various

Lecture Notes of Spring 2013

Algorithms I

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

The accuracy of its content is not guaranteed and the author(s) do not assume responsibilitty for possible damages of any kind. This lecture notes are not an official document released by employees of the University of Mannheim, hence those do not assume responsibility, as well.

Many thanks to Ingo Bürk, who initially published the underlying LaTeX template here. This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 3.0 Germany License.

Contents

1

TODO: 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 $0 < |i(e)| \leq 2$.
> 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 $|i(e)| = 1$ 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
TODO: Graphic missing!

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

Proof: $\sum\limits_{i=1}^{n} degree(v_i) = 2 * |E|$

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 $e_1, e_2, \ldots$ 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!]
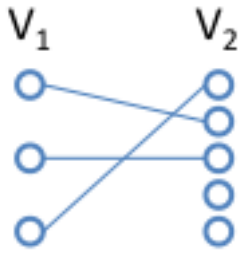
Definition 1.3
TODO

Lemma 1.2
TODO

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.
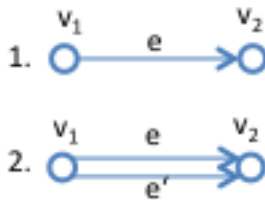
Graphically:

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.

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

> **Lemma** 1.3
> $$\sum_{v \in V} g_{in}(v) = \sum_{v \in V} g_{out}(v)$$

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) <u>cycle</u>, 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 <u>simple</u>, if it does not contain parallel edges.

Definition 1.8
A directed graph is called <u>strongly connected</u> 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 $\sim$ mean are "connected"?) if there is a directed path from x to y and vice versa.
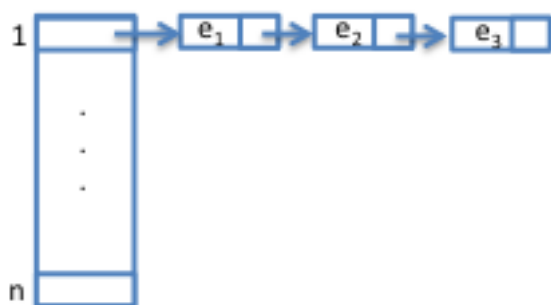
The equivalence classes of this relation $cV xV$ are called strongly connected components. (Analogously: Define connected components for undirected graphs)

ℹ️ 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 $\equiv$ transactions of user; Edges $\equiv$ waiting situations

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

> **Lemma** 1.5
>
> 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.
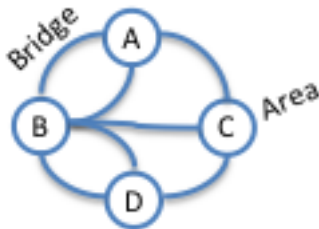
2

Euler Graphs and Hamilton Graphs

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.
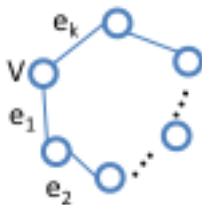
Hence every node has even degree.

" $<$ " 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. $a \circ \to \circ \to ... \circ b$
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

numbers represent the degree of each edge

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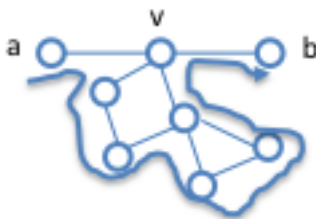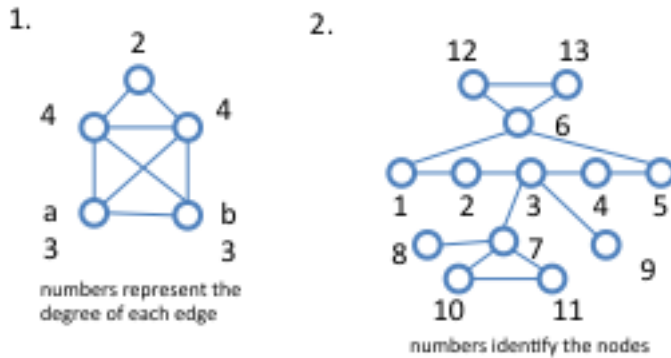
In the directed case a directed Euler path is a directed path on which every edge appears exactly once. Directed Euler cycle analogously.

> **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

1.

2.

3.

is hamiltonian!   is hamiltonian!   not hamiltonian!

The problem, given an arbitrary undirected graph: Is it Hamiltonian? $\Rightarrow$ NP complete $\Rightarrow$ 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 $x, y \in V \ with \ x \neq y$ 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

K_3 K_4 K_5

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.
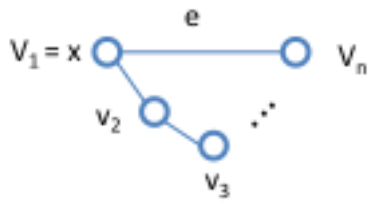
$v_i \neq v_j \, for \, i \neq j$

$S = \{v_i : 1 \leq i \leq n$ 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) $|S \cup T| < |V| = n$
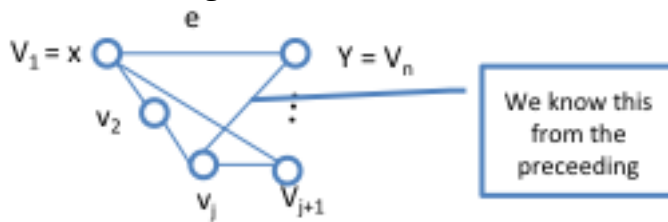
  iii) $deg(x) = |S|$
       $deg(y) = |T|$

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.
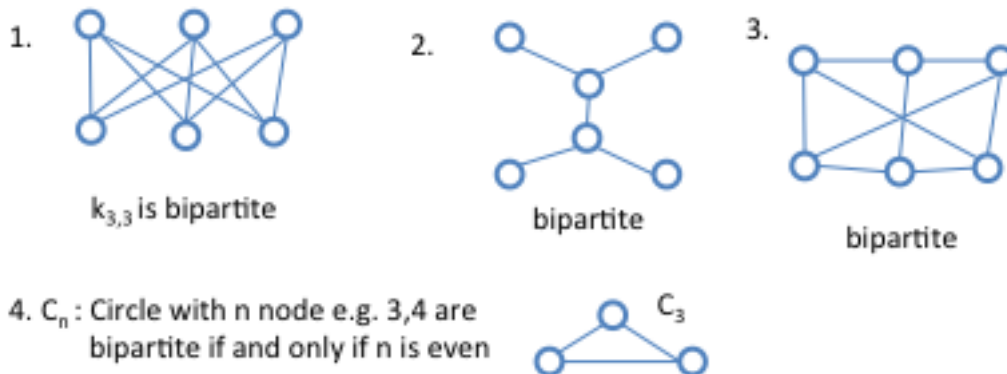Now remove edge e and there is a Hamiltonian left.
Cost of checking the condition $O(M^2)|E| \leq |V|^2$

2.3 Bipartite Graphs

Example

1. $k_{3,3}$ is bipartite

2. bipartite

3. bipartite

4. C_n : Circle with n node e.g. 3,4 are bipartite if and only if n is even

C_3

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$\}$
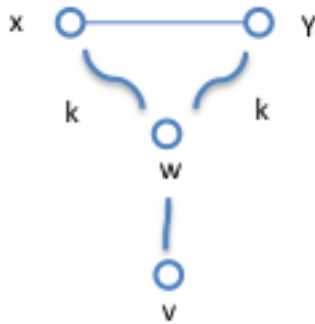$V \in V_2, V = V_1 \dot\cup V_2$ (disjoint union)
Claim: Ther is no edge e with both endpoints in V_1 respectively V_2. Assume there is an edge e with both endpoints in V_1. Let the end points be x, y
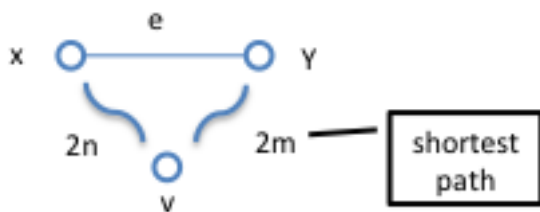
$2m \leq 2n + 1 \text{and} 2n \leq 2m + 1 \Rightarrow m = n$

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.
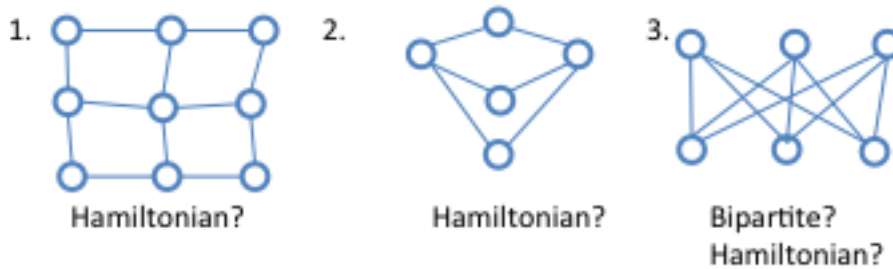
14

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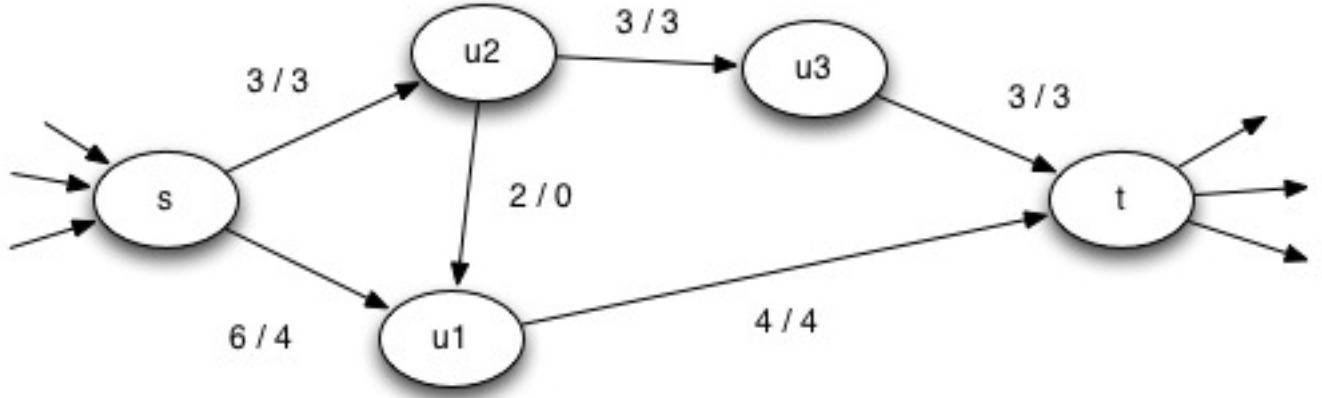$\sum_{e \in \alpha(v)} f(e) = \sum_{e \in \beta(v)} f(e)$
This is called "conservation function".
The total flow of the flow function is given by $F = \sum_{e \in \alpha(t)} f(e) - \sum_{e \in \beta(t)} f(e)$

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. $t \in \bar{S}$)
$E_{S\bar{S}} = \{e : \text{all edges with starting point in S and endpoint in } \bar{S}\}$
$E_{\bar{S}S} = \{e : \text{all edges with start point in } \bar{S} \text{ and end point in S}\}$
$E_{S\bar{S}} \cup E_{\bar{S}S}$ 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.
$$F = \sum_{e \in \alpha(t)} f(e) - \sum_{e \in \beta(t)} f(e)$$

17

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

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 $S \subseteq V$, $s \in S$, $t \notin S$

$$F \le c(S)$$

Proof. From lemma 3.1 we know

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

□

Korollar 3.1 **Max Flow - Min Cut Statement**

If $F = c(S)$ then the total flow F is <u>maximal</u> and the capacity of the cut defined by S is <u>minimal</u>.

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

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

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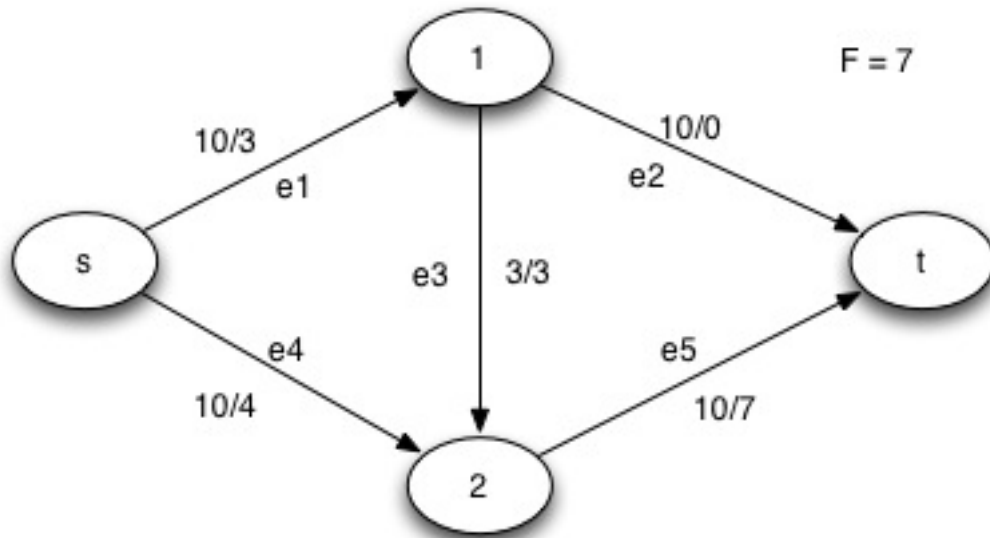
□

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

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

Example 3.3

Which of the following is an augmenting path?

- $e_1 e_2$

- $e_1 e_3 e_5$

- $e_4 e_3 e_2$

- $e_4 e_5$

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

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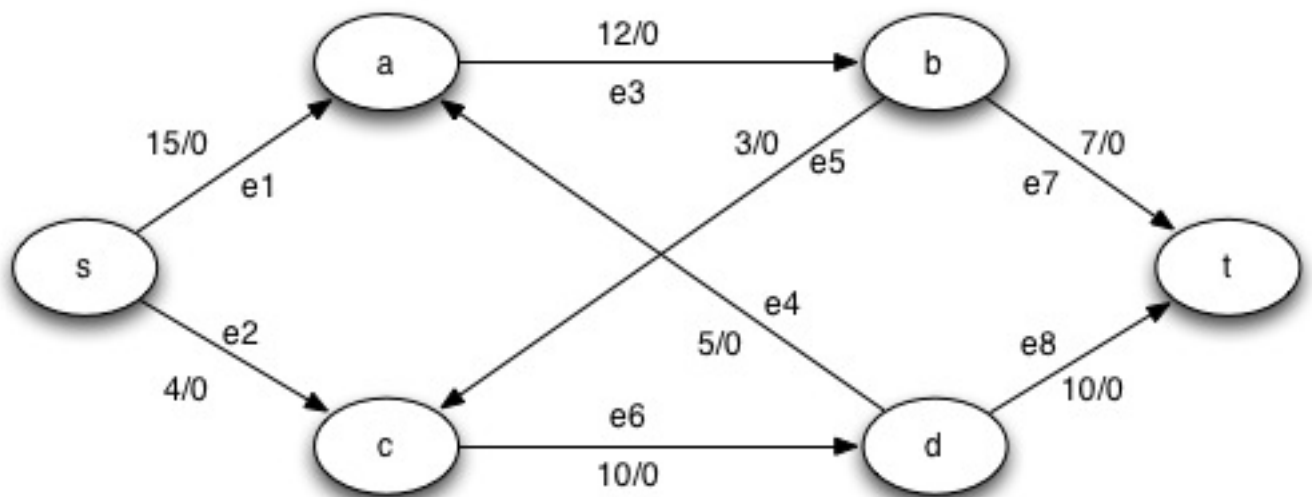
Augmenting path:
$$s* \xrightarrow{e_2} c* \xrightarrow{e_6} d* \xrightarrow{e_4} a* \xrightarrow{e_3} b* \xrightarrow{e_7} t*$$
Compute deltas:

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

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:

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

- $s \rightarrow a \rightarrow b \rightarrow t$

- $s \rightarrow a \rightarrow d \rightarrow t$

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

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 $s \rightarrow v_0 \rightarrow v_1 ... v_{f_{e-1}} \rightarrow v_{f_e} \rightarrow t$ 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 $s = v_0 \rightarrow ... \rightarrow v_2 = t$ 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.

$$F \overset{(3.1)}{=} \sum_{e \in E_{S\bar{S}}} f(e) - \underbrace{\sum_{e \in E_{\bar{S}S}} f(e)}_{=0} = \sum_{e \in E_{S\bar{S}}} c(e) = c(e)$$
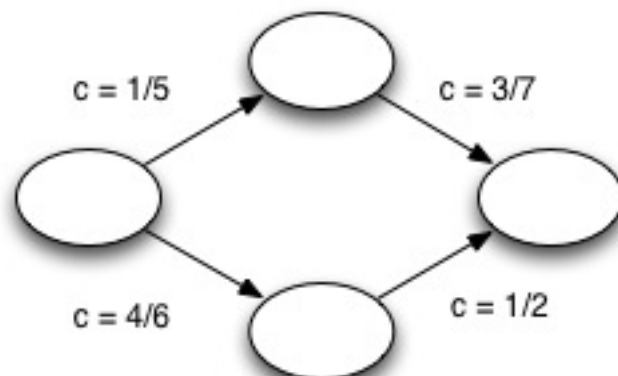
f is a flow function with maximal total flow.   ☐

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.
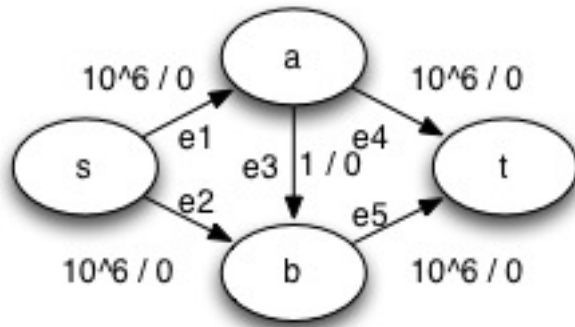

What about $c : E - > \mathbf{R}$?
Turn the capacities into natural numbers, calculate the results and divide them later.   ☐

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.

Theorem 3.1
If we use breadth-first-search when labelling and always select the shortest augmenting pathe, 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).

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 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

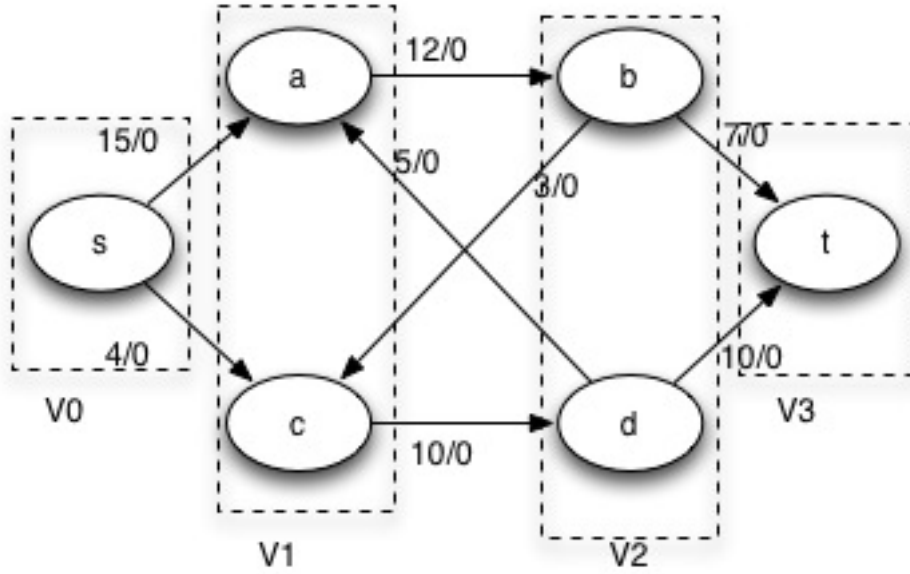$E(i) = \{e$: e is an useful edge between some node in V_{n-1} and $V_i\}$

Example 3.6
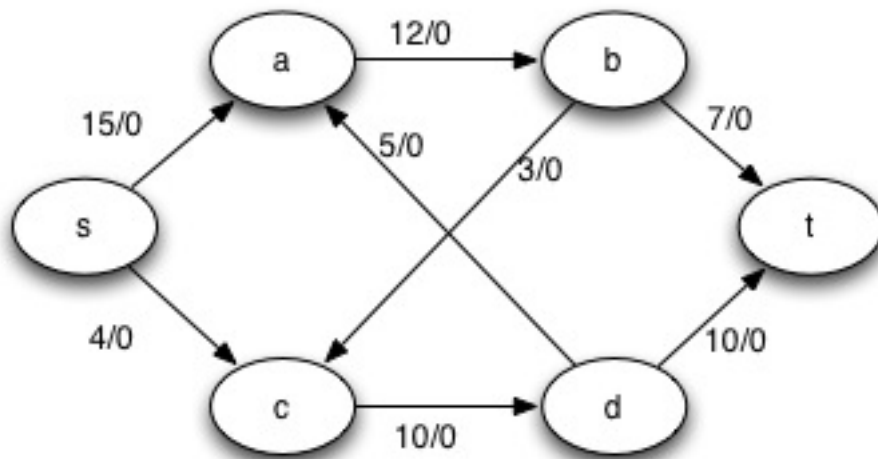
Layers:

- $V_0 = \{s\}$

- $V_1 = \{a, c\}$

- $V_2 = \{b, d\}$

- $V_3 = \{t\}$

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:
$c(e) > 0$ 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 <u>any</u> 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: Denic'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 $f_{new}(e) := f_{old}(e) + \bar{f}(e)$

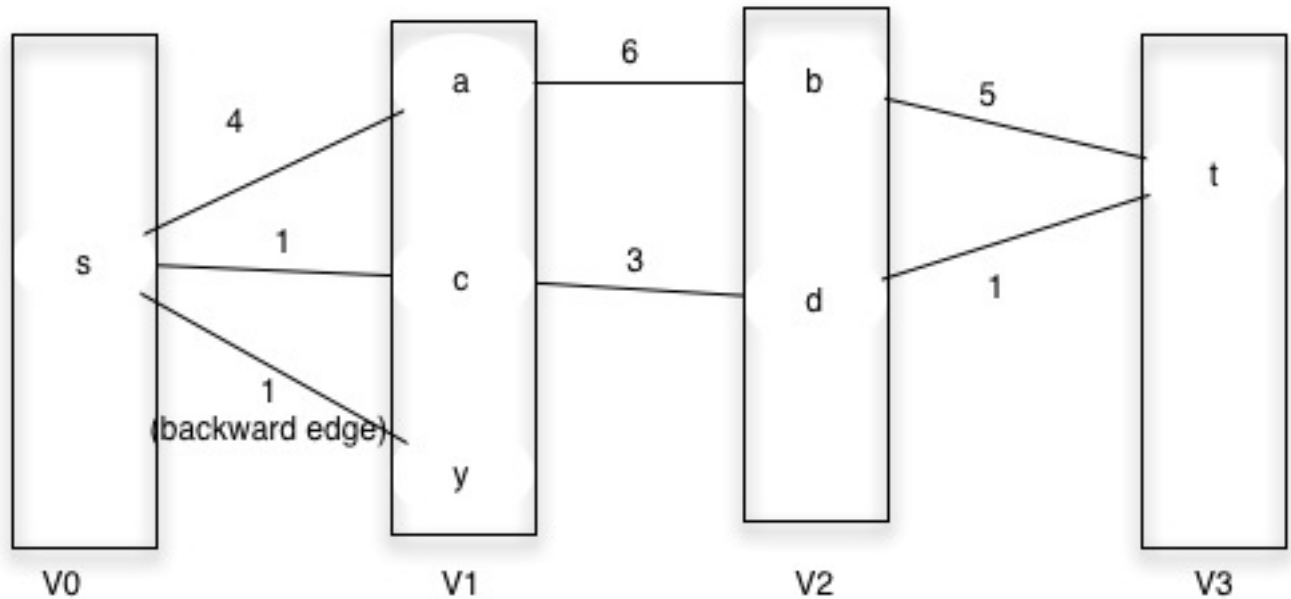ii) if $u \xleftarrow{e} v$ with $u \in V_{i-1}, v \in V_i$ then $f_{new}(e) := f_{old}(e) - \bar{f}(e)$

Algorithm of Denic:

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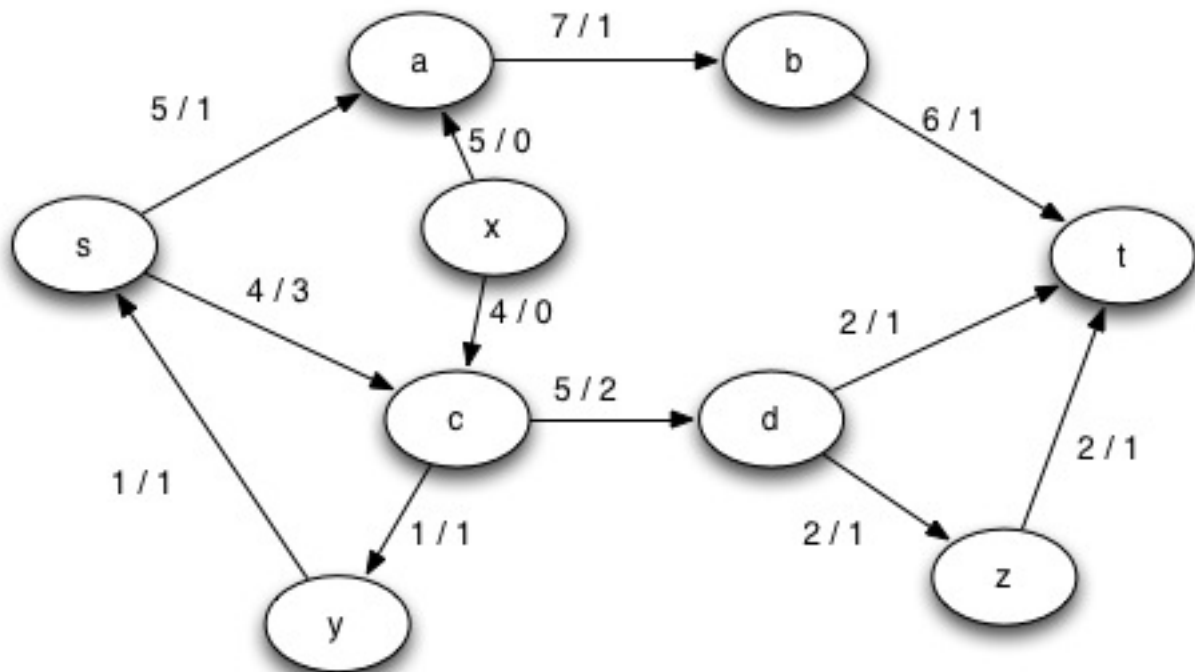
Given the example above the total flow of the network is $F = 3$. We will proceed with the first layering as follows:

26

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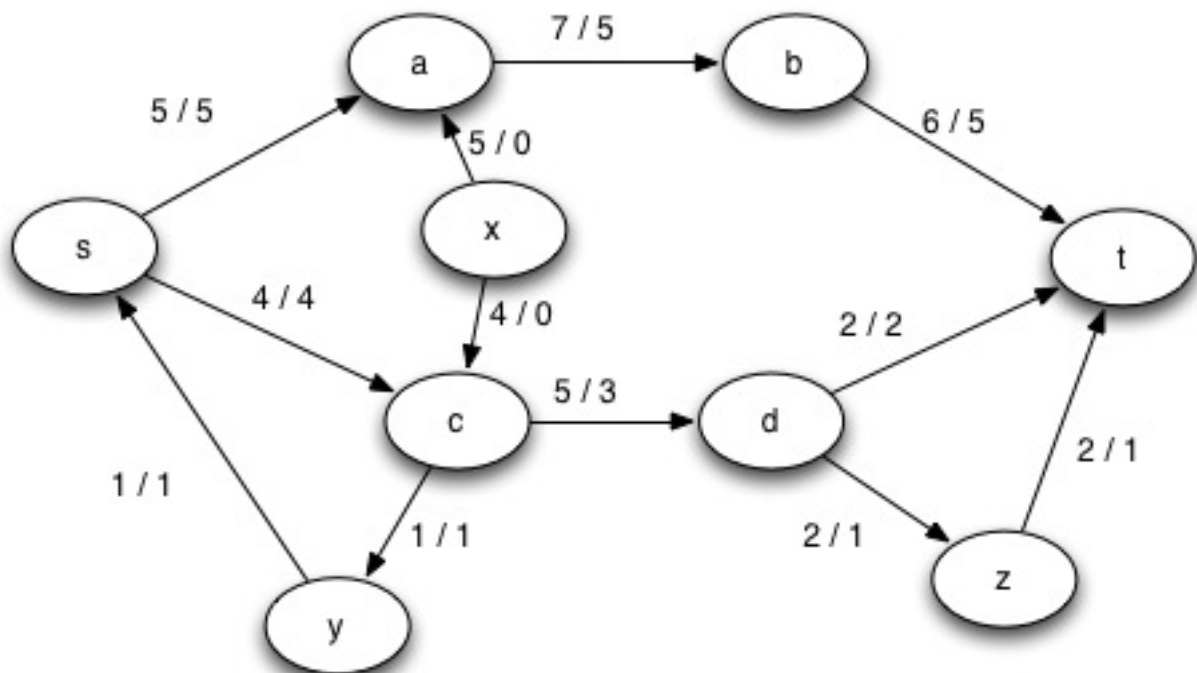
After determining $\bar{f}$ we have to update the flows on our graph and gain a new total flow $F = 8$.

Second layering:
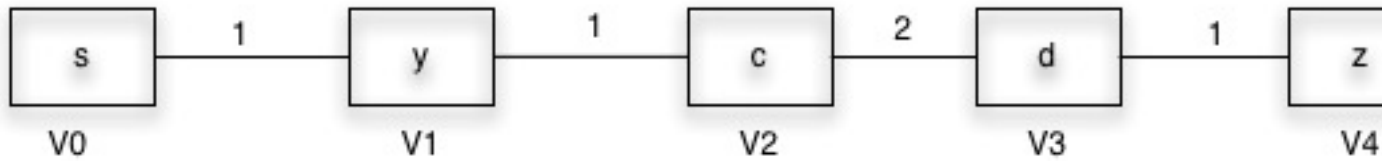
There is only one path from s to t. A flow function $\bar{f}$ that saturates at least one edge is the one that sends one unit along this path.
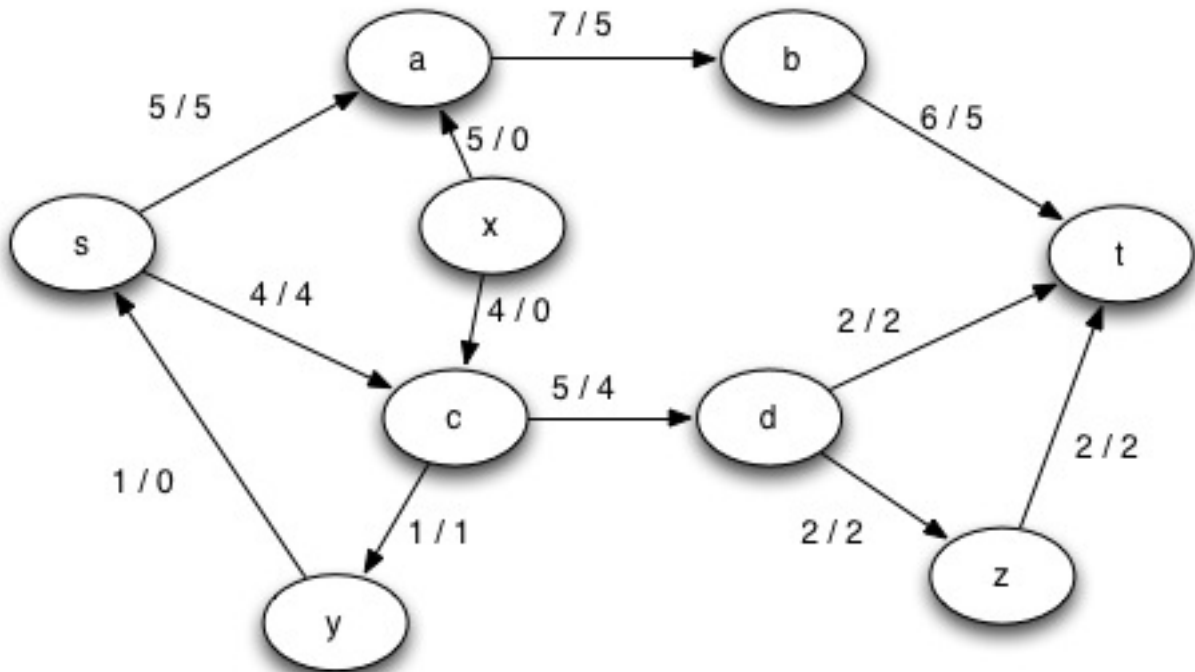
We update again the flows of the used edges and the total flow increases by one, so we have:
$F = 9$.

Third layering

$\overline{V_0 = \{s\}}$ 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 $0 \leq f_{new}(e) \leq c(e) \forall e \in E$ and that the flow condition holds for every node.

- Show $0 \leq f_{new}(e) \leq c(e)$:

    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:
    $u \xrightarrow{e} v$

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

    M: Bitte prüft 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 $e \in E(i), u \xrightarrow{e} v, u \in V_{i-1}, v \in V_i$
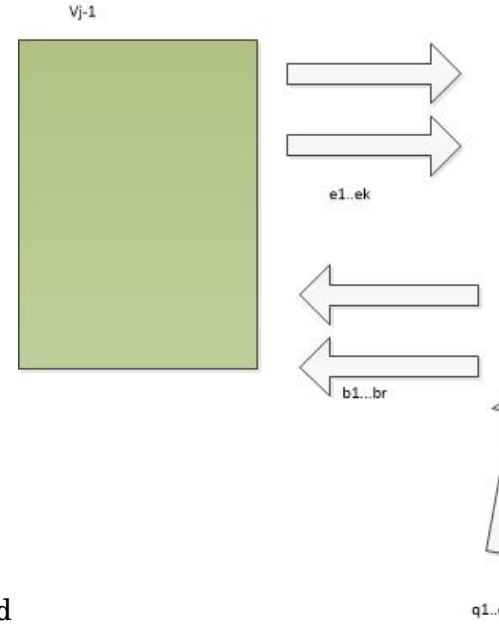
    – Case 1: $u \xrightarrow{e} v$
    $$f_{new}(e) = f_{old}(e) + \bar{f}(e) \geq 0$$

    – Case2: $u \leftarrow v$
    $$f_{new}(e) = f_{old}(e) - \bar{f}(e) \geq f_{old}(e) - \bar{c}(e) \stackrel{Def.\bar{c}}{=} 0$$

<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.

□