various

Lecture Notes of Spring 2013

Algorithms I

University of Mannheim
2013

This script originates from the course "‘Algorithms I"’ at the University of Mannheim as lecture notes.

Many thanks to Ingo Bürk, who initially published the underlying LaTeX template here.

Contents

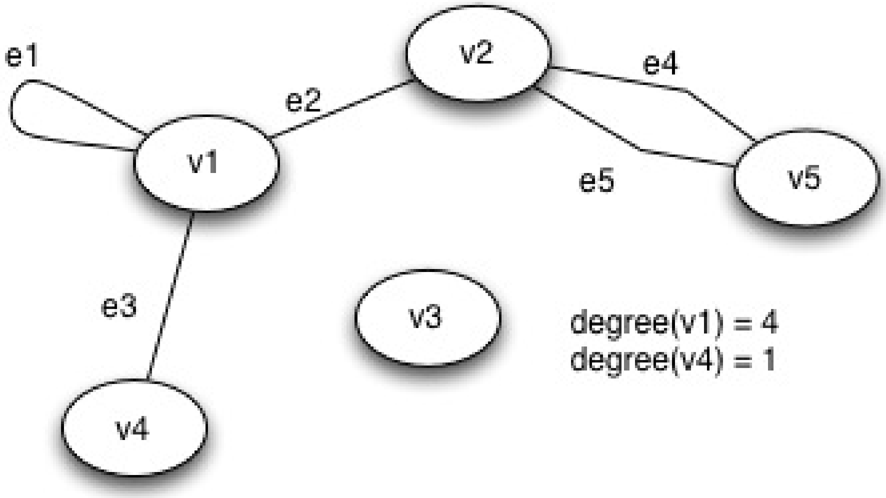
1

Elementary Notions about Graphs

Definition 1.1
An undirected graph is a pair G=(V, E) where V is a set of nodes and E is a set of edges, together with a function i: E $\rightarrow \gamma(v)$ such that $0 < \mid i(e) \mid \leq 2$.
If u,v i(e) we call u,v endpoints of e. If V and E are a finite set we call G a finite graph. If i(e_1) = i(e_2) we call e_1, e_2 parallel edges.

If $\mid i(e) \mid = 1$ we call e a loop. The degree of a node v is the number of edges for which v is an endpoint where loops a counted twice. Is the degree of v = 0 then we call v isolated.

Example



Lemma 1.1
In a finite graph the number of nodes with odd degree is even.

Proof: $\sum_{i=1}^{n} degree(v_i) = 2 * |E|$
This is because we start with a graph, where each node is isolated. Then we insert one edge after another.
Case 1: i(e) = x then the degree of x is increased by 2
Case 2: i(e) = x, y then the degree of x and y are increased by 1
We asume that $v_1 ... v_i$ have an even degree and $v_i + 1 ... v_n$ have odd degree.

$$\sum_{k=1}^{i} degree(v_k) + \sum_{k=i+1}^{n} degree(v_k) = 2|E|$$

$\sum_{k=1}^{i} degree(v_k)$ is an even number

$\sum_{k=i+1}^{n} degree(v_k)$ must be an even number and hence the number of nodes with odd degree must be even

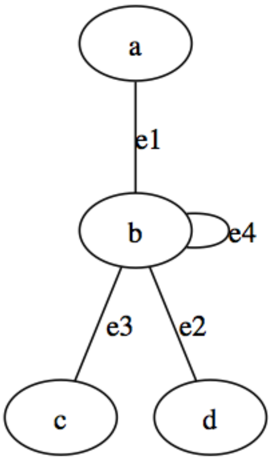
$2|E|$ is an even number

Definition 1.2
If G=(V,E) is a graph and $v_1, v_2 \in V$ with $i(e) = \{v1, v2\}$ then we say that v_1, v_2 are neighbours. A path in G is a sequence of edges $e_1, e_2, ...$ such that:

i) $e_i, e_i + 1$ share an endpoint

ii) if e_i is not a loop and neither the first nor the last edge. Then e_i shares one endpoint with $e_i - 1$ and the other with $e_i + 1$ [MS: does this make sense? sounds strange!]

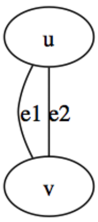
Example



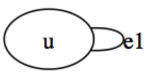
A finite graph is graphically represented by: $v_0 - v_1 - v_2 - ... - v_i$
v_0 is called start point and v_i is called end point. The length of the path e_1 ... e_i is i.

A cycle (circle) is a path where the end point coincide with the start point.
A path is called simple if every node in V occures at most once.
A cycle of length $\neq 2$ is called simple if every node except of the start/end node occurs at most once. A cycle of length 2 (e_1, e_2) is called simple if $e_1/not = e_2$ and if each node except for the start/end node occurs at most once.

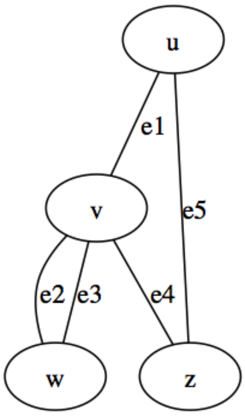
Example



Simple

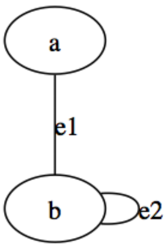


Simple



Not simple

A Graph is connected if for every pair of nodes (u, v) there is a path between u and v.



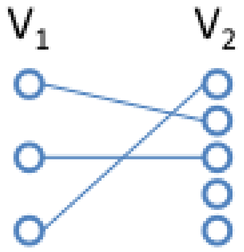
An infinite Graph has finite and infinite paths. Every path between two nodes is finite. The graph is connected. There are infinitely paths.
e.G. e_1 (finite)
or e_1, e_2
...
and there is also an infite path e_1, e_2, e_2, e_2, e_2, e_2, ...

Definition 1.3
Let $G(V,E)$ be a connected graph $a \in V$ is called a seperation point (articulation point) if there are nodes v,w such that every path connecting v and w visits a. If G has such a point G is called [word missing - MS]
An edge is called bridge if there exist nodes v, w such that every path connecting v and w contains this edge.

Example
Real-Life examples where seperation points are important are in computer networks or the information distribution (flow of information) within a company. So a seperation point can be regarded as a kind of coordinator between a and b.

Definition 1.4
Let $G = (V, E)$ be a graph without loops. If there exists $V_1, V_2 \leq V$ and $V_1 \cup V_2 = V$ such that $V_1 \cap V_2 = \emptyset$ and every edge e has one endpoint in V_1 and the other in V_2, then we call G a bipartite.

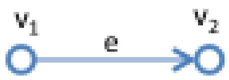
Example

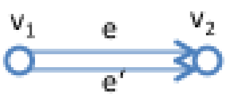


Definition 1.5
A directed graph is a pair $G = (V, E)$ where V is a set of nodes (vertices) and E is a set of edges together with a function $i : E -> V x V$. If $i(e) = (v_1, v_2)$ then v_1 is called start point, v_2 is called end point.

Graphically:

If $i(e) = (v_1, v_2)$ we draw 1.
If $i(e') = (v_1, v_2)$ then this indices a second edge (2.).
If $i(e_1) = i(e_2)$ we call e_1, e_2 parallel.
If $i(e) = (v, v)$ then e is called a directed loop.

1. 

2. 

$g_{out}(v)$ is the number of edges that have starting point v.
$g_{in}(v)$ is the number of edges with endpoint v.

Lemma 1.2
$\sum_{v \in V} g_{in}(v) = \sum_{v \in V} g_{out}(v)$

Proof: We start with a graph without edges. Then we insert one after the other edges in E. Each edge contributes 1 to both sides of the equation.

Definition 1.6
A directed path is a sequence of edges $e_1, e_2...$ such that the end point of e_i is the start point of $e_1 + 1, i > 1$([NW] +1 seems strange to me, correct?).

A directed path $e_1...e_k$ is called a (directed) cycle, if the start point of e_1 and the end point of e_k coincide.
A simple (directed) path is a path where every node occurs at most once.
A directed cycle is called simple if every node except for the start and end node occurs at most once.

Definition 1.7
A graph directed or undirected is called simple, if it does not contain parallel edges.

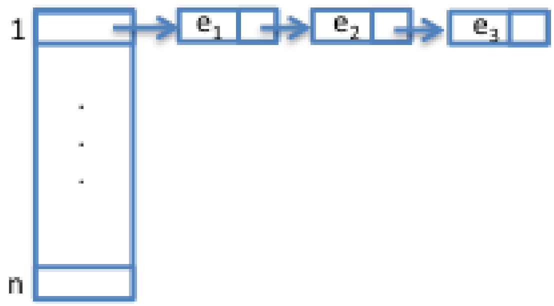
Definition 1.8
A directed graph is called strongly connected if for any pair of nodes (u,v) there is a directd path from u to v.

Let G be a directed graph $G = (V,E)$. $x, y \in V x \sim y$ ([NW] does ~mean are "connected"?) if there is a directed path from x to y and vice versa.

The equivalence classes of this relation $cVxV$ are called strongly connected components. (Analogously: Define connected components for undirected graphs)

We should know how the following terms are defined: reflexivity, symetry, transitivity.

Implementation:
1. Adjacency Lists $V = 1...n, E = e_1...e_t$



2. Dynamically changing graphs:
e.g. multi user databses: Nodes ≡ transactions of user; Edges ≡ waiting situations

1	1	2
2	1	3
3	1	2

Graph is used to detect dead locks. Waiting arises when data are locked by a user that modifies these data.
U_1 $write(d), read(d')$
U_2 $read(d), write(d')$

Definition 1.9
An undirected graph is called a tree if it is connected and does not have simple cycles.
Let G be a directed graph, $G = (V,E)$. A node r is called root if every other node can be reached from r via a directed path.
A directed graph is called a tree if it has a root and the underlying undirected graph is a tree.
Let G be a directed graph. A node is called source if $g_{in}(v) = 0$. v is called sink if $g_{out}(v) = 0$

Lemma 1.3
NW: what was lemma 1.3? the next one was 1.4 in my notes

Lemma 1.4
If $G = (V, E)$ is a directed graph without directed cycles, then tere is always a source and sink.

We use this theorem to detect cycles

Proof: Source (sink analogously): Select an arbitrary node v_1. If v_1 is a source we are done. If it is not, then there must be an edge e_1 leading to it $v_2 \xrightarrow{e_1} v_1$.
If v_2 is a source we are done. If not, there must be an edge e_2 leading to it $v_3 \xrightarrow{e_2} v_2 \xrightarrow{e_1} v_1$. We continue this process. It must stop because there are only finitely many nodes and if a node would appear once more on such a path, there would be a directed cycle.

2

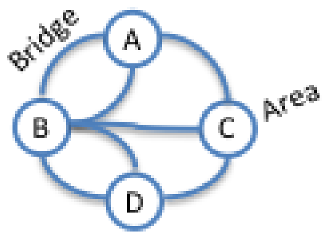
Euler Graphs and Hamilton Graphs

TODO

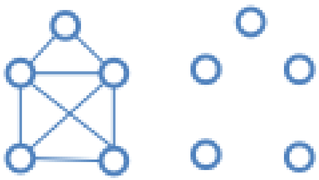
2.1 Euler Graphs

2.1.1 Euler 1736: Königsberger Brückenproblem

Is it possible to do a round walk crossing every bridge exactly once?



Example 2.1



Definition 2.1
Let G be a finite undirected graph. A path $e_1..e_t$ is called a euler path if every edge in E occurs exactly once in the list.
A graph is a euler graph if it has a euler path.

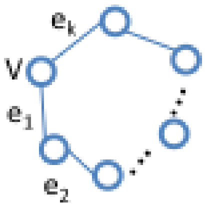
Theorem 2.1
A finite connected graph is a euler graph if and only if:

i) It ha eiter exactly two nodes of odd degree. or

ii) All nodes have even degree.

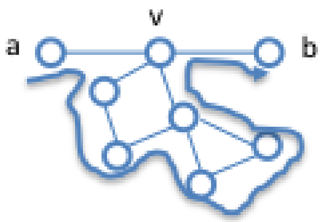
In the last case the path is a cycle. In the first case no euler path is a cycle. Check is possible in linear time.

Proof: " > " Let $G = (V, E)$ be a graph that has a euler path that is not a cycle. Let $|E| = k$ $\circ \xrightarrow{e_1} \circ \xrightarrow{e_2} ...\circ \xrightarrow{e_k}$ In this path v_1 and v_{k+1} have od degreee and all other nodes have even degree. Now consider the case teht G has a euler cycle.



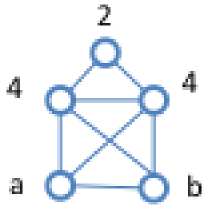
Hence every node has even degree.

" < " Let G be a graph with exactly two nodes with odd degree, let this be a and b. We contradict a euler path as follows:
Start at node a and folow an edge ?inktt? on a. $a\circ \rightarrow \circ \rightarrow ...\circ b$
Case 1: All edges have been used -> done
Case 2: Still edges unused. Then because G is connected there must be some node v on the path from which there is an unused edge. We construct a path starting from v as before. This path must end in v.
$\Rightarrow$ Repeat until there are no more unused edges.
Analogously we proceed when the degree of all nodes is even.

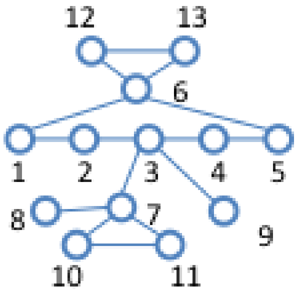


Example

1.



2.



In the directed case a directed Euler path is a directed path on which every edge appears exactly once. Directed Euler cycle analogously.

Theorem 2.2
A finite directed graph is a directed Euler graph if and only if its underlying undirected graph is connected.

i) There is one node a with $g_{out}(a) = g_{in}(a) + 1$ and another node $b g_{out}(b) = g_{in}(b) + 1$ and for all other nodes $v g_{in}(v) = g_{out}(v)$. Or

ii) For all nodes $g_{in}(v) = g_{out}(v)$ (directed Euler cyle)

2.2 Hamiltonian Graphs

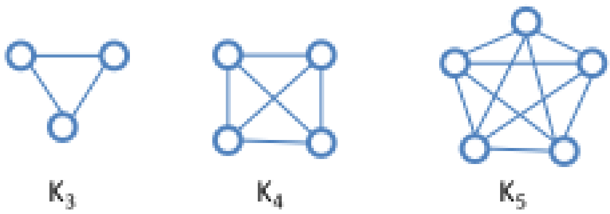
Definition 2.2
Let $G = (V, E)$ be a graph. A Hamiltonian cycle C is a cycle on which every node $\in V$ occurs exactly once. If G has a Hamiltonian cycle it is called Hamiltonian.

Example

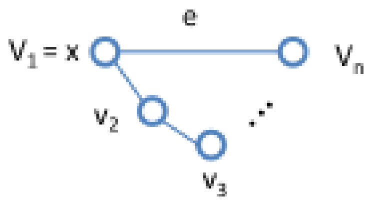
The problem, given an arbitrary undirected graph: Is it Hamiltonian? ⇒ NP complete ⇒ no polinomial time algorithm is known and it is assumed there is no such.
One way out of the complexity issue is to derive conditions that can be tested explicitly and if they are satisfied the desired property is ensured.

Theorem 2.3
Let $G = (V, E)$ be an undirected finite graph without loops and without parallel edges. Let $|V| = n$. If for all $x, y \in Vwithx \neq y$ and no edge with end points x, y the following holds:
$deg(x) + deg(v) \geq |V| = n$
Then G has a Hamiltonian Cycle.

Example



Proof: Assume there is a graph $G = (V, E)$ with $deg(x) + deg(y) \geq |V|$ for all x and y with $x \neq y$ and no edges between them, but is not Hamiltonian. Among all graphs with nodes in V, we choose one that has this property and has the maximal number of edges, we call graph $G_0 = (V, E_0)$. As the complete graph (every node is connected with every other node) is Hamiltonian, we know there must be an edge e connecting some x and y and $e \in E_0$.
We add edge e to the graph and obtain a new graph $G_1 = (V, E_1)$ that still satisfies the degree conditions and must be Hamiltonian because G_0 was the one with the largest number of edges.
We know that the Hamiltonian cycle must contain the edge e.



$v_i \neq v_j for i \neq j$

$S = \{v_i : 1 \leq i \leq n$ x,y are connected with an edge in $E_0\}$
$T = \{v_i : 1 \leq i \leq n$ there is an edge between y and v_i in $E_0\}$

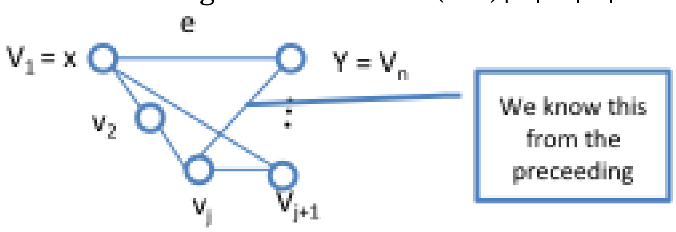
Observation:

i) $y = v_n \in S \cup T$

ii) $|S \cup T| < |V| = n$

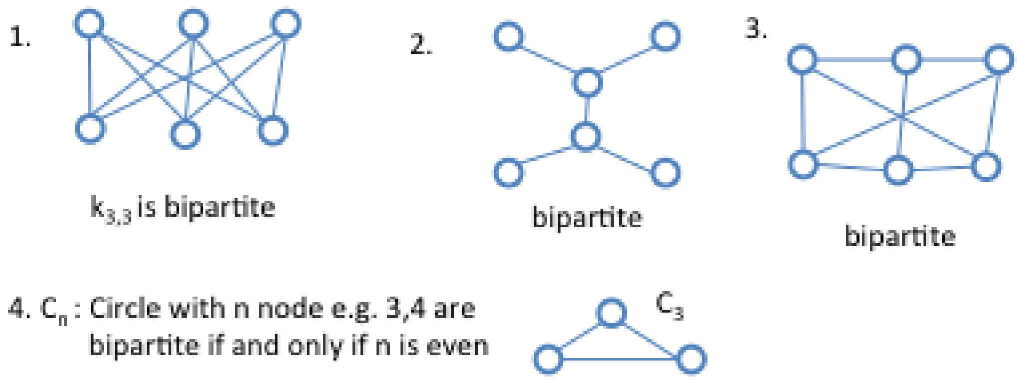
iii) $deg(x) = |S|$
$deg(y) = |T|$

Hence $S \cap T \neq \emptyset, let v_j \in S \cap T$ hence there is an edge between x, v_{j+1} and an edge between y, v_j.
Now remove edge e and there is a Hamiltonian left.
Cost of checking the condition $O(M^2)\,|E| \leq |V|^2$



2.3 Bipartite Graphs

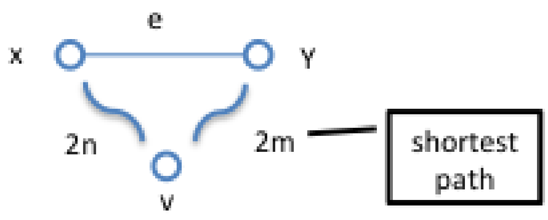
Example



> **Theorem** 2.4
> Let $G = (V, E)$ be a connected undirected graph without loops and parallel edges. G is bipartite if and only if it does not contain any circle of odd length.

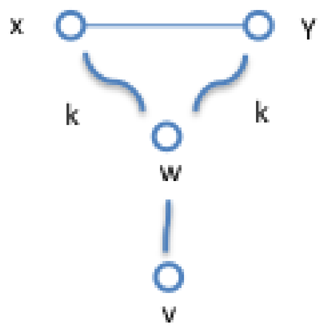
Corollary: All trees are bipartite

Proof: $\Rightarrow$ IF G contains a cicle of odd length then it is not bipartite. $\Leftarrow$ Let G not have any circle of odd length we choose node v.
$V_1 = \{u \in V$ a shortest path between u and v is of odd length$\}$
$V_2 = \{u \in V$ a shortest path between u and v is of even length$\}$
$V \in V_2, V = V_1 \dot\cup V_2$ (disjoint union)
Claim: Ther is no edge e with both endpoints in V_1 respectively V_2. Assume there is an edge e with both endpoints in V_1. Let the end points be x, y



$2m \leq 2n + 1 \text{and} 2n \leq 2m + 1 \Rightarrow m = n$

Let $P(x)$ a shortest path from v to x, analogously $P(y)$ let w be the last node on the paths starting at v that lies on both paths.

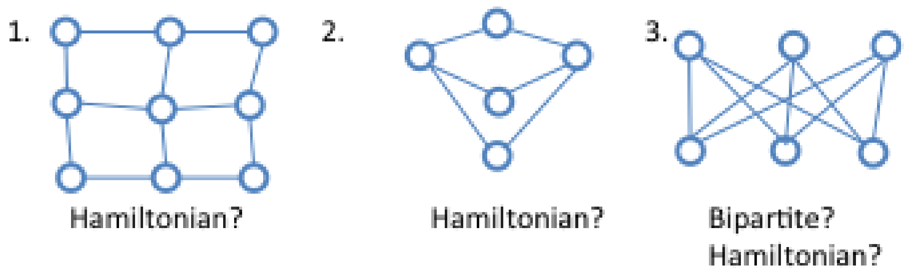


The length of the path from w to x coincides with the length of path from w to y.
The circle w - x - y - w is of odd length i.e. $2k+1 \Rightarrow$ contradiction!

Corollary 2.5: A bipartie graph with an odd number of nodes cannot be Hamiltionian

Proof: Assume if were Hamiltionian then there is a cycle where node appears exactly once. This cycle is of odd length $\Rightarrow$ contradicts Theorem 2.4

Example 2.2



$\Rightarrow$ We have two theorems to check:

i) Count degrees

ii) Corollary 2.5

3
(Network) Flow Problems

TODO

3.1 Network Flow Problems

Example 3.1
Example: Oil field + transportation

Definition 3.1
A network N consists of

i) A finite directed graph $G=(V,E)$ without loops and parallel edges

ii) a function $c: E->\mathbb{R}^+$, which assigns a capacity to each edge

iii) two designated nodes s and t, called source and sink

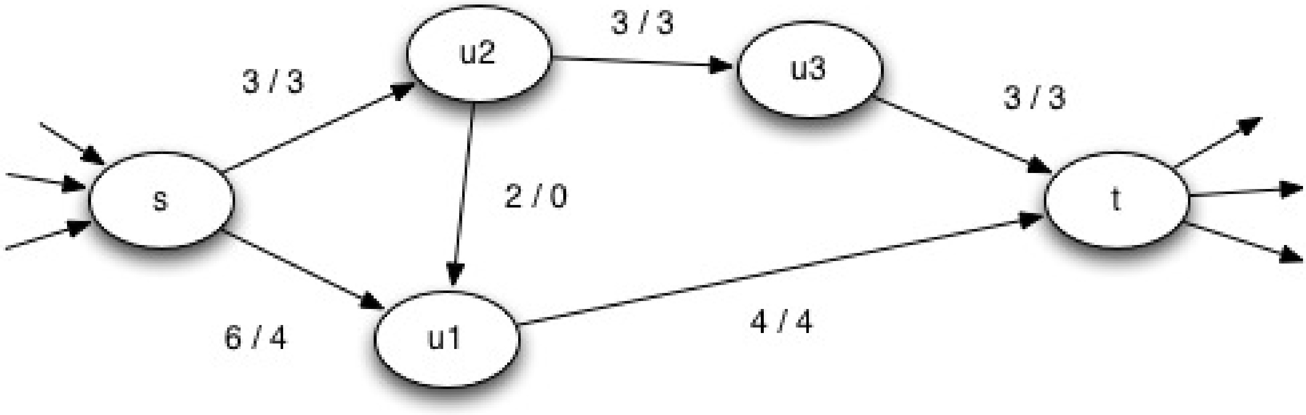
Short: $N=(G,c,\{s,t\})$

Definition 3.2
Let $N=(G,c,\{s,t\})$ be a network. A flow function on N is a function $f: E->\mathbb{R}$ such that

- $0 \le f(e) \le c(e), \forall e \in E$
- $\alpha(v) := \{e : \text{endpoint of e is v}\}, v \in V$
  $\beta(v) := \{e : \text{startpoint of e is v}\}, v \in V$

For every $v \in V \setminus \{s,t\}$
$\sum_{e\in\alpha(v)} f(e) = \sum_{e\in\beta(v)} f(e)$
This is called "conservation function".
The total flow of the flow function is given by $F=\sum_{e\in\alpha(t)} f(e) - \sum_{e\in\beta(t)} f(e)$

Example 3.2



Notion: 6/4 describes the capacity and the flow of an edge. In this example the capacity of the edge is 6 and the flow is 4.

Problem:
Given an arbitrary network N, find a flow function f, where the total flow F is maximal.

Definition 3.3
Let N = $(G, c, \{s, t\})$ be a network. Let $S \subseteq V$ with s $\in S, t \notin S$
$\bar{S} = V \setminus S$ (i.e. $t \in \bar{S}$)
$E_{S\bar{S}} = \{e$: all edges with starting point in S and endpoint in $\bar{S}\}$
$E_{\bar{S}S} = \{e$: all edges with start point in $\bar{S}$ and end point in S $\}$
$E_{S\bar{S}} \cup E_{\bar{S}S}$ is the cut defined by S.
The capacity of a cut defined by S: $c(S) = \sum_{e \in E_{S\bar{S}}} c(e)$

Lemma 3.1
Let $N = (G, c, \{s, t\})$ be a network, $f : E \to \mathbb{R}$ be a flow function then for any $S \subseteq V$ with $s \in S, t \notin S$:

$$F = \sum_{e \in E_{S\bar{S}}} f(e) - \sum_{e \in E_{\bar{S}S}} f(e)$$

Proof.

$$F = \sum_{e \in \alpha(t)} f(e) - \sum_{e \in \beta(t)} f(e)$$

$$0 = \sum_{e \in \alpha(v)} f(e) - \sum_{e \in \beta(v)} f(e); \forall v \in \bar{S} \setminus \{t\}$$

We add all these equations up. Left hand side: F remains. Right hand side: Let $x \xrightarrow{e} y$ be an edge. We need to consider 4 cases:

i) $x, y \in S$, then the value $f(e)$ does not occur in the summation

ii) $x, y \in \bar{S}$, then $f(e)$ occurs one time positive in the summation, namely for y
f(e) occurs one time negative in the summation, namely for x

iii) $x \in S; y \in \bar{S}$, f(e) occurs positive for y and nowhere else and $e \in E_{S\bar{S}}$

iv) $x \in \bar{S}; y \in S$, then $f(e)$ occurs negative for x and nowhere else and $e \in E_{\bar{S}S}$

This leads to the following equation:

$$F = \sum_{e \in E_{S\bar{S}}} f(e) - \sum_{e \in E_{\bar{S}S}} f(e)$$

Only case **??** and **??** contribute. □

Lemma 3.2
For every flow function f with total flow F and any ser $S \subseteq V$, $s \in S$, $t \notin S$

$$F \leq c(S)$$

Proof. From lemma 3.1 we know

$$F = \sum_{e \in E_{S\bar{S}}} f(e) - \sum_{e \in E_{\bar{S}S}} f(e) \leq \sum_{e \in E_{S\bar{S}}} f(e) \leq \sum_{e \in E_{S\bar{S}}} c(e) = c(S)$$

□

Korollar 3.1 **Max Flow - Min Cut Statement**
If $F = c(S)$ then the total flow F is maximal and the capacity of the cut defined by S is minimal.

Proof. Let $F = c(S)$, consider another flow function f' with total flow F'.

i) $F' \leq c(S)$ (Lemma 3.2) // $F' \leq c(S) = F$
Hence, f is a flow function with maximal total flow.

ii) Let S' with $s \in S'$, $t \notin S'$ be given. $c(S) = F \leq c(S')$. Hence the capacity $c(S)$ is minimal among all other capacities.

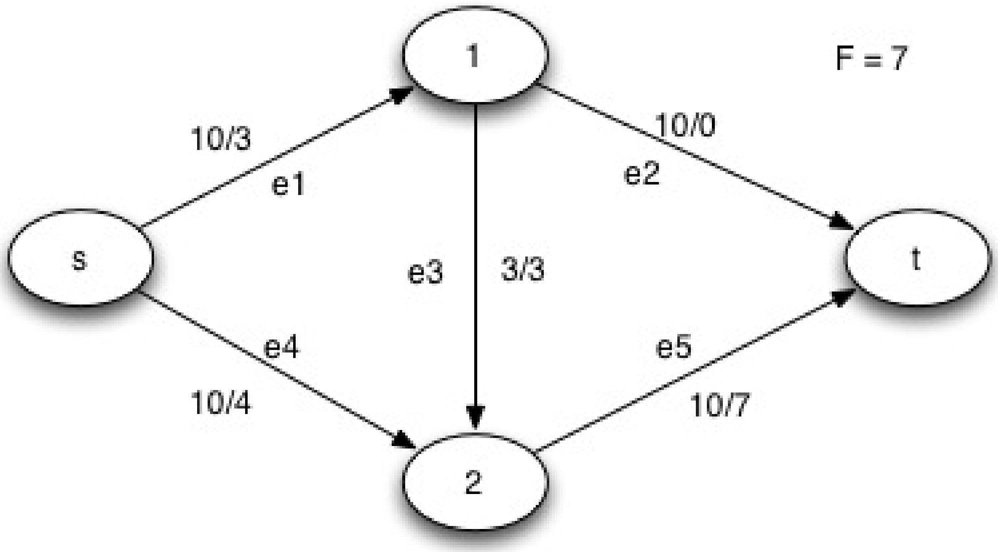
□

An augmenting path is a simple path from s to t, that is not necessarily directed. And for which the following two cases hold: Let e be an edge on this path:

i) $s \to \circ \to \circ \to \ldots \to \underset{s_i}{\circ} \xrightarrow{e} \underset{s_{i+1}}{\circ} \ldots \underset{t}{\circ}$ then we request that $f(e) < c(e)$

ii) $s \to \circ \ldots \underset{s_i}{\circ} \xleftarrow{e} \underset{s_{i+1}}{\circ} \ldots \underset{t}{\circ}$ then we request that $f(e) > 0$

Example 3.3



Which of the following is an augmenting path?

- e_1e_2
- $e_1e_3e_5$
- $e_4e_3e_2$
- e_4e_5

Solution:
The first, third and fourth example are augmenting paths. The second path violates case 1.

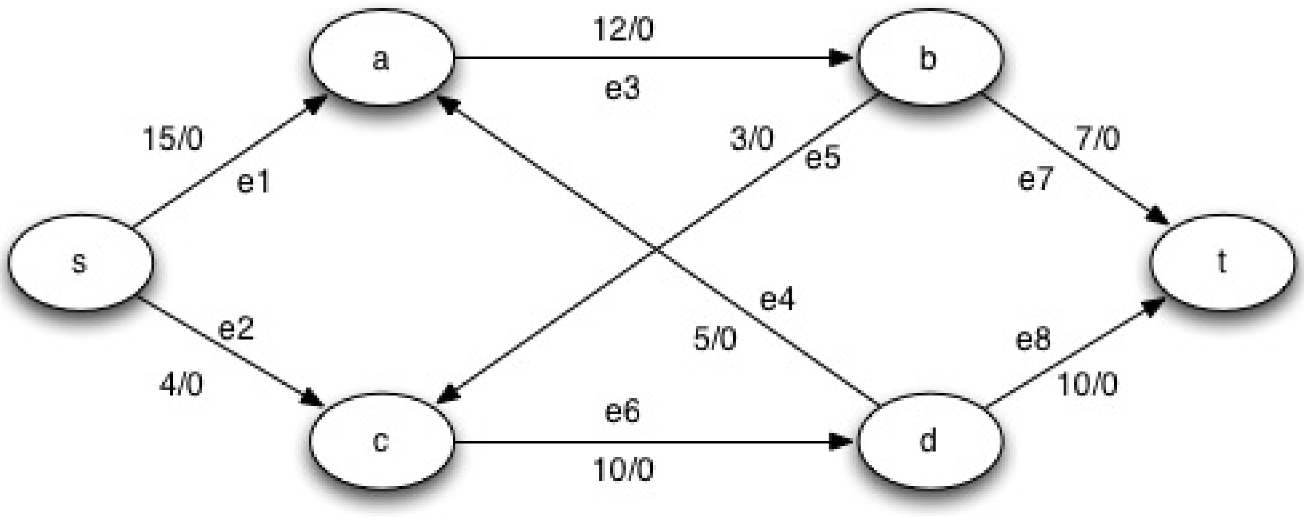
We use $e_4e_3e_2$ to improve the flow function as follows:

- For forward egdes $e : c(e) - f(e)$:
  - $e_4 : 6$
  - $e_2 : 10$
- For backward edges $e : f(e)$
  - $e_3 : 3$

We chose the minimum from the values and add the value to the flow of forward egdes and substract it from backward egdes. The flows of the edges change as follows:

- $e_4 = {}^{10}/_{7}$
- $e_2 = {}^{10}/_{3}$
- $e_3 = {}^{3}/_{0}$

Example 3.4



Augmenting path:
$s* \xrightarrow{e_2} c* \xrightarrow{e_6} d* \xrightarrow{e_4} a* \xrightarrow{e_3} b* \xrightarrow{e_7} t*$
Compute deltas:

- $\Delta_{(e_2)} = 4$
- $\Delta_{(e_6)} = 10$
- $\Delta_{(e_4)} = 5$
- $\Delta_{(e_3)} = 12$
- $\Delta_{(e_7)} = 7$

The minimum Δ is 4, so the flow of the edges will be increased by 4.

- $e_2 = {}^{4}/_{4}$
- $e_6 = {}^{10}/_{4}$
- $e_4 = {}^{5}/_{4}$
- $e_3 = {}^{12}/_{4}$
- $e_7 = {}^{7}/_{4}$

The next steps or paths would be:

- $s \to a \to b \to c \to d \to t$
- $s \to a \to b \to t$
- $s \to a \to d \to t$

The application of this paths leads to a new flow: $F = 14$.

3.1.1 The Algorithm of Ford Fulkerson

See handout for a description of the algorithm.

Lemma 3.3
When executing a step in the algorithm, the actual function f is a flow function.

Proof. The assumption is obviously true for step 1 because $f \equiv 0$ is a flow function. It is obviously true for steps 2, 3 and 5, too, because f is not modified.
Step 4:
Let f be a flow function when we enter step 4. We have to show that after performing step 4, the newly calculated function f is still a flow function.
Let f_{old} be the function with which we enter step 4 and f_{new} the newly calculated one. f_{old} is a flow function. Hence,

$$\sum_{e\in\alpha(v)} f_{old}(e) = \sum_{e\in\beta(v)} f_{old}(e); \forall v : v \neq s, v \neq t$$

Let $s \to v_0 \to v_1 ... v_{f_{e-1}} \to v_{f_e} \to t$ be an augmenting path used in step 4. By definition of $\Delta f_{new}(e) < c(e)$ and $f_{new}(e) > 0$.
For step 4: Let $s = v_0 \to ... \to v_2 = t$ be the path along which we achieved the marking. Only the flow value of the edges along this path is modified, so we have to check only the edges respectively nodes along this path. We have to check:

i) $0 \leq f_{new}(e) \leq c(e) \forall e$: e edge on the path

ii) $\sum_{e\in\alpha(v)} f_{new}(e) = \sum_{e\in\beta(v)} f_{new}(e), \forall v$, v on the path, $v \neq s, v \neg t$

The check:

i) 1 holds by the definition of Δ

ii) Let $v_i, v_i \neq s, v_i \neq t$ be a node on the path:

   a) $\xrightarrow{e_i} v_i \xrightarrow{e_{i+1}}, e_i \in \alpha(v_i), e_{i+1} \in \beta(v_i)$ for both edges f_{new} is obtained from f_{old} by adding Δ, so 2 holds in this case

b) $\xrightarrow{e_i} v_i \xleftarrow{e_{i+1}}, e_i, e_{i+1} \in \alpha(v_i)$. One contributes Δ, the other contributes $-\Delta$, so 2 holds for v_i

c) $\leftarrow v_i \leftarrow$ analogously

d) $\leftarrow v_i \rightarrow$ analogously, too

□

Lemma 3.4
If the algorithm by Ford-Fulkerson terminates then the determined flow function has maximal total flow.

Proof. If the algorithm terminates, then it does so in step 3. That means we started labeling but did not reach t. Let S be the set of nodes that were marked in the last round. Then $s \in S$ and $t \notin S$. $\bar{S} = V \setminus S$. Let $x \xrightarrow{e} y$ be an edge in $E_{S\bar{S}}$, this means x is labelled, y is not labelled. So, we know that $f(e) = c(e)$ because otherwise we could have marked y.
If $x \xrightarrow{e} y$ is an edge in $E_{\bar{S}S}$ $(x \in \bar{S}, y \in S)$. So, y is labelled and x is not. Then we can conclude that $f(e) = 0$, otherwise we could have marked e.

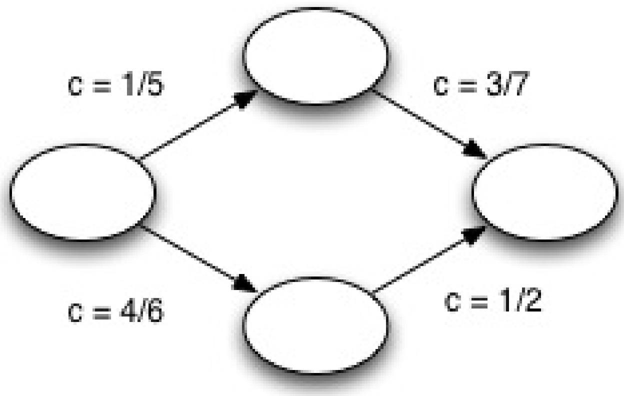
$$F \overset{(3.1)}{=} \sum_{e \in E_{S\bar{S}}} f(e) - \underbrace{\sum_{e \in E_{\bar{S}S}} f(e)}_{=0} = \sum_{e \in E_{S\bar{S}}} c(e) = c(e)$$

f is a flow function with maximal total flow. □

Termination

Lemma 3.5
If $c: E -> \mathbf{N}$ then the algorithm terminates.

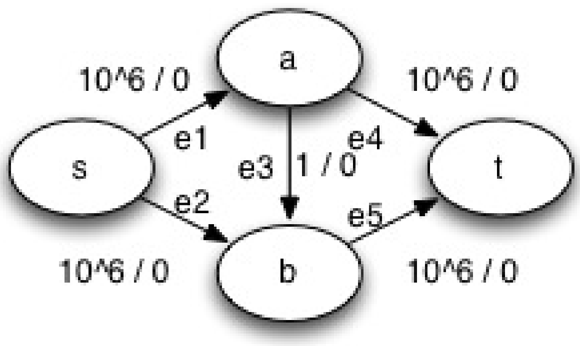
Proof. This holds because the algorithm starts with $f \equiv 0$ and the total flow F is increased by Δ and Δ is a natural number as $c(e) \in \mathbf{N} \forall e$ and because $F \leq c(s)$ for all S with $s \in S, t \notin S$ i.e. F is bordered.



What about $c: E -> \mathbf{R}$?
Turn the capacities into natural numbers, calculate the results and divide them later. □

Proposition 3.1
The example with $c: E -> \mathbf{R} \backslash \mathbf{Q}$ is an example, for which we can apply the algorithm in a way that is does not terminate.

Example 3.5



Maximal total flow: $2 * 10^6$
Selecting the augmenthing paths:

- $e_1 e_4$

- $e_2 e_5$

This would solve the problem within two steps, but...
Choosing the following augmenting paths:

- $e_1 e_3 e_5$ with $\Delta = 1$ and
- $e_2 e_3 e_4$ with $\Delta = 1$

would lead to $2 * 10^6$ rounds to solve the problem.

3.1.2 The Edmonds-Karp Algorithm

An algorithm which improves Ford Fulkerson by finding the shortest augmenting path and therefore reducing the runtime bound.

Theorem 3.1
If we use breadth-first-search when labelling and always select the shortest augmenting path, then the algorithm terminates and uses $O(|V|^3 * |E|)$ steps for any $c : E -> \mathbf{R}$ (where it is assumed that any real number can be manipulated in one step).

3.1.3 The Algorithm of Dinic

Definition 3.4
Let e be an edge between u and v with flow value $f(e)$. e is called useful from u to v if

i) $u \xrightarrow{e} v$ and $f(e) < c(e)$

ii) $u \leftarrow v$ and $f(e) > 0$

Let $G = (V, E)$ be the directed graph for the network and f a flow function for the network.
A layering for the network is defined as follows:

i) $V_0 = \{s\}, i \leftarrow 0$

ii) $T := \{v \in V, v \notin V_j, j \leq i$ and there is an useful edge from a node in V_i to v $\}$

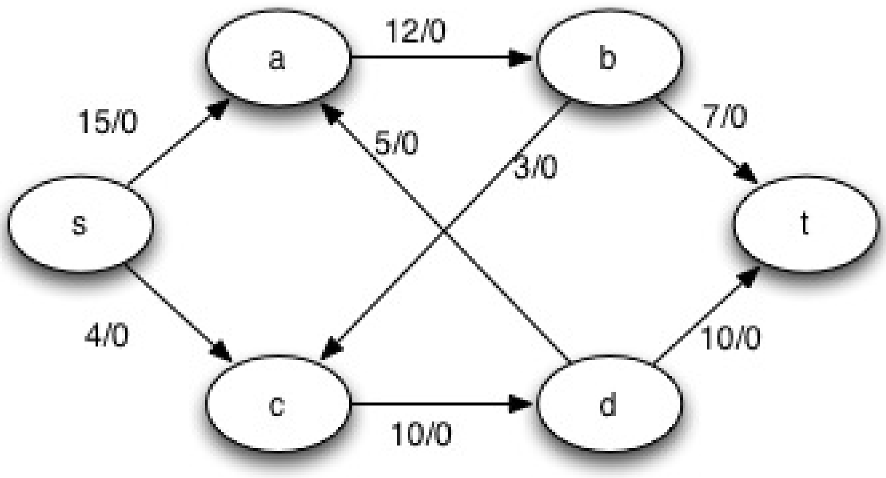
iii) If $T = \emptyset$ then the actual flow function has maximal total flow and the algorithm stops.

iv) If $t \in T$ then put $l := i+1, V_l = \{t\}$ and stop the algorithm.

v) $V_{i+1} := T, i \leftarrow i+1$ and go to step 2

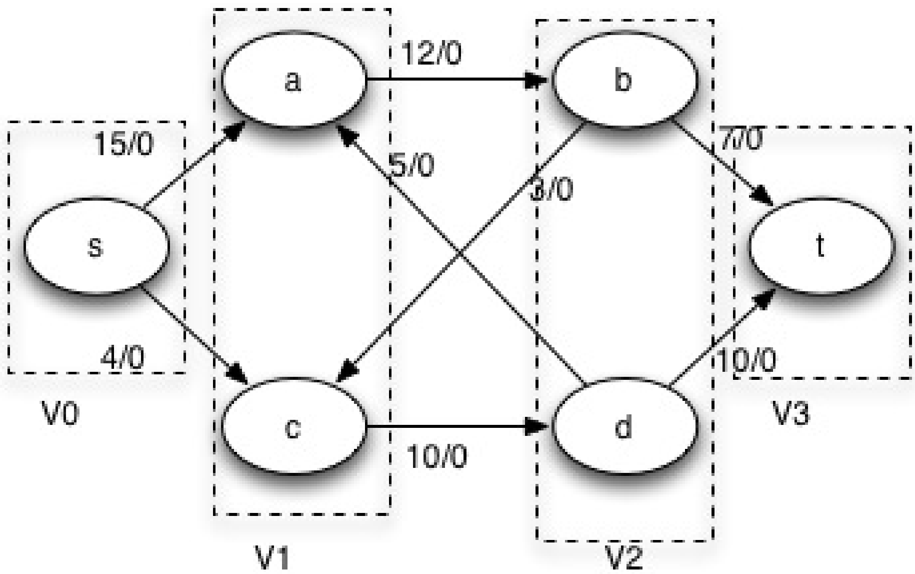
$E(i) = \{e$: e is an useful edge between some node in V_{n-1} and $V_i\}$

Example 3.6



Layers:

- $V_0 = \{s\}$
- $V_1 = \{a, c\}$
- $V_2 = \{b, d\}$
- $V_3 = \{t\}$



Theorem 3.2
When the algorithm for layering stops in step 3, then the actual flow function has maximal total flow.

Proof. Determine a set S with $F = c(S)$. What S?
$S = \bigcup_{i=0}^{i} V_j$, $\bar{S} = V \setminus S, s \in S, t \notin S$
For any edge $u \xrightarrow{e} v$, $e \in E_{S\bar{S}}$, we know $f(e) = c(e)$ because otherwise T would not be $\emptyset$. And for any edge $u \xleftarrow{e} v \in E_{\bar{S}S}$ we know $f(e) = 0$ and continue as for lemma 3.5. □

Definition 3.5
Associate capacities $\bar{c}$ to the edges in $E(i)$. Let c be an edge in $E(i)$, $u \xrightarrow{e} v$

i) if $u \in V_{i-1}, v \in V_i$, then $\bar{c}(e) := c(e) - f(e)$

ii) $u \in V_{i-1}, v \in V_i, u \xleftarrow{e} v$, then $\bar{c}(e) := f(e)$

Remark:
$\bar{c}(e) > 0$ in both cases as only useful edges are considered in $E(i)$

In the new network we look for a flow function $\bar{f}$ such that on any path from s to t

$$s - v_1 - v_2 - \dots - t, v_j \in V_j, e_j \in E_j$$

there is at least one edge e with $\bar{f}(e) = c(e)$.
Given such a function $\bar{f}$ (see handout: Dinic's algorithm) we modify the original flow function f_{old} as follows:

i) if $u \xrightarrow{e} v$ with $u \in V_{i-1}, v \in V_i$ then $f_{new}(e) := f_{old}(e) + \bar{f}(e)$

ii) if $u \xleftarrow{e} v$ with $u \in V_{i-1}, v \in V_i$ then $f_{new}(e) := f_{old}(e) - \bar{f}(e)$

Algorithm of Dinic:

i) Initialize with a flow function f, e.g. $f(e) \equiv 0 \forall e \in E$

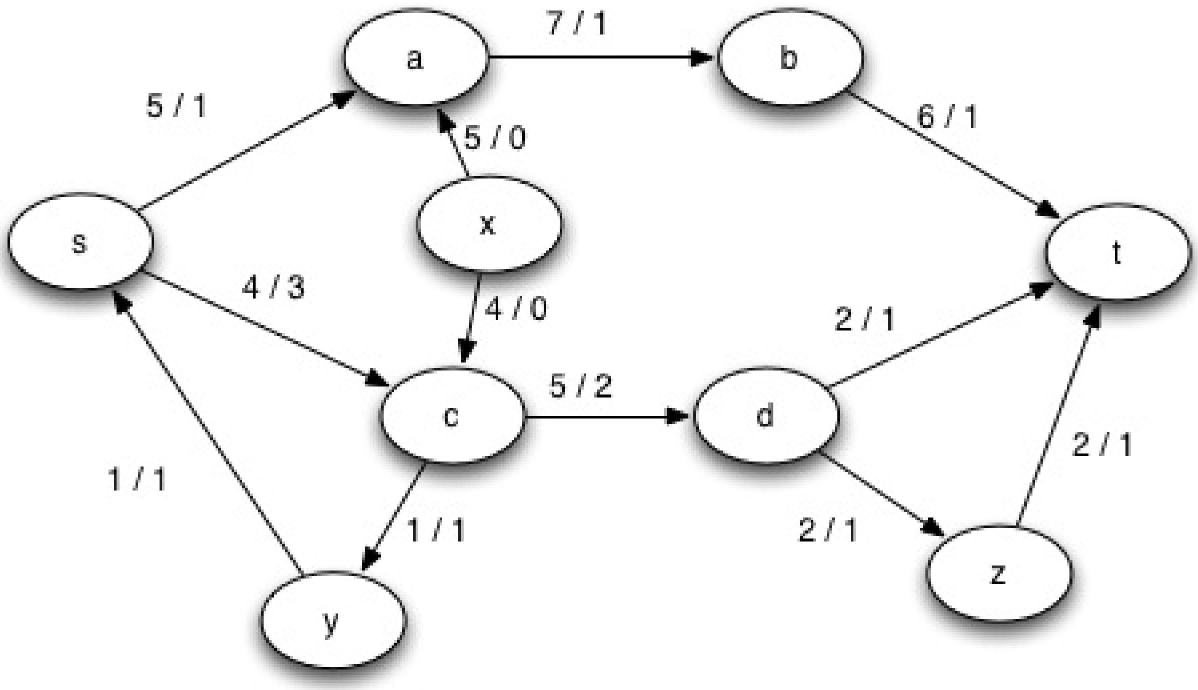
ii) Construct a layering with respect of f (remember: halt twice)

iii) Determine $\bar{f}$

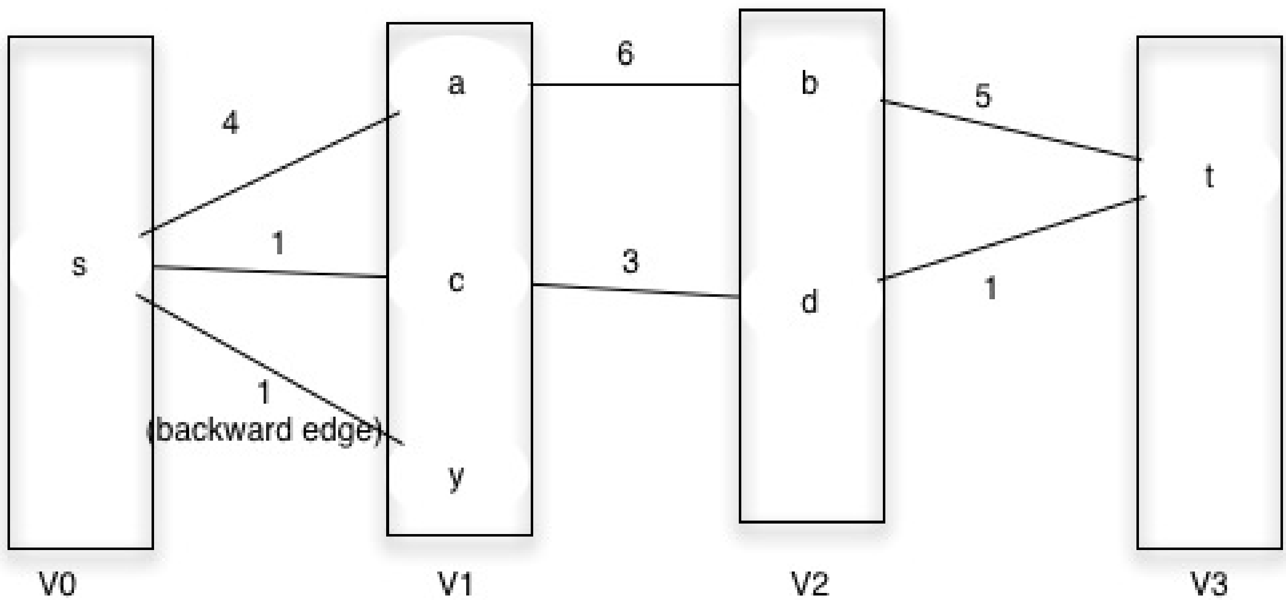
iv) From f and $\bar{f}$ determine the new flow function

v) Go to step 2

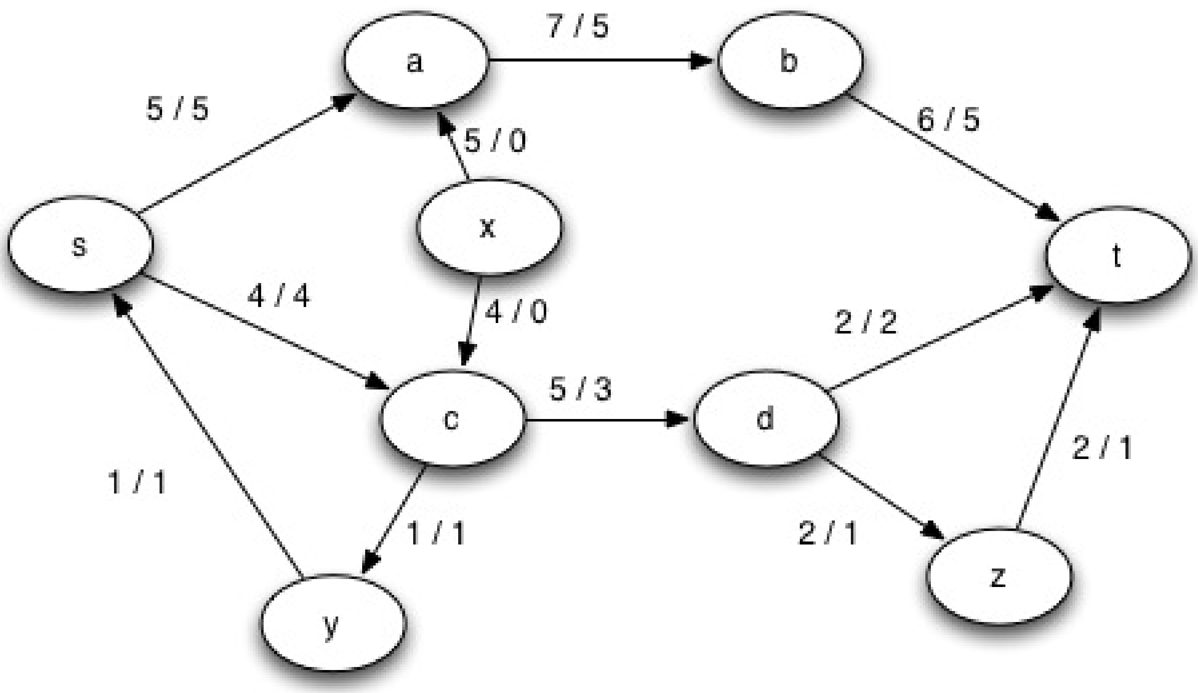
Example 3.7



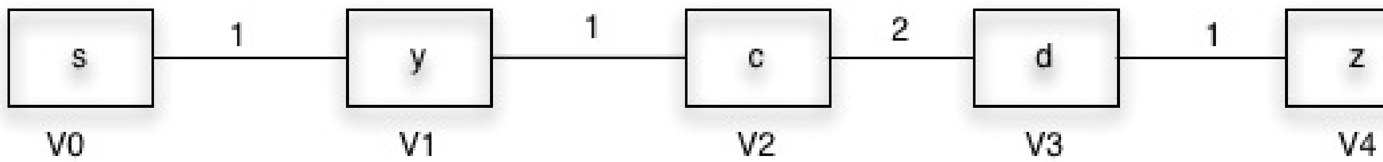
Given the example above the total flow of the network is $F = 3$. We will proceed with the first layering as follows:



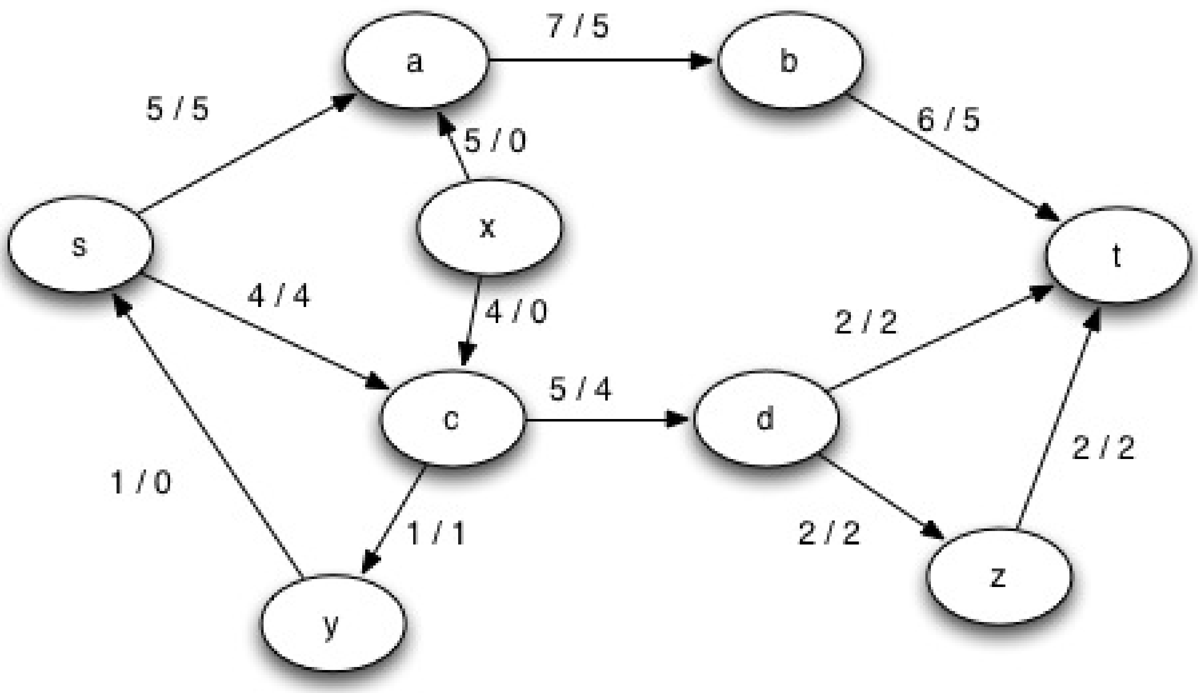
The nodes x, z disappeared while layering.
Determine $\bar{f}$: there are two paths from s to t, find $\bar{f}$ for each of them that saturates at least one edge on the path. The lower path yields a flow function that transports one unit from s to t. The upper path yields a flow function that transports four units from s to t.



After determining $\bar{f}$ we have to update the flows on our graph and gain a new total flow $F = 8$.
Second layering:



There is only one path from s to t. A flow function $\bar{f}$ that saturates at least one edge is the one that sends one unit along this path.



We update again the flows of the used edges and the total flow increases by one, so we have: $F = 9$.

Third layering

$V_0 = \{s\}$ can not be continued as there are no more useful edges from V_0.

Lemma 3.6
Let N be a network with flow function f. $\bar{f}$ is the flow function of the layered network. Then

i) the new calculated function f is a flow function and

ii) the new total flow is obtained by adding the old total flow and $\bar{F}(F_{old} + \bar{F} = F_{new})$.

Proof. For i) we have to show that $0 \le f_{new}(e) \le c(e) \forall e \in E$ and that the flow condition holds for every node.

- Show $0 \le f_{new}(e) \le c(e)$:
  i) Oberservation: $\forall e \notin \bigcup E(i)$ nothing has changed

ii) Let $e \in E(i)$, that is $u \xrightarrow{e} v, u \in V_{i-1}, v \in V_i$ and e is useful.

- Case 1:
  $u \xrightarrow{e} v$

$$\bar{f}(e) \leq \bar{c}(e) \overset{Def.}{=} c(e) - f(e)$$
$$f_{new}(e) = f(e) + \bar{f}(e) \leq f(e) - \bar{f}(e) = c(e)$$

  M: Bitte prÃ¼ft mal jemand den Teil mit f(e) - f(e)!

- Case 2:
  analogously

• Show $0 \leq f_{new}(e) \forall e \in E$:

i) Oberservation: all edges $\notin \bigcup E(i)$ are not affected

ii) Let $e \in E(i), u \xrightarrow{e} v, u \in V_{i-1}, v \in V_i$

- Case 1: $u \xrightarrow{e} v$

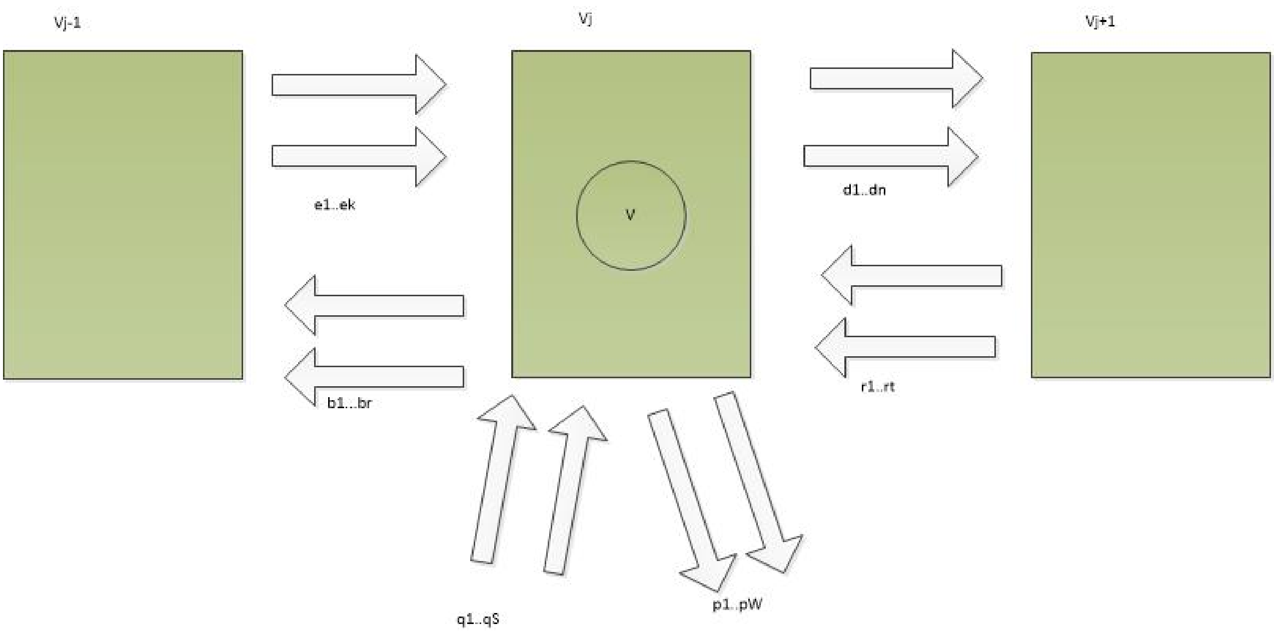
$$f_{new}(e) = f_{old}(e) + \bar{f}(e) \geq 0$$

- Case2: $u \leftarrow v$

$$f_{new}(e) = f_{old}(e) - \bar{f}(e) \geq f_{old}(e) - \bar{c}(e) \overset{Def. \bar{c}}{=} 0$$

<u>Flow condition:</u>

i) Oberservation: Nodes that do not appear in a layer are unaffected



ii) $\bar{f}$ is a flow function in the layered network.

$$a) \sum_{e \in E \cup B} \bar{f}(e) = \sum_{e \in D \cup R} \bar{f}(e)$$

Because f_{old} is a flow function, we know that
b) $\sum_{e \in E \cup Q \cup R} f_{old}(e) = \sum_{e \in B \cup F \cup D} f_{old}(e)$ (original direction of edges)
For f_{new} we get

$$\sum_{e \in \cup P \cup R} f_{new}(e) = \sum_{e \in E} f_{new}(e) + \sum_{e \in Q} f_{new}(e) + \sum_{e \in R} f_{new}(e)$$

$$= \sum_{e \in E} (f_{old}(e) + \bar{f}(e)) + \sum_{e \in Q} f_{old}(e) + \sum_{e \in R} (f_{old}(e) - \bar{f}(e))$$

$$\sum_{e \in B \cup P \cup D} f_{new}(e) = \sum_{e \in B} f_{new}(e) + \sum_{e \in P} f_{new}(e) + \sum_{e \in D} f_{new}(e)$$

$$= \sum_{e \in B} (f_{old}(e) + \bar{f}(e)) + \sum_{e \in D} (f_{old}(e) + \bar{f}(e)) + \sum_{e \in P} f_{old}(e)$$

Because of a) the $\bar{f}$ terms are compensated and because of b) the f_{old} termas are compensated.

□

Lemma 3.7
Let l_k be the index of the last layer (the one which contains t) in the k th layering step. Then $l_{k+1} > l_k$.

Proof. When we reached the (k+1)-layering (where we assume that it's not the last layering), then there is a part in the layered network $s = v_0 \xrightarrow{e_1} v_1 \xrightarrow{e_2} \ldots \xrightarrow{e_{l_{k+1}}} v_{l_{k+1}} = t,\ v_i \in V_i^{k+1}$ (i th layer in the (k+1)-layering).

case 1:
All nodes of this path appeared already in the k-th layering step. $v_i \in \bigcup_{j=1}^{l_k} V_j^{(k)},\ i = 0, \ldots, l_{k+1}$.

- Claim: if $v_a \in V_b^{(b)}$ then $a \geq b$
  Proof by induction on a
- Ing stduction: $a = 0$, $a = v_0 \in V_0^{(k)} = ba \geq b$
- Induction assumption: let $v_a \in V_b^{(k)}$ then $a \geq b$
- Induction step: $a \to a+1$, so let $v_a \in V_b^{(b)}$

Let $v_{a+1} = V_c^{(k)}$, show that $a + 1 \geq c$.

i) $c \leq b + 1$
   $v_{a+1} \in V_c^{(k)}$, by induction assumption $a \geq b$ then $a + 1 \geq b + 1 = c$

ii) $c > b+1$

$$V_b^{(k)} V_{b+1}^{(k)} V_c^{(k)}$$

This would mean that the edge l_{a+1}, that is useful for (k+1)layering was not used in the k-th layering, but it was already useful at that time, hence this case can not happen.
In particular we know:

$$t = v_{l_{k+1}} \in V_{l_k}^{(k)}, hence l_{k+1} \geq l_k$$

The equality symbol in $l_{k+1} \geq l_k$ can not occur because otherwise the whole path $v_0 \to v_1 ... v_{l_{k+1}}$ would have existed in the k-th layering. But when we determine the function$\bar{f}$ we dit it in a way that at least one edge e of the path is saturated. This edge e is no longer useful for the next layering.

case 2:
$\to$ Not all v_i are contained in the case layering. Proof is analogously to case 1. □

Korollar 3.2
The repeated part of Dimic's algorithm

i) Initialize f, e.g. $f \equiv 0$

ii) Determine a layering

iii) Determine $\bar{f}$

iv) Calculate $f = f_{old} + \bar{f}$

v) Go to step 1

is performed at most $|V| - 1$ times.

Proof. As $l_k < l_{k+1}$ and each layer contains at least one node. □

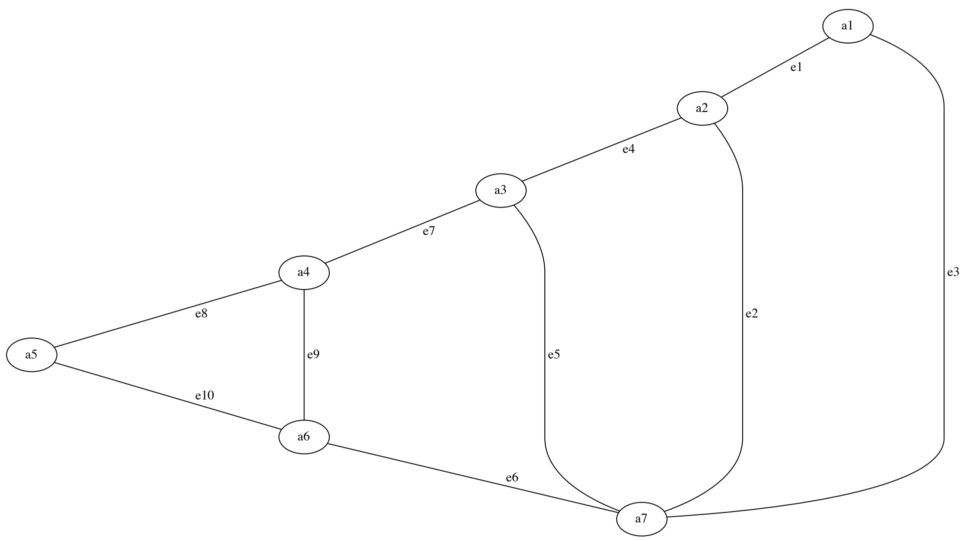
4
Matching

Matchings in undirected graphs without parallel edges and loops

4.1 Elementary Definitions

Definition 4.1
A matching in an undirected (finite) graph $G = (V, E)$ is a set $M \subseteq E$ such that no two edges share a node. A matching M is called *maximal* if, for any matching M' with $M \subseteq M'$, $M = M'$ holds. M is of maximum cardinality if $|M| \geq |M'|$ for any matching M'. M is *perfect* if every node is incident on one edge.

Example



Remark: Perfect matching not possible with $|V|$ being odd. Maximum cardinality = 3. Empty set is a matching. All subsets of matchings are matchings.
Maximum cardinality: $\{e_3, e_1, e_9\}$ $\{e_{10}, e_5, e_1\}$
Maximal matching: $\{e_2, e_9\}$

Lemma 4.1
If M is a matching of maximum cardinality it is a maximal matching.

Proof. Let M' be a matching with $M \subseteq M'$, assume $M \subset M'$ then $|M'| > |M|$ $\lightning$(as M has maximum cardinality). □

Remark: There are maximum matchings without maximum cardinality.

Lemma 4.2
If M is perfect then $2|M| = |V|$.

Proof. Obvious. □

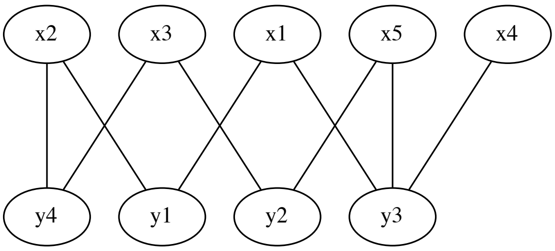
Lemma 4.3
If a graph has a perfect matching M then every matching of maximum cardinality is perfect. Clearly, M is of maximum cardinality.

Proof. Obvious. □

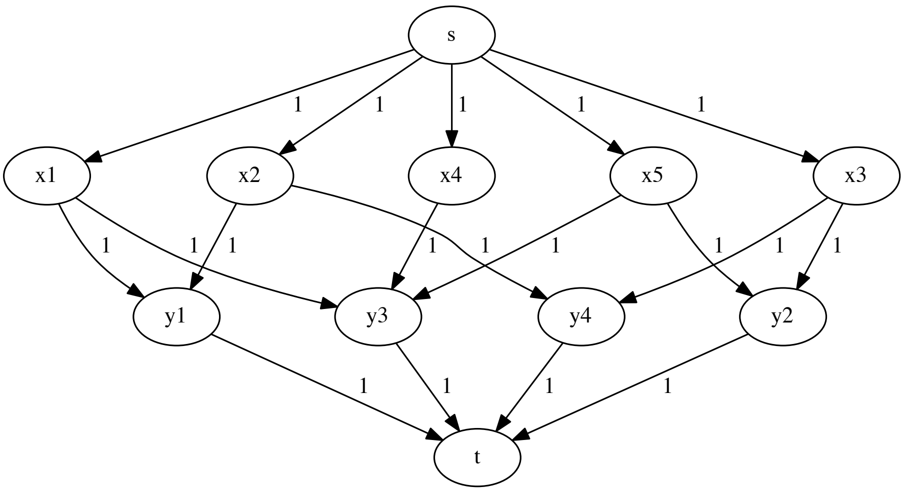
4.2 Matching in bipartite graphs

Problem: Find a matching of maximum cardinality in a bipartite graph.
Solution: Construct an associated network as follows:
Let $G = (V, E)$ in a bipartite graph.
$V = X \cup Y$
$\overline{V} = V \cup \{s, t\}$
$X \cap Y = \emptyset$
$\overline{E} = \{s \to x : x \in X\} \cup \{y \to t : y \in Y\}$
$\overline{c}(\overline{e}) = 1, \forall \overline{e} \in \overline{E}$

Example



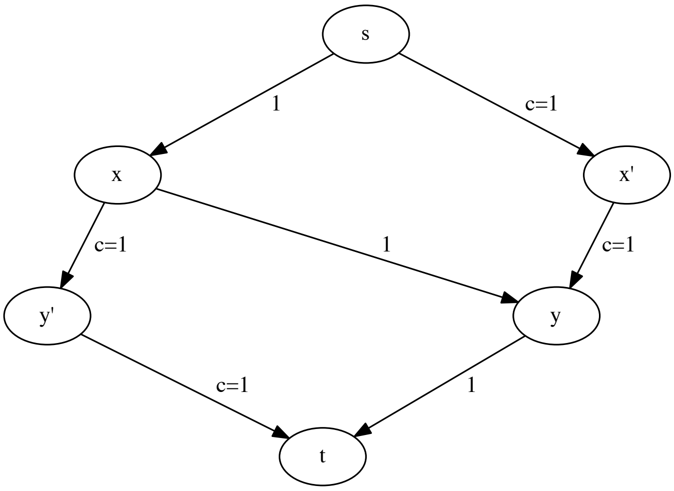
One solution:



UNKLARHEIT IN NOTIZEN

Theorem 4.1
The number of edges in a matching M of maximum cardinality coincides with F_{max}, i.e. the maximum total flow in the associated network.

Proof. Let M be a matching of maximum cardinality. For every edge $x - y$ in M we transport one unit along the path $s - x - y - t$. We can do that as the edges in M are node disjoint, i.e. they do not share nodes. This defines a flow function f' with $F' = |M|$ and hence $F_{max} \geq F' = |M|$. Now let f be an arbitrary flow function for the associated network, without loss of generality we may choose f such that $f(e) \in \mathbb{N}\, \forall e$. All paths that connect s with t have the form $s \to x \to y \to t$. If such a path is used to transport a unit value then it is clear that no flow can happen along edges of the form $x \to y'$ or $x' \to y$.



Let $N = \{x - y : f(x \to y) = 1\}$. N is a matching and the total flow of f, i.e. F, satisfies $F = |N| \leq |M|$, hence also F_{max}, because M was assumed to be of maximum cardinality. Hence $|M| = F_{max}$ because we chose f arbitrarily. $\square$

This theorem yields an algorithm to determine a matching of maximum cardinality. Given a bipartite graph $G = (V, E)$ with $V = X\hat{A} \cup Y$ and $X\hat{A} \cap Y = \emptyset$:

i) Construct the associated network.

ii) On this network, calculate a flow function with maximum total flow.

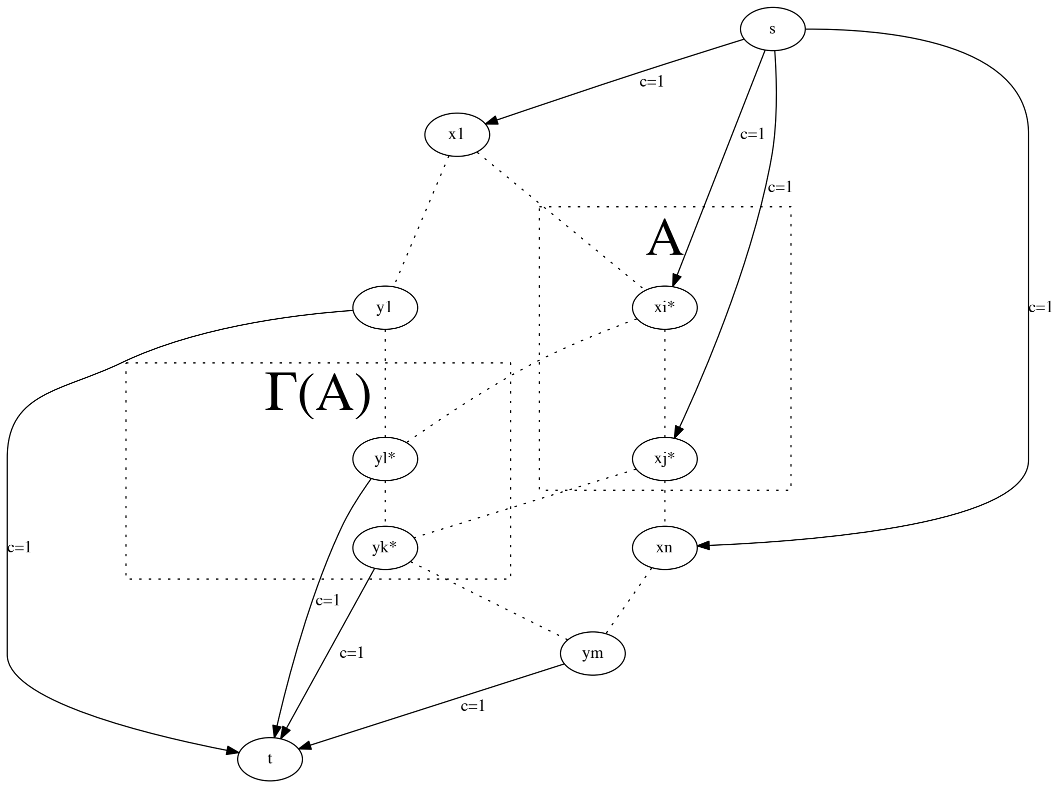
iii) The matching of maximum cardinality is obtained by $N = \{x - y : f(x \to y) = 1\}$.

Definition 4.2
Let G be a bipartite graph, $V = X \cup Y; X \cap Y = \emptyset$. Let $A \subseteq X$. Let $\Gamma(A) = \{y \in Y : \exists x \in A : x - y \in E\}$. ($\Gamma(A)$: potential partners for the elements in A). A matching M is called *complete* if $|M| = |X|$, i.e. every $x \in X$ gets a partner.

Theorem 4.2
Wedding or marriage theorem: A bipartite graph has a complete matching if and only if for every $A \subseteq X$ $|\Gamma(A)| \geq |A|$.

Proof. ">" Let G have a complete matching M, let $A \subseteq X$. Then every $x \in X$ has a partner for itself in M, hence for every $A \subseteq X |\Gamma(A)| \geq |A|$.
"<" Let $|\Gamma(A)| \geq |A|$ for every $A \subseteq X$. Assume there is no complete matching. Let S be the set

of nodes that were marked in the last step of Ford-Fulkerson in the associated network. The calculated flow function determines a matching M with i) $F = |M|$. As we assumed that there is no complete matching we get ii) $|M| < |X|$.



Marking in the last round of Ford-Fulkerson. S = the marked nodes.
$A := X \cap S$
Let $y \in \Gamma(A)$ then there is an $x \in A = X \cap S$ x marked and $x - y$ is an edge in E. <- NOTIZEN UNKLAR
In the last successfull construction of an augmenting path there was no flow determined along the edge $s \to x$, otherwise we could not mark x now. Hence the flow along the edge $x \to y$ must have been 0 before as well. Hence the remaining capacity $x \to y = 1$.

4.3 Stable matching (marriage)

Given:
1. n men $b_1 \dots b_n$, n women $a_1 \dots a_n$
2. each person has a preferene list of the other sex:
$a_1 : b_2, b_3, b_1 \dots$, where a_1 likes b_2 best. □

5
Networks with costs, respectively upper/lower bounds

5.1 Networks with costs

Get rid of parallel and antiparallel edges, cost: $E \to \mathbb{R}$

Example 5.1
case a)

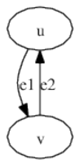


Figure 5.1: With antiparallel edges

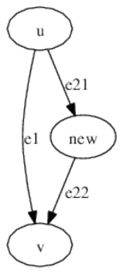


Figure 5.2: With antiparallel edges

?? is transformed to **??**
$c(e_1)$ as before c.
$cost(e_21) = cost(e_2)$
$cost(e_22) = 0$

$cost(e_2 1) = cost(e_2 2) = cost(e_2)$

case b)

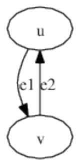


Figure 5.3: With antiparallel edges

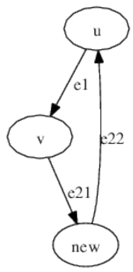


Figure 5.4: After transformation - without antiparallel edges

?? is transformed to **??**

From now on we consider only networks without parallel/antiparallel edges.

Definition 5.1
Let $G = (V, E)$ with $s, t \in V, c : E \to \mathbb{R}^+$ be a network. Let cost: $E \to \mathbb{R}$ be a function that associates "cost" to every edge. Let f be a flow function for this network.

$$cost(f) = \sum_{e \in E} f(e) * cost(e)$$

Task:
Given:

i) a network $G = (V, E), s, t \in V, c, cost$

ii) $w \in \mathbb{R}^+$

Find a flow function f with total flow $F = w$ and minimal costs.

Definition 5.2
$G=(V,E), s,t\in V, c, cost, f flowfunction$
The graph $G_f=(V,E_f)$ with $\tilde{c}$, $\tilde{cost}s$.
Let $e=(u,v)\in E$

i) If $f(e)<c(e)$ then $e_1=(u,v)\in E_f$
$\tilde{c}(e_1)=c(e)-f(e)>0$
$\tilde{cost}(e_1)=cost(e)$

ii) If $0<f(e)$ then $e_2=(v,u)\in E_f$
$\tilde{c}(e_2)=f(e)$
$\tilde{cost}(e_2)=-cost(e)$

Observation:

a If $e\in E$ is an edge with $0<f(e)<c(e)$

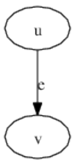


Figure 5.5: $G=(V,E)$

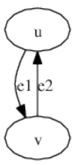


Figure 5.6: G_f

b If $e\in E$ is an edge with $0<f(e)=c(e)$ - as shown in **??**.

c If $e\in E$ is an edge with $0=f(e)<c(e)$ - as shown in **??**.

d If $e\in E$ is an edge with $0=f(0)=c(0)$ - as shown in **??**.

Definition 5.3
Let p be a directed cycle in G_f. $cost(p):=\sum_{e\in P} cost(e)$

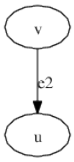


Figure 5.7: G_f

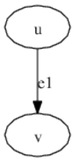


Figure 5.8: G_f

Example 5.2
directed cycles of negative costs: $t-h-b-d-t \rightarrow -1$
$s-c-g-h-b-a-s \rightarrow -5$

Theorem 5.1
Let a network with cost function be given and a flow function f with total flow $F = w$. f has least costs among all flow functions with total flow w if and only if G_f does not have any directed cycles of negative cost.

Proof. "$\Longrightarrow$" Let f be a cost minimal flow function with total flow $F = w$. Assume that G_f contains a directed cycle of negative costs. Adapting the flow values along this cycle in the original network we can reduce the cost while maintaining the total flow. □

Example 5.3
For edges that correspond to forward edges (of type e_1) we may raise the flow value. For edges that correspond to backward edges (of type e_1) we may reduce the flow.

Lemma 5.1
Let f be flow function with total flow $F = w$ and let f be cost minimal. Let p be an augmenting path that is cost minimal then the resulting flow f' constructed from f and p is cost minimal for $w + \Delta$ (Δ from the augmenting path).

Figure 5.9: G_f

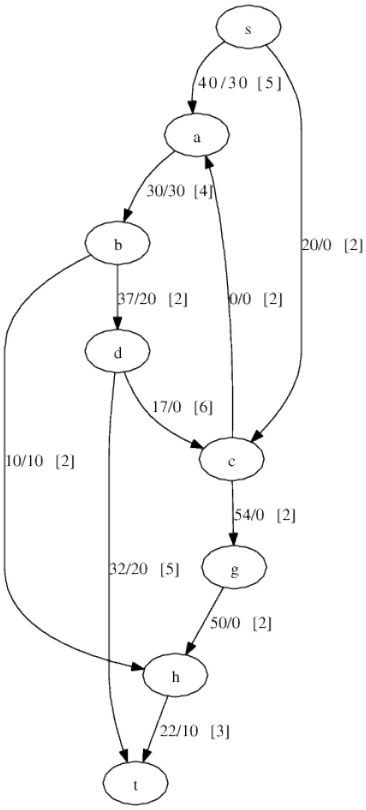


Figure 5.10: G

5.2 Networks with upper and lower bounds

we associate with each edge e a lower bound $b(e)$ and request for the flow function additionally that:
$b(e) <= f(e)$
A flow function obeying this additional constraint is called <u>legal flow function</u>.

Example 5.4
Graphic of no legal flow missing

To answer the question if there is a legal flow for a network with lower/upper bounds we proceed on follows. Construct the Graph $\bar{G} = (\bar{V}, \bar{E})$.

i) $\bar{V} = V \cup \{\bar{s}, \bar{t}\}$

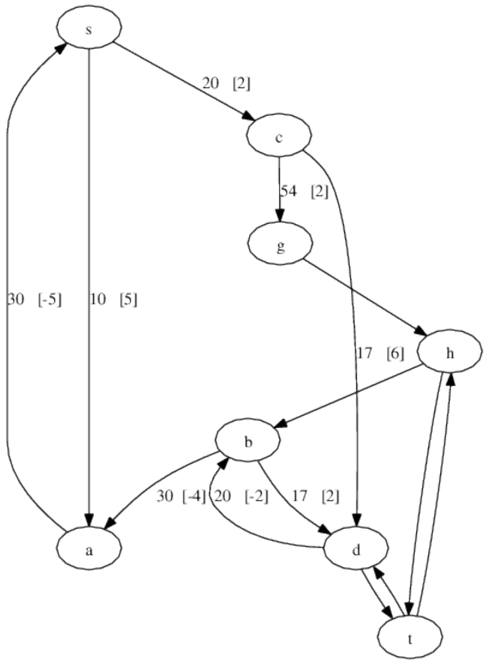


Figure 5.11: G_f

ii) for every node $v \in V$ introduce an edge e from v to $\bar{t}$.
$\bar{c}(e) = \sum_{e \in \beta(v)} b(e)$
$\bar{b}(e) = 0$

iii) for every node $v \in V$ introduce an edge e from $\bar{s}$ to v.
$\bar{c}(e) = \sum_{e \in \alpha(v)} b(e)$
$\bar{b}(e) = 0$

iv) The edges in E remain but with new bunds:
$\bar{c}(e) = c(e) - b(e)$
$\bar{b}(e) = 0$

v) Introduce edges from $s \xrightarrow{e} t$, $t \xrightarrow{e'} s$ with:
$\bar{c}(e) = \bar{c}(e') = \infty$
$\bar{b}(e) = \bar{b}(e') = 0$

BarE consists of the edges in E plus the newly introduced edges.

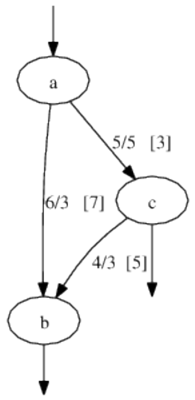


Figure 5.12: G_f

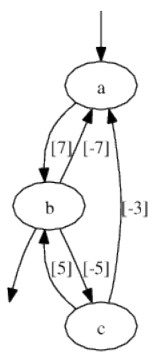


Figure 5.13: G_f

Lemma 5.2
The original network with upper and lower bounds has a legal flow if and only if the maximum flow of the auiliary network saturates all edges emanating from $\bar{s}$.

Hint:
Let $Barf$ be a flow function for the associated network with maximal total flow then $f(e) = Barf(e) + b(e)$ is a legal flow function ($e \in E$).

6
NP-Completeness

Definition and analyisis of different complexity classes

6.0.1 Motivation/Examples

Solvable in Polinomial Time	NP complete
Euler Path "a path with all edges" Shortest path from X to Y ⇒ Dijkstra'salgorithm	Hamiltonian Path "node ocurs exactly once" longest simple Path between X and Y
2-CNF (conjuntive normal form) e.g. $(x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_3)$ Can we find an assignment, that the formula becomes true in poly. time?	3-CNF
Two processor scheduling	3 or more processor schedulung ⇒ NP-complete or unknown
Football game until 1955 loss 0 points, draw 1 point, win 3 points. The season has already started, is my team still able to win the championchip?	loss 0 points, draw 1 point, win 3 points

6.0.2 Introduction

Definition 6.1
P is the class of problems that can be solved in polinomial time.

NP is the class of porblems, where given a solution one may check in polinomial time, that it is indeed one.

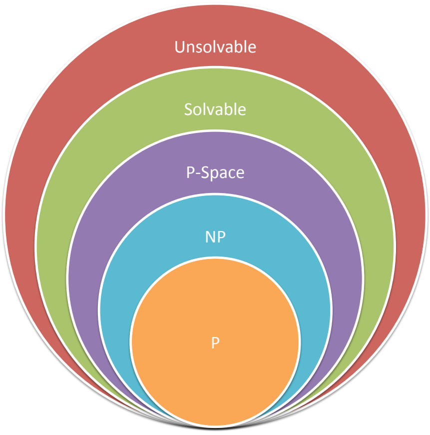
Example 6.1
Example:

i) Given a connected undirected-finite graph. Is there a circle such taht every node appears exatly once on the circle. Given such a circle $< v_1, \ldots v_n >$ it can easily checked that it has the property. $\Rightarrow$ Hamiltonian Cycle $\in$ NP.

ii) 3-CNF; given values for the variables, we can check that the formula becomes true, with this values in polynomial time.

Clarity: $P \subseteq NP$ open problem; $P = NP$??

Overview:



6.0.3 Problem Types/Issues

Decision Problems	**Optimization Problems**
Yes or no answer, is there a path from X to Y?	Find the shortest path from X to Y?

Definition 6.2
Complexity Theory deals with decision problems. Optimization problems are tried to transform into decision problems.

Example 6.2
Choose an integer k, treat the decision problem: "Is there a path from X to Y no longer than k?"

To show that an optimization problem is hard to solve it is sufficient to consider the related decision problem. If the latter is hard to solve, so is the first.

Reductions
Reductions are used to show that problem are hard to solve. A reduction reduces a decision problem to another one.
$A \Rightarrow B$ (reduction)

Properties of the reduction should be: "it can be done in polynomial time".
If A and B the transformation of AB is an instance of B then A should have the answer yes if and only if β has the answer yes.
Call this a polynomial reduction.
If $A \rightarrow B$ and B can be solved in poly. time, then instances of A can be solved in poly. time.
If $A \rightarrow B$ and A is hard to solve, then we may conclude that B is hard to solve.

Lecture of 08.05.13 - beginning: redundancy to former parts?

Consider the following classes: P, NP, NPC
Class $A \xrightarrow{poly} B$
Instance $\alpha \rightarrow \beta$

1 Observation: If the instances of the problem B can be solved in poly time, then this is true for instances of A.

2 Observation: If the instances of A are "hard" to solve, so are the instances of B.

3 Observation: If A is in NPC and $B \in NP$, then we can conclude B is also in NPC.

Problem: To find a first problem class in NPC (St. Cook 1971 3-SAT problem)

6.1 Polynomial Time

1. Problem solvable in $O(n^100)$?
   Speed-up theorem of Blum → reduce export
2. Polynomial time is independent of the computer model
3. The class of polynomials is closed with respect to addition, multiplication and compostion.
4. Practical reasons.

Definition 6.3 **Abstract Decision Problem**
An Abstract Decision Problem is a mapping $\mathbb{Q} : I \to \{0,1\}$. I: is a set of problem instances

Example 6.3
Is there a path from u to v in the directed graph G of length ≤ k $I = \{(G,u,v,k) : k \in \mathbb{N}, G$ finite directed Graph, u,v are nodes in G }
$Q((G,u,v,k)) = 1$ if ∃ path ... and $Q((G,u,v,k)) = 0$ otherwise.

6.1.1 Conding and concrete problems

A coding of as set S is a mapping $e : S \to \{0,1\}$.
A concrete problem is a problem, the instances of which are in { 0,1 }.
$Q : S' \leq \{0,1\} \to \{0,1\}$
An algorithm solves a concrete problem in $O(T(n))$ if it delivers the answer for an input x with $|x| = n$ on $O(T(n))$ steps.
A concrete problem is said to be solvable in polynomial time if there is an algorithm solving it in $O(n^k)$ for some k.
P is the class of concrete problems that can be solved in poly time.
Let $Q : I \to \{0,1\}$ be an abstract problem and $e : I \to \{0,1\}^*$ be an encoding.
The resulting concrete problem $Qe : e(I) \to \{0,1\}$ such that $Q(i) = 1$ iff $Qe(e(i)) = 1 \subseteq \{0,1\}^*$.

Example 6.4
Consider an algorithm for an abstract problem an $\mathbb{N}$ that needs $\Theta(k)$ steps for input $k \in \mathbb{N}$.
encoding e: $\mathbb{N} \to \{0,1\}$ $e(k) = 11111$ (k times), then the algorithm is linear in the length of the input.
encoding e': $e'(k)$ =binary representation of k length$e'(k) = \left|e'(k)\right| = \log_2 k + 1$ (log hat eigentlich so eckige klammern)

The algorithm with this encoding produces costs exponential in the length $\left|e'(k)\right|$ of the input $f : \{0,1\}* \to \{0,1\}*$ is said to be compatible in poly time if there is an algorithm that computes $f(x), x \in \{0,1\}*$ in poly time.

Two encodings are called polinomically connected if there are two functions f_{12} and f_{21} with $f_{12}(e_1(i)) = e_2(i), f_{21}(e_2(i)) = e_1(i), i \in I$ that are polinomially computable.

Lemma 6.1
Let Q be an abstract decision problem.

$Q: I \to \{0,1\}$
$e_1, e_2 : I \to \{0,1\}*$ encodings that are polynomially connected, then $Q_{e_1} \in P$ iff $Q_{e_2} \in P$

Proof. Steps:

i) $Q_{e_1} : e_1(I) \subseteq \{0,1\}* \to \{0,1\}$ be solvable in $O(n^k)$ for some k for input of length n

ii) Let $e_1(i)$ be computable from $e_2(i)$ in $O(n^c)$ where $n = |e_2(i)|$.
We want to show, that $Q_{e_2} \in P$ to solve Q_{e_2} we proceed as follows:
$Q_{e_2} : e_2(I) \to \{0,1\}$ consider $e_2(i)$. First determine $e_1(i)$ from $e_2(i)$ and then run the algorithm for Q_{e_1} on the input $e_1(i)$.
The first step costs $O(n^c), n = |e_2(i)|$ and the result is $e_1(i)$.
In th second step the algorithm for Q_{e_1} is run on $e_1(i), |e_1(i)| = O(|e_2(i)|^c)$.
It needs $O((n^c)^k) = O(n^{ck})$

Sidenote: length $\to |e_1(i)| \leq O(n^2) \to P \leq PSPACE$

□

6.2 Formal Language Representation

Let ϵ be an alphabet (=finite set, the elements are called symbols).
A language over ϵ is a subset of ϵ^*, ϵ^* all finite sequences of over ϵ.
$L \leq \epsilon^*$ (Grammar density?)

Appendix 4: Application of general matching

6.3 Two processor scheduling

Given:

i) Two identical agents/processors

ii) Collection of n jobs or taskstogether with a directed acyclic graph with n nodes which correspond to the jobs, that describe the preceedence among the jobs.

Example 6.5
2 Graphics missing

How can the jobs be scheduled on the two processors such that the jobs are completed as quickly as possible?

Solution:
using matching!
But:
for three processors the problem is NP-Complete.

To solve the two processors problem we proceed as follows. We construct a graph $G*$ thathas the same nodes as G and there is an edge x,y in $G*$ if and only if there is no path from x to y in G and no path from y to x in G.

1. Observation:
if we have a schedule we can obtain a matching. If the schedule is optimal the matching is of maximal cardinality.

2. Interessting:
We can use matching of maximal cardinality to produce a schedule that is optimal.

Let S be a set of nodes with indegree 0. Let $M*$ be a matching of max. cardinality for $G*$. Apply the following rules repeatly:

i) If there is an unmatched node in S then schedule it. Remove it from G.

ii) If there is a pair of jobs in S that are unmatched in $M*$then we schedule this pair in the output and delete the two nodes (corresponding to the jobs) in G.

If neither rule 1 nor rule 2 applies and there are still jobs left to be scheduled then we know that S contains only nodes thatare matched and for every node in S its partner ($M*$) is not in S.

$J_1 \in S$ ——$M*$—— $J'_1 \notin S$

$J_2 \in S$ ——$M*$—— $J'_2 \notin S$

Observation:
$\nexists$ path J_1 to J_2