \times B = \{(a, b) \mid a \in A, b \in B\}$ ordered pairs of elements.

$A \otimes B = \{\{a, b\} \mid a \in A, b \in B \text{ or } a \in B, b \in A\}$ unordered pairs of elements.

Remarks: $\{a, b\}$ is not considered as a set, because a can be equal to b.

The multigraph

multigraph
$G = (V, E, \mathcal{G})$, where V is a nonempty set of *vertices* (or *points* or *nodes*), E is a set of *edges* and $\mathcal{G} : E \to V \otimes V$

To be more precisely, we can use the following notation too:
$G = (V(G), E(G), \mathcal{G}(G))$.

$n = |V|$ the *order* of the multigraph G
$m = |E|$ the *size* of the multigraph G
G is an (n, m) multigraph
If $\mathcal{G}(e_1) = \mathcal{G}(e_2)$ then e_1 and e_2 are *parallel* edges. If $\mathcal{G}(e) = \{a, a\}$ then e is a *loop.*

If $\mathcal{G}(e) = \{a, b\}$, then a and b are joined (linked) by the edge e, a and b are *adjacents* (e is *incident* with a or b)

The set of edges which join the vertices a and b is:

$$\mathcal{G}^{-1}(a, b) = \{e \in E \mid \mathcal{G}(e) = \{a, b\}\}.$$

Let x be a vertex in G. $N_G(x)$ or $N(x)$ is the set of vertices adjacents to x, the *neighborhood* of x:

$$N_G(x) = \{y \in V(G) \mid \exists e \in E(G), \mathcal{G}(e) = \{x, y\}\}$$

or

$$N_G(x) = \{y \in V(G) \mid \mathcal{G}^{-1}(x, y) \neq \emptyset.\}$$

In a multigraph G the set of edges (which are not loops) incidents to a vertex x is:

$$I_G(x) = \{e \in E(G) | \exists y \in V(G), y \neq x, \mathcal{G}(e) = \{x, y\}\}$$

The set of loops incidents to a vertex x is:

$$L_G = (x) = \{e \in E(G) | \mathcal{G}(e) = \{x, x\}\}$$

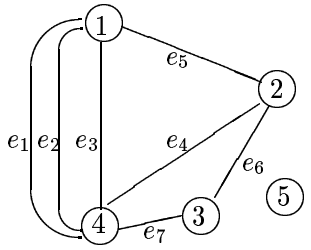
The *degree* of a vertex x denoted bey $\deg(x)$ is number of edges incidents to x (a loop is counted twice). $\deg(x) = |I_G(x)| + 2|L_G(x)|$.

If $\deg(x) = 0$, then x is an *isolated* vertex. If $\deg(x) = 1$, then x is an *end-vertex* or sometimes a *leaf*.

A multigraph without loops and parallel edges is a *graph*. In the case of a graph G we have $|\mathcal{G}^{-1}(a, b)| \leq 1$ for all $a, b \in V$, so instead of writing $\mathcal{G}(e) = \{a, b\}$ we can write $\{a, b\}$ only, to denote an edge. In this case to denote a graph we write $G = (V, E)$ only.

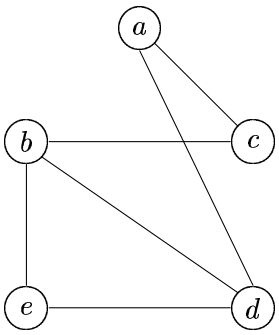
In graphs the *degree* of x, noted by $\deg(x)$ or $\deg_G(x)$, is the number of vertices in $N_G(x)$: $\deg(x) = |N_G(x)|$.

Examples.



The multigraph G_1

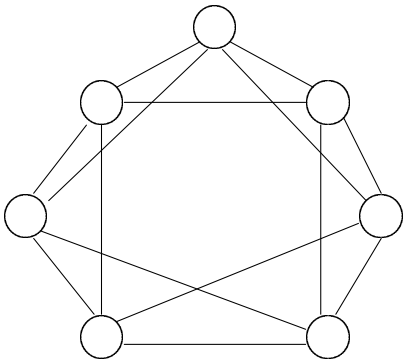
$V(G_1) = \{1, 2, 3, 4, 5\}$, $E(G_1) = \{e_1, e_2, e_3, e_4, e_5, e_6, e_7\}$,
$\mathcal{G}(e_1) = \mathcal{G}(e_2) = \mathcal{G}(e_3) = \{1, 4\}$, $\mathcal{G}(e_4) = \{2, 4\}$, $\mathcal{G}(e_5) = \{2, 1\}$,
$\mathcal{G}(e_6) = \{2, 3\}$, $\mathcal{G}(e_7) = \{3, 4\}$.
$\deg(1) = 4$, $\deg(2) = 3$, $\deg(3) = 2$, $\deg(4) = 5$, $\deg(5) = 0$.



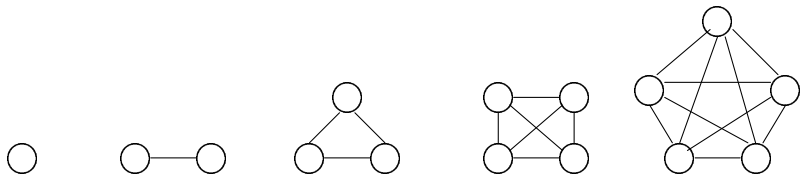
The graph G_2

$V(G_2) = \{a, b, c, d, e\},\ E(G_2) = \big\{\{a, c\}, \{a, d\}, \{b, c\}, \{b, e\}, \{b, d\}\{e, d\}\big\}$

If all vertices of a multigraph have the same degree r, then the multigraph is *regular* of degree r, or *r-regular*. The following graph is an (7,14) 4-regular graph.

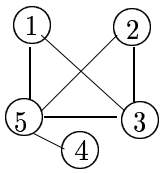


A graph in which all pairs of distinct vertices are adjacents is called a *complete graph*. A complete graph of order n is denoted by K_n.

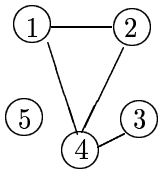


K_1 K_2 K_3 K_4 K_5

The graph $\overline{G} = (\overline{V}, \overline{E})$ is the *complement* of the graph $G = (V, E)$, if $\overline{V} = V$ and $\overline{E} = \big\{\{a, b\} \mid \{a, b\} \notin E\big\}$.



G



$\overline{G}$

If G is a graph of order n, then $E(G) \cup E(\overline{G}) = E(K_n)$.

Two graphs G_1 and G_2 are *isomorphic* if there is a one-to-one function

$$\varphi \, : \, V(G_1) \to V(G_2),$$

such that

$$\text{if } \{a, b\} \in E(G_1), \text{ then } \{\varphi(a), \varphi(b)\} \in E(G_2).$$

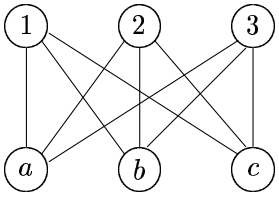
We can define the isomorphism for multigraphs too. Two multigraphs G_1 and G_2 are isomorphic if there is a one-to-one function

$$\varphi \, : \, V(G_1) \to V(G_2),$$

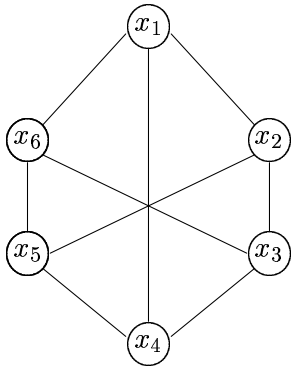
such that

$$|\mathcal{G}_1^{-1}(a, b)| = |\mathcal{G}_2^{-1}(\varphi(a), \varphi(b))| \text{ for all } a, b \in V(G_1).$$

Example of isomorphic graphs



G_1



G_2

The function φ is the following:

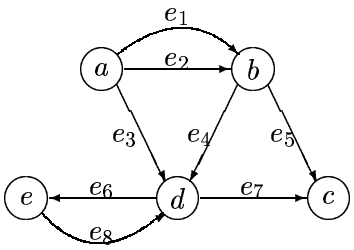
x	1	2	3	a	b	c
$\varphi(x)$	x_1	x_5	x_3	x_2	x_6	x_4

In isomorphic graphs $\deg(x)=\deg(\varphi(x))$ for all $x \in V(G_1)$.

Directed multigraph

directed multigraph

$\vec{G} = (V, E, \vec{\mathcal{G}})$, where
V is a nonempty set of *vertices* (or *points* or *nodes*),
E is a set of *arcs* and
$\vec{\mathcal{G}} : E \to V \times V$



In this directed multigraph the arcs e_1 and e_2 are parallel arcs, but e_6 and e_8 are not.

Let $(\vec{G})$ be a directed multigraph.

$$N^{\text{in}}_{\vec{G}}(y) = \{x \in V(\vec{G}) \mid \vec{\mathcal{G}}^{-1}(x, y) \neq \emptyset\}$$

$$N^{\text{out}}_{\vec{G}}(y) = \{z \in V(\vec{G}) \mid \vec{\mathcal{G}}^{-1}(y, z) \neq \emptyset\}$$

The indegree of x in a multigraph is the number of arcs entering x, the outdegree of x is the number of arcs leaving x. For a graph this definitions can be done by neighborhoods.

The indegree of x in a graph:

$$\text{in-deg}(x) = |N^{\text{in}}(x)|$$

and the outdegree of x in a graph:

$$\text{out-deg}(x) = |N^{\text{out}}(x)|.$$

Sometimes instead of *directed graph* we us the term *digraph.*

Representing the multigraphs

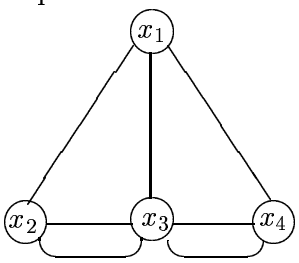
1) – representation by definition

2) – geometrical representation

3) – adjacency matrix
$G = (E, V, \mathcal{G})$, $V = \{x_1, x_2, \ldots, x_n\}$
$A = (a_{ij})_{i,j=\overline{1,n}}$ the adjacency matrix when

$$a_{ij} = \begin{cases} |\mathcal{G}^{-1}(x_i, x_j)| & \text{if } i \neq j \\ 2|\mathcal{G}^{-1}(x_i, x_j)| & \text{if } i = j \end{cases}$$

Example:



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

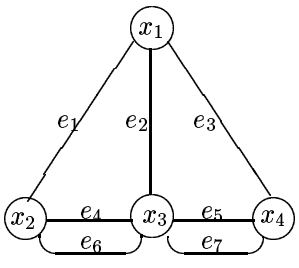
$$\deg(x_i) = \sum_{j=1}^{n} a_{ij}, \text{ for all } i = 1, 2, \ldots, n$$

The adjacency matrix of a graph has only 0s and 1s. In the case of a directed multigraph the representation can be made in the same way.

4) – incidency matrix

$G = (E, V, \mathcal{G})$, $V = \{x_1, x_2, \ldots, x_n\}, E = \{e_1, e_2, \ldots, e_m\}$
$B = (b_{ij})_{i=\overline{1,n}, j=\overline{1,m}}$,

$$b_{ij} = \begin{cases} 1 & \text{if } x_i \text{ is incident with } e_j \text{ and } e_j \text{ is not a loop} \\ 2 & \text{if } x_i \text{ is incident with } e_j \text{ and } e_j \text{ is a loop} \\ 0 & \text{if } x_i \text{ is not incident with } e_j \end{cases}$$



B =

	e_1	e_2	e_3	e_4	e_5	e_6	e_7
x_1	1	1	1	0	0	0	0
x_2	1	0	0	1	0	1	0
x_3	0	1	0	1	1	1	1
x_4	0	0	1	0	1	0	1

5) – lists

a)

x_1	x_2	x_3	x_4		
x_2	x_1	x_3	x_3		
x_3	x_1	x_2	x_2	x_4	x_4
x_4	x_1	x_3	x_3		

also by linked lists

b)

x_2	x_3	x_4	$*$	x_1	x_3	x_3	$*$	x_1	x_2	x_2	x_4	x_4	$*$	x_1	x_3	x_3	$*$	$*$

c)

x_2	x_3	x_4	x_1	x_3	x_3	x_1	x_2	x_2	x_4	x_4	x_1	x_3	x_3
1	4	7	12										

Properties:

1) $G = (V, E, \mathcal{G}), |E(G)| = m$, then $\sum_{x \in V(G)} \deg x = 2m$.

2) The number of odd vertices is even.

3) In every graph there exists at least two vertices with the same degree.

Proof by pigeonhole principle. If $|V(G)| = n$, the degrees can be
$0, 1, \ldots, n-2$ or
$1, 2, \ldots, n-1$.

$\delta(G) = \min_{x \in V(G)} \deg x$ minimal degree in G

$\Delta(G) = \max_{x \in V(G)} \deg x$ maximal degree in G

Submultigraph, subgraph

A multigraph $H = (V(H), E(H), \mathcal{H})$ is a *submultigraph* of $G = (V(G), E(G), \mathcal{G})$ if $V(H) \subseteq V(G)$ and $E(H) \subseteq E(G)$.

For graphs the definition is the same.

induced subgraphs

spanning subgraph, if $V(H) = V(G)$

Walks, trails, paths

In a multigraph $G = (V, E, \mathcal{G})$ an alternating sequence

$$W : v_0, e_1, v_1, e_2, v_2, \ldots v_{i-1}, e_i, v_i, \ldots v_{n-1}, e_n, v_n, \qquad n \geq 0$$

of vertices and edges, where $\mathcal{G}(e_i) = \{v_{i-1}, v_i\}, i = 1, 2, \ldots, n$ is a *walk*. The edges and vertices are not necessary distincts. This is usually called an v_0–v_n walk. In a graph instead of this alternating sequence we can use the sequence of vertices only: $v_0, v_1, \ldots, v_n$, because $|\mathcal{G}(v_{i-1}, v_i)| = 1$ for all $i = 1, 2, \ldots, n$. The number n of edges in a walk is the *length* of that walk.

Special walks:

trail: if no edge is repeated

walk: if no vertice is repeated

Theorem. *Every v_0–v_n walk contains a v_0–v_n path.*

Proof. By inductoin on length of the walk. If $n = 1$ the walk is a path. Let us consider the property true for all length less than n, and let

$$W : v_0, e_1, v_1, \ldots, v_{i-1}, e_i, v_i, \ldots, v_{j-1}, e_j, v_j, \ldots e_n, v_n$$

be a walk which is not a path. So there exists vertices v_i and v_j such that $v_i = v_j$. The walk $W' : v_0, e_1, v_1, \ldots, v_{i-1}, e_i, v_i, e_{j+1} v_{j+1}, \ldots e_n, v_n$ (which results by removing the v_i–v_j walk) is shorter than W. So in W' by the induction hypothesis there exists a v_0–v_n path which is a path in W too.

A u–v walk is a *closed walk* if $u = v$. If a walk is not closed, it is *open*. A closed trail is a *circuit*. A closed path is a *cycle*.

Two vertices u and v in a multigraph are *connected* if there exists a u–v path. A (multi)graph is *connected* if every pair of its vertices are connected. If a (multi)graph is not connected, it is called *disconnected*. If a (multi)graph is disconnected, it has a number of *connected components* or for short *components*. Let us denote by $k(G)$ the number of components of G.

Connected Component Algorithm

Let $x \in V(G)$. Let us define recursively the following sets:

$$U_0 := \{x\}$$

$$U_i := U_{i-1} \cup \left(\bigcup_{y \in U_{i-1}} N(y) \right)$$

$$\exists k : U_k = U_{k-1} = U.$$

The subgraph induced by U is a component.

In the case of directed multigraphs (digraphs):

directed walk

$$v_0, e_1, v_1, e_2, v_2, \ldots v_{i-1}, e_i, v_i, \ldots v_{n-1}, e_n, v_n, \qquad n \geq 0,$$

where $\mathcal{G}(e_i) = (v_{i-1}, v_i), i = 1, 2, \ldots, n$.

directed trail, directed path

directed circuit, directed cycle

semiwalk

$$v_0, e_1, v_1, e_2, v_2, \ldots v_{i-1}, e_i, v_i, \ldots v_{n-1}, e_n, v_n, \qquad n \geq 0,$$

where $\mathcal{G}(e_i) = (v_{i-1}, v_i)$, or $\mathcal{G}(e_i) = (v_i, v_{i-1})$, for $i = 1, 2, \ldots, n$.

weakly connected multigraph if for every pair u, v of vertices there exists a directed u–v path or a directed v–u path, but not the both.

strongly connected multigraph if for every pair u, v of vertices there exists a directed u–v path.

weighted (multi)graph $G = (V, E, \mathcal{G}, \mathcal{W})$

$$\mathcal{W} : E \to \mathbf{R}$$

the length of a weighted path P is $l(P) = \sum_{e_i \in P} \mathcal{W}(e_i)$.

If a multigraph is not weighted, we always can consider that $\mathcal{W}(e) = 1$, for every edge e.

The definitions are the same for directed multigraphs.

Distance between two vertices $d(u, v)$: the length of the shortest u–v path.

$$d(u, v) = \min_{P\ u-v\ \text{path}} l(P)$$

Distance matrix for the graph: $G = (V, E)$, $V = \{v_1, v_2, \ldots, v_n\}$:

$$D = \left(d_{ij}\right)_{i,j=\overline{1,n}} \text{ where } d_{ij} = d(v_i, v_j)$$

Warshall algorithm

$D_0, D_1, D_2, \ldots D_{k-1} = D_k = D$

$D_0 := (d_{ij}^{(0)})_{i,j=\overline{1,n}}$

where $d_{ij}^{(0)} = \begin{cases} \mathcal{W}(v_i, v_j) & \text{if } \{v_i, v_j\} \in E \\ 0 & i = j \\ \infty & \text{if } \{v_i, v_j\} \notin E, i \neq j \end{cases}$

For $k > 0$: $d_{ij}^{(k)} := \min\left(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\right)$ for $i, j = 1, 2, \ldots, n$.

Algorithm to compute the distance matrix based on the Warshall algorithm:

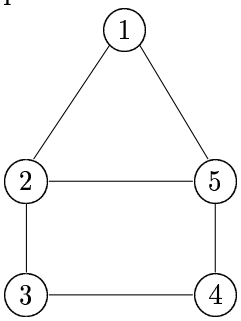
```
D := D_0
for k := 1 to n do
    for i := 1 to n do
        for j := 1 to n do
            d_ij := min(d_ij, d_ik + d_kj)
        endfor
    endfor
endfor
```

The proof that this algorithm is correct is given in: S. Baase: *Computer Algorithms. Introduction to Design and Analysis*, Addison-Wesley Publ. Co, 1983. Chapter 6.

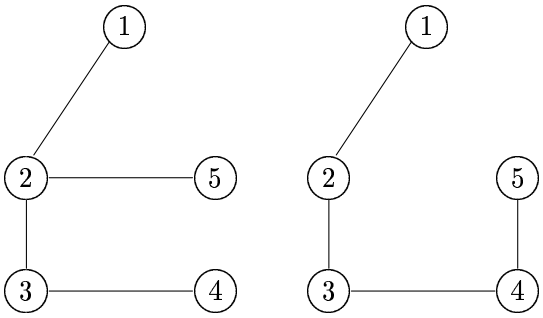
Connected graphs without cycles are called *trees*. Disconnected graphs without cycles are called *forests*.

Rooted tree $\longrightarrow$ digraph

Sometimes we want to visit all the vertices of a (multi)graph. The are two major methods:
– breadth-first search
– depth-first search

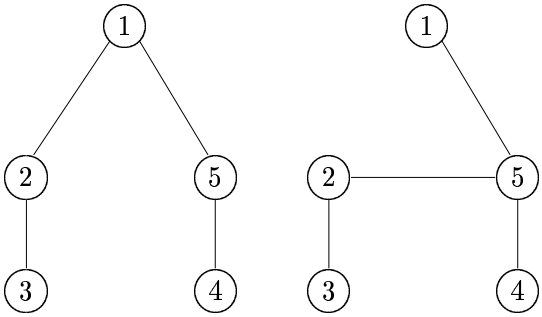


A. Breadth-first search – visiting the vertices as wave is propagated



3, 2, 4, 1, 5 (from 3 to 2, 4, from 2 to 1, 5) or
3, 2, 4, 1, 5 (from 3 to 2, 4, from 2 to 1, from 4 to 5)

B. Depth-first search – visiting the vertices as man is walking

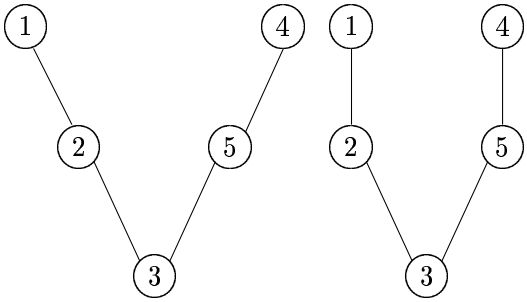


3, 2, 1, 5, 4

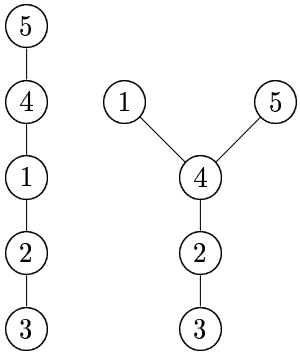
or

3, 2, 5, 1, (5,) 4

A



B



A - wide

B - deep

3 Shortest paths in graphs

Moore's algorithms

An algorithm to find shortest paths from a vertex to all other vertices.

A breadth-first search algorithm.
Let $G = (V, E)$ be a graph. We will use
– u to denote the first vertex,
– $l(v)$ to denote the distance from u to v,
– $p(v)$ to denote the previous vertex of v in the shortest path,
– Q to denote a queue (an element is added to at an end, and removed at the other end).

Notation:
$v \to Q$ adding v to the queue,
$Q \to v$ get the first element from the queue in v and removing it from queue

a) Algorithm to find all distances from a vertex u to all other

Let
$l(u) := 0$
$l(v) := \infty,\ \forall v \in V(G),\ v \neq u$
Q an empty queue
$u \to Q$
while Q is not empty **do**
$Q \to x$
$\forall y \in N(x)$ **if** $l(y) = \infty$ **then**
$p(y) := x$
$l(y) := l(x) + 1$
$y \to Q$
endif
endwhile

b) Algorithm to find a shortest u–v path

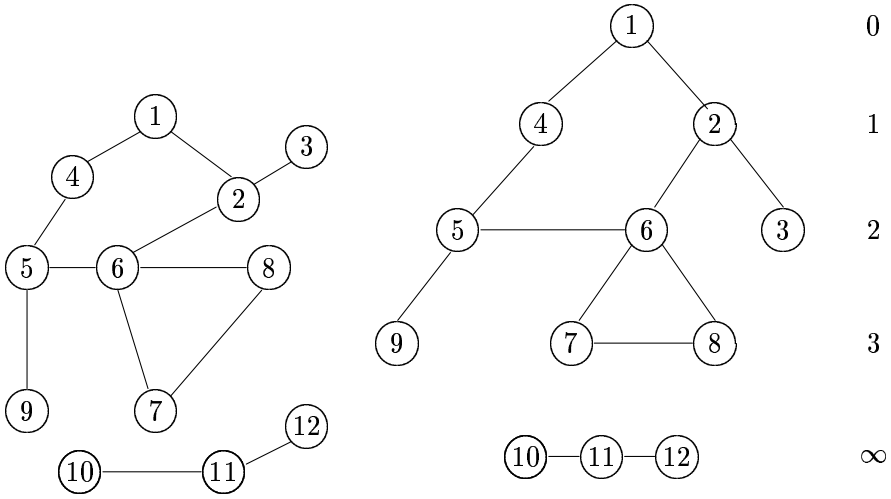
Let
$k := l(v)$
$u_k := v$
while $k \neq 0$ **do**
$u_{k-1} := p(u_k)$

$k := k - 1$

endwhile

The resulting paths is: $u_0, u_1, \ldots, u_k$

Changing $N(x)$ in $N^{\text{out}}(x)$, the algorithm will work for digraphs too.



	1	2	3	4	5	6	7	8	9	10	11	12
l	0	1	2	1	2	2	3	3	3	∞	∞	∞
p		1	2	1	4	2	6	6	5			

Example.
$u = 1$, $v = 8$ so $k := 3$
$u_3 := 8$, $u_2 := 6$, $u_1 := 2$, $u_0 := 1$.
The shortest path between 1 and 8 is: 1, 2, 6, 8.

Shortest paths in weighted graphs

Shortest Paths From A Vertex To All Vertices

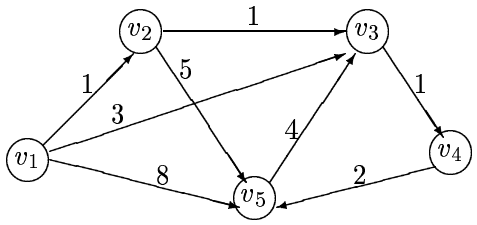
1. Dijkstra's algorithm

$G = (V, E, \mathcal{W})$, where $\mathcal{W} : E \to \mathbf{R}^+$.
Let u the first vertex
$S := \{u\}$, $T := V \setminus S$
$l(u) := 0$
$l(v) := \infty$, $\forall v \in V,\ v \neq u$.
$x := u$

while $T \neq \emptyset$ **do**
  for all $v \in N(x) \cap T$ **do**
    if $l(v) > l(x) + \mathcal{W}(x,v)$ **then** $l(v) := l(x) + \mathcal{W}(x,v)$
      $p(v) := x$
    endif
  endfor
  Let x be the vertex from T for which $l(x)$ is minim: $l(x) = \min_{y \in T} l(y)$.
  $S := S \cup \{x\}$, $T := T \setminus \{x\}$
endwhile

Changing $N(x)$ in $N^{\text{out}}(x)$, the algorithm will work for digraphs too.

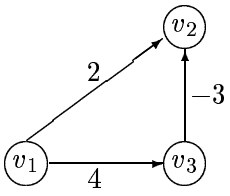
The shortest path can be found by the same **b)** algorithm.



For different values of u we obtain the followings:

	v_i	v_1	v_2	v_3	v_4	v_5
$u = v_1$	l_i	0	1	2	3	5
	p_i	–	v_1	v_2	v_3	v_4
$u = v_2$	l_i	∞	0	1	2	4
	p_i	–	–	v_2	v_3	v_4
$u = v_3$	l_i	∞	∞	0	1	3
	p_i	–	–	–	v_3	v_4
$u = v_4$	l_i	∞	∞	6	0	2
	p_i	–	–	v_5	–	v_4
$u = v_5$	l_i	∞	∞	4	5	0
	p_i	–	–	v_5	v_3	–

For negative weights the algorithm doesn't work.



2. Ford's algorithm

Let $V = \{v_1, v_2, \ldots, v_n\}$ be the set of vertices of the weighted graph.

Let $i := 1$
  $l(v_1) := 0$
  $l(v_i) := \infty$, for $i = 2, 3, \ldots, n$.
while $i \leq n$ **do**
    $\forall v \in N(v_i) : l(v) := \min\big(l(v_i), l(v_i) + \mathcal{W}(v_i, v)\big)$
    if the value $l(v)$ is changed and $(v = v_j$ where $j < i)$ **then** $i := j - 1$
    endif
$i := i + 1$
endwhile

A more detailed description with the vector p of previous vertices.

Let $l(v_1) := 0$
  $l(v_i) := \infty$, for $i = 2, 3, \ldots, n$.
$i := 1$
while $i \leq n$ **do**
  $j := 1$
  while $j \leq n$ **do**
    if $l(v_j) - l(v_i) > \mathcal{W}(v_i, v_j)$ **then**
      $l(v_j) := l(v_i) + \mathcal{W}(v_i, v_j)$
      $p(v_j) := v_i$
      if $j < i$ **then**
        $i := j - 1$
        $j := n$
      endif
    endif
    $j := j + 1$
  endwhile
  $i := i + 1$
endwhile

For digraphs change $N(v_i)$ in $N^{\text{out}}(v_i)$.

Example. Let us consider the weighted graph with the following adjacency matrix:

$$\begin{pmatrix} 0 & 10 & 4 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 7 & 0 & 0 \\ 0 & 3 & 0 & 2 & 2 & 0 & 5 & 15 \\ 0 & 0 & 0 & 0 & 3 & 5 & 0 & 0 \\ 0 & 3 & 0 & 0 & 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 7 & 8 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 6 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

The Ford's algorithm fo $u = 1$ will give the followings:

i :	1	2	3	4	5	6	7	8
l_i:	0	7	4	6	6	11	9	15
p_i:	0	3	1	3	3	4	3	7

This algorithms works in the case of negative weights too, if there is no negative length directed cycles in the digraph.

The Bellman-Kalaba algorithm

This is a matrix algorithm which works on adjacency martix. We choose a vertex (a column in the matrix), and try to obtain distances from all vertices to this one. Let us denote this column by $V^{(1)} = \left(V_i^{(1)}\right)_{i=\overline{1,n}}$. If the adjacency matrix is $A = \left(a_{ij}\right)_{i,j=\overline{1,n}}$, where $a_{ij} = d_{ij}^{(0)}$, that is:

$$a_{ij} = \begin{cases} \mathcal{W}(v_i, v_j) & \text{if } \{v_i, v_j\} \in E(G) \ \ (\text{or } (v_i, v_j) \in E(\vec{G})) \\ 0 & \text{if } i = j \\ \infty & \text{if } \{v_i, v_j\} \notin E(G) \ \ (\text{or } (v_i, v_j) \in E(\vec{G})) \end{cases}$$

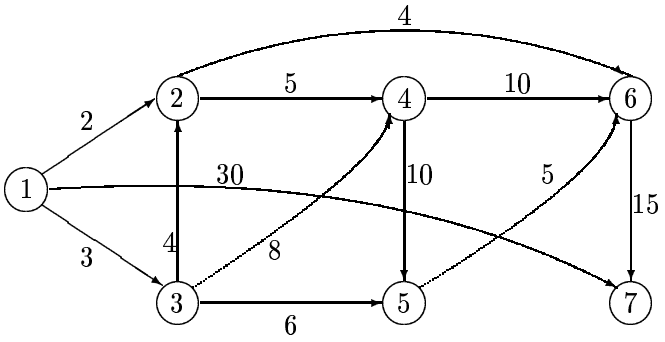
We will compute the following vectors for $k = 1, 2, \ldots$:

$$V_i^{(k)} = \min_{j=\overline{1,n}} \left(a_{ij} + V_j^{(k-1)}\right) \text{ for } i = 1, 2, \ldots, n$$

until $V^{(l)} = V^{(l-1)}$ for an l.

Example.

Let us compute for the following graph the distances to the vertex 7. So the vector $V^{(1)}$ will be the 7^{th} column.



	1	2	3	4	5	6	7	$V^{(1)}$	$V^{(2)}$	$V^{(3)}$	$V^{(4)}$
1	0	2	3	∞	∞	∞	30	30	30	21	21
2	∞	0	∞	5	∞	4	∞	∞	19	19	19
3	∞	4	0	8	6	∞	∞	∞	∞	23	23
4	∞	∞	∞	0	10	10	∞	∞	25	25	25
5	∞	∞	∞	∞	0	5	∞	∞	20	20	20
6	∞	∞	∞	∞	∞	0	15	15	15	15	15
7	∞	∞	∞	∞	∞	∞	0	0	0	0	0

The length of the shortest path betwwen 1 and 7 is 21. How can we find the path efectively? Let us denote $V = V^{(4)}$.

$V_1 = 21$	
$V_1 - V_2 = 2 = \mathcal{W}(1,2)$ $V_1 - V_3 = -2 \neq \mathcal{W}(1,3)$ ... $V_1 - V_7 = 21 \neq \mathcal{W}(1,3)$	so vertex 2 is on the shortest path
$V_2 - V_3 = -4 \neq \mathcal{W}(2,3)$ $V_2 - V_3 = -4 \neq \mathcal{W}(2,4)$ ... $V_2 - V_6 = 4 = \mathcal{W}(2,6)$ $V_2 - V_7 = 21 \neq \mathcal{W}(2,7)$	so vertex 6 is on the shortest path
$V_6 - V_7 = 15 = \mathcal{W}(6,7)$	
The shortest path is: 1, 2, 6, 7.	

Shortest Paths From All Vertices To All Vertices

Using the Warshall-type algorithm to compute the distance matrix, we will complete with a P matrix to get the paths too, not only the distances.

The Floyd-Warshall algorithm

Initially $p_{ij} := i$ if $d_{ij} \neq \infty$ and $i \neq j$, and in other cases $p_{ij} := 0$.

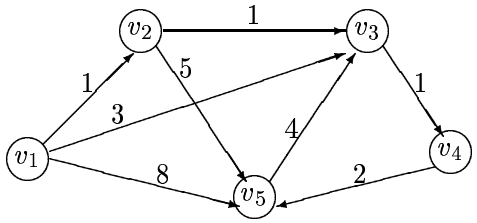
```
D := D_0
for k := 1 to n do
    for i := 1 to n do
        for j := 1 to n do
            if d_ij > d_ik + d_kj then
                d_ij := d_ik + d_kj
                p_ij := p_kj
            endif
        endfor
    endfor
endfor
```

An x–y paths is determined by the following algorithms:

$k := n$:
$u_k := y$
while $u_k \neq x$ **do**
$u_{k-1} := p_{xu_k}$
$k := k - 1$
endwhile

The path is: $u_k, u_{k+1}, \ldots, u_n$

Example.



The adjacency matrix of the weighted graph given in the above figure:

$$D_0 = \begin{pmatrix} 0 & 1 & 3 & \infty & 8 \\ \infty & 0 & 1 & \infty & 5 \\ \infty & \infty & 0 & 1 & \infty \\ \infty & \infty & \infty & 0 & 2 \\ \infty & \infty & 4 & \infty & 0 \end{pmatrix}$$

The distance matrix D and the matrix P of previous vertices:

$$D = \begin{pmatrix} 0 & 1 & 2 & 3 & 5 \\ \infty & 0 & 1 & 2 & 4 \\ \infty & \infty & 0 & 1 & 3 \\ \infty & \infty & 6 & 0 & 2 \\ \infty & \infty & 4 & 5 & 0 \end{pmatrix} \qquad P = \begin{pmatrix} 0 & 1 & 2 & 3 & 4 \\ 0 & 0 & 2 & 3 & 4 \\ 0 & 0 & 0 & 3 & 4 \\ 0 & 0 & 5 & 0 & 4 \\ 0 & 0 & 5 & 3 & 0 \end{pmatrix}$$

The complexity of algorithms

Which algorithm from the above is better? How can the complexity be computed?

The complexity of an algorithm measures the amount of work done by the algorithm when solves a problem. The complexity of an algorithm is a function of the size of input data. Instead of using the computation time in a computer, we shall use the number of basic operations or the number of steps given by algorithm. Only in a few cases we can compute the exact value of

the number of steps or basic operations, so we will compute this number in the worst case, and the complexity will be called *worst-case complexity.* In the following we shall use this type of complexity.

The notation $O\big(f(n)\big)$

$f, g : \mathbf{N} \to \mathbf{Z}$. If there exists a positive real constant C and a nonnegative integer n_0 such that

$$|f(n)| \leq C|g(n)| \quad \text{for all} \quad n \geq n_0,$$

then the *order* of f is lower than or equal to the order of g, and in this case we will write:

$$f(n) = O\big(g(n)\big).$$

If $f(n) = O\big(g(n)\big)$ and $g(n) = O\big(f(n)\big)$ the function f and g have the same order, so we will write: $f(n) = \Theta\big(g(n)\big)$.

Examples. The complexity of the usual matrix multiplication for $A \times B$, when both matrices are $n \times n$ matrices, is $O(n^3)$ (basic operation: multiplication of two numbers).

The complexity of a sequential search algorithm is $O(n)$, the complexity of the bubblesort algorithm is $O(n^2)$ (basic operation: comparison of two elements).

The complexity of the above shortest path algorithms.

algorithm	complexity	complexity for "all to all" case
Moore	$O(n^2)$	$O(n^3)$
Dijkstra	$O(n^2)$	$O(n^3)$
Ford	$O(n^3)$	$O(n^4)$
Bellman-Kalaba	$O(n^3)$	$O(n^4)$
Floyd-Warshall	$O(n^3)$	$O(n^3)$

4 Critical Path Method

First model: activity represented by arc

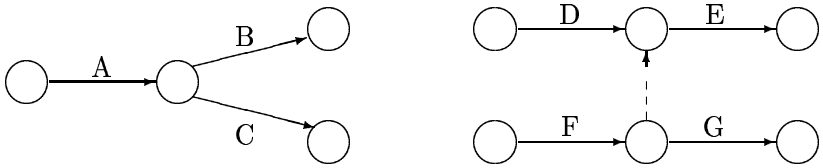
Activity digraph: a connected, acyclic digraph (has no directed cycles) $\vec{G} = (V, E, \mathcal{W})$, where $V = \{v_1, v_2, \ldots, v_n\}$ with the following properties:

– the arcs represent activities, and the corresponding weights are the units of time to complete that activities (execution times of the activities), let us denote $d_{ij} = \mathcal{W}(v_i, v_j)$,

– there exists a *start vertex*, let this be v_1, for which: $N^{\text{in}}(v_1) = \emptyset$,

– there exists a *terminal vertex*, let this be v_n, for which: $N^{\text{out}}(v_n) = \emptyset$.

Connection between activities:



In the above example activity A must be completed before activities B and C start. May exist activities with 0 execution time, these are used only to force some succession (these will be drawn by broken lines). Activity E can start only when both D and F are completed (we have here a dummy activity), but G can start when F is completed.

We are interesting to see the maximal amount of time to complete the entire project. This is the length of the longest path in the activity digraph between the start and terminal vertices. To solve this problem we can use the algorithms for the shortest paths, changing "min" to "max" everywhere. But there is another algorithms too.

Decomposition on levels

The vertices of an acvtivity digraph can be distributed on levels. The start vertex is on level 1. If (v_i, v_j) is an arc, the level of v_i is lower than the level of v_j.

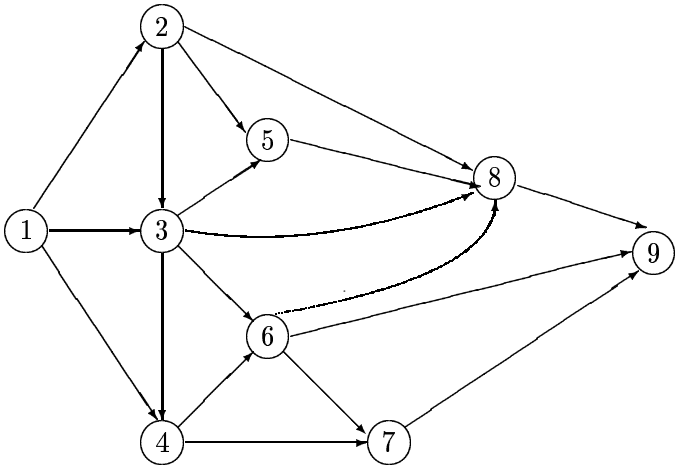
The algorithm for decompostion on levels:

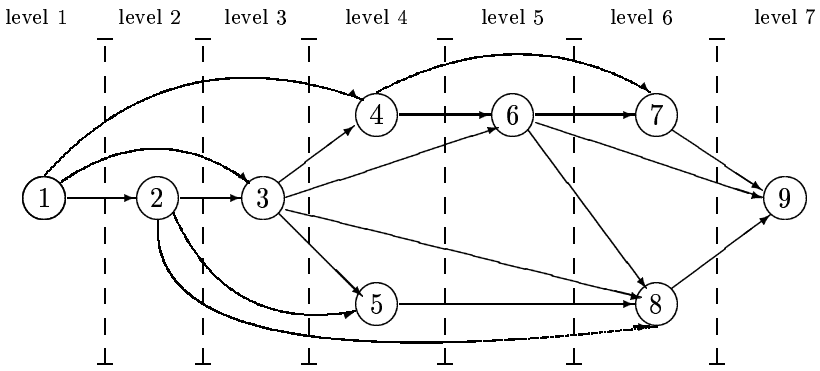
for $i = 1$ **to** n **do**
$l(i) := 1$

```
endfor
for i = 1 to n do
    call next(i)
endfor
```

where:

```
procedure next(i):
            for j = 1 to n do
                if (a_ij ≠ 0) and (l(j) ≤ l(i)) then
                   l(j) := l(i) + 1
                   if j < i then call next(j)
                   endif
                endif
            endfor
end procedure
```

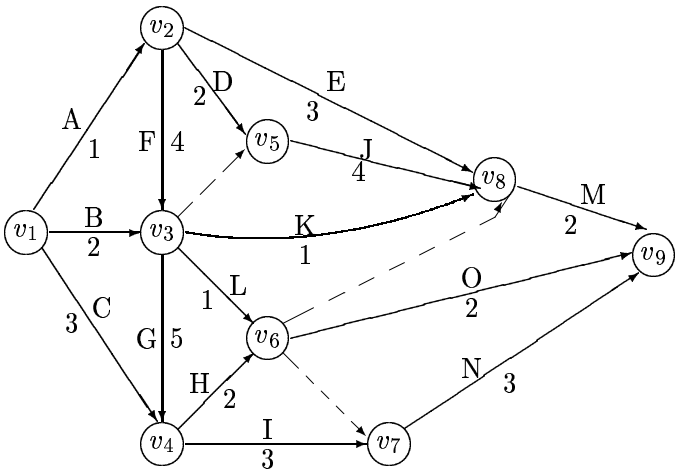




Let the following project to be completed.

activity	precedent activities	execution time
A	–	1
B	–	2
C	–	3
D	A	2
E	A	3
F	A	4
G	B, F	5
H	C, G	2
I	C, G	3
J	B, F, D	4
K	B, F	1
L	B, F	1
M	E, H, J, K, L	2
N	H, I, L	3
O	H, L	2

The corresponding acvtivity digraph is the following:



If we consider that vertices $v_1, v_2, \ldots, v_n$ are distributed on levels in this order, than the following algorithm (the Critical Path Method – CPM) will give us the time moments t_i and t_i^* attached to each vertex v_i in acvtivity digraph. Vertices in acvtivity digraph can be considered as some events is the project. If the moment of beginning the project is considered to be 0, than the time moment t_i represents the earliest time and t_i^* the latest one when all activities from this event v_i can start (these activities are the arcs outgoing from v_i).

$t_1 := 0$
for $j = 2, 3, \ldots n$ **do**

$$t_j := \max_{v_i \in N^{\text{in}}(v_j)} (t_i + d_{ij})$$

endfor
$t_n^* := t_n$
for $i = n-1, n-2 \ldots 1$ **do**

$$t_i^* := \min_{v_j \in N^{\text{out}}(v_i)} (t_j^* - d_{ij})$$

endfor

For our example we have:

vertex	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9
t_i	0	1	2	10	5	12	13	12	16
t_i^*	0	1	5	10	10	13	13	14	16

We define now some time resources that are availables running the project.

$R_t(v_i, v_j) = t_j^* - t_i - d_{ij}$ the *total time resources*. Activity (v_i, v_j) can be started later with this amount of time without influencing the duration of project.

$R_f(v_i, v_j) = t_j - t_i - d_{ij}$ the *free time resources*. Activity (v_i, v_j) can be started later with this amount of time without influencing the time moment t_j.

$R_s(v_i, v_j) = \max(t_j - t_i^* - d_{ij}, 0)$ the *safe time resources*. Activity (v_i, v_j) can be finished later with this amount of time without influencing the duration of project.

Vertices for which all these time resources are equal to 0 are on a path called the *critical path*. The activities on this path must be completed without lateness.

These time resources for our example are the following:

activity	execution time	total time resources: R_t	free time resources: R_f	safe time resources: R_s
A	1	0	0	0
B	2	3	3	3
C	3	7	7	7
D	2	7	2	2
E	3	10	8	8
F	4	0	0	0
G	5	0	0	0
H	2	1	0	0
I	3	0	0	0
J	4	5	3	0
K	1	8	6	6
L	1	7	6	6
M	2	2	2	0
N	3	0	0	0
O	2	2	2	1

We can modify the Floyd-Warshall algorithm to obtain the length of the

longest paths between any two vertices. In D_0 we have:

$$d_{ij}^{(0)} = \begin{cases} \mathcal{W}(v_i, v_j) & \text{if } (v_i, v_j) \in E \\ 0 & \text{if } i = j \\ -\infty & \text{if } (v_i, v_j)y \notin E \end{cases}$$

$D := D_0$
for $k := 1$ **to** n **do**
    for $i := 1$ **to** n **do**
        for $j := 1$ **to** n **do**
            if $d_{ij} < d_{ik} + d_{kj}$ **then** $d_{ij} := d_{ik} + d_{kj}$
            endif
        endfor
    endfor
endfor

The earliest and the latest time for an activity are the following:

$t_i := d_{1i}$ for $i = 1, 2, \ldots, n$
$t_i^* := d_{1n} - d_{in}$ for $i = 1, 2, \ldots, n$

For the above example the length of the longest paths between vertices are:

	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9
v_1	0	1	5	10	5	12	13	12	16
v_2	$-\infty$	0	4	9	4	11	12	11	15
v_3	$-\infty$	$-\infty$	0	5	0	7	8	7	11
v_4	$-\infty$	$-\infty$	$-\infty$	0	$-\infty$	2	3	2	6
v_5	$-\infty$	$-\infty$	$-\infty$	$-\infty$	0	$-\infty$	$-\infty$	4	6
v_6	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	0	0	0	3
v_7	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	0	$-\infty$	3
v_8	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	0	2
v_9	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	0

So, the time moments t_i and t_i^* are:

vertex	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9
t_i	0	1	2	10	5	12	13	12	16
t_i^*	0	1	5	10	10	13	13	14	16

If activities have no precise execution time, but only an optimistic and a pessimistic value, we use the PERT method to evaluate the project time. (PERT = *Programme Evaluation and Review Technique* or *Programme Evolution Research Task*). In this case we shall use some probabilistic methods. The execution time of an activity will be considered as a random values from the interval given by the optimistic and pessimistic values.

Second model: activity represented by vertex

The activity digraph is modified as follows. The vertices will be the activities and the arcs will represent the successions of activities. Activities are considered to be decomposed on levels in the order: $v_1, v_2, \ldots, v_n$. Here the start vertex and the terminal vertex are condired not real activities, they are *dummy activities* (this is to have only one start vertex and only one terminal vertex). We shall use the following for each vertex:

$t_m(v_i)$	v_i	$t_m^*(v_i)$
$t_M(v_i)$	d_i	$t_M^*(v_i)$

d_i – the execution time of activity v_i
$t_m(v_i)$ – the earliest time when activity v_i starts
$t_m^*(v_i)$ – the earliest time when activity v_i ends
$t_M(v_i)$ – the latest time when activity v_i starts
$t_M^*(v_i)$ – the latest time when activity v_i ends

The following algorithm compute all these values:

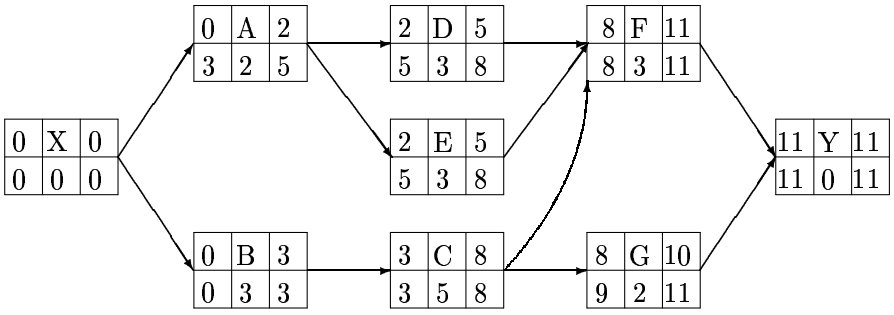
$t_m(v_1) := 0$
$t_m^*(v_1) := d_1$
for $j := 2, 3 \ldots, n$ **do**
    $t_m(v_j) := \max\limits_{v_i \in N^{\text{in}}(v_j)} t_m^*(v_i)$
    $t_m^*(v_j) := t_m(v_j) + d_j$
endfor
$t_M^*(v_n) := t_m^*(v_n)$
$t_M(v_n) := t_M^*(v_n) - d_n$
for $i := n-1, n-2, \ldots, 1$ **do**
    $t_M^*(v_i) := \min\limits_{v_j \in N^{\text{out}}(v_i)} t_M(v_j)$

$t_M(v_i) := t^*_M(v_i) - d_i$
endfor

v is a critical activity if $t_m(v) = t_M(v)$ (and of course $t^*_m(v) = t^*_M(v)$). A path with all activities critical is a critical path.

Let us consider the following example.

activity	precedent activities	execution time
A	–	2
B	–	3
C	B	5
D	A	3
E	A	3
F	C, D, E	3
G	C	2



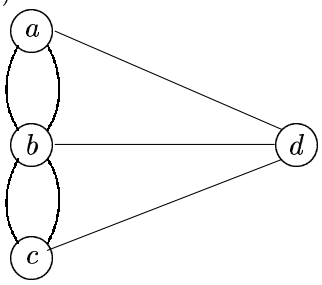
The project can be done in 11 time units. The critical path is: X, B, C, F, Y.

5 Eulerian graphs

A trail is eulerian if it contains all edges of a multigraph. A circuit which contains all edges of a multigraph is an eulerian circuit. A (multi)graph having an eulerian circuit is an *eulerian (multi)graph.* (Take care! In a trail no edges are repeated. In circuit too.)

Theorem 1 *A multigraph without isolated vertices is eulerian if and only if it is connected and the degree of each vertex is even.*

So the graph corresponding to the Königsberg Bridge Problem is not eulerian. The degree of all its vertices is odd ($\deg(a) = \deg(c) = \deg(d) = 3$ and $\deg(b) = 5$).



Proof. **a)** If the multigraph has an eulerian circuit then it must be connected and each vertex has even degree, because the eulerian circuit enter a vertex by an edge and leave it by another one.

b) Let us consider a connected multigraph with all vertices having even degree. We shall prove by induction on number of edges that there is an eulerian circuit in it. If the multigraph is connected and the degree of vertices is even, we easily can find a circuit. If this is not containing all edges of the multigraph, let us delete from multigraph the edges of this circuit. The resulting multigraph, not neccessary connected, will have even degree vertices, but less edges. In each its connected component there will be an eulerian circuit by the induction hypothesis. Let these be $\mathcal{E}_1, \mathcal{E}_2, \ldots \mathcal{E}_k$. The original circuit with these circuits $\mathcal{E}_1, \mathcal{E}_2, \ldots \mathcal{E}_k$ will be an eulerian circuit in the original multigraph. □

Theorem 2 *The connected multigraph G has an eulerian trail if and only if G contains vertices u and v with odd degree and all the other vertices with even degree.*

Proof. By introducing a new edge between u and v the multigraph obtaining will have an eulerian circuit (see Theorem 1). Removing the added edge $\{u, v\}$ an eulerian trail will result. □

Theorem 3 *If in a connected multigraph G the number of odd degree vertices is $2k$ then there exist in G k edge-independent paths which together will cover all edges of the G.*

Proof. Two paths are edge-independent if they have no edges in common. By adding k new edges between the $2k$ vertices to have only even degree vertices and using Theorem 1 the proof is immediately. □

Algorithm to find an eulerian circuit (Fleury)

An edge in a multigraph is a *bridge*, if its deletion increase by one the number of connected components of the multigraph.

Let us check if the multigraph is eulerian. If yes, let us begin with an arbitrary edge. Remove it. Select an adjacent edge to the deleted one which is not a bridge, delete it, and continue until the last edge is selected which must be adjacent to the first one selected. The selected edges will produce an eulerian circuit in order in which were deleted.

Another algorithm based on the proof of the Theorem 1 can easily be found.

For digraphs (and for directed multigraphs too) the following results are true.

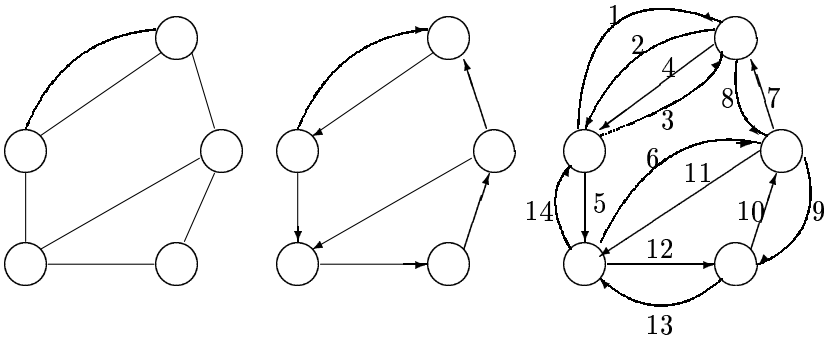
Theorem 4 *A connected digraph contains a directed eulerian circuit if and only if for each vertex* out-deg (v) = in-deg (v).

Theorem 5 *A connected digraph G contains a directed eulerian trail if and only if G contains vertices u and v such that* out-deg (u) = in-deg $(u) + 1$, out-deg (v) = in-deg $(u) - 1$ *and* out-deg (x) = in-deg (x) *for all other vertices x of G. The trail begins at u and ends at v.*

An interesting result on multigraphs:

Theorem 6 *The edges of a connected multigraph can be traversed such that each edge is traversed exactly twice (once in each direction) and this traversal will terminate at the first vertex where it began.*

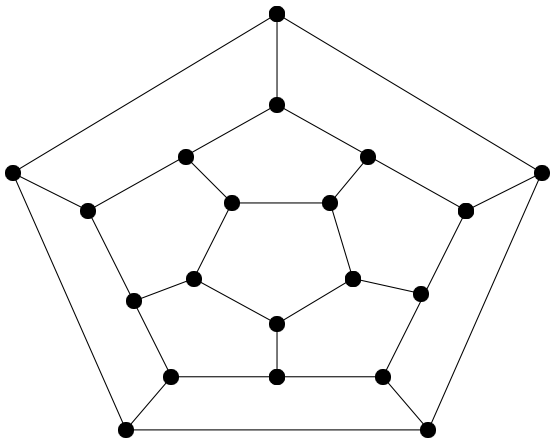
Proof. After any direction of the edges of the multigraph add an arc (v, u) for each arc (u, v). For the resulted directed multigraph we can apply Theorem 4, which we'll give the corresponding circuit.



□

6 Hamiltonian graphs

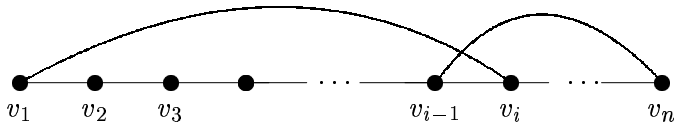
In 1857 the Irish mathematician Sire William R. Hamilton invented a game which used a regular dodecahedron (which has 20 vertices, 30 edges and 12 faces, each face is a regular pentagon). Each vertex was named after a city and the game asks to visit all cities once and return to the first one. The corresponding graph is the following:



A path is *hamiltonian* if contains all vertices of a graph. A cycle which contains all vertices of a graph (a spanning cycle) is a *hamiltonian cycle.* A graph (or maybe a multigraph) is hamiltonian if contains a hamiltonian cycle. This problem is more complicated than the eulerian one. At this time we know only sufficient conditions for a graph to be hamiltonian.

Theorem 7 [Ore] *If in a graph of order* $n \geq 3$ *for each two nonadjacent vertices* u *and* v *we have that* $\deg(u) + \deg(v) \geq n$ *then the graph is hamiltonian.*

Proof. Let us consider the graphs with the above property, but which are not hamiltonian. From these graphs let us consider one which has the maximal number of edges. Adding the edge $\{u, v\}$ the graph will become hamiltonian. So in the original graph there exists a hamiltonian path $u = v_1, v_2, \ldots, v_n = v$.



On this path must exist two adjacent vertices v_{i-1} and v_i such that v_1 is adjacent with v_i and v_n with v_{i-1}, otherwise $\deg(v_1) \leq n-1-\deg(v)$, which is a contradiction. In this case the cycle $v_1, v_2, \ldots, v_{i-2}, v_{i-1}, v_n, v_{n-1}, \ldots v_{i+1}, v_i, v_1$ is a hamiltonian one. □

Theorem 8 [G. Dirac] *If in a graph with at most $2k$ vertices each vertex has degree at least k ($k > 1$) then the graph is hamiltonian.*

Proof 1. Use the Ore's theorem!
Proof 2. (Independent proof)
To prove the theorem we shall use the following lemmas.

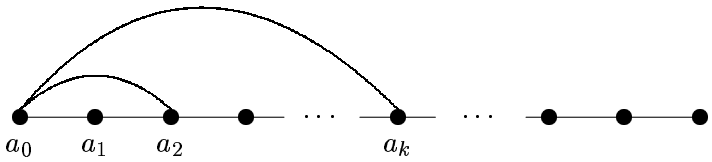
Lemma 1 *If in a graph with at most $2k$ vertices each vertex has a degree at least k then the graph is connected.*

Proof. Let us consider that the graph has at most $2k$ vertices, each vertex has degree at least k, however the graph is not connected. If this is the case, the graph has at least two components and their is a component with no more than k vertices. In this component the maximal degree is $k-1$, which contradicts our assumption that each vertex has degree at least k. □

This property is not true for multigraphs. Give an example!

Lemma 2 *In a graph in which each vertex has degree at least $k > 1$ there exists a cycle with length at least $k+1$.*

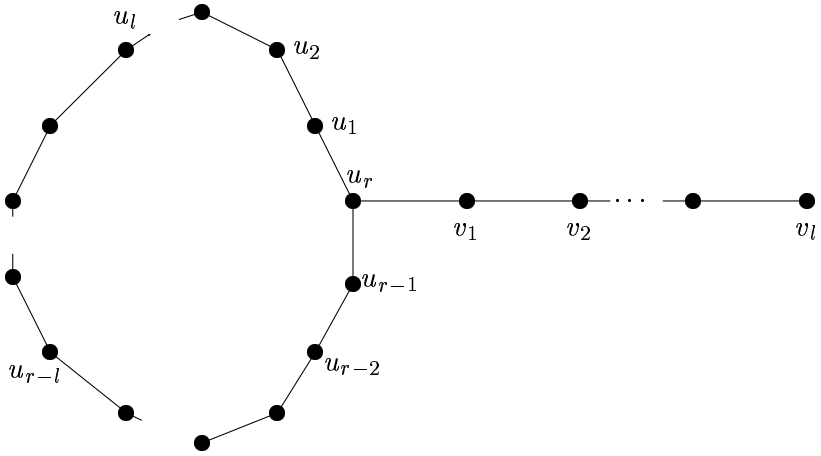
Proof. Let us consider a longest path $a_0, a_1, a_2 \ldots$:



For each vertex v: $\deg(v) \geq k$, so a_0 has degree at least k too. But because the path $a_0, a_1, a_2, \ldots$ is a longest one, a_0 can be adjacent only with

vertices from this path. a_0 has degree at least k, so it must be joined in the worst case with vertices $a_1, a_2, \ldots a_k$. In this case a cycle with at least $k+1$ vertices spring up. □

Proof of the theorem 8. By lemma 1 the graph is connected. By lemma 2 in the graph there exists a cycle with length $r \geq k+1$.



If this is not a hamiltonian one ($r < 2k$), from one vertex of it, maybe u_r, there exists a longest path $u_r, v_1, \ldots, v_l$ which vertices are not on the cycle. Because this is longest path and $\deg(v_l) \geq k$, vertex v_l is adjacent only with vertices from this path or from the cycle. But v_l cannot be adjacent with the vertices $u_1, u_2 \ldots, u_l$ and $u_{r-l}, u_{r-l+1}, \ldots u_r$ (in this case a longer cycle will result). Vertex v_l cannot be adjacent with two adjacent vertices $u_{l+1}, u_{l+2}, \ldots, u_{r-l-1}$ because after replacing the edge between these two adjacent vertices by the two adjacent edges with v_l, a longer cycle result. So v_l can be adjacent only with at most $\dfrac{r-2l}{2}$ vertices from the cycle. But v_l must have at least $k-l$ adjacent vertices from the cycle (because it can have at most l adjacent vertices on the path), so $\dfrac{r-2l}{2} \geq k-l$, that is $r \geq 2k$, which is a contradiction. □

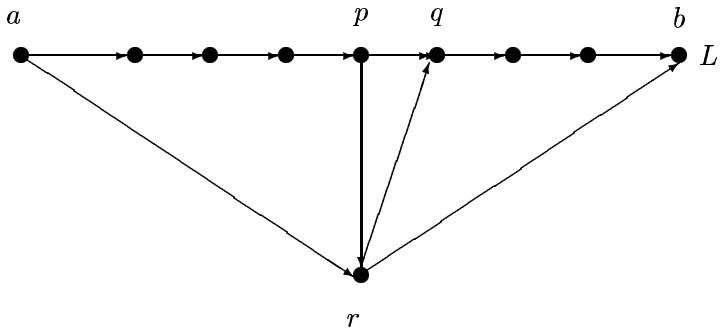
Theorem 9 *Let G be a graph of order $n \geq 2$. If $deg(v) \geq \dfrac{n-1}{2}$ for every vertex v of G, then G contains a hamiltonian path.*

Proof. Let G' be the graph obtained from G by adding a new vertex x and all the edges from this to the other. In G' $\deg(x) = n$ and $\deg(v) \geq$

$\frac{n-1}{2} + 1 = \frac{n+1}{2}$ for all v in G, but G' is of order $n+1$, so by Theorem 8 the graph G' contains a hamiltonian cycle. Removing x from this cycle, a hamiltonian path in G results. □

Theorem 10 [L. Rédei] *After any orientation of edges of a complete graph K_n ($n \geq 2$) the resulting digraph contains a directed hamiltonian path.*

Proof. Let us consider a longest directed path L in the K_n after the orientation of the edges.

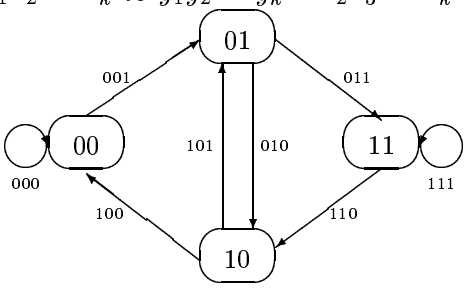


We have the following arcs: (a, r) (if there exists the arc (r, a), L is not the longest path) and (r, a). In this case must exist the arcs: (p, r) and (r, q) and then the path $a, \ldots p, r, q, \ldots, b$ is longer than L, which ia contradiction. □

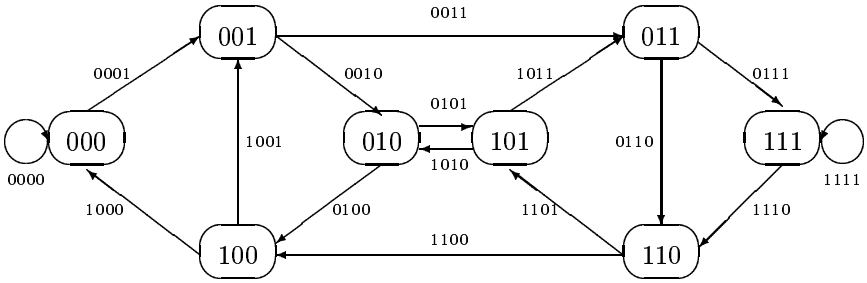
De Bruijn graphs

Let $\mathcal{A}$ be an alphabet of n letters. $\mathcal{A}^k$ is the set of all words of length k over $\mathcal{A}$.

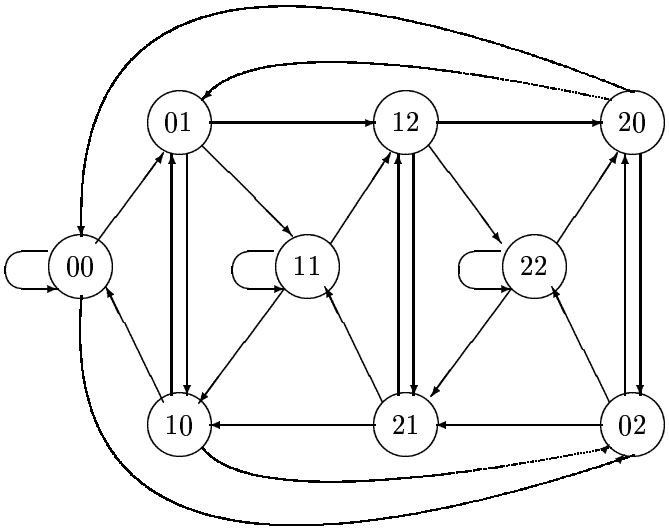
De Bruijn graph: $B(n,k) = (V(n,k), E(n,k))$ where $V(n,k) = \mathcal{A}^k$ is the set of vertices, $E(n,k) = \mathcal{A}^{k+1}$ the set of arcs. There is an arc from $x_1x_2 \ldots x_k$ to $y_1y_2 \ldots y_k$ if $x_2x_3 \ldots x_k = y_1y_2 \ldots y_{k-1}$.



The de Bruijn graph $B(2,2)$.



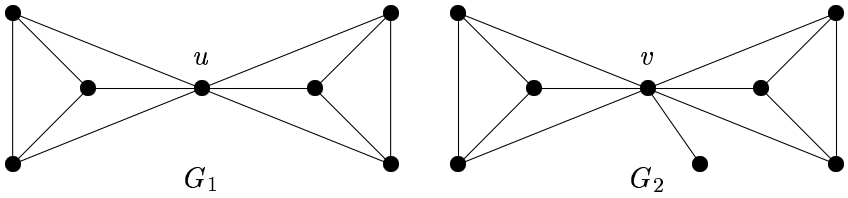
The de Bruijn graph $B(2,3)$.



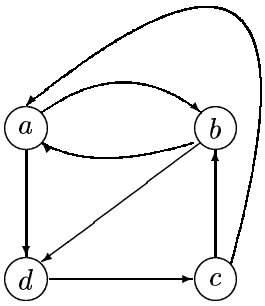
The de Bruijn graph $B(3,2)$.

Properties of hamiltonian graphs

Theorem 11 *If a connected graph G after deleting k vertices becomes a nonconnected graph with more than k components, then G doesn't contain a hamiltonian cycle. If after deleting the k vertices G becomes a nonconnected graph with more than $k+1$ components, then G doesn't contain even a hamiltonian path.*


u
v
G_1
G_2

Latin matrix (or square)



$$A = \begin{pmatrix} 0 & ab & 0 & ad \\ ba & 0 & 0 & bd \\ ca & cb & 0 & cd \\ 0 & 0 & dc & 0 \end{pmatrix} \qquad A^* = \begin{pmatrix} 0 & b & 0 & d \\ a & 0 & 0 & d \\ a & b & 0 & d \\ 0 & 0 & c & 0 \end{pmatrix}$$

$A^{(2)} = A \otimes A^*$
$A^{(k)} = A^{(k-1)} \otimes A^*$, for $k \geq 2$.

$$A^{(2)} = \begin{pmatrix} 0 & 0 & adc & abd \\ 0 & 0 & bdc & bad \\ cba & cab & 0 & \left\{ \begin{matrix} cad \\ cbd \end{matrix} \right\} \\ dca & dcb & 0 & 0 \end{pmatrix}$$

$$A^{(3)} = \begin{pmatrix} 0 & adcb & abdc & 0 \\ bdca & 0 & badc & 0 \\ 0 & 0 & 0 & \left\{ \begin{matrix} cbad \\ cabd \end{matrix} \right\} \\ dcba & dcab & 0 & 0 \end{pmatrix}$$

$$A^{(4)} = \begin{pmatrix} \left\{ \begin{matrix} adcba \\ abdca \end{matrix} \right\} & 0 & 0 & 0 \\ 0 & \left\{ \begin{matrix} bdcab \\ badcb \end{matrix} \right\} & 0 & 0 \\ 0 & 0 & \left\{ \begin{matrix} cbadc \\ cabdc \end{matrix} \right\} & 0 \\ 0 & 0 & 0 & \left\{ \begin{matrix} dcbad \\ dcabd \end{matrix} \right\} \end{pmatrix}$$

7 Trees and forests

Let us consider a multigraph $G = (E, V, \mathcal{G})$, where $m = |E|$, $n = |V|$, and $k = \kappa(G)$ is the number of connected components. We define the *cyclomatic number* as $\nu(G) = m - n + k$.

Theorem 12 *Let G be a multigraph and G' the graph obtained by adding an edge to G.*

a) If the new edge is a loop or join two vertices in the same component, then

$$\nu(G') = \nu(G) + 1.$$

b) If the edge join two vertices from two components, then

$$\nu(G') = \nu(G).$$

Proof. In case a) by adding an edge will increase only the number of edges by 1, so $\nu(G') = \nu(G) + 1$. In case b) the number of edges will increase by 1, but the number of component will decrease by 1. So $\nu(G') = \nu(G)$. □

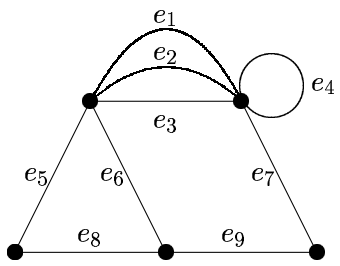
Let $E = \{e_1, e_2, \ldots, e_m\}$. A cycle is represented by a (characteristic) vector $\mathbf{v} = (v_1, v_2, \ldots v_m)$ where $v_i = 1$ when the edge e_i belongs to cycle and $v_i = 0$ otherwise. We define the addition of vectors: $\mathbf{v} = \mathbf{v^1} + \mathbf{v^2}$, where

$$v_i = \begin{cases} 1 & \text{if } v_i^1 + v_i^2 = 1 \\ 0 & \text{if } v_i^1 + v_i^2 = 2 \text{ or } v_i^1 + v_i^2 = 0 \end{cases}$$

The cycles $c_1, c_2, \ldots c_p$ are independent if for the correspondig characteristic vectors $\mathbf{v^1}$, $\mathbf{v^2}$, ..., $\mathbf{v^p}$ we will have for any $\alpha_i \in \{0, 1\}$:

$$\alpha_1\mathbf{v^1} + \alpha_2\mathbf{v^2} + \ldots + \alpha_p\mathbf{v^p} = \mathbf{0} \;\Rightarrow\; \alpha_1 = 0, \alpha_2 = 0, \ldots \alpha_p = 0,$$

where $\mathbf{0} = (0, 0, \ldots, 0)$.



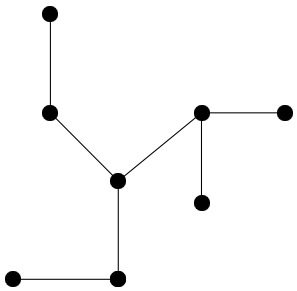
The characteristic vector for the cycle $[e_1, e_6, e_9, e_7]$ is $(1,0,0,0,0,1,1,0,1)$. The cycles $[e_5, e_6, e_8]$, $[e_6, e_3, e_7, e_9]$ and $[e_3, e_5, e_8, e_9, e_7]$ are not independent, because for their characteristic vector we have:
$(0,0,0,0,1,1,0,1,0) + (0,0,1,0,0,1,1,0,1) + (0,0,1,0,1,0,1,1,1) = 0$.

A maximal independent vector set is a fundamental (or basic) cycle system. Every cycle not in the basic cycle system can be written as a linear combinations of cycles (vectors) in the basic system.

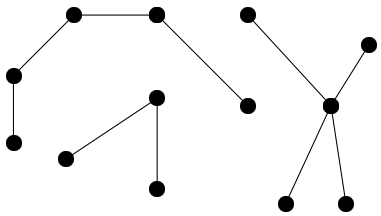
$\nu(G)$ represents the maximal number of independent cycles (or the cardinal of the basic cycles system) in G. If $\nu(G) = 0$ then G has no cycles.

A *tree* is a connected graph without cycles. If a graph has no cycles it is a *forest* (which is a union of trees). So for a forest we shall have $\nu(G) = 0$. (A tree is a connected forest.)

Vertices with degree 1 are called *leaves.* In each tree there are at least two leaves.

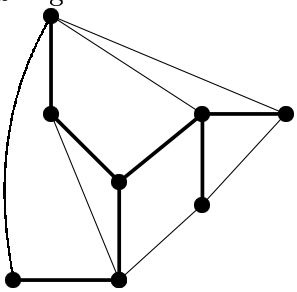


tree



forest

A *spanning tree* of a graph G is a subgraph of G which is a tree and contains all vertices of G (it is spanning subgraph which is a tree). In a tree every edge is bridge.



Theorem 13 *Let G be a graph of order n. The following statements are equivalent and characterize the trees.*

(1) G is connected and without cycles,

(2) G is without cycles and it contains $n-1$ edges,

(3) G is connected and it contains $n-1$ edges,

(4) G is without cycles, but adding an edge between any two not adjacents vertices, a cycle will result,

(5) G is connected, and after eliminating any edge it become disconnected,

(6) any two vertices of the graph are connected by a single path.

Proof. *(1) $\Rightarrow$ (2)*: If G is connected then $k = \kappa(G) = 1$ (composed by only one component), and without cycles, then $\nu(G) = m - n + 1 = 0$, so $m = n - 1$.

(2) $\Rightarrow$ (3): G without cycles: $\nu(G) = m - n + k = 0$ and $m = n - 1 \Rightarrow k = 1$.

(3) $\Rightarrow$ (4): $k = 1$ and $m = n - 1 \Rightarrow \nu(G) = 0$. Adding a new edge, in $\nu(G)$ only the value of m will increase, so $\nu(G) = 1$, a cycle will appear.

(4) $\Rightarrow$ (5): If G is not connected, adding a new edge between two vertices in two components a new cycle doesn't result, which is a contradiction with *(4)*, so G must be connected.

G is without cycles and connected, so $\nu(G) = m - n + 1 = 0$ and $m = n - 1$, eliminating any edge, only m will decrease, and k must be equal to 2 to have $\nu(G) = m - n + k = 0$.

(5) $\Rightarrow$ (6): G is connected, and if between any two vertices exist 2 different paths, a cycle result, but in this case by eliminating an edge from cycle, G will remain connected, which is contradiction with *(5)*.

(6) $\Rightarrow$ (1): If any two vertices are connected by a path, the graph is connected. If G contains a cycle between two vertices of this, there are two paths between them, which is a contradiction. □

Theorem 14 *A spanning tree T of a graph has the following properties:*

(1) T is connected,

(2) T is without cycle,

(3) T has n vertices,

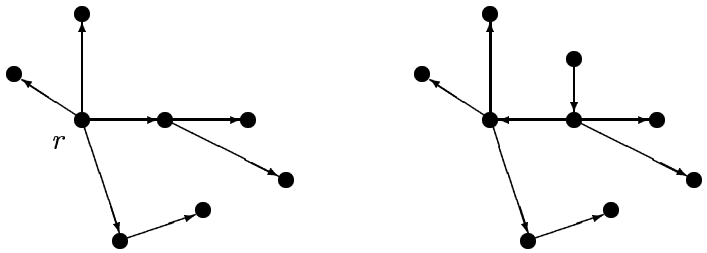
(4) T has $n-1$ edges.

If a subgraph T of a graph G has any three of the above properties, then T is spanning tree.

Remark. Properties *(2)* and *(4)* guarantee the spanning tree, it is not necessary a third property.

Rooted trees.

A rooted tree is a directed tree in which there is a special vertex r, named *root* and for each vertex v there is an r–v directed path.



a rooted tree a digraph which is not a rooted tree (but can be)

Binary trees.

Binary trees are defined recursively:

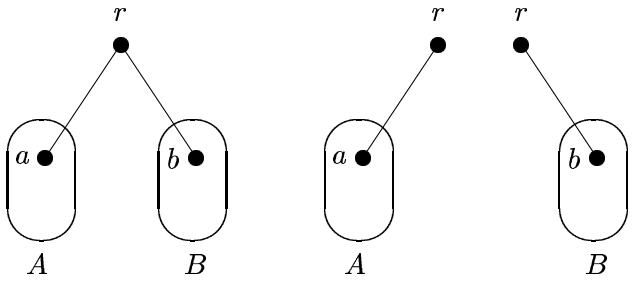
1. A vertex is a binary tree and is considered as a root.
2. If A and B are binary trees with roots a and b then the following are binary trees in which A is left subtree and B a right subtree:

– a tree with a root r, which is joined by an edge with a and by one with b,

– a tree with a root r, which is joined by an edge with a,

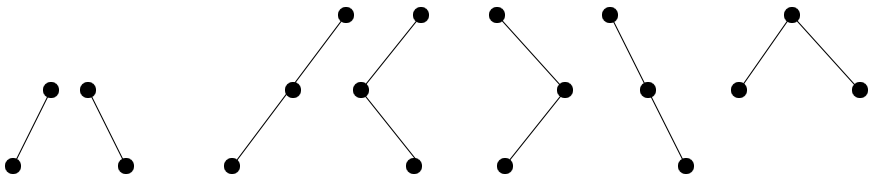
– a tree with a root r, which is joined by an edge with b.

Graphically:



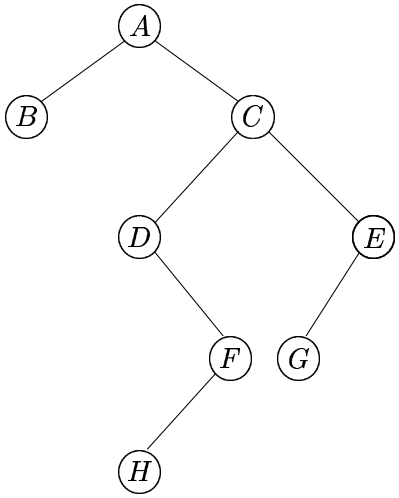
Binary trees are not directed graphs, but we can consider that their are, because they have a root and edges can be considered as arcs oriented from root towards leaves. We always draw binary trees with the root in top and the other vertices below.

We enumerate all binary trees with 2 and 3 vertices.



n=2 n=3

There are three methods of visiting the vertices of a binary tree:
– preorder traversal
– inorder tarversal
– postorder traversal



preorder: A, B, C, D, F, H, E, G

inorder: B, A, D, H, F, C, G, E

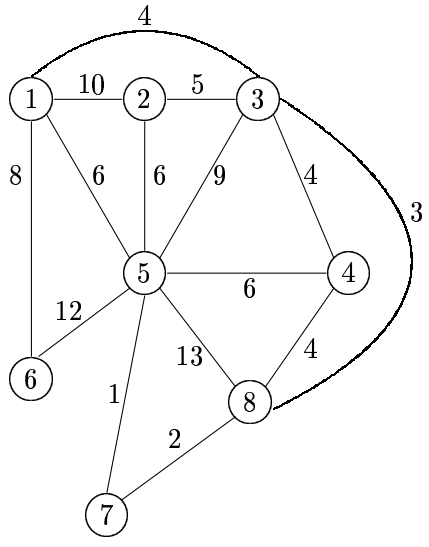
postorder: B, H, F, D, G, E, C, A

```
procedure preorder(a:bin);
Begin
 if a<>nil then
 begin
     write (a^.info:3);
     preorder (a^.left);
     preorder (a^.right);
 end;
End;

procedure inorder(a:bin);
Begin
```

```
 if a<>nil then
 begin
     inorder (a^.left);
     write (a^.info:3);
     inorder (a^.right);
 end;
End;

procedure postorder(a:bin);
Begin
 if a<>nil then
 begin
     postorder (a^.left);
     postorder (a^.right);
     write (a^.info:3);
 end;
End;
```

Minimal spanning tree

Let as consider a weighted graph. The value of a spanning tree is the sum of the weight of edges in spanning tree. We are interested in the spanning tree with minimal value, which is called the *minimal spanning tree*. There are two well-known algorithm to find a minimal spanning tree.

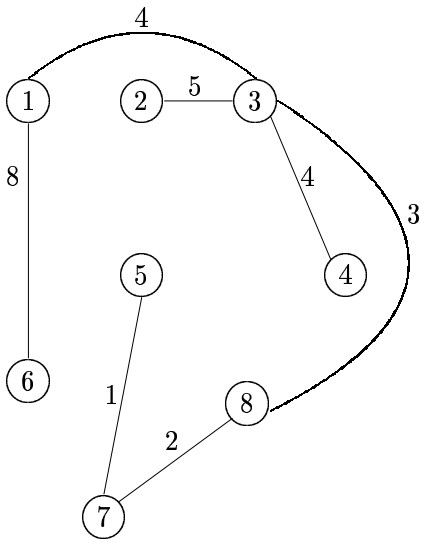


Kruskal algorithm

Let us arrange the edges in ascending order by the weights. The first edge will be included in the future spanning tree (the included edge is marked below by a star). At the beginning each edge is in a separate set. We put an edge in the spanning tree only if its incident vertices are in different sets, and these sets will be unified. The algorithm finds the minimal spanning tree when all vertices belong to the same set.

```
                          {1}, {2}, {3}, {4}, {5}, {6}, {7}, {8}

{5,7}      1    *     {5,7}, {1}, {2}, {3}, {4}, {6}, [8}
{7,8}      2    *     {5,7,8}, {1}, {2}, {3}, {4}, {6}
{3,8}      3    *     {3,5,7,8}, {1}, {2}, {4}, {6}
{1,3}      4    *     {1,3,5,7,8}, {2}, {4}, {6}
{3,4}      4    *     {1,3,4,5,7,8}, {2}, {6}
{4,8}      4
{2,3}      5    *     {1,2,3,4,5,7,8}, {6}
{1,5}      6
{2,5}      6
{4,5}      6
{1,6}      8    *     {1,2,3,4,5,6,7,8}
{3,5}      9
{1,2}     10
{5,6}     12
{5,8}     13
```

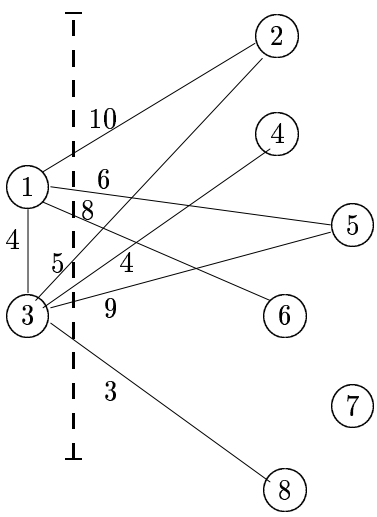
The edges marked by star are on the minimal spanning tree. The minimal spanning tree is the following:

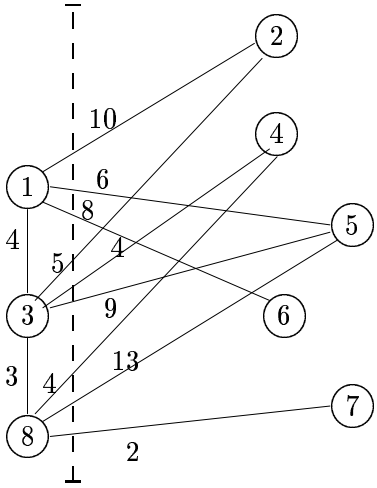


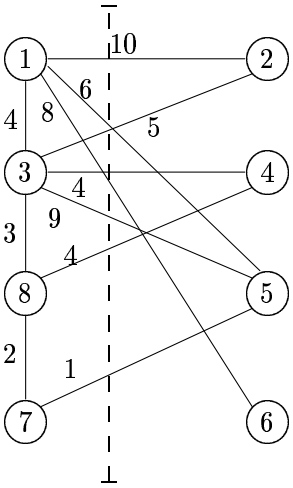
Prim algorithm

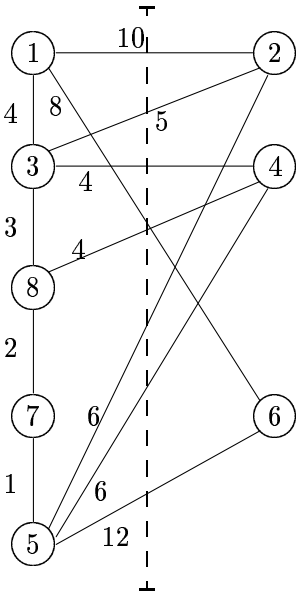
The basic idea of this algorithm is that among all incident edges to a given vertex x the one with minimal value must be in the minimum spanning tree. If not, by putting this edge in the minimum spanning tree, a cycle will arise, and after deleting any other edge incident to vertex x from this cycle, a new spanning tree with less value will appear, which is a contradiction.

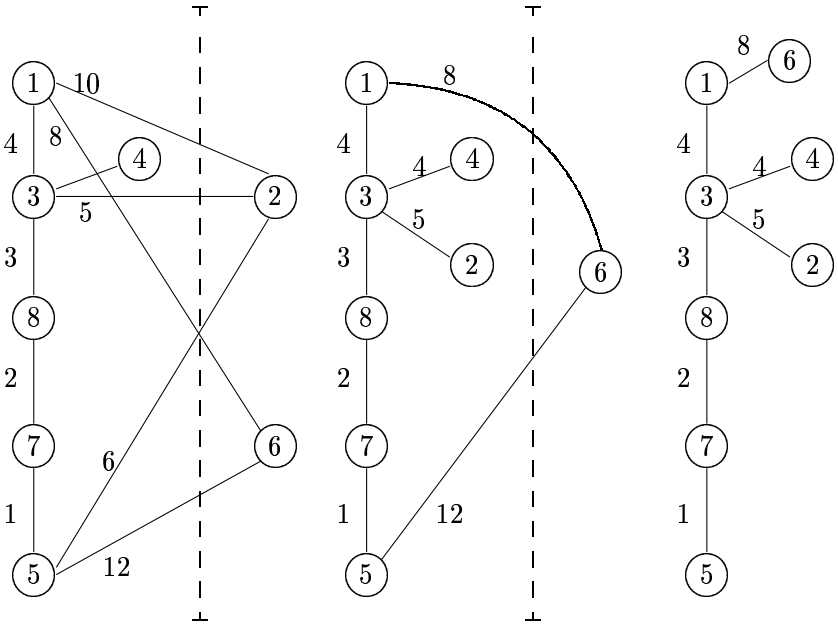
Let's begin with any vertices. This vertex will be put in a set A, rest of the vertices remain in a set B. (Always $A \cup B = V$.) Let us consider all edges between these two sets. Select the edge from these with the minimal weight. Put the second incident vertex to this in the set A. Continue until all vertices get in A. The selected edges are the edges of a minimum spanning tree.


1
2
3
4
5
6
7
8
10
6
8
4
5
4
9
3


1
2
3
4
5
6
7
8
10
6
8
4
5
4
9
13
3
4
2

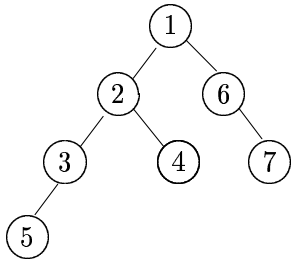

1
2
3
4
5
6
7
8
10
6
4
8
5
4
9
3
4
2
1


1
2
3
4
5
6
7
8
10
4
8
5
4
3
4
2
6
1
6
12



Prüfer code

The Prüfer code of a rooted tree is sequence of natural numbers. Let us label the vertices of a tree by consecutive naturals in any order. From leaves select the one with the minimal label. Put in the sequence the parent of this vertex. Delete from tree the selected leaf and continue until no vertex remains. The obtained naturals represent the Prüfer code of the tree.



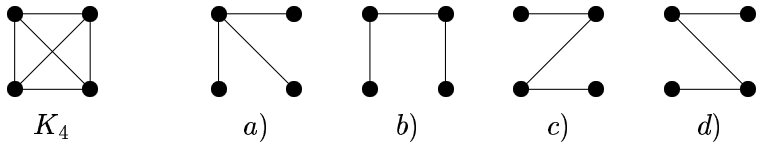
2,3,2,1,6,1

For a tree with n vertices, this code will have $n-1$ naturals. In the last position in this code always will stay the label of the root. The first $n-1$ positions in this sequence represent an $(n-2)$-permutations with repetition of n elements. There are n^{n-2} such different permutations. So, this is the number of all different rooted trees with n vertices. (Between these can be isomorphic trees but with different labels.)

Theorem 15 (Cayley) *If we label with consecutive naturals the vertices*

of a complete graph of order n, then this will have n^{n-2} different spanning trees.

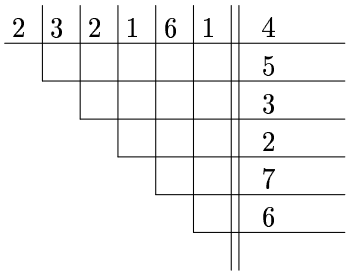
Example. For K_4 we will have $4^2 = 16$ different spanning trees.

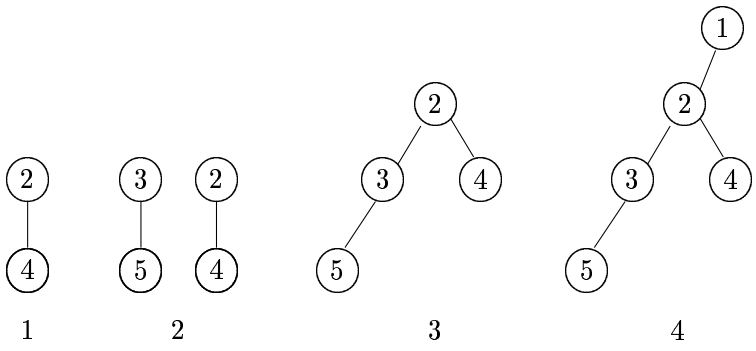


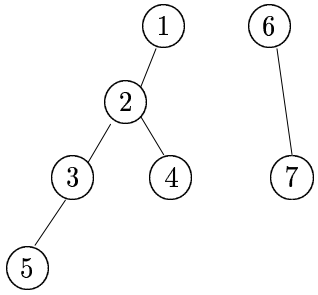
There are four spanning trees of type a), four of type b), four of type c) and four of type d).

The decoding algorithm:

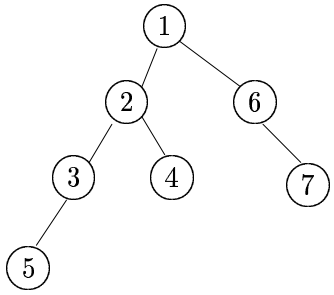
Let a be the first natural in the the Prüfer code. Consider the least natural b which is not in the Prüfer code (sequence). Put in the tree an edge from a to b. Delete the first number from the sequence and put in the number b. Continue until all numbers in the original sequence will be deleted.







5



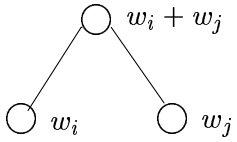
6

Huffman algorithm

Let us consider a rooted tree with the leaves $v_1, v_2, \ldots, v_k$ labeled by the following weights: $w_1, w_2, \ldots, w_k$ (in this order). If the length of the path from the root to the leaf v_j is l_j, then we are interesting in the value $\sum_{j=1}^{k} w_j l_j$, the *weighted pathlength.*

The problem is: *Find a rooted binary tree with the minimal weighted pathlength for a sequence of numbers (which label the leaves).*

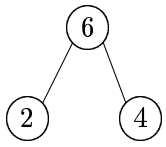
Select the two least numbers in the sequence, let these be w_i and w_j, delete them from the sequence, add to the sequence the number $w_i + w_j$, put in tree the following subtree



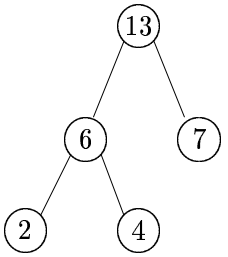
Continue until the sequence will reduce to a single number. The corresponding tree will be with the minimal weighted pathlenght.

Example.

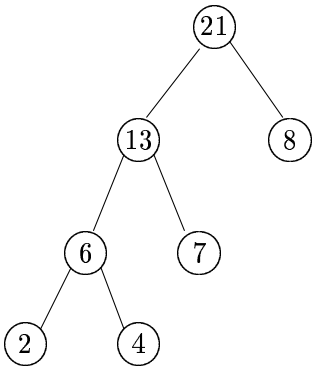
The sequence is: $2, 4, 7, 8, 13, 19, 20$



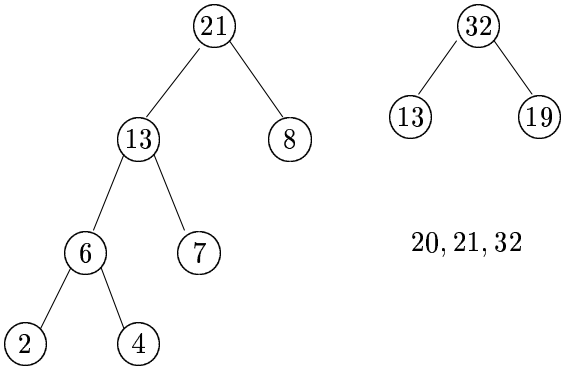
$6, 7, 8, 13, 19, 20$



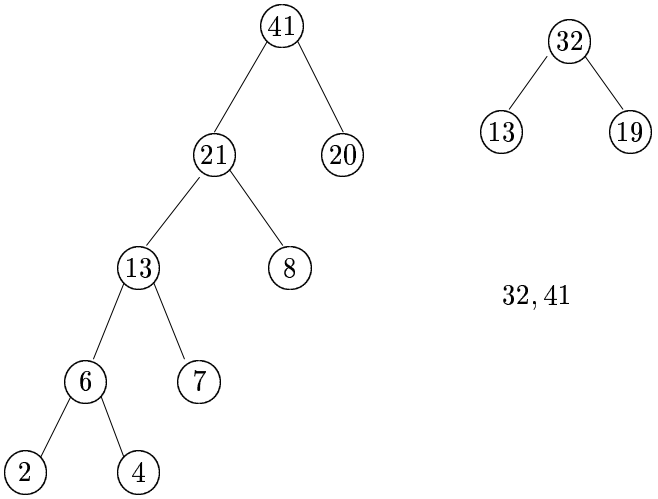
$8, 13, 13, 19, 20$



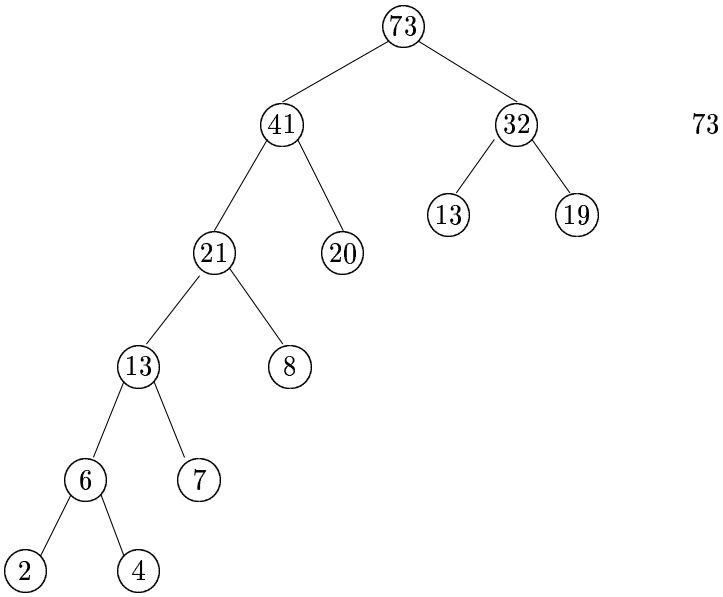
$13, 19, 20, 21$



$20, 21, 32$



$32, 41$

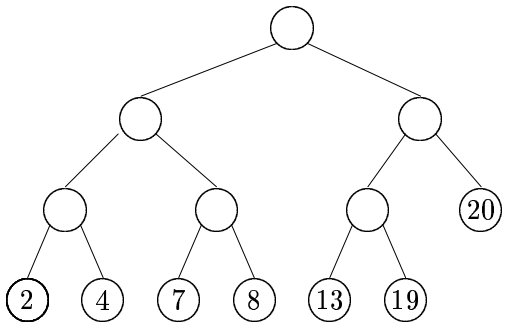


73

The weighted pathlength for this tree is:

$$2 \cdot 5 + 4 \cdot 5 + 7 \cdot 4 + 8 \cdot 3 + 20 \cdot 2 + 13 \cdot 2 + 19 \cdot 2 = 186,$$

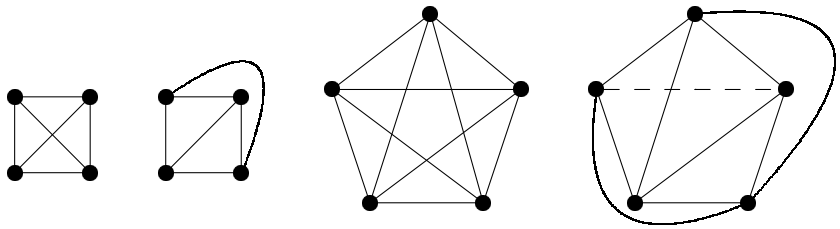
which is minimal. Any other binary tree has a larger weighted pathlength. For the following tree



the weighted pathlength is: $(2 + 4 + 7 + 8 + 13 + 19) \cdot 3 + 20 \cdot 2 = 199$.

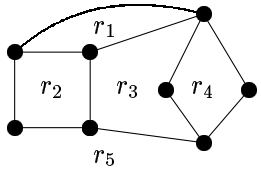
8 Planar graphs

A graph is *planar* if it can be drawn in the plane without any intersection of its edges.



K_4 K_5

For example in the figures above the K_4 can be drawn without intersection of its edges, but K_5 cannot (At this moment we cannot say that K_5 is nonplanar, this will be proved later.)



$n = 8$
$m = 11$
$r = 5$

If we draw a planar graph such that any two edges of it do not intersect than this is called a *plane graph*. The vertices and the edges of a plane graph determines some regions (see regions r_1, r_2, r_3, r_4 and r_5 in the above graph). From these only one is unbounded, the other are bounded regions.

Theorem 16 *In a plane graph the boundaries of the bounded regions form a basic cycle system.*

Proof. The proof is made by induction on the number of regions. Any new region contains at least a new edge, so if the number of regions increases by one, the number of basic cycles will increase by one too. □

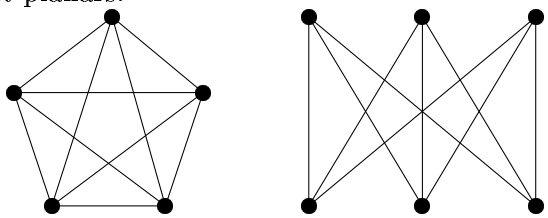
So the number of regions is equal to the cyclomatic number plus 1 (corresponding to the unbounded region): $r = \nu(G) + 1$. From this the following theorem results (because $\nu(G) = m - n + 1$).

Theorem 17 (Euler) *If a connected plane graph has n vertices, m edges and r regions, then*

$$n - m + r = 2.$$

Another proof: Let us consider a plane graph and the number $n-m+r$. If we eliminate an edge from a cycle, the number of edges will decrease by 1, and the number of regions too. So the number $n-m+r$ is the same for this new graph, and will remain constant by this operation of eliminating an edge from a cycle. If we eliminate all edges from cycles, the resulted graph will be a tree for which the number $n-m+r$ is the same constant and is equal to 2 (because $r=1$ and $m=n-1$). So the original number $n-m+r$ is equal to 2.

This formula of Euler can be used to prove that the graphs K_5 and $K_{3,3}$ are not planars.



K_5 $\qquad$ $K_{3,3}$

Theorem 18 *K_5 is nonplanar.*

Proof. If K_5 is planar, we can use the Euler's formula. We have: $n=5$, $m=10$ and each region contains at least 3 edges on boundaries. So

$$3r \leq 2m \quad \Rightarrow \quad r \leq \frac{2m}{3} = \frac{20}{3}.$$

So, because r is an integer, we must have $r \leq 6$. But from the Euler's formula $r = 2-n+m = 7$, which is a contradiction. □

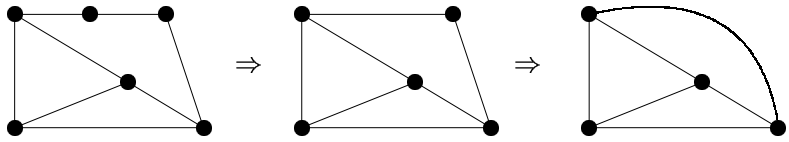
Theorem 19 *$K_{3,3}$ is nonplanar.*

Proof. If $K_{3,3}$ is planar, we can use the Euler's formula. We have: $n=6$, $m=9$ and each region contains at least 4 edges on boundaries. So

$$4r \leq 2m \quad \Rightarrow \quad r \leq \frac{m}{2} = \frac{9}{2}.$$

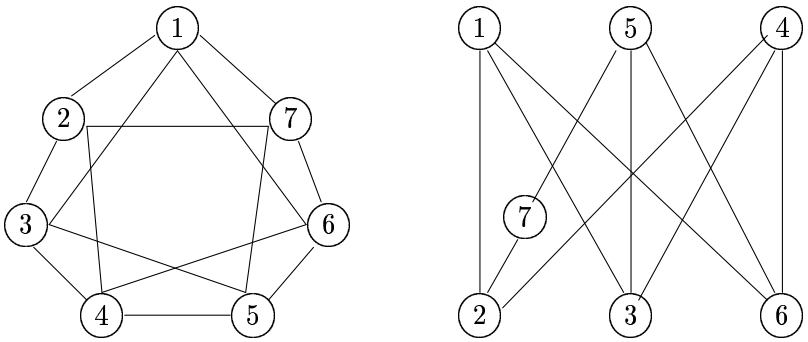
So, because r is an integer, we must have $r \leq 4$. But from the Euler's formula $r = 2-n+m = 5$, which is a contradiction. □

We define the *contraction* on graphs as being the operation by what we eliminate a vertex of degree 2 and join the corresponding two vertices by an edge.

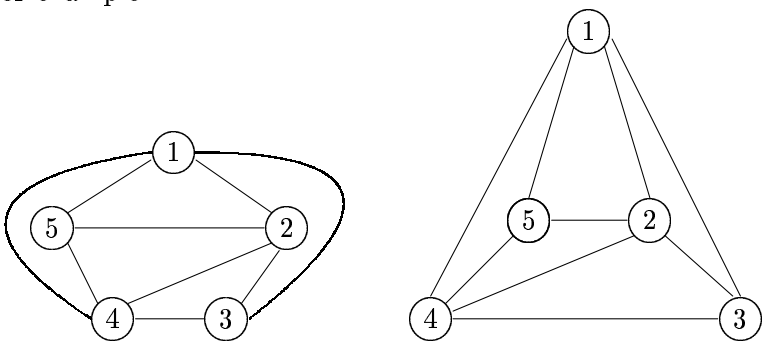


Theorem 20 (Kuratowski) *A connected graph G is planar if and only if G does not contain any subgraph which can be contracted to a K_5 or $K_{3,3}$.*

In the below graph after eliminating the edges $\{1,7\}, \{6,7\}$ and $\{2,3\}, \{4,5\}$ a graph results, which easily can be contracted to $K_{3,3}$, so the original graph is nonplanar.



Each plane graph can be drawn such that the edges are straightline segments. For example:



Theorem 21 *In a connected plane graph with $n \geq 3$ the following are true:*
a) $r \leq 2(n-2)$
b) $m \leq 3(n-2)$.

Proof. *a*) Each region has a boundary of at least 3 edges, so $3r \leq 2m$. Using the Euler's formula

$$3r \leq 2m = 2(n + r - 2) \quad \text{from this} \quad r \leq 2n - 4$$

b) Using the result in *a*) we will have $m = n + r - 2 \leq 3n - 6$. □

Theorem 22 *In a planar graph always there exists at least a vertex with degree at most 5.*

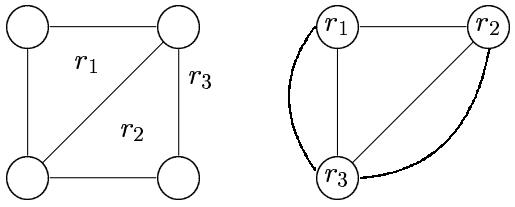
Proof. If all vertices are of degree at least 6, then

$$2m = \sum_{v \in V(G)} \deg(v) \geq 6n$$

From this $m \geq 3n$ results in contradiction with the result of the previous theorem, that says $m \leq 3n - 6$. □

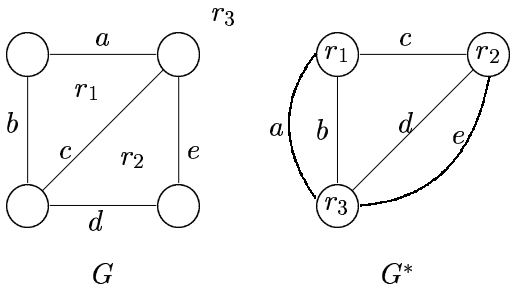
Dual graphs

For a plane graph (or maybe plane multigraph) G let us define the *dual graph* G^*. To each region in G will correspond a vertex in G^*. Two vertices in G^* are joined by p parallel edges if the corresponding regions in G have p edges in common.



G G^*

A set of edges in a graph is an *(edge) cutset* if by deleting these edges the graph becomes disconnected. Let us draw the above graphs by labelling the edges with letter $a, b, c, \ldots$ in G. If two regions has in common an edge, the corresponding edge in G^* will be labelled by the same letter.



G G^*

Let us notice that the set of edges of a cycle in G is an edge cutset in G^* and conversely a set of edges of an edge cutset in G^* correspond to the edges of a cycle in G.

By this remark we can define the duality more general.

Two graphs G and G^* are duals each of other if we can make a 1-1 correspondence between edges of them such that to any set of edges of a cycle in G corresponds a set of edges which is an edge cutset in G^* and conversely, to a set of edges of an edge cutset in G^* corresponds a set of edges of a cycle in G.

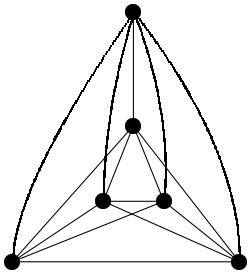
Theorem 23 *A connected graph G is planar if and only if G has a dual graph.*

Crossing number

The crossing number of a graph is the minimal number of crossing edges among all drawings of the graph in plane. This number will be denoetd by $c(G)$. For all planar graphs the crossing number is 0. As we have been seen before $c(K_5) = 1$ and $c(K_{3,3}) = 1$.

Theorem 24 $c(K_6) = 3$.

Proof. Instead of K_6 let us consider a new graph: put in every crossing point a new vertex. Then the resulting graph will have $n' = 6 + c$ vertices and $m' = 15+2c$ edges (because K_6 has 6 vertices and 15 edges). In this new graph, which is a planar one, we have $m' \leq 3n' - 6$, so $15 + 2c \leq 18 + 3c - 6$, and $c \geq 3$ results. But we can draw K_6 with exactle 3 crossings, so $c(K_6) =$



3. □

Easily can be proved by the same method, that:

$$c(K_n) \geq \frac{(n-3)(n-4)}{2}$$

For upper bound we have:

$$c(K_n) \leq \frac{1}{4} \left\lfloor \frac{n}{2} \right\rfloor \left\lfloor \frac{n-1}{2} \right\rfloor \left\lfloor \frac{n-2}{2} \right\rfloor \left\lfloor \frac{n-3}{2} \right\rfloor$$

For $n \leq 10$ the equality holds, and it is conjectured for all n.

For a complete bipartite graph $K_{m,n}$ we have:

$$c(K_{m,n}) \leq \left\lfloor \frac{m}{2} \right\rfloor \left\lfloor \frac{m-1}{2} \right\rfloor \left\lfloor \frac{n}{2} \right\rfloor \left\lfloor \frac{n-1}{2} \right\rfloor$$

For $1 \leq \min(m, n) \leq 10$ the equality holds.

9 Flows in networks

Let us consider some notations which will be useful in the following. Let X be an arbitrary set, $\mathbf{Z}$ the set of integers. If g is a function $g : X \times X \to \mathbf{Z}$ we shall use the following extension on sets.

$$\text{If } A, B \subseteq X, \text{ then } g(A,B) = \sum_{\substack{x \in A \\ y \in B}} g(x,y).$$

Properties of this function:

1) $g(A, B \cup C) = g(A,B) + g(A,C) - g(A, B \cap C)$
2) $g(A \cup B, C) = g(A,C) + g(B,C) - g(A \cap B, C)$
3) If $B \cap C = \emptyset$ then
$g(A, B \cup C) = g(A,B) + g(A,C)$
$g(B \cup C, A) = g(B,A) + g(C,A)$
4) If $f : X \times X \to \mathbf{Z}$, $g : X \times X \to \mathbf{Z}$, $h : X \times X \to \mathbf{Z}$ and $f = g + h$ then
$f(A,B) = g(A,B) + h(A,B)$ for $A, B \subseteq X$.

If $A = \{a\}$ then instead of $f(\{a\}, B)$ we shall write $f(a,B)$. In the following we shall use capital letters to denote sets and small letters to denote elements of sets.

A *network* is a connected directed graph (V,E) with the properties:
a) $\exists s \in V : N^{\text{in}}(s) = \emptyset$, s is called *source*,
b) $\exists t \in V : N^{\text{out}}(t) = \emptyset$, t is called *target* or *sink*,
c) A *capacity function* is defined: $\alpha : V \times V \to \mathbf{Z}_+$ such that
$\alpha(x,y) > 0$ if $(x,y) \in E$,
$\alpha(x,y) = 0$ if $(x,y) \notin E$.
Such a network will be denoted by (V, E, α, s, t).

A *flow* in network is defined as a function $f : V \times V \to \mathbf{Z}_+$ with the properties:
1° $f(x,y) \leq \alpha(x,y)$ (*capacity constraint*),
2° $f(x,V) = f(V,x)$ for all $x \in V \setminus \{s,t\}$ (*conservation equation*).

The quantity $v(f) = f(s,V)$ is the *value of the flow* f.

Theorem 25 *$v(f) = f(s,V) = f(V,t)$ for any flow.*

Proof.

$$\sum_{x\in V}\big(f(V,x)-f(x,V)\big)=f(V,V)-f(V,V)=0,$$

otherwise

$$\sum_{x\in V}\big(f(V,x)-f(x,V)\big)=\underbrace{f(V,s)}_{=0}-f(s,V)+$$

$$+\sum_{\substack{x\in V\\ x\neq s,t}}\underbrace{\big(f(V,x)-f(x,V)\big)}_{=0}+$$

$$+f(V,t)-\underbrace{f(t,V)}_{=0}=f(V,t)-f(s,V).$$

□

By this theorem we can reformulate the restrictions on flow:

1° $f(x,y)\le\alpha(x,y)$,

2° $f(x,V)-f(V,x)=\begin{cases} v(f), & \text{if } x=s\\ 0, & \text{if } x\neq s, x\neq t\\ -v(f), & \text{if } x=t\end{cases}$

Problem.

Find a maximal value flow in a network. A maximal value flow is called a **maximal flow** or **maximum flow**.

Examples.

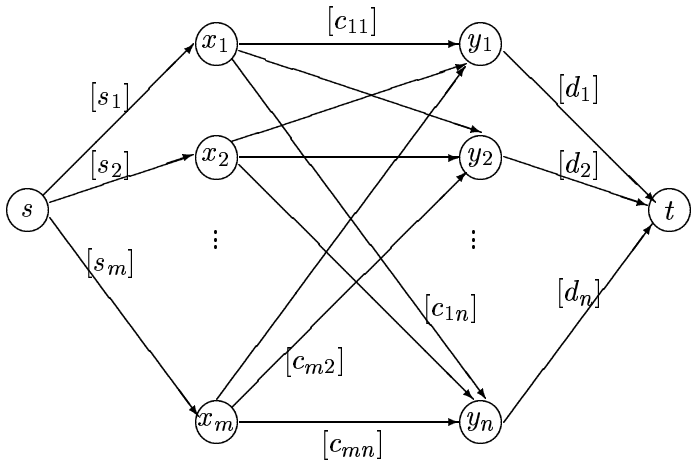
1) *Maritime transportation*

There are m harbours $x_1,x_2,\dots x_m$ with some goods in the amount: $s_1,\ s_2,\ \dots s_m$ respectively, and n harbours $y_1,y_2,\dots,y_n$ with the demands $d_1,d_2,\dots,d_n$ respectively. The problem is how can be transported the maximal amount of goods by ships from source harbours x to destination harbours y if we know that the maximal amount of goods that can be transported from each x_i to each y_j in a fixed time interval is c_{ij} (capacity of ships).

Let us consider a digraph with vertices $x_1,x_2,\dots x_m$ and $y_1,y_2,\dots,y_n$ with arcs from each x_i to each y_j. Add two new vertices to this graph s and t with arcs from s to each x_i and from each y_j to t. Complete this graph

with capacities: s_i on the arcs (s, x_i), d_i on the arcs (y_i, t), and c_{ij} on each arc (x_i, y_j).

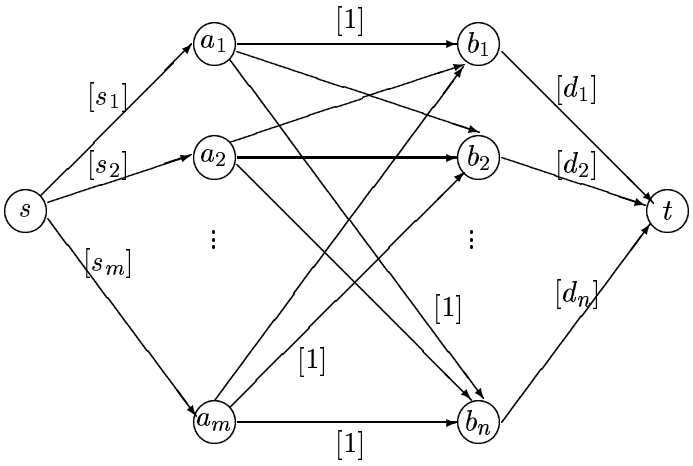
The problem is to find a maximal flow in this network.



2) *Family trip*

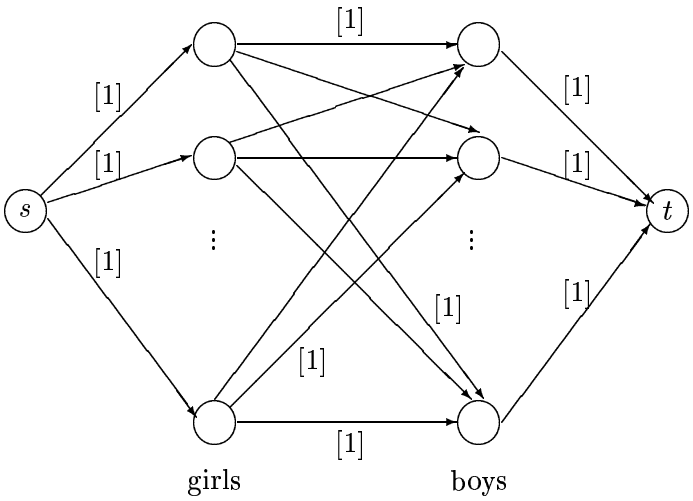
Families $a_1, a_2, \ldots, a_m$ of members $s_1, s_2, \ldots, s_m$ respectively want to go on a trip by buses. There are n buses $b_1, b_2, \ldots, b_n$ of places $d_1, d_2, \ldots, d_n$ respectively. The problem is the following: can be organized such a trip in which every two members of a family are in different buses?

The attached network is similar to the previous one. The vertices are $a_1, a_2, \ldots, a_m$ and $b_1, b_2, \ldots, b_n$, with arcs from each a_i to each b_j with the capacity 1 each of them. Add two new vertices s and t with arc from s to each a_i with the capacity s_i arc from b_j to t with the capacity d_j. The problem is to find a maximal flow in this network. The problem has a solution only if this flow saturate all arc from s. An arc is *saturated* (or is a *saturation arc*) if the flow on it is identical to the capacity.



3) *College dance*

Can we organize a dance in a college such that every girl has as partner an acquainted boy. The acquaintance is denoted by an arc? In the attached network each arc has capacity 1.



The problem has a solution if the maximal flow in the attached network saturate all arcs from s.

The cut

If in a network we have $A \subseteq V$, $\overline{A} = V \setminus A$ and $s \in A$ and $t \in \overline{A}$ then

$(A, \overline{A})$ is a *cut*. The *capacity* of the cut $(A, \overline{A})$ is defined as:

$$\alpha(A, \overline{A}) = \sum_{\substack{x \in A \\ y \in \overline{A}}} \alpha(x, y).$$

A cut in a network with the minimal capacity is a *mininal cut*.

Theorem 26 *If $(A, \overline{A})$ is a cut in a network, then for every flow f we have*

$$v(f) = f(A, \overline{A}) - f(\overline{A}, A) \leq \alpha(A, \overline{A}).$$

Proof. The following are true:
$f(s, V) - f(V, s) = v(f)$,
$f(x, V) - f(V, x) = 0$ for $x \neq s, t$.
After summing up this equations for all $x \in A$ we have:

$$v(f) = \sum_{x \in A} \big(f(x, V) - f(V, x)\big) = f(A, V) - f(V, A)$$

but $V = A \cup \overline{A}$, and $A \cap \overline{A} = \emptyset$, and we have (after using the equations from the beginning of this section):

$$v(f) = f(A, V) - f(V, A) = f(A, A \cup \overline{A}) - f(A \cup \overline{A}, A)$$

$$= f(A, A) + f(A, \overline{A}) - f(A, A) - f(\overline{A}, A) = f(A, \overline{A}) - f(\overline{A}, A) \leq \alpha(A, \overline{A}),$$

because $f(\overline{A}, A) \geq 0$ always. □

Theorem 27 (Ford–Fulkerson) *In a network a maximal flow has a value equal to the capacity of a minimal cut.*

Proof. By the previous theorem it is enough to prove that there is a maximal flow and a cut $(A, \overline{A})$ for which:
$f(A, \overline{A}) = \alpha(A, \overline{A})$ and
$f(\overline{A}, A) = 0$.
Let us define recursively the set A.

1) First, put s in A.

2) If there is an arc (x, y) such that $x \in A$ and $y \notin A$ and $f(x, y) < \alpha(x, y)$, then put y in A.

3) If there is an arc (y, x) such that $x \in A$ and $y \notin A$ and $f(y, x) > 0$, then put y in A.

We claim that by this algorithm $t \in \overline{A}$. If not, then there exists a sequence $x_1, x_2, \ldots, x_r$ of vertices such that $x_1 = s$, $x_r = t$ and for each $i = 1, 2, \ldots r-1$

$(x_i, x_{i+1}) \in E$ and $f(x_i, x_{i+1} < \alpha(x_i, x_{i+1})$, or
$(x_{i+1}, x_i) \in E$ and $f(x_{i+1}, x_i) > 0$.

Let us denote:

$$\varepsilon_1 = \min_{\forall i:\ (x_i, x_{i+1}) \in E} \big(\alpha(x_i, x_{i+1}) - f(x_i, x_{i+1})\big)$$

$$\varepsilon_2 = \min_{\forall i:\ ;\ (x_{i+1}, x_i) \in E} f(x_{i+1}, x_i).$$

And let be: $\varepsilon = \min\{\varepsilon_1, \varepsilon_2\}$. We can define a new flow f^*:

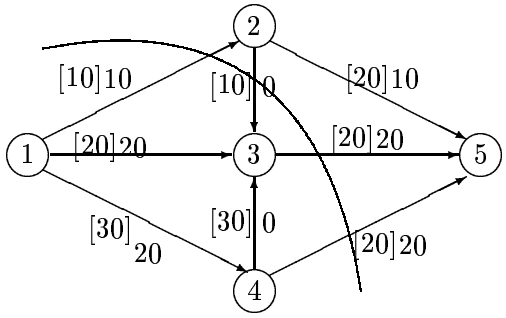
1) $f^*(x_i, x_{i+1}) = f(x_i, x_{i+1}) + \varepsilon$ if $(x_i, x_{i+1}) \in E$
2) $f^*(x_{i+1}, x_i) = f(x_{i+1}, x_i) - \varepsilon$ if $(x_{i+1}, x_i) \in E$
3) $f^*(x, y) = f(x, y)$ for arcs out of the sequence $x_1, x_2, \ldots, x_r$.

It is easy to verify that this function is a flow with the value $v(f^*) = v(f) + \varepsilon$, which is a contradiction, because we have been supposed that f is maximal.

By the definition of the set A we can conclude:

$\forall (x, y) \in (A, \overline{A})$: $\quad f(x, y) = \alpha(x, y)$
$\forall (y, x) \in (\overline{A}, A)$: $\quad f(y, x) = 0$, which proves the theorem.

An example:



In the above example the capacities are written in brackets, the value of flow on arcs without brackets. The value of the considered flow is 50, which is a maximal flow, because there exists a cut $A = \{1, 3, 4\}$, $\overline{A} = \{2, 5\}$ with the capacity 50 too. All arcs from A to $\overline{A}$: $(1, 2), (3, 5), (4, 5)$ are saturated, the single arc from $\overline{A}$ to A: $(2, 3)$ has 0 flow on it. □

This theorem is also called *the max-flow min-cut theorem.*

A max-flow min-cut algorithm (The Ford–Fulkerson algorithm)

Based on the above theorem an algorithm to find the maximal flow is considered. The algorithm labels step by step the vertices of the network. A label of the vertex y is $(x^+, \Delta y)$ or $(x^-, \Delta y)$, where x^+ means that on the arc (x, y) the flow can be increased by Δy and x^- means that on the arc (y, x) the flow can be decreased by Δy.

At first we consider that there is a null flow (0 on each arc).

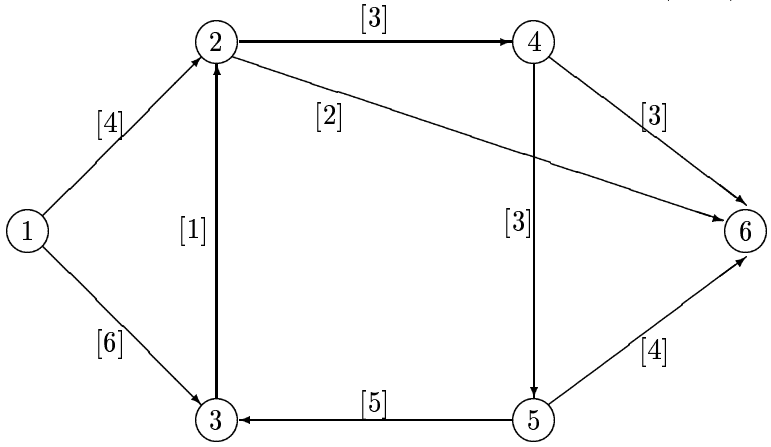
Let us begin the algorithm by labeling vertex s with $(-, \infty)$.

At each step if $(x, y) \in E$, x is labeled, y is not labeled and $f(x, y) < \alpha(x, y)$, then we label vertex y with $(x^+, \Delta y)$, where $\Delta y = \min\big(\Delta x, \alpha(x, y) - f(x, y)\big)$.

If $(y, x) \in E$, x is labeled, y is not labeled and $f(y, x) > 0$, then we label vertex y with $(x^-, \Delta y)$, where $\Delta y = \min\big(\Delta x, f(y, x)\big)$.

If we can label vertex t, then there exists a semipath from s to t on which the flow can be modified with the value Δt, the first element in the label determines which is the previous vertex and how the flow must be modified. We continue with the label of the previous vertex, and so on.

If we cannot label vertex t, then the flow is maximal, the labeled vertices are in the set A and the others in $\overline{A}$ of the minimal cut $(A, \overline{A})$.



The labeling process

The flow is equal to 0 on each arc.

vertex	1	2	4	6
label	$(-, \infty)$	$(1^+, 4)$	$(2^+, 3)$	$(4^+, 4)$

The flow is changed as follows: $f(4,6) = 3$, $f(2,4) = 3$, $f(1,2) = 3$. After deleting the labels, we will continue.

vertex	1	2	6
label	$(-, \infty)$	$(1^+, 1)$	$(2^+, 1)$

The flow is changed as follows: $f(2,6) = 1$, $f(1,2) = 4$. After deleting the labels, we will continue.

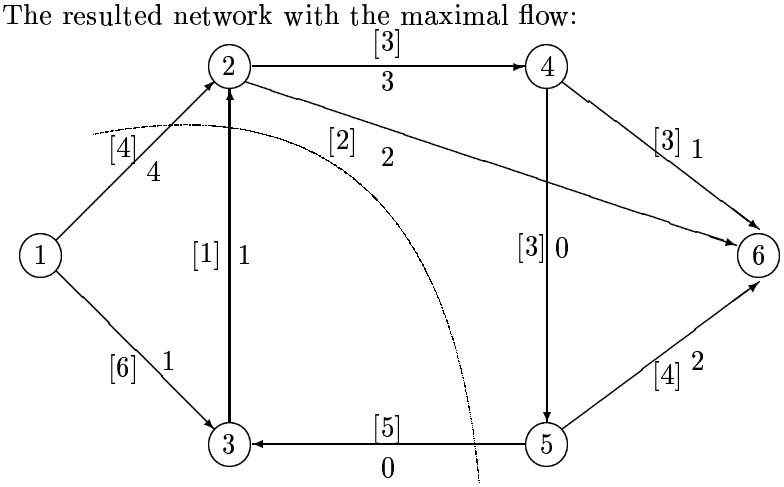
vertex	1	3	2	6
label	$(-, \infty)$	$(1^+, 6)$	$(3^+, 1)$	$(2^+, 1)$

The flow is changed as follows: $f(2,6) = 2$, $f(3,2) = 1$, $f(1,3) = 1$. Continue, after deleting the labels.

vertex	1	3
label	$(-, \infty)$	$(1^+, 5)$

At this moment vertex t cannot be labeled, so the flow is maximal. The minimal cut is: $A = \{1, 3\}$, $\overline{A} = \{2, 4, 5, 6\}$. The value of the flow is 5.

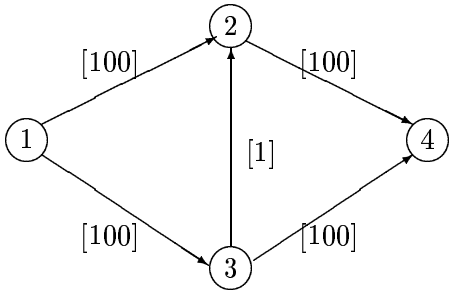
The resulted network with the maximal flow:



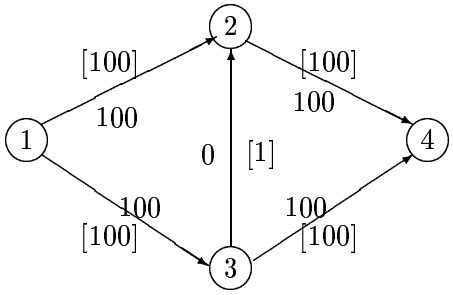
Analysis of the Ford-Fulkerson algorithm

The semipath on which the Ford-Fulkerson algorithm improves the flow is the *augmenting semipath.*

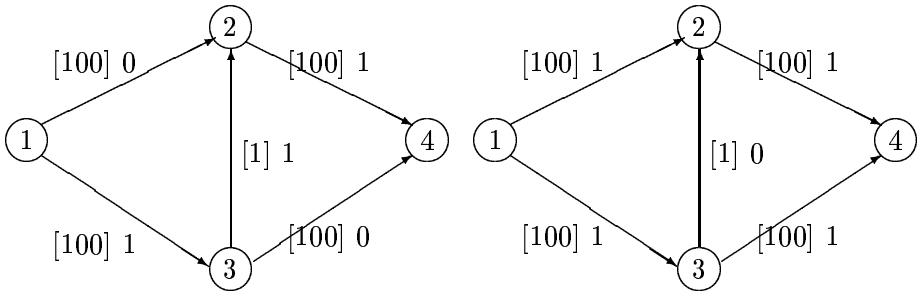
Let us consider the following example:

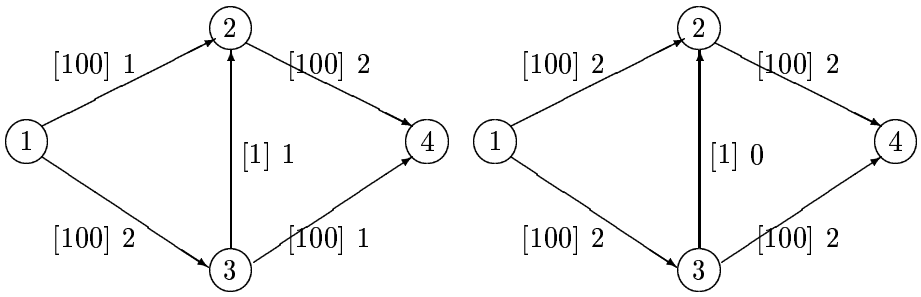


Using the Ford-Fulkerson algorithm, we can get the solution in two steps. Starting with the null flow (value 0 on each arc), on the path $1, 2, 4$ we can increase the flow by 100, and in the same way on the path $1, 3, 4$ two. So the value of the maximum flow is 200.



But equally the algorithm can use firstly the path $1, 3, 2, 4$ and increases the flow by 1. After this, using the semipath $1, 2, 3, 4$, the flow can be modified by 1 too.





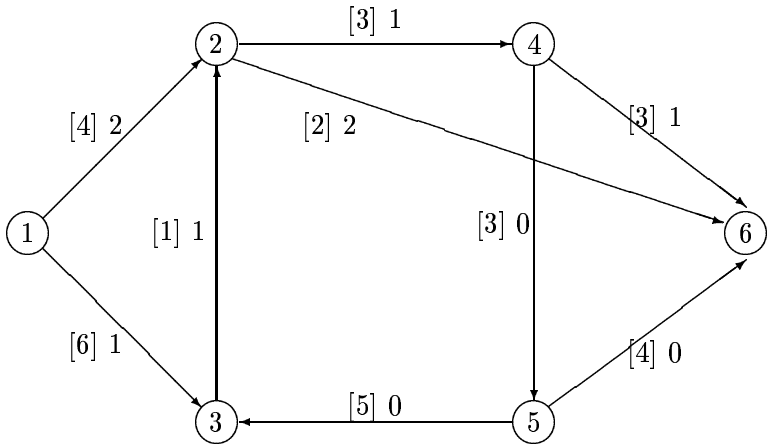
Continuing in this way, the result will be found after 200 steps. So the complexity of the algorithms depends on the value of the flow. If $m = |E(G)|$ is the number of arcs of the network, $v(f)$ is the value of the maximum flow, then the complexity of the algorithm will be $O(mv(f))$ or $O(n^2v(f))$, if n is the number of vertices of the network (pseudopolynomial algorithm).

This method can be improved, if we use an implementation in which always the shortest semipath is used (arcs are considered to have weight equal to 1). The augmenting semipath is computed by a breadth-first search algorithm, so it will be a shortest one.

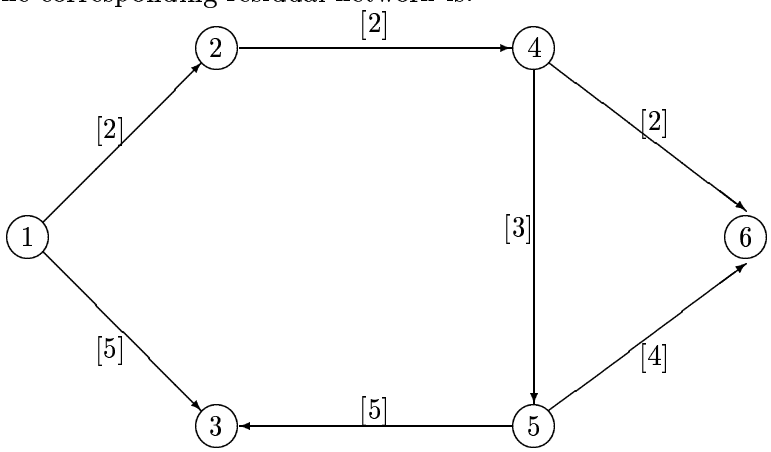
This new version of the Ford-Fulkerson algorithm will be called the *Edmonds-Karp algorithm.* The complexity of this algorithm is $O(nm^2)$ or $O(n^5)$.

The residual network. Let $N = (V, E, \alpha, s, t)$ be a network and f a flow in it. The corresponding *residual network* is $N_f = (V, E', \alpha', s, t)$ where $\alpha' = \alpha - f$ and E' is obtained from E by deleting the arcs with capacity equal to 0.

For the following network and flow



the corresponding residual network is:



If f is a flow in a network, and f' is a flow in the corresponding residual network to f, then $f + f'$ is a flow in the original network with the value $v(f) + v(f')$.

Residual networks can be used to prove the correctness of the Edmonds-Karp algorithm.

Flow in generalized networks

Generalized network $G = (V, E, \beta, \alpha, s, t)$, where V, E, s, t have same meanings as in the case of flow networks. Functions β and α, defined on

$V \times V$ and with nonnegative integer values, are the capacity functions (the lower and the upper capacity functions): $\beta \leq \alpha$ and

1) $\alpha(x,y) > 0$ if $(x,y) \in E$

2) $\alpha(x,y) = 0$ if $(x,y) \notin E$

A *generalized flow* (compatible flow) in a generalized network is a funtion $f : V \times V \to \mathbf{Z}_+$ with the properties:

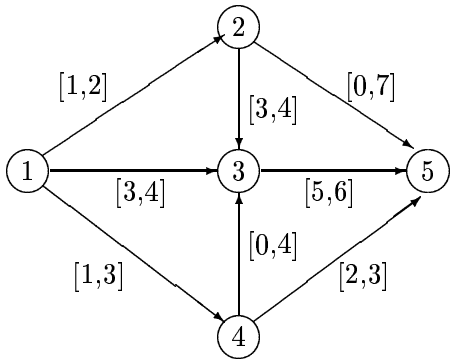
1) $\beta(x,y \leq f(x,y) \leq \alpha(x,y)$ for $x,y \in V$

2) $f(V,x) = f(x,V)$ for $x \in V \setminus \{s,t\}$.

The value of the generalized flow is $v(f) = f(s,V) = f(V,t)$.

We are interested in maximum generalized flow (or sometimes maybe in minimum generalized flow).

There is a problem: *When a generalized flow exists?*



In vertex 2 the maximal value of incoming flow can be 2, but the outgoing flows must be 3 (3+0), so a generalized flow cannot exist.

To answer the question *when a generalized flow exists*, we shall trace back the problem to a classical flow network. Let us attach to a generalized flow network $G = (V,E,\alpha,s,t)$ a flow network $G^* = (V^*,E^*,\alpha^*,a,z)$ in the following way:

$$V^* := V \cup \{a,z\}$$

$$E^* := E \cup \left\{(a,x) \mid N_G^{\text{in}}(x) \neq \emptyset\right\} \cup \left\{(x,z) \mid N_G^{\text{out}}(x) \neq \emptyset\right\} \cup \left\{(t,s)\right\}$$

$$\alpha^*(x,y) := \alpha(x,y) - \beta(x,y), \quad \forall x,y \in V$$

$$\alpha^*(a,x) := \beta(V,x)$$

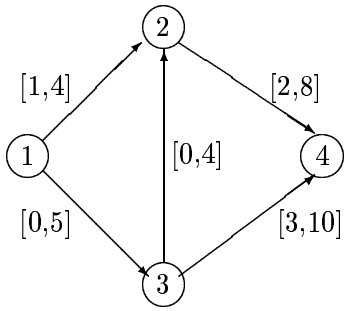
$$\alpha^*(x,z) := \beta(x,V)$$

$\alpha^*(t,s) := \infty$

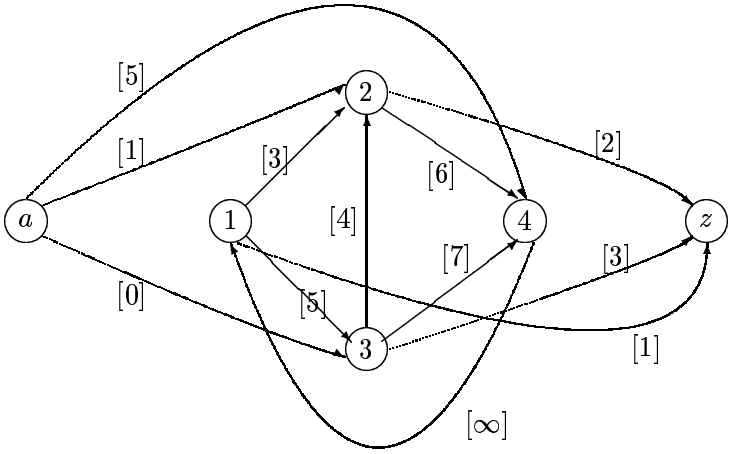
Remember that:

$$\beta(V,x) = \sum_{y \in N_G^{\text{in}}(x)} \beta(y,x), \qquad \beta(x,V) = \sum_{y \in N_G^{\text{out}}(x)} \beta(x,y)$$

An example of a generalized flow network.



The corresponding flow network:



The following theorem answer our previous question.

Theorem 28 *In the generalized flow network G there exists a generalized flow if and only if in the attached flow network G^* there exists a flow which saturates all arcs from a.*

Proof. I. Let f^* be a flow in G^* which saturates the arcs from the source vertex a. But because

$$\alpha^*(a, V^*) = \beta(V, V) = \alpha(V^*, z)$$

this flow will saturates the arcs to z too.

Let us define $f := f^* + \beta$ and let us prove that this is a generalized flow in the original network.

1. *Capacity constraint.*

From

$$0 \leq f^*((x, y) \leq \alpha^*(x, y) = \alpha(x, y) - \beta(x, y) \quad \text{for } x, y \in V$$

the following results

$$\beta(x,y) \leq f(x,y) \leq \alpha(x,y) \quad \text{for } x,y \in V$$

2. *Conservation constraint.*
For $x \in V \setminus \{s,t\}$:

$$f(V,x) - f(x,V) = f^*(V,x) + \beta(V,x) - f^*(x,V) - \beta(x,V)$$

But

$$\beta(V,x) = \alpha^*(a,x) = f^*(a,x) \quad \text{for } x \in V \setminus \{s,t\} \quad \text{(saturation)}$$

$$\beta(x,V) = \alpha^*(x,z) = f^*(x,z) \quad \text{for } x \in V \setminus \{s,t\} \quad \text{(saturation)}$$

and because of $V^* = V \cup \{a,z\}$ we have

$$f^*(V,x) + \beta(V,x) = f^*(V,x) + f^*(a,x) = f^*(V^*,x)$$

$$f^*(x,V) + \beta(x,V) = f^*(x,V) + f^*(x,z) = f^*(x,V^*)$$

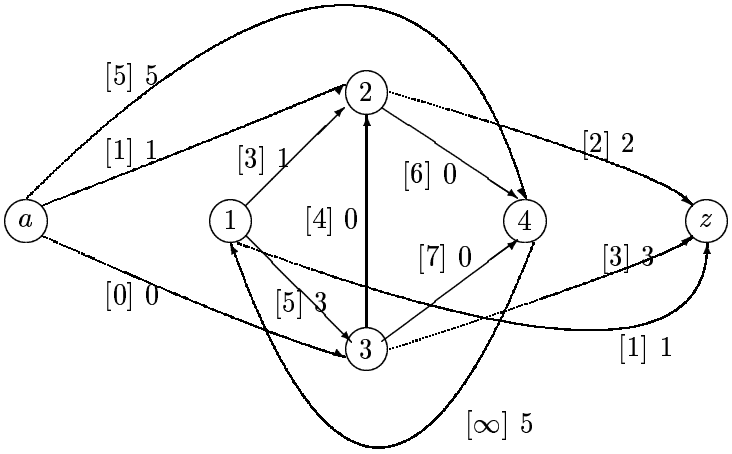
Because f^* is a flow in G^*, we shall obtain:

$$f(V,x) - f(x,V) = f^*(V^*,x) - f^*(x,V^*) = 0 \quad \text{for } x \in V \setminus \{s,t\}.$$

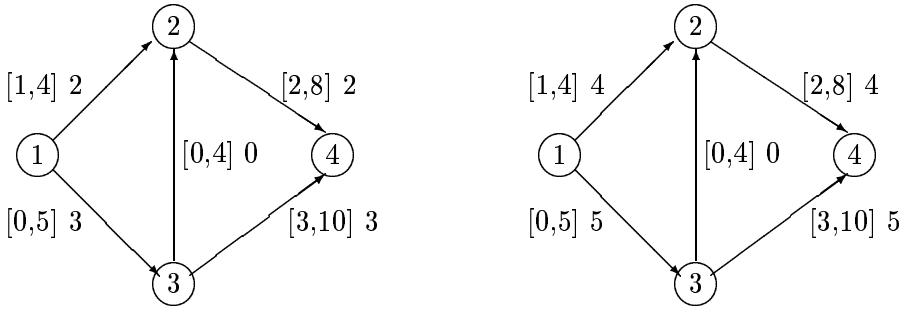
These two prove that f is generalized flow in the original network, so such a flow there exists.

II. Conversely, if f there exists, we can prove in the same way that f^* is a flow which saturates the arcs from a. □

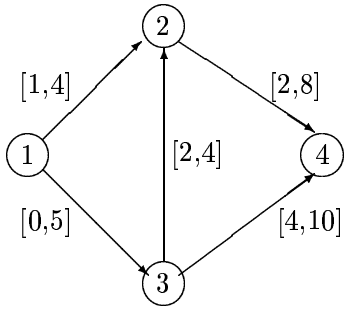
The maximum flow f^* in G^* is:



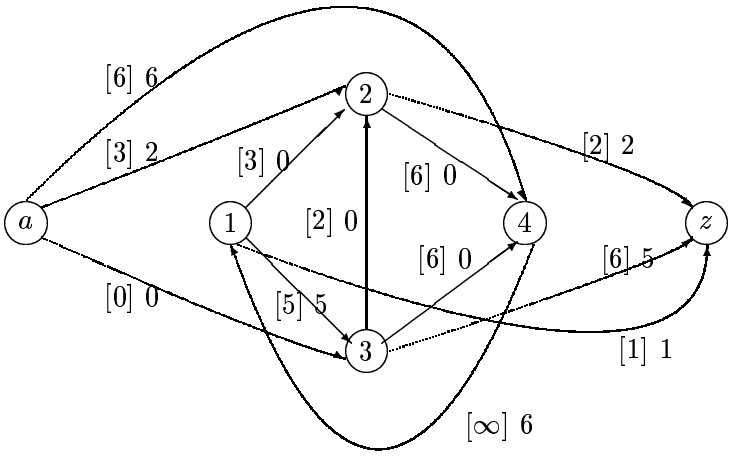
This flow saturates the arcs from a, so the function $f := f^* - \beta$ computed on the arcs of the network G will be a generalized flow (figure in the left). The maximum generalized flow, obtained from this, after applying the Ford-Fulkerson method, if we do not consider the lower capacity, will be the one in the right figure.



An example when the generalized flow does not exist.



The corresponding network and flow.



This flow ia a maximum one. The cutset is $A = \{a, 1, 2, 4\}$ and $\overline{A} = \{3, z\}$. Because this flow does not saturates the arcs to z (arc $(3, z)$ is not a saturating one), there is no generalized flow in the original network.

A minimum cost maximum flow

Let us consider a flow network $G = (V, E, \alpha, s, t)$ and a cost function $c : V \times V \to \mathbf{R}_+$ (where $\mathbf{R}_+$ is the set of nonnegative real numbers). If f is a flow in G then the *cost of the flow* f is defined as:

$$\mathcal{C}(f) := \sum_{x,y \in V} c(x,y) f(x,y)$$

Usually $c(x,y) = 0$ for $(x,y) \notin E$, but by our definition this is not important (because in this case the flow is equal to 0).

We are interesting in minimum cost maximum flow. How can we know if a given maximum flow is or not of minimum cost? We solve this problem by attaching to it a weighted digraph in which the absence of a negative length cycle will correspond to a minimum cost maximum flow.

For a network $G = (V, E, \alpha, s, t)$ a cost function c and a maximum flow f let us define the following weighted digraph $\overline{A}G$ with the same vertex set as in G:

• If on the arc (x,y) in G the flow $f(x,y) = 0$, then in $\overline{A}G$ put an arc (x,y) with the weight

$$\mathcal{W}(x,y) := c(x,y)$$

• If on the arc (x,y) in G the flow $f(x,y) = \alpha(x,y)$, then in $\overline{A}G$ put an arc (y,x) with the weight

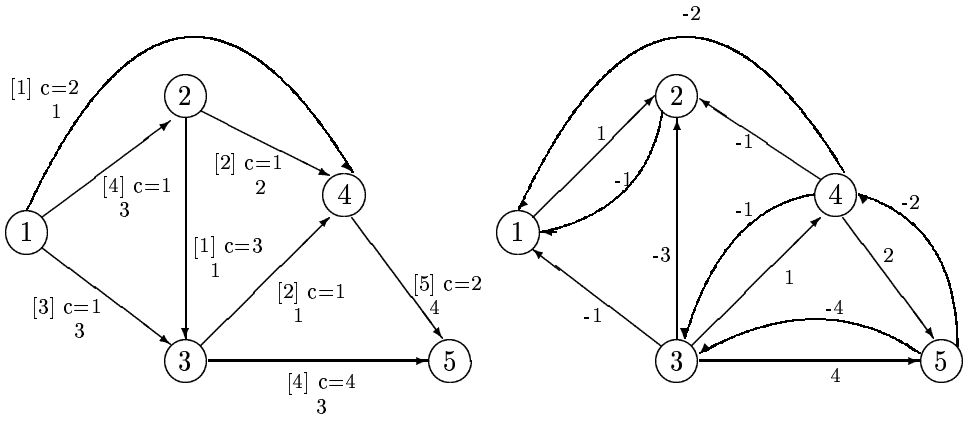
$$\mathcal{W}(y,x) := -c(x,y)$$

• If on the arc (x,y) in G the flow $0 < f(x,y) < \alpha(x,y)$, then in $\overline{A}G$ put the both arcs (x,y) and (y,x) with the weights:

$$\mathcal{W}(x,y) := c(x,y) \qquad \text{and}$$

$$\mathcal{W}(y,x) := -c(x,y).$$

An example:



The cost of this flow is $\mathcal{C}(f) = 3{\cdot}1{+}3{\cdot}1{+}2{\cdot}1{+}1{\cdot}3{+}1{\cdot}2{+}1{\cdot}1{+}3{\cdot}4{+}4{\cdot}2 = 34$.

Theorem 29 (Busacker–Saaty) *A maximum flow in the network G is of minimum cost if and only if the corresponding weighted graph $\overline{A}G$ has no negative length directed cycle.*

Proof. I. Let us prove that if f is a minimum cost flow in G, then in $\overline{A}G$ there exists no negative length cycles. This assertion is equivalent to the following: *if in $\overline{A}G$ there exists a negative length cycle, then the corresponding flow f in G in not of minimum cost.*

Let K be a negative length cycle in $\overline{A}G$, and let us define the following function (on arcs only):

$\overline{A}f(x,y) := f(x,y) + 1$ if (x,y) is on the cycle K and $\mathcal{W}(x,y) > 0$,
$\overline{A}f(x,y) := f(x,y) - 1$ if (y,x) is on the cycle K and $\mathcal{W}(y,x) < 0$,
$\overline{A}f(x,y) := f(x,y)$ otherwise

Can be proved that this function $\overline{A}f$ is a flow in G too. Let us use the following notation:

$$f_c(A) = \sum_{x,y)\in A} c(x,y)f(x,y) \qquad \text{where } A \subseteq E$$

Can be proved the following:
- if $A \cap B = \emptyset$ then $f_c(A \cup B) = f_c(A) + f_c(B)$

- if $\overline{A}f = f+1$ then $\overline{A}f_c(A) = f_c(A)+c(A)$ where $c(A) := \sum_{(x,y)\in A} c(x,y)$

For simplicity let us denote by K too the set of arcs of the cycle K. Then $E = (E \setminus K) \cup K$ and $(E \setminus K) \cap K = \emptyset$. Then

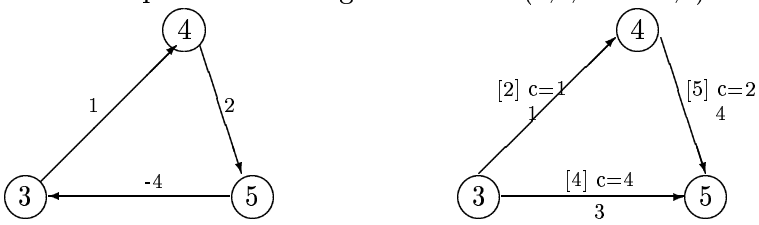
$$\overline{A}f_c(E) = \overline{A}f_c(E \setminus K) + \overline{A}f_c(K) = f_c(E \setminus K) + f_c(K) + c(K)$$

$= f_c(E) + c(K) < f_c(E)$ becuase $c(K) < 0$ is the length of the cycle K which is contradiction with the minimality of f.

II. Conversely, can be proved, in the same way, the following: if f is no a minimum flow, then in $\overline{A}G$ there exists a negative length cycle.

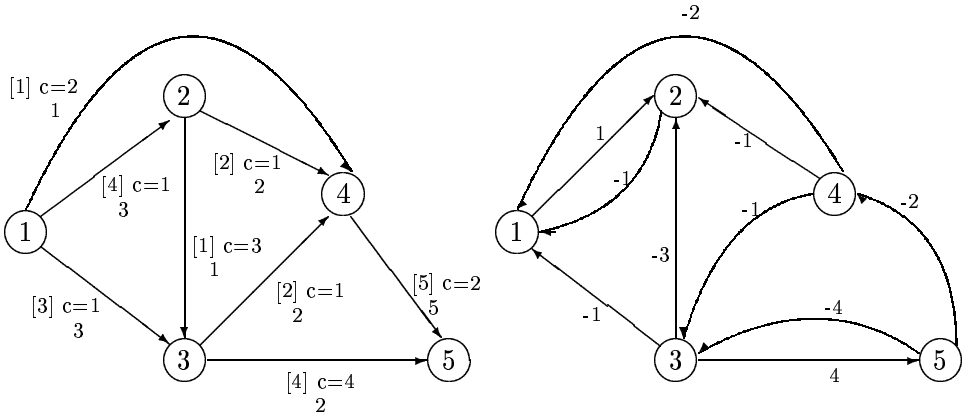
□

In our example a directed cycle with length -1 exists: 3,4,5,3 wich correspond to two paths in the original network (3,4,5 and 3,5).



This means that the cost of the flow can be descreased by 1, if we increase the value of the flow on the path 3,4,5 by 1, and descrease on the path 3,5 (here only one edge) by 1.

The new flow and the corresponding weighted graph are the following:



The cost of this flow is 33 (the modifications are: $+1 \cdot 1 + 1 \cdot 2 - 1 \cdot 4$). In this new graph there is no directed cycle with negative length.

To find the negative length cycles, we can use the Floyd–Warshall algorithm. This algorithm doesn't give us the distance between vertices if there are negative weights in graph, but we can check if there are or not negative length cycles. If the result of this algorithm is the matrix D and there exists a $d_{ii} < 0$ for some i, then in the graph there is at least a negative length cycles.

For our above example the adjacency matrix is:

$$D_0 = \begin{pmatrix} 0 & 1 & \infty & \infty & \infty \\ -1 & 0 & \infty & \infty & \infty \\ -1 & -3 & 0 & 1 & 4 \\ -2 & -1 & -1 & 0 & 2 \\ \infty & \infty & -4 & -2 & 0 \end{pmatrix}$$

and the result of the Floyd-Warshall algorithm is

$$D = \begin{pmatrix} 0 & 1 & \infty & \infty & \infty \\ -1 & 0 & \infty & \infty & \infty \\ -5 & -4 & -1 & 0 & 2 \\ -6 & -5 & -2 & -1 & 1 \\ -9 & -8 & -5 & -4 & -2 \end{pmatrix}$$

Because of the negative values on the main diagonal, in graph there exists a cycle with negative length.

But we can modify the Floyd–Warshall algorithm to find the shortest paths even in the case of negative weights and negative length cycles. When a negative value on the main diagonal appears we will take it 0. In this case the negative length cycles does not influence any more the length of the paths.

$D := D_0$
$cycle := 0$
for $k := 1$ **to** n **do**
  for $i := 1$ **to** n **do**
    for $j := 1$ **to** n **do**
      if $d_{ij} > d_{ik} + d_{kj}$ **then**
        $d_{ij} := d_{ik} + d_{kj}$
        if $d_{ij} < 0$ and $i = j$ **then** $d_{ij} := 0$
                                $cycle := i$
        endif
      endif
    endfor
  endfor
endfor

For our above example:

$$D = \begin{pmatrix} 0 & 1 & \infty & \infty & \infty \\ -1 & 0 & \infty & \infty & \infty \\ -5 & -4 & 0 & 0 & 3 \\ -6 & -5 & -2 & 0 & 2 \\ -8 & -7 & -4 & -3 & 0 \end{pmatrix}$$

If at the end of the algorithm $cycle > 0$, then the graph contains negative length cycles, and vertex v_{cycle} is on that cycle.

To find the negative length cycle let use the following algorithm (in which $d0_{ij}$ is the general element of D_0):

$i := cycle$
$j := \min\limits_{1 \le s \le n} (d0_{is} + d_{si})$ if $j \neq i$. The arc (v_i, v_j) is on the cycle.
while $j \neq i$ **do**
  $k := \min\limits_{1 \le s \le n} (d0_{js} + d_{si})$. The arc (v_j, v_k) is on the cycle.
  $j := k$
endwhile

Resuming the flow problems

– flow networks: maximum flow

methods:	Ford–Fulkerson	$O(n^2 v(f))$
	Edmonds–Karp	$O(n^5)$
	preflow	$O(n^4)$

– minimum cost maximum flow

– generalized flow networks: generalized (or compatible) flow
maximum generalized flow
minimum generalized flow

10 Matching in bipartite graphs

bipartite graph: $G(A, B)$ with
$V = A \cup B$, $A \cap B = \emptyset$, if $\{x, y\} \in E(G)$, then $x \in A, y \in B$.
complete bipartite graph: $K_{m,n}$ (all possible edges)

Theorem 30 *In a bipartite graph there is no odd length cycle.*

Theorem 31 *A graph G is a bipartite one if and only if G has no odd length cycle.*

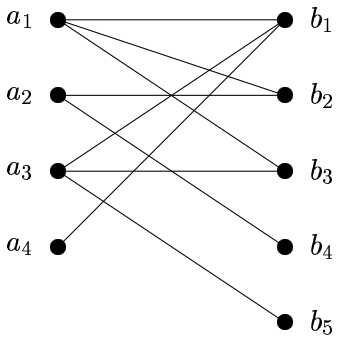
matching: a set of independent edges
complete (or perfect) matching: if every vertices of the graph is incident to an edge of the matching
maximum matching: a matching with the maximum number of edges

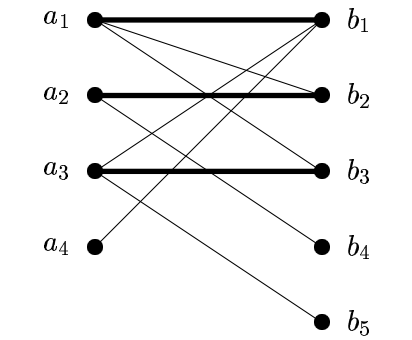
Theorem 32 (Hall) *In a bipartite graph $G(A, B)$ there is a matching which spans A if and only if for all $X \subseteq A$ we have $|N(X)| \geq |X|$.*

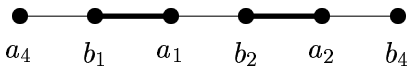
Corollary 1 *There is a matching in $G(A, B)$ which spans both A and B if and only if*
1) $|A| = |B|$
2) $\forall X \subseteq A \Rightarrow |N(X)| \geq |X|$.

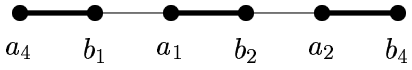
Methods to find maximum matching in bipartite graphs

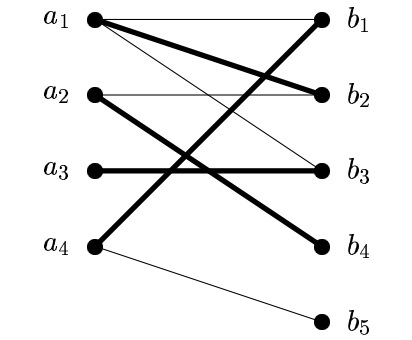
1. Alternating paths method



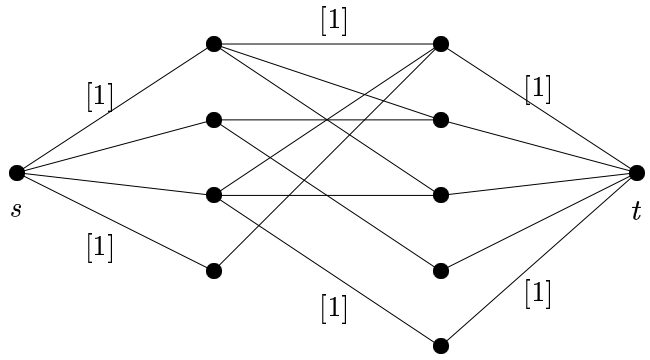








2. Flow method



3. Independent 1's method

	b_1	b_2	b_3	b_4	b_5
a_1	1^*	1	1	0	0
a_2	0	1^*	0	1	0
a_3	1	0	1^*	0	1
a_4	1	0	0	0	0
				L	L

	b_1	b_2	b_3	b_4	b_5	
a_1	$\boxed{1^*}$	$\boxed{1}$	1	0	0	b_2*
a_2	0	$\boxed{1^*}$	0	$\boxed{1}$	0	b_4*
a_3	1	0	1^*	0	1	b_5*
a_4	$\boxed{1}$	0	0	0	0	b_1*
	a_1*	a_2*	a_3*	$L*$	$L*$	

	b_1	b_2	b_3	b_4	b_5
a_1	1	1^*	1	0	0
a_2	0	1	0	1^*	0
a_3	1	0	1^*	0	1
a_4	1^*	0	0	0	0

maxmatch(G): the number of edges in a maximum matching

mincov(G) : the minimum number of vertices such that every edge of G is incident to at least one vertex of the considered vertices

Theorem 33 *In each graph G:* maxmatch(G) $\leq$ mincov(G)

Theorem 34 (König) *In a bipartite graph G:* $\text{maxmatch}(G) = \text{mincov}(G)$

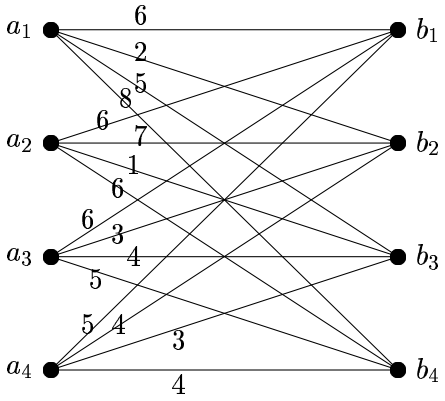
A minimum weight maximum matching
a weighted complete bipartite graph
the weight of a matching = the sum of weights in the matching

The Hungarian method
for a weighted complete bipartite graph $K_{n,n}$

1. Subtraction from rows and columns to have at least one 0 in each row and column
2. Find a maximum number of independent 0's.
3. While the number of independent 0's is not n do
   3.1. minimum covering, modification.
   3.2. Find a maximum number of independent 0's.

The complexity is $O(n^3)$.

Example.



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

Subtract 2 from the first row, 1 from the second row, 3 from the third and fourth row:

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

Subtract 2 from the first column, 1 from the fourth column:

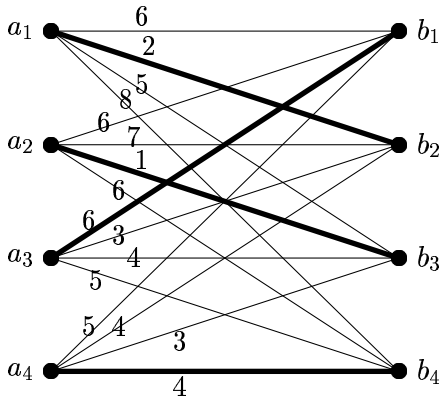
2	0	3	5
3	6	0	4
1	0	1	1
0	1	0	0

Mark by stars a maximum independent set of 0's. Cover by three lines all the 0's.

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

Modification of the elements (Subtract 1 — the minimum uncovered value — from all uncovered elements, leave the covered elements by one line unmodified, add 1 to the double covered elements.) Mark by a star a maximum independent set of 0's.

1	0*	3	4
2	6	0*	3
0*	0	1	0
0	2	1	0*



The value of the maximum matching: $2 + 1 + 6 + 4 = 13$.

If the graph is a $K_{m,n}$ with $m \neq n$ we must transform the matrix to the one with $m = n$ by adding dummy rows or column (with 0 elements only).

Why doesn't work the algorithm for $m \neq n$? A solution represents a set of independent values from the matrix with minimum sum. If $m > n$, our algorithm gives us a set of independent values from only n rows, and maybe we can choose what row to be without selected value and initial the subtracted values were different for these rows.

Example.

3	7	5	8
6	3	2	3
3	5	8	6
5	8	6	4
6	5	7	3

After adding a null column, the problem will be:

3	7	5	8	0
6	3	2	3	0
3	5	8	6	0
5	8	6	4	0
6	5	7	3	0

A possible solution is:

0^*	2	1	5	0
5	0	0^*	2	2
0	0^*	4	3	0
2	3	2	1	0^*
3	0	3	0^*	0

To obtain a maximum weight maximum matching, the algorithm can be transformed as follows:

– subtract all elements of the matrix from the maximum value of the matrix,

– apply the algorithm for the new matrix

– the minimum weight matching of this matrix will correspond to the maximum weight matching in the original matrix.

Example.

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

After transformation:

2	6	3	0
2	1	7	2
2	5	4	3
3	4	5	4

A minimum weight maximum matching is:

2	6	1	0^*
1	0^*	4	1
0	3	0^*	1
0^*	1	0	1

which corresponds in the original matrix to the solution : $8+7+4+5 = 24$.

Matching in general case: in any graph – an independent set of edges.

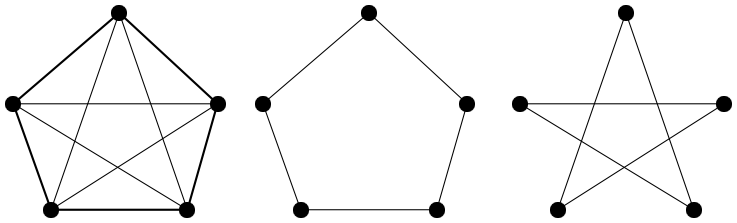
A *factor* of a graph G is a spanning subgraph of G. An *r-factor* is an *r-regular* (all degrees are the same) subgraph.

1-factor = a perfect matching

2-factor = a Hamiltonian cycle

A graph is *factorable* if there are a set of factors which cover all edges of the graph. A graph is *r-factorable* if can be factored in *r-factors.*

For example K_5 can be factored in 2 Hamiltonian cycles.

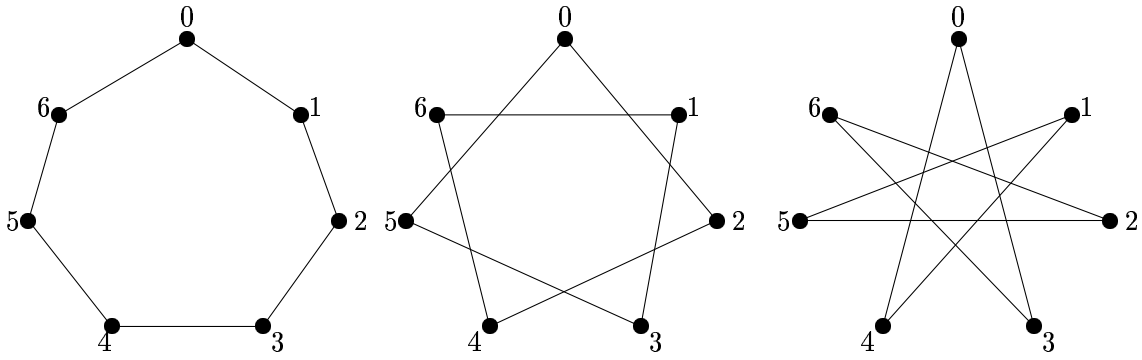


Theorem 35 *The complete graph K_{2n+1} ($n \geq 1$) can be factored in n Hamiltonian cycles.*

Proof. In the following the indices of the vertices will be considered *modulo* $2n+1$. If the vertices are $v_0, v_1, v_2, \ldots, v_{2n}$, the corresponding Hamiltonian cycles are:

H_1 contains the edges: $\{v_i, v_{i+1}\}$ for $i = 0, 1, 2, \ldots, 2n$.
H_2 contains the edges: $\{v_i, v_{i+2}\}$ for $i = 0, 2, 4, \ldots, 4n$.
...
H_j contains the edges: $\{v_i, v_{i+j}\}$ for $i = 0, j, 2j, \ldots, 2nj$.
...
H_n contains the edges: $\{v_i, v_{i+n}\}$ for $i = 0, n, 2n, \ldots, 2n^2$. □

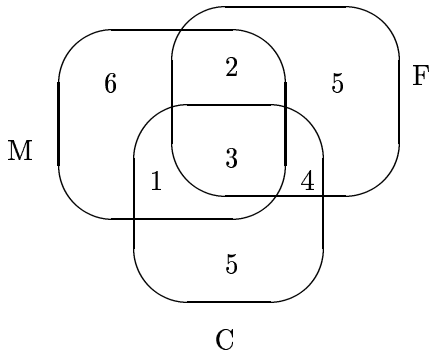
For K_7 the factorization will give us the following factors.


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

11 Extremal problems in graph theory

The principle of inclusion and exclusion

Let us consider the following problem: In a class there are 30 students. From their 12 like mathematics, 14 physics, 13 chemistry. There also 5 students who like both mathematics and physics, 4 who like mathematics and chemistry and 7 who like physics and chemistry. And there are 3 student who like all these three subjects. How may students are in the class who don't like any of these 3 subjects?



$$6 + 2 + 5 + 1 + 3 + 4 + 5 = 26$$

or

$$30 - (12 + 14 + 13) - (5 + 4 + 7) - 3 = 4$$

By generalization:

A is a set, $A_1, A_2, \ldots, A_n$ subsets of A. Let $|A|$ be the number of elements in A. Then the number of elements of A which are not in the subsets A_i $(i = 1, 2, \ldots, n)$ is equal to:

$$|A| - \sum_{i=1}^{n} |A_i| + \sum_{1 \le i < j \le n} |A_i \cap A_j| - \sum_{1 \le < i < j < k \le n} |A_i \cap A_j \cap A_k| + \ldots$$

$$\ldots + (-1)^n |A_1 \cap A_2 \cap \ldots \cap A_n| \qquad (1)$$

Let us use the following notation:

$$\bigcup_{i=1}^{n} A_i = A_1 \cup A_2 \cup \ldots \cup A_n.$$

The following formulas can be proved by induction:

$$|\bigcup_{i=1}^{n} A_i| = \sum_{i=1}^{n} |A_i| - \sum_{1\le i<j\le n} |A_i \cap A_j| + \ldots + (-1)^{n+1} |\bigcap_{i=1}^{n} A_i| \qquad (2)$$

$$|\bigcap_{i=1}^{n} A_i| = \sum_{i=1}^{n} |A_i| - \sum_{1\le i<j\le n} |A_i \cup A_j| + \ldots + (-1)^{n+1} |\bigcup_{i=1}^{n} A_i| \qquad (3)$$

The proof of (2): By induction. For $n = 2$ the formula is true.

$$|A_1 \cup A_2| = |A_1| + |A_2| - |A_1 \cap A_2|. \qquad (4)$$

By the hypothesis of induction:

$$|\bigcup_{i=1}^{n-1} A_i| = \sum_{i=1}^{n-1} |A_i| - \sum_{1\le i<j\le n-1} |A_i \cap A_j| + \ldots + (-1)^{n} |\bigcap_{i=1}^{n-1} A_i|$$

Let us use the following decomposition: $\bigcup_{i=1}^{n} A_i = \left(\bigcup_{i=1}^{n-1} A_i\right) \cup A_n$ and the formula (4):

$$|\bigcup_{i=1}^{n} A_i| = \left(\bigcup_{i=1}^{n-1} A_i\right) \cup A_n = \left|\bigcup_{i=1}^{n-1} A_i\right| + |A_n| - \left|\left(\bigcup_{i=1}^{n-1} A_i\right) \cap A_n\right| =$$

$$= \sum_{i=1}^{n-1} |A_i| - \sum_{1\le i<j\le n-1} |A_i \cap A_j| + \ldots + (-1)^{n} \left|\bigcap_{i=1}^{n-1} A_i\right| +$$

$$+|A_n| - \left|\left(\bigcup_{i=1}^{n-1} A_i\right) \cap A_n\right| \qquad (5)$$

The last expression can be write:

$$\left(\bigcup_{i=1}^{n-1} A_i\right) \cap A_n = \bigcup_{i=1}^{n-1} (A_i \cap A_n),$$

which after the substition in (5) will give the requested formula.

Theorem 36 (Zarankiewicz) *If G is a graph of order n, which does not contain complete subgraphs of orfer k, then for the minimum degree δ of vertices we have:*

$$\delta \leq \left\lfloor \frac{(k-2)n}{k-1} \right\rfloor$$

Proof. Let $f = \left\lfloor \frac{(k-2)n}{k-1} \right\rfloor$. The following formula results:

$$(k-2)n = f(k-1) + r, \qquad \text{where } 0 \leq r < k-1 \tag{6}$$

□

Let us consider that the graph $G = (V, E)$ satisfies the condtions in the theorem, but the degree of each vertex is at least $f+1$. A contradiction will arise.

Let $x_1 \in V$ and $x_2 \in N(x_1)$, then

$$|N(x_1) \cap N(x_2)| = |N(x_1)| + |N(x_2)| - |N(x_1) \cup N(x_2)| \geq$$

$$\geq (f+1) + (f+1) - n = 2(f+1) - n > 0$$

To prove that $2(f+1) - n > 0$ let us consider (6), and we have for n:

$$n < \frac{(f+1)(k-1)}{k-2} = (f+1)\left(1 + \frac{1}{k-2}\right) \tag{7}$$

For $k \geq 3$ we have $n < 2(f+1)$.

So there is a vertex $x_3 \in N(x_1) \cap N(x_2)$ (so $k > 3$). By a similar computation:

$$|N(x_1) \cap N(x_2) \cap N(x_3)| \geq 3(f+1) - 2n > 0$$

The last inequality is from(7) too, for $k \geq 4$.

So we shall obtain:

$$|N(x_1) \cap \ldots N(x_{k-1})| = |N(x_1)| + |\bigcap_{j=2}^{k-1} N(x_j)| -$$

$$- \left| N(x_1) \cup \left(\bigcap_{j=2}^{k-1} N(x_j) \right) \right| \geq (f+1) + (k-2)(f+1) - (k-3)n - n =$$

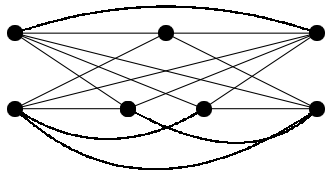
$$= (k-1)(f+1) - (k-2)n = (k-1)(f+1) - f(k-1) - r =$$

$$= k - 1 - r > 0$$

So $x_k \in N(x_1) \cap N(x_2) \cap \ldots \cap N(x_{k-1})$, but these vertices $x_1, x_2, \ldots, x_k$ are the vertives of a complete subgraph of order k, which is contradiction, which proves the theorem.

A k-partite graph is a generalization of the bipartite graph: There are k sets of independent vertices and edges are only between vertices from different sets. $K_{n_1,n_2,\ldots,n_k}$ is a complete k-partite graph.

A complete 3-partite graph: $K_{2,3,2}$:



Theorem 37 (Turán) *In a graph of order n which does not contain complete subgraphs of order $(k+1)$ the maximal number of edges is:*

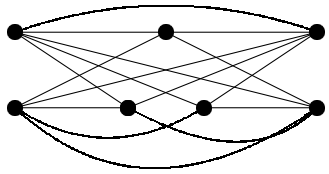
$$e \leq \frac{1}{2}\left(n^2 - r^2\right)\frac{k-1}{k} + \frac{r(r-1)}{2}$$

where $n = hk + r$, $0 \leq r < k$.

The extremal graphs (for which the equality holds) are $K_{n_1,n_2,\ldots,n_k}$ with r independent sets of $h+1$ vertices each and $k-r$ independent sets with h vertices each (So from $n_1, n_2, \ldots, n_k$ r are equal to $h+1$ and $k-r$ are equal to h).

Example.

Let be $n = 7$, $k = 3$, and because $7 = 2 \cdot 3 + 1$ we have $h = 2$ and $r = 1$. $n_1 = 3, n+2 = 2, n_3 = 2$, and the extremal graf is $K_{3,2,2}$ (which is equivalent to $K_{2,3,2}$):



Proof. By induction on n. For $n = 1, 2, \ldots, k$ the theorem is true and $e \leq \frac{1}{2}k(k-1)$. Let us consider that the theorem is true for all $n' \leq n-1$, and let's consider a graph G of order n. By the theorem of Zarankiewicz

$$\delta = \min_{y \in V} \deg(y) \leq \left\lfloor \frac{(k-1)n}{k} \right\rfloor$$

and let $x \in V$ for which $\deg(x) \leq \delta$. Delete vertex x and its incident edges from G to obtain the graph G_x. This graph does not contain a K_{k+1}, and if its number of edges is not maximal, replace it by a graph of order $n-1$ and with maxiaml number of edges. For G_x the theorem is true by the induction hypothesis, so is of the form requested: ak-partite graph with r' independent sets, each of $h'+1$ vertices and $k-r'$ independent sets, each of h' vertices, where $n-1 = h' \cdot k + r'$. If we add to this graph the deleted vertex x in a set of h' vertices and join it to all vertices in other independent sets, the resulted graph is with same structure, an extremal graph. $\deg(x) = n-1-h' < \left\lfloor \frac{(k-1)n}{k} \right\rfloor$. So this graph has the same structure.

This extremal graph can be found by deleting from K_n the edges of r subgraphs K_{h+1} and the edges of $k-r$ subgraphs K_h. So in the extremal graph

$$e = \binom{n}{2} - r\binom{h+1}{2} - (k-r)\binom{h}{2} =$$

$$= \binom{n}{2} - r\frac{(h+1)h}{2} - (k-r)\frac{h(h-1)}{2} =$$

$$= \frac{n(n-1)}{2} - \frac{h}{2}\big(2r + kh - k\big)$$

but $h = \dfrac{n-r}{k}$ and

$$e = \frac{n(n-1)}{2} - \frac{h}{2}\big(r + n - k\big) = \frac{n(n-1)}{2} - \frac{n-r}{2k}\big(r + n - k\big) =$$

$$= \frac{n^2k - nk - (n^2 - r^2) + kn - kr}{2k}$$

and by adding and subtracting r^2k we have

$$e = \frac{n^2k - r^2k + -(n^2 - r^2) + r^2k - kr}{2k} = \frac{k(n^2 - r^2) - (n^2 - r^2)}{2k} + \frac{r^2 - r}{2} =$$

$$= \frac{n^2 - r^2}{2} \cdot \frac{k-1}{k} + \frac{r(r-1)}{2}$$

□

Extremal problems of Ramsey type

Let $R(m,k)$ be the least number for which any graph of order $n \geq R(m,k)$ contains a K_m or its complement contains a K_k. Extremal graphs are those graphs of order $n-1$ for which this property is not true.

As we know: $R(3,3) = 6$. An extremal graph is a cycle of 5 vertices.
It is easy to prove that:

$$R(1,k) = R(k,1) = 1$$

$$R(2,k) = R(k,2) = k$$

Theorem 38 ***a)*** *If* $R(m-1,k)$ *and* $R(m,k-1)$ *there exist, then* $R(m,k)$ *there exists too, and:*

$$R(m,k) \leq R(m-1,k) + R(m,k-1).$$

b) *If* $R(m-1,k) = 2p$ *and* $R(m,k-1) = 2q$ *for* $p, g \in \mathbf{N}^*$ *then*

$$R(m,k) < R(m-1,k) + R(m,k-1).$$

($\mathbf{N}^*$ represents the positive integers)
Proof. **a)** Let be $n = R(m-1,k) + R(m,k-1)$. Let us color the edges of the graph K_n by red and blue. A vertex r in this graph has degree $n-1$, and it is incident to n_1 red edges which span a graph G_1, and n_2 blue edges which span a graph G_2. So

$$n_1 + n_2 + 1 = R(m-1,k) + R(m,k-1).$$

There are two cases:

— If $n_1 \geq R(m-1,k)$ then in G_1 there is a red K_{m-1} or a blue K_k. But the graph K_{m-1} with the vertex r and red edges incidents to it will form a red K_m.

– If $n_1 < R(m-1,k)$ then $n_2 \geq R(m,k-1)$ must be. The proof is similar to the previous case.

b) Let us consider a $K_{2p+2q-1}$ with red and blue edges. Each vertex has degree $2p+2q-2$. Let r be a vertex of the graph. There three cases:

— Between the edges incidents to r there are at least $2p$ red edges. In the graph spanned by these edges there is an a red K_{m-1} or a blue K_k. The red K_{m-1} with vertex r and its incidents red edges form a red K_m.

— Between the edges incidents to r there are at least $2q$ blues edges. In the graph spanned by these edges there is an a red K_m or a blue K_{k-1}. The blue K_{k-1} with vertex r and its incidents red edges form a red K_k.

— There are exactly $2p-1$ red and $2q-1$ blue edges which are incidents to r. But this case cannot happen for all vertices of the graph, because in this case in the red graph there is an odd number of vertices all of odd degree. □

The mathematical induction on two variables

Let $A(m,k)$ be a property depending on naturals m and k.

If $A(1,k)$ and $A(m,1)$ are true, and if from $A(m-1,k)$ true and $A(m,k-1)$ true, we can prove that $A(m,k)$ is true too, then this property is true for all naturals m and k.

Theorem 39 *$R(m,k)$ there exists for all $m,k \in \mathbf{N}^*$ and*

$$R(m,k) \leq \binom{m+k-2}{m-1}.$$

Proof. Let the property $A(m,k)$ the following: $R(m,k) \leq \binom{m+k-2}{m-1}$.

$$1 = R(1,k) \leq \binom{k-1}{0} = 1 \quad \text{is true}$$

$$1 = R(m,1) \leq \binom{m-1}{m-1} = 1 \quad \text{is true}$$

By induction hypothesis:

$$R(m-1,k) \leq \binom{m+k-3}{m-2}$$

$$R(m,k-1) \leq \binom{m+k-3}{m-1}$$

But from the theorem 38

$$R(m,k) \leq R(m-1,k)+R(m,k-1) = \binom{m+k-3}{m-2}+\binom{m+k-3}{m-1} = \binom{m+k-2}{m-1}.$$

□

Some known values:

$R(3,3) = 6,\ R(3,5) = 14,\ R(3,6) = 18.$

$R(k,k) > 2^{k/2}$(Erdős)

Unsolved:

There exists the limit $\lim_{k\to\infty} R(k,k)^{1/k}$?

Generalization

$$R(k_1+1, k_2+1, \ldots, k_n+1) \leq \frac{(k_1! + k_2! + \ldots + k_n!)}{k_1!k_2! \cdot \ldots \cdot k_n!}$$

Paul Erdős proved that:

$$R(\underbrace{3,3,\ldots,3}_{r \text{ times}}) \ \leq\ r! \sum_{k=0}^{r} \frac{1}{k!} + 1$$

and he conjectured that the equality always holds.
Known values are: $R(3,3) = 6$, $R(3,3,3) = 17$ for which the equality holds.