Algorithmen II

Peter Sanders

Übungen:

Moritz Laupichler, Nikolai Maas

Institut für Theoretische Informatik

Web:

`algo2.iti.kit.edu/AlgorithmenII_WS23.php`

5 Maximum Flows and Matchings

[mit Kurt Mehlhorn, Rob van Stee]

Folien auf Englisch

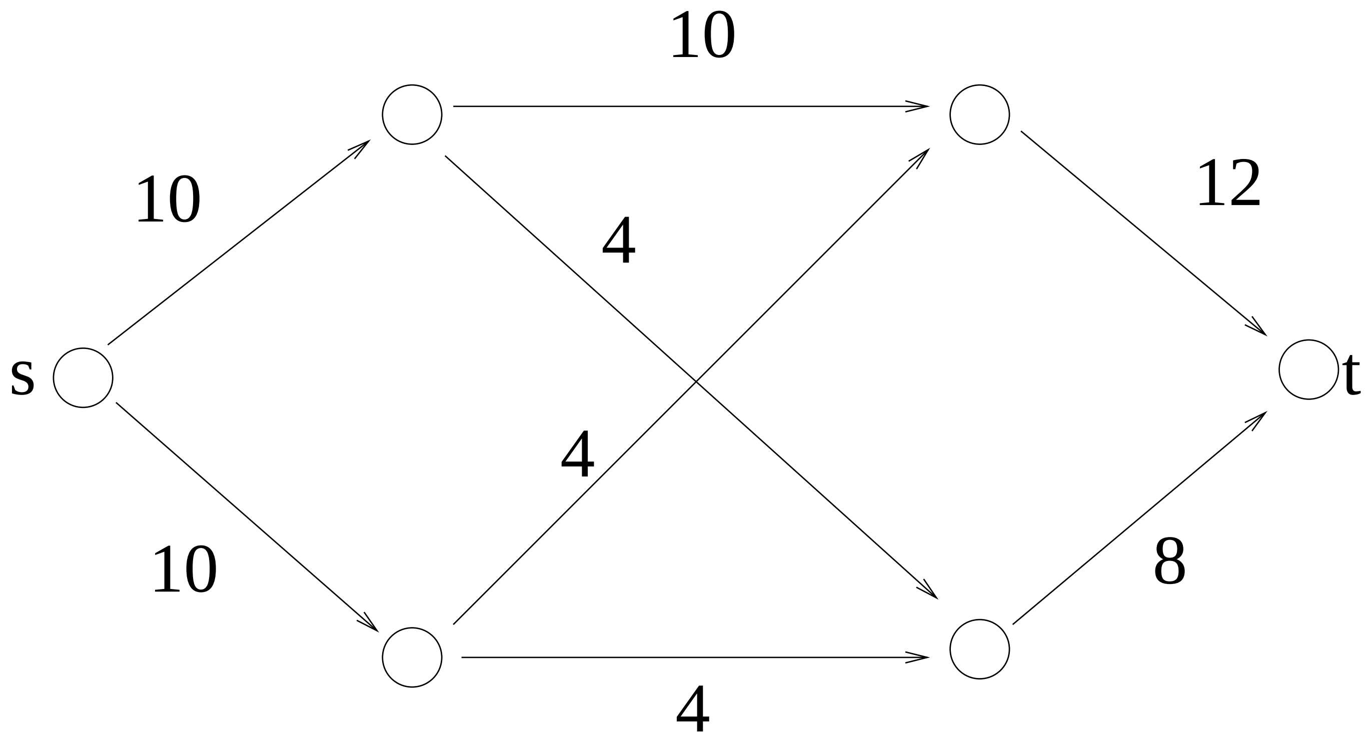
Literatur:

[Mehlhorn / Näher, The LEDA Platform of Combinatorial and Geometric Computing, Cambridge University Press, 1999]
`http://www.mpi-inf.mpg.de/~mehlhorn/ftp/LEDAbook/Graph_alg.ps`

[Ahuja, Magnanti, Orlin, Network Flows, Prentice Hall, 1993]

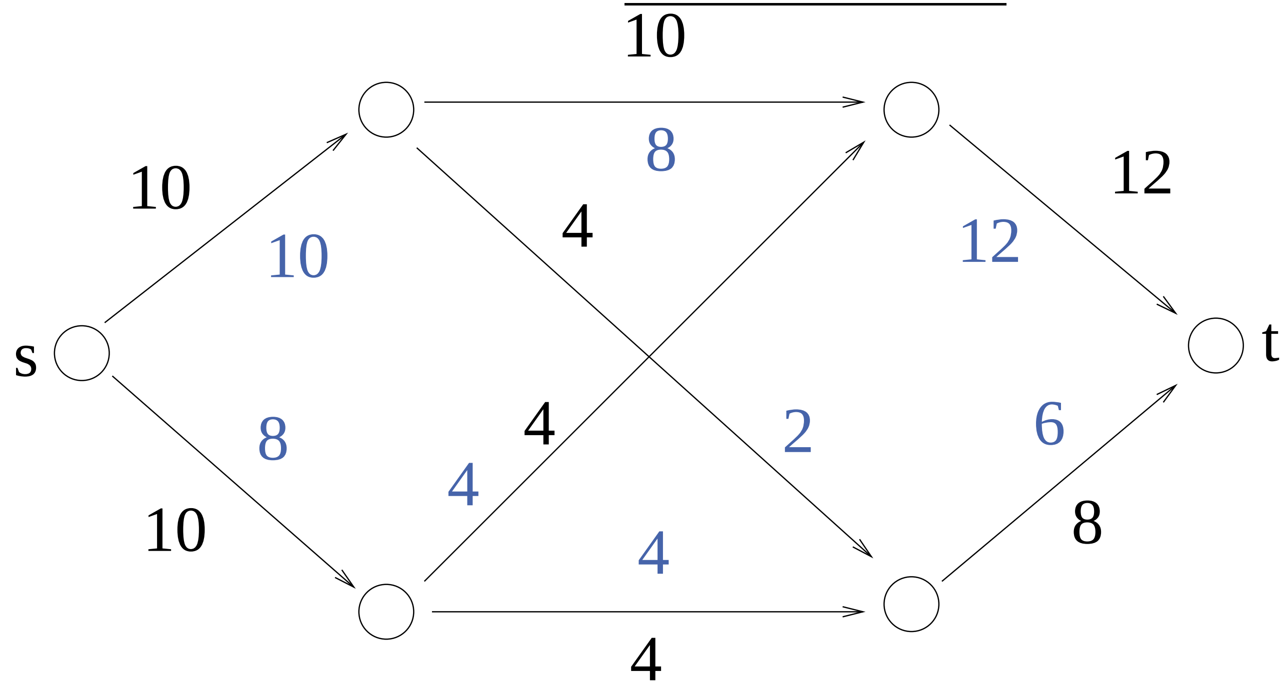
Definitions: Network

- Network = directed weighted graph with source node s and sink node t
- s has no incoming edges, t has no outgoing edges
- Weight c_e of an edge e = capacity of e (nonnegative!)



Definitions: Flows

- Flow = function f_e on the edges, $\forall e : 0 \leq f_e \leq c_e$
  $\forall v \in V \setminus \{s,t\}$: total incoming flow = total outgoing flow
- Value of a flow $\mathbf{val}(f) =$
  total outgoing flow from $s =$ total flow going into t
- Goal: find a flow with maximum value

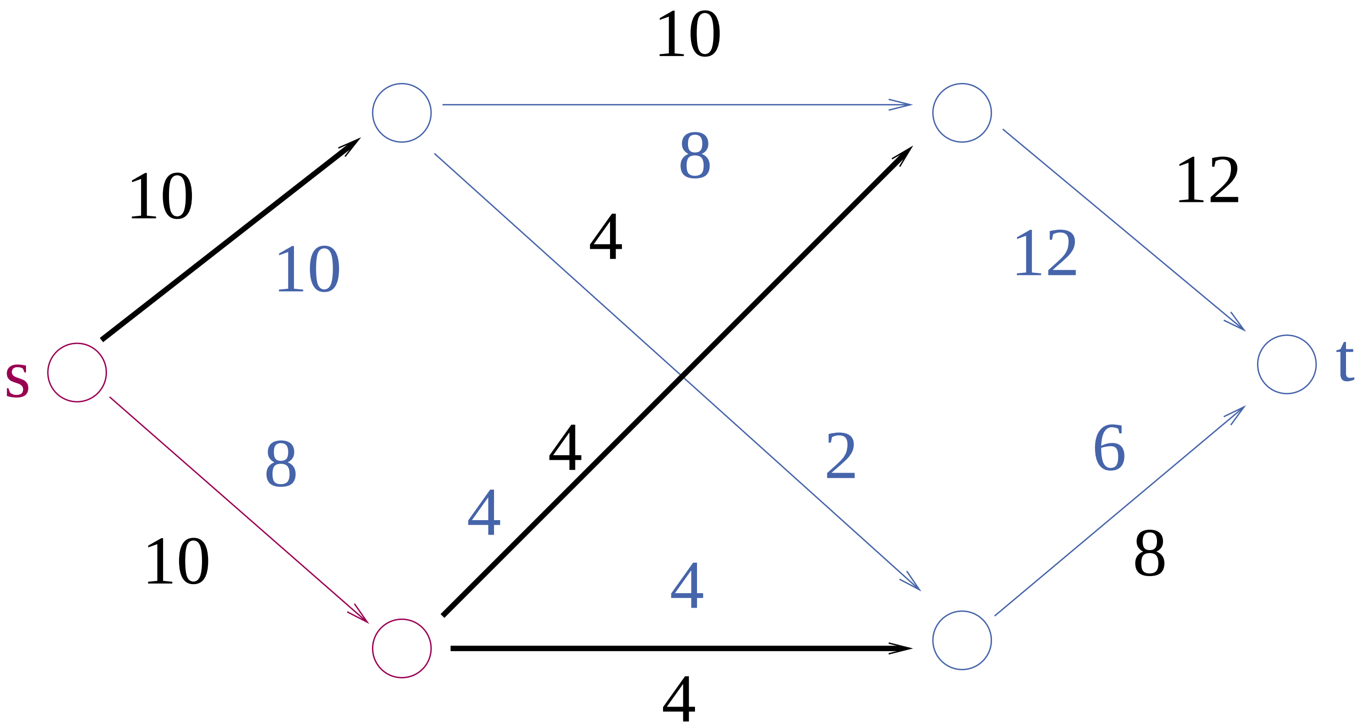


Definitions: (Minimum) s-t Cuts

An s-t cut is partition of V into S and T with $s \in S$ and $t \in T$.

The capacity of this cut is:

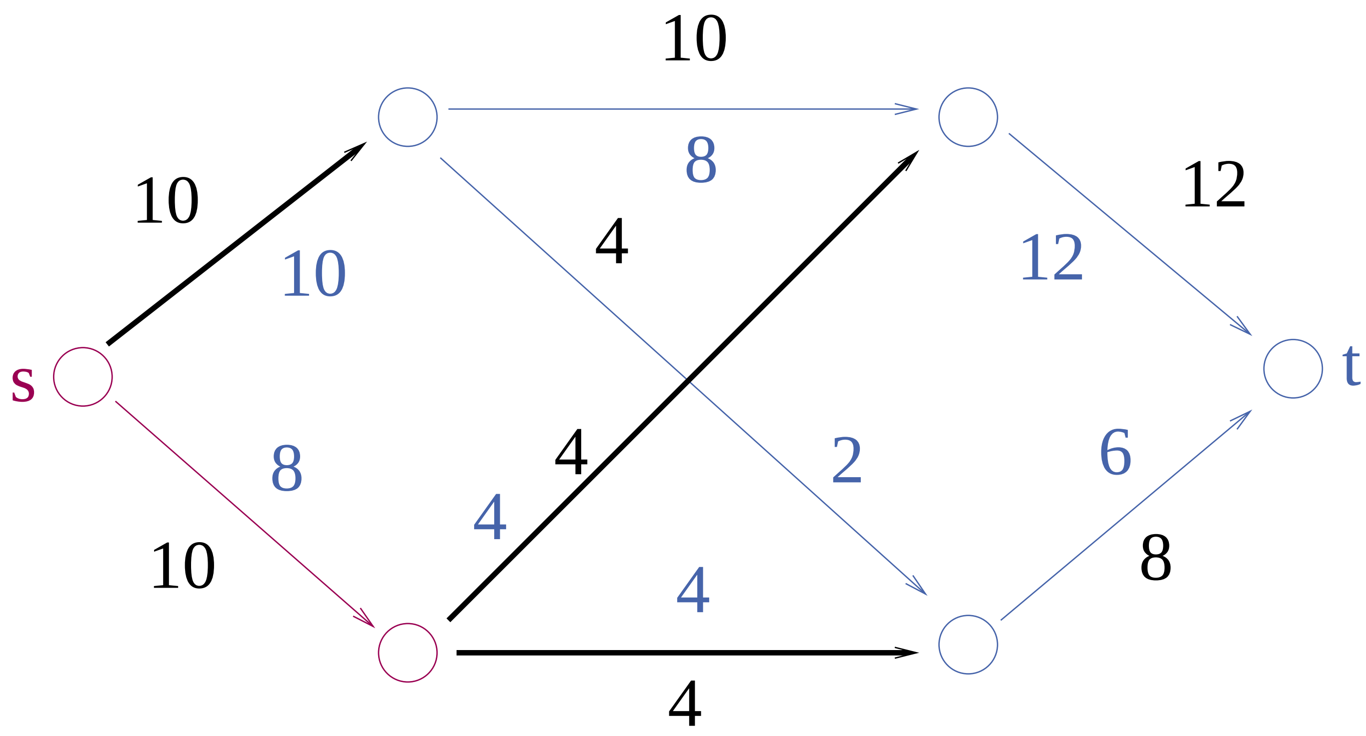
$$\sum \{c_{(u,v)} : u \in S, v \in T\}$$



Duality Between Flows and Cuts

Theorem:[Elias/Feinstein/Shannon, Ford/Fulkerson 1956]

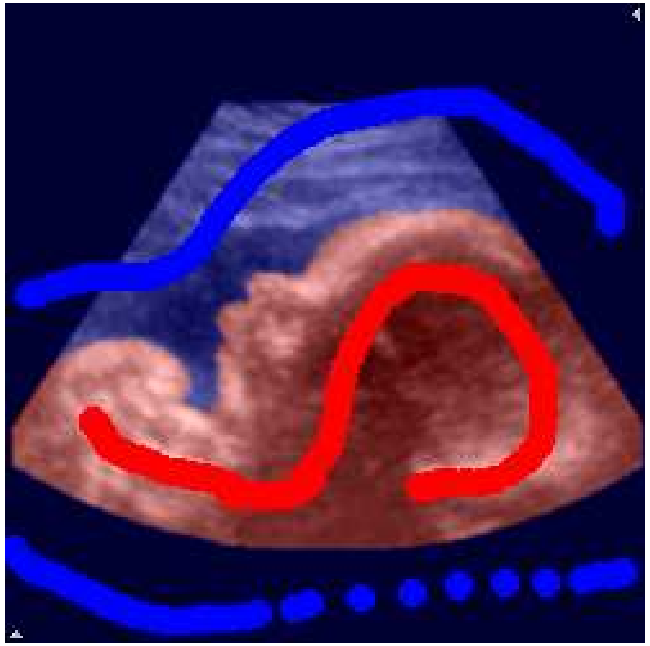
Value of an s-t max-flow $=$ minimum capacity of an s-t cut.



Proof: later

Applications

- Oil pipes
- Traffic flows on highways
- Image Processing `http://vision.csd.uwo.ca/maxflow-data`
  - segmentation
  - stereo processing
  - multiview reconstruction
  - surface fitting
- disk/machine/tanker scheduling
- matrix rounding
- …



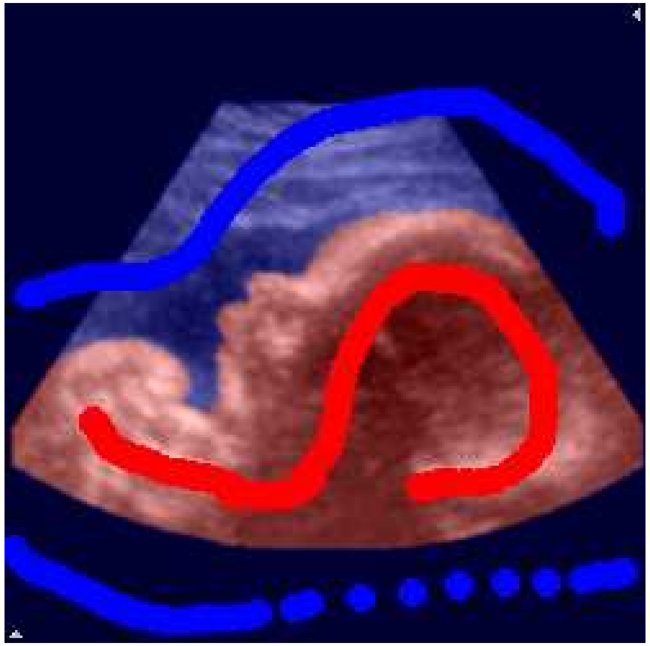
Current Research Challenge: AI versus Optimal Algorithms

Many image processing applications are currently taken over by deep convolutional neural networks.

- \+ Often better results
- \+ No ad-hoc definitions of s, t, c
- − "Optimality" is thrown over board
- − Lots of training examples needed

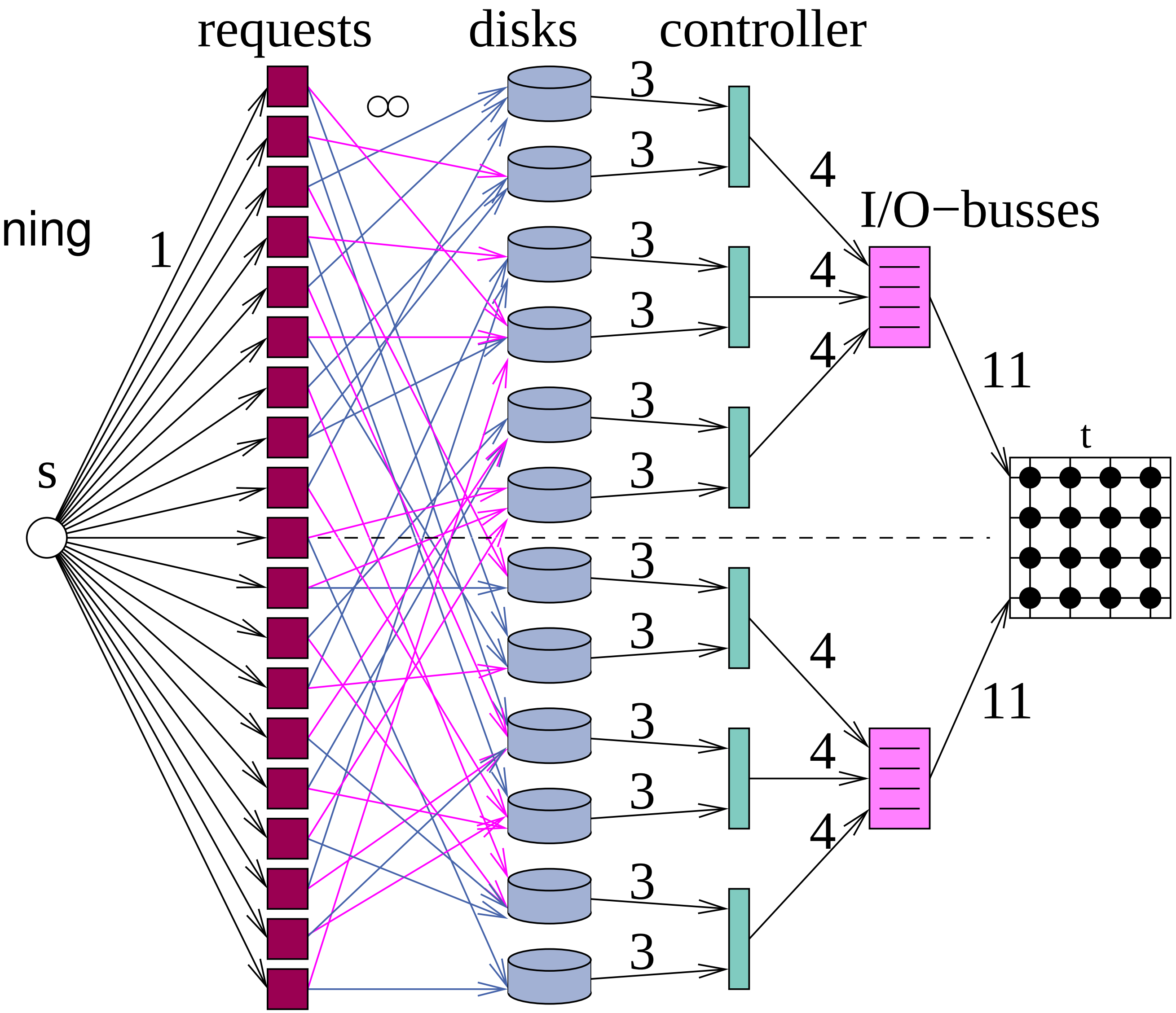
Is there a middle way?

Learn s, t, c then optimize?



Applications in our Group

- multicasting using
  network coding
- balanced k partitioning
- disk scheduling



Option 1: linear programming

- Flow variables x_e for each edge e
- Flow on each edge is at most its capacity
- Incoming flow at each vertex = outgoing flow from this vertex
- Maximize outgoing flow from starting vertex

We can do better!

Algorithms 1956–now

Year	Author	Running time
1956	Ford-Fulkerson	$O(mnU)$
1969	Edmonds-Karp	$O(m^2n)$
1970	Dinic	$O(mn^2)$
1973	Dinic-Gabow	$O(mn\log U)$
1974	Karzanov	$O(n^3)$
1977	Cherkassky	$O(n^2\sqrt{m})$
1980	Galil-Naamad	$O(mn\log^2 n)$
1983	Sleator-Tarjan	$O(mn\log n)$
1986	Goldberg-Tarjan	$O(mn\log(n^2/m))$
1987	Ahuja-Orlin	$O(mn+n^2\log U)$

$n =$ number of nodes

$m =$ number of arcs

$U =$ largest capacity

Year	Author	Running time
1987	Ahuja-Orlin-Tarjan	$O(mn\log(2+n\sqrt{\log U}/m))$
1990	Cheriyan-Hagerup-Mehlhorn	$O(n^3/\log n)$
1990	Alon	$O(mn+n^{8/3}\log n)$
1992	King-Rao-Tarjan	$O(mn+n^{2+\varepsilon})$
1993	Philipps-Westbrook	$O(mn\log n/\log\frac{m}{n}+n^2\log^{2+\varepsilon} n)$
1994	King-Rao-Tarjan	$O(mn\log n/\log\frac{m}{n\log n})$ if $m\geq 2n\log n$
1997	Goldberg-Rao	$O(\min\{m^{1/2},n^{2/3}\}m\log(n^2/m)\log U)$
2014	Lee-Sidford	$O(m\sqrt{n}\log^2 U)$
2020	v. d. Brand et al.	$O(m+n^{\frac{3}{2}}\log U\log^{?} m)$
2021	Gao-Liu-Peng	$O(m^{\frac{3}{2}-\frac{1}{328}}\log U\log^{?} m)$
2022	v.d. Brand et al.	$O(m^{\frac{3}{2}-\frac{1}{58}}\log U\log^{?} m)$
2022	Chen, Kyng et al.	$O(m^{1+o(1)}\log U)$

Augmenting Paths (Rough Idea)

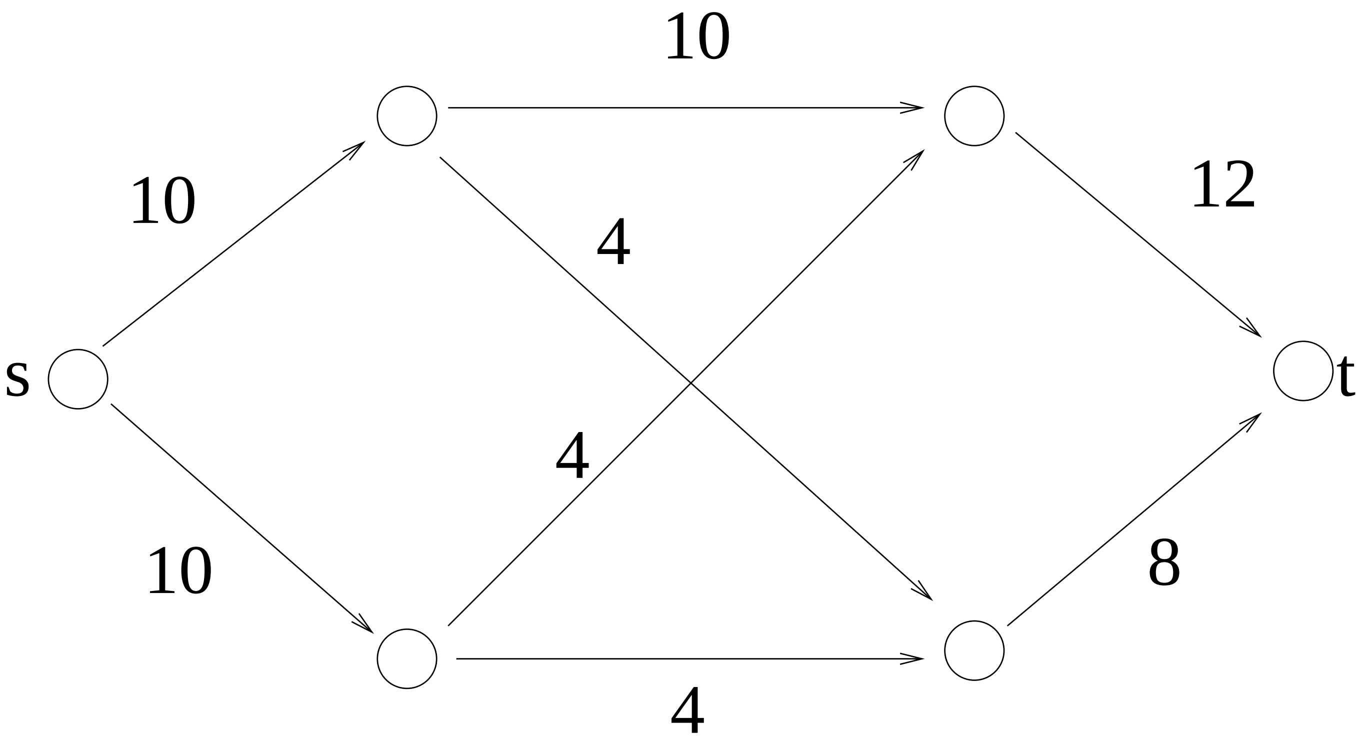
Find a path from s to t such that each edge has some spare capacity

On this path, saturate the edge with the smallest spare capacity

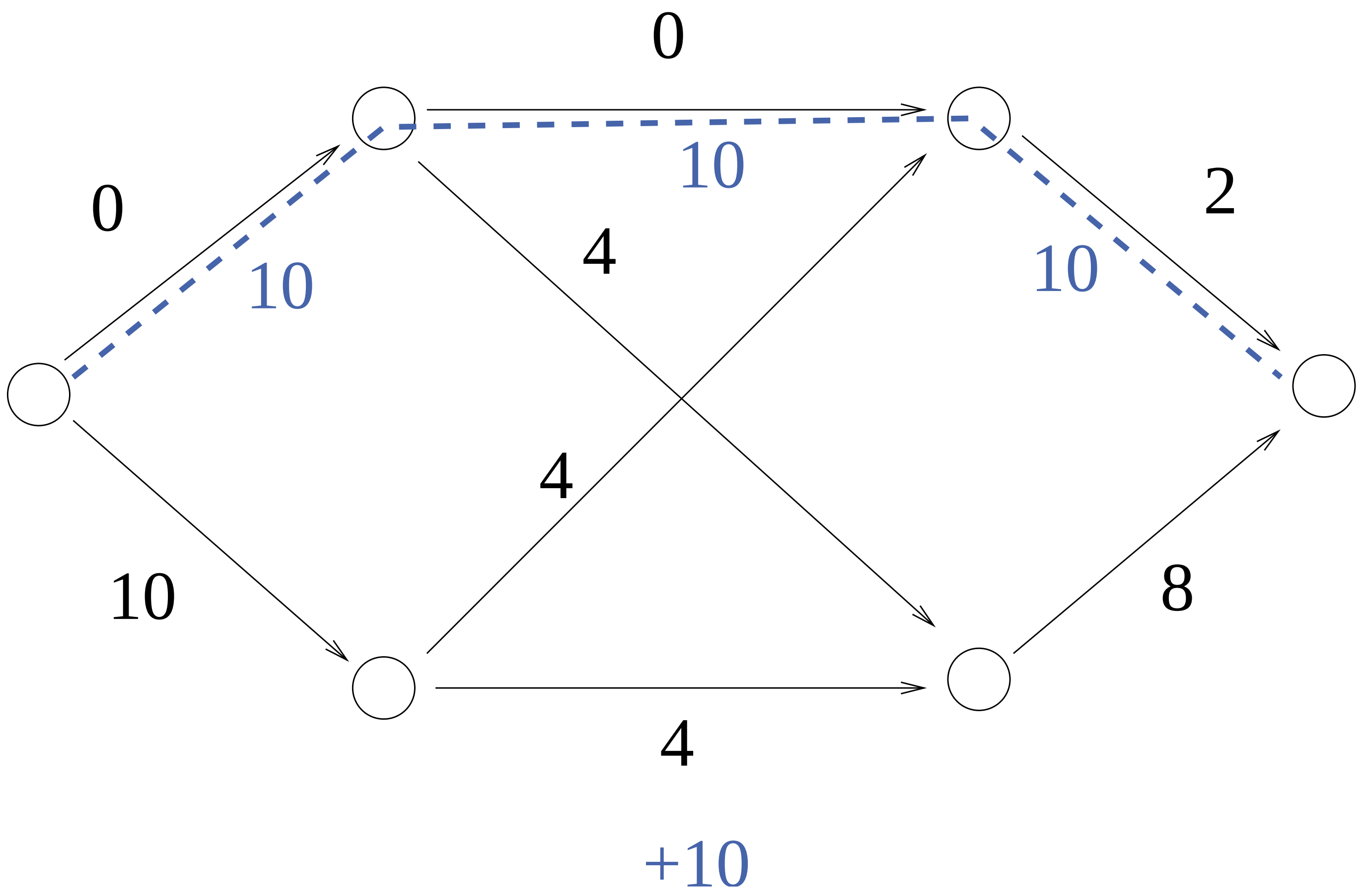
Adjust capacities for all edges (create residual graph) and repeat

A typical greedy algorithm

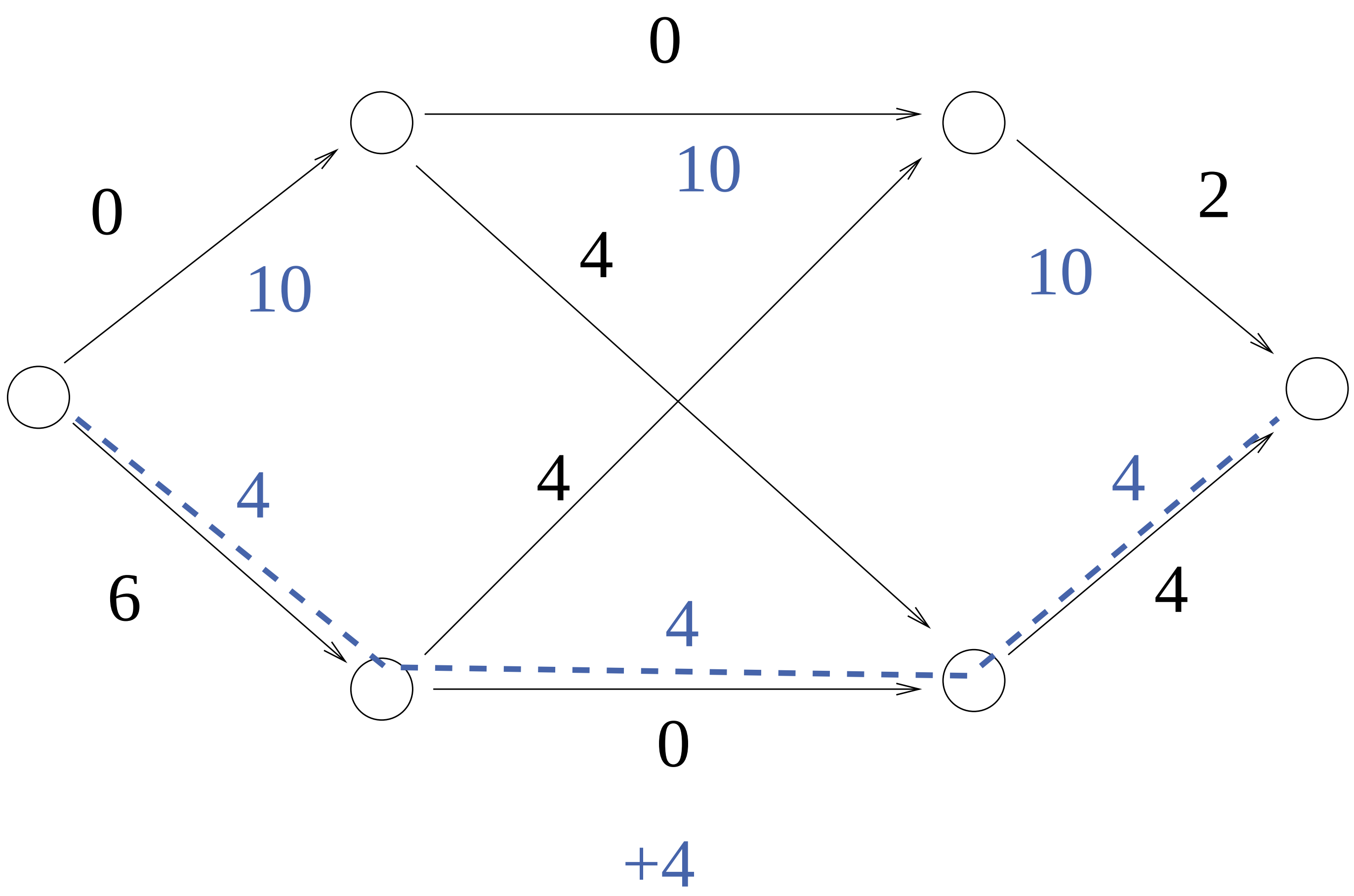
Example



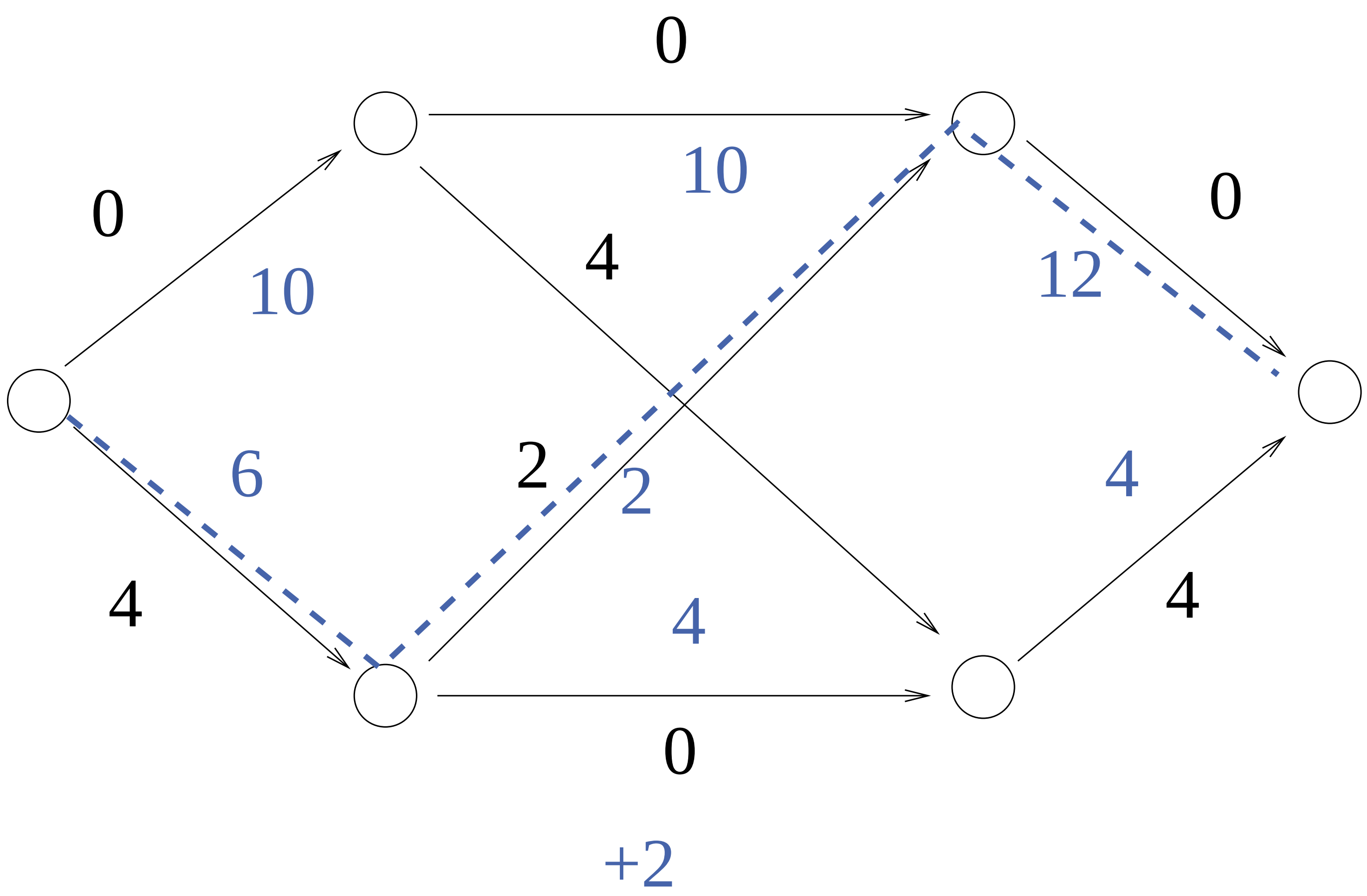
Example



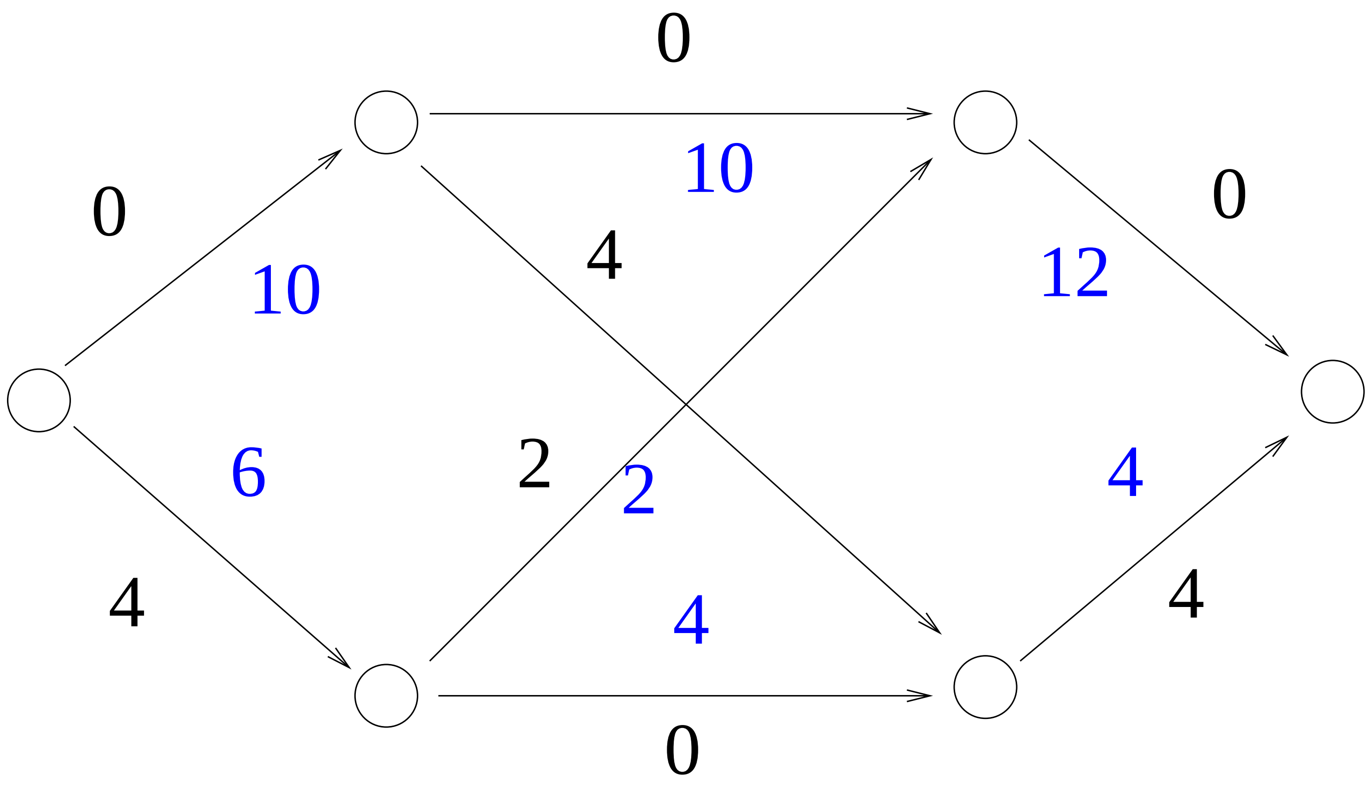
Example



Example

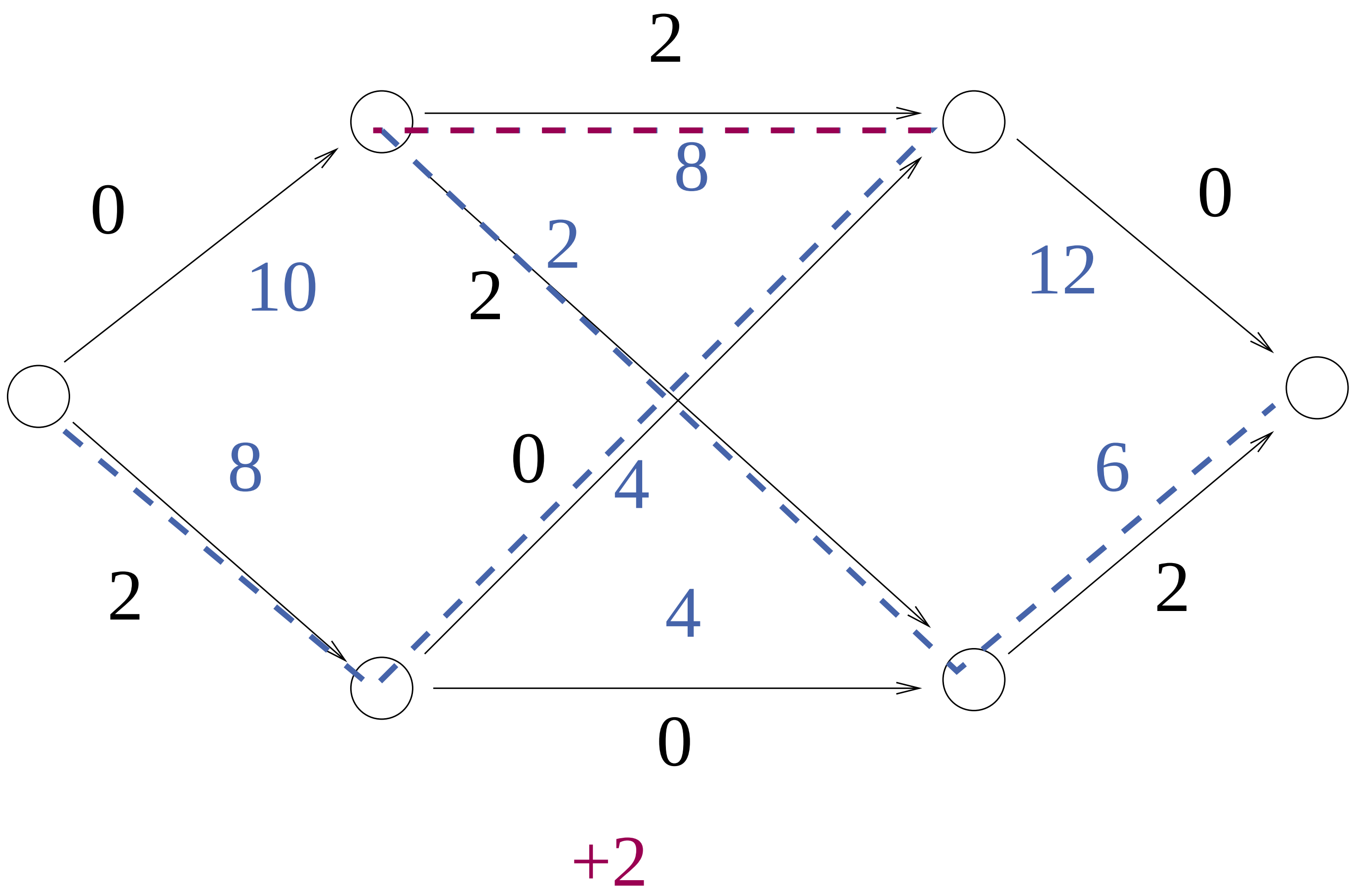


Example



are we done?

Example

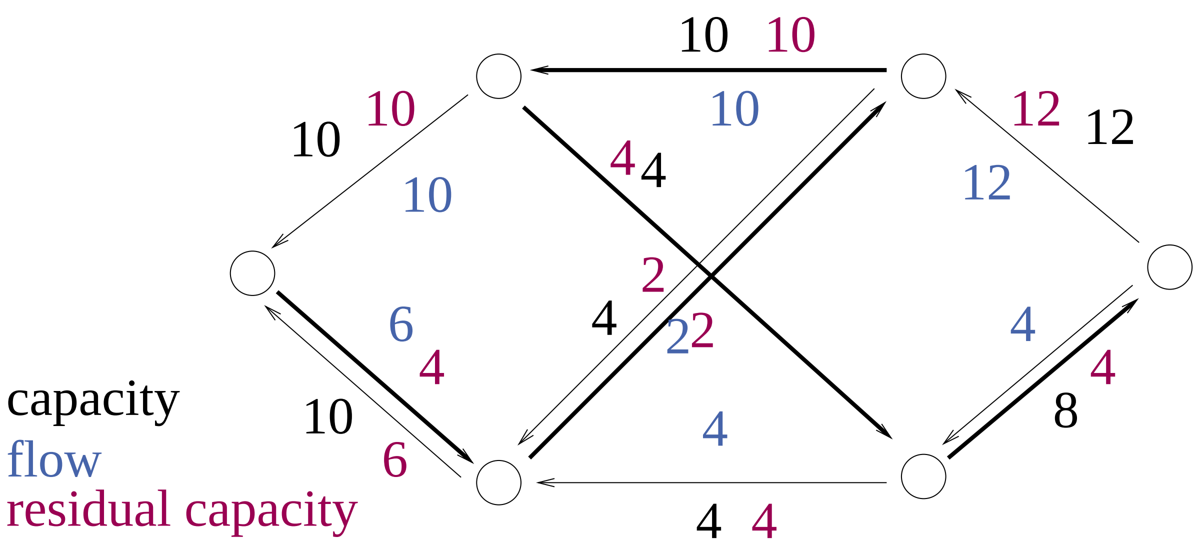


Residual Graph

Given, network $G = (V, E, c)$, flow f

Residual graph $G_f = (V, E_f, c^f)$. For each $e \in E$ we have

$$\begin{cases} e \in E_f \text{ with } c_e^f = c_e - f(e) & \text{if } f(e) < c(e) \\ e^{\text{rev}} \in E_f \text{ with } c_{e^{\text{rev}}}^f = f(e) & \text{if } f(e) > 0 \end{cases}$$



Augmenting Paths

Find a path p from s to t such that each edge e has nonzero residual capacity c_e^f

$$\Delta f := \min_{e \in p} c_e^f$$

foreach $(u,v) \in p$ **do**

  if $(u,v) \in E$ **then** $f_{(u,v)} += \Delta f$

  else $f_{(v,u)} -= \Delta f$

Ford Fulkerson Algorithm

Function FFMaxFlow$(G = (V, E), s, t, \mathsf{c} : E \to \mathbb{N}) : E \to \mathbb{N}$

$f := 0$

while $\exists$path $p = (s, \ldots, t)$ in G_f **do**

augment f along p

return f

time $\mathrm{O}(m\mathsf{val}(f))$

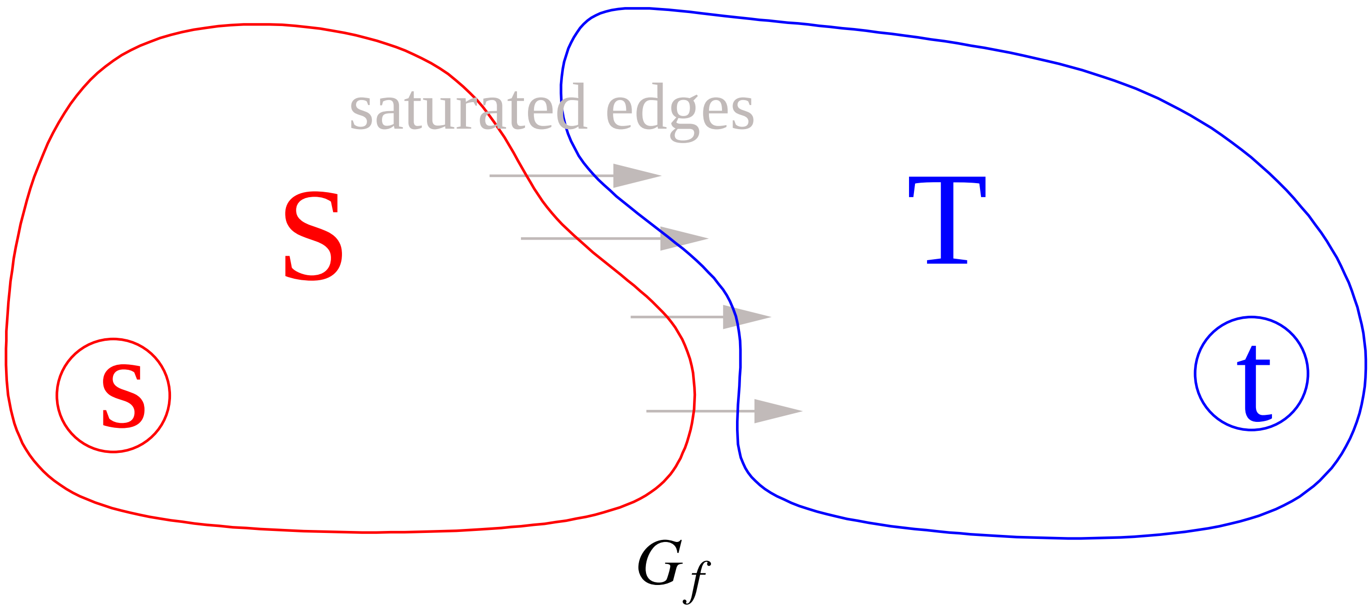
Ford Fulkerson – Correctness

"Clearly" FF computes a feasible flow f. (Invariant)

Todo: flow value is maximal

At termination: no augmenting paths in G_f left.

Consider cut $(S, T := V \setminus S)$ with
$S := \{v \in V : v$ reachable from s in $G_f\}$



A Basic Observations

Lemma 1: For any cut (S,T):

$$\mathbf{val}(f) = \overbrace{\sum_{e \in E \cap S \times T} f_e}^{S \rightarrow T \text{ edges}} - \overbrace{\sum_{e \in E \cap T \times S} f_e}^{T \rightarrow S \text{ edges}} .$$

Ford Fulkerson – Correctness

Todo: $\mathbf{val}(f)$ is maximal when no augmenting paths in G_f left.

Consider cut $(S, T := V \setminus S)$ with

$S := \{v \in V : v$ reachable from s in $G_f\}$.

Observation: $\forall (u,v) \in E \cap T \times S : f(u,v) = 0$

otherwise $c^f(v,u) > 0$ contradicting the definition of S.

$$\mathbf{val}(f) = \sum_{e \in E \cap S \times T} f_e - \sum_{e \in E \cap T \times S} f_e \qquad \text{Lemma 1}$$

$$= \sum_{e \in E \cap S \times T} f_e \qquad \text{Observation above}$$

$$= \sum_{e \in E \cap S \times T} c_{(u,v)} = (S,T) \text{ cut capacity}$$

see next slide

Max-Flow-Min-Cut theorem

Theorem: Max-flow = min-cut

Proof:

obvious: any-flow $\leq$ max-flow $\leq$ min-cut $\leq$ any-cut

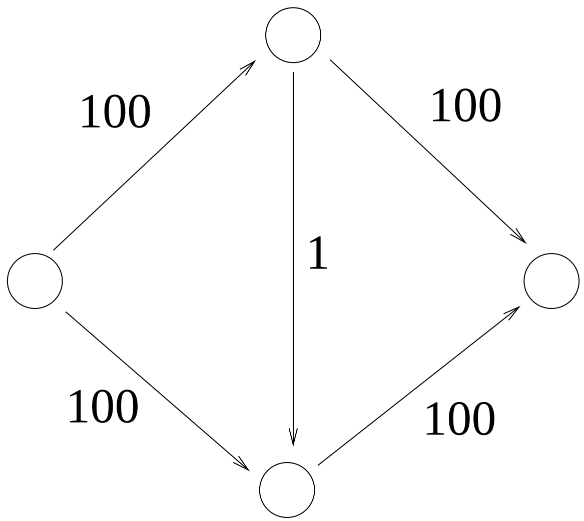
previous slide:

(S,T) flow $=$ (S,T) cut capacity

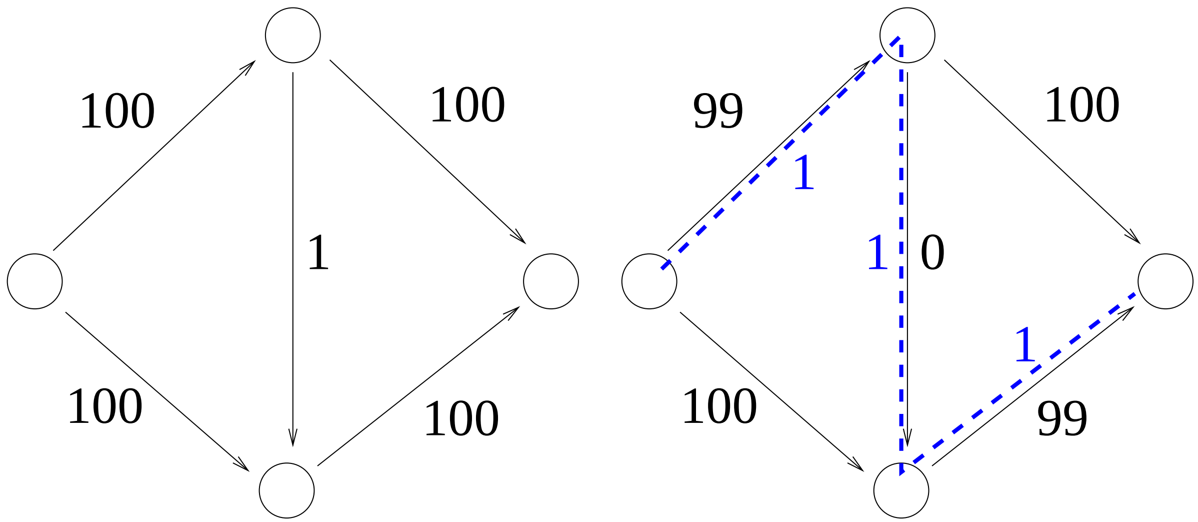
$\Rightarrow$

(S,T) flow $=$ max-flow $=$ min-cut

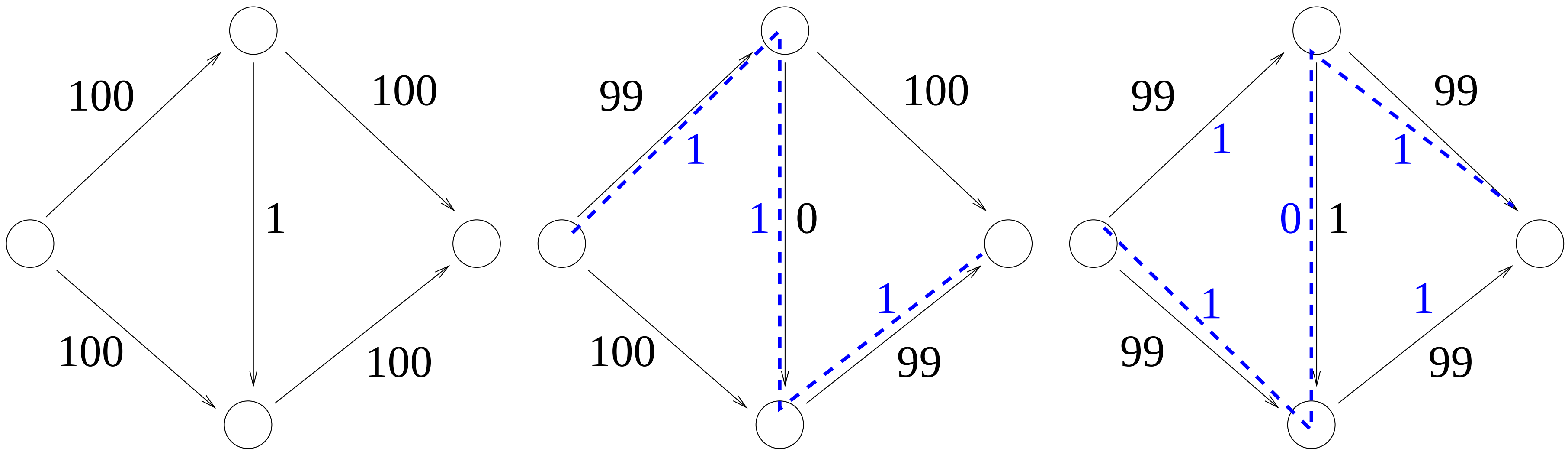
A Bad Example for Ford Fulkerson



A Bad Example for Ford Fulkerson



A Bad Example for Ford Fulkerson

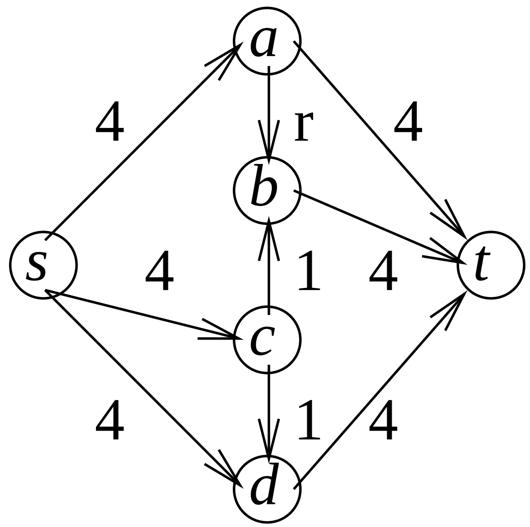


An Even Worse Example for Ford Fulkerson

[U. Zwick, TCS 148, p. 165–170, 1995]

Let $r = \dfrac{\sqrt{5}-1}{2}$.

Consider the graph



And the augmenting paths

$p_0 = \langle s, c, b, t \rangle$

$p_1 = \langle s, a, b, c, d, t \rangle$

$p_2 = \langle s, c, b, a, t \rangle$

$p_3 = \langle s, d, c, b, t \rangle$

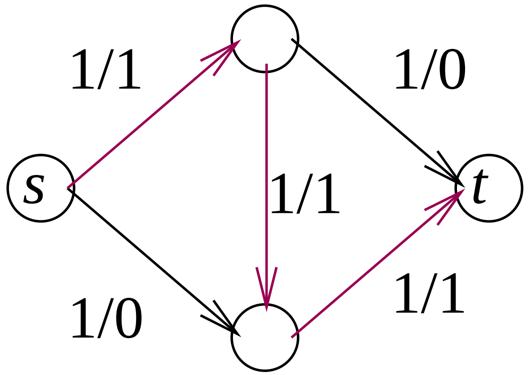
The sequence of augmenting paths $p_0(p_1, p_2, p_1, p_3)^*$ is an infinite sequence of positive flow augmentations.

The flow value does not converge to the maximum value 9.

Blocking Flows

f_b is a **blocking flow** in H if

$$\forall \text{paths } p = \langle s, \ldots, t \rangle : \exists e \in p : f_b(e) = c(e)$$



Dinitz Algorithm

Function DinitzMaxFlow$(G = (V,E), s, t, c : E \to \mathbb{N}) : E \to \mathbb{N}$

$f := 0$

while $\exists$path $p = (s, \ldots, t)$ in G_f **do**

$d{=}G_f$.reverseBFS(t) $\;: V \to \mathbb{N}$

$L_f = (V, \{(u,v) \in E_f : d(v) = d(u) - 1\})$ // layer graph

find a blocking flow f_b in L_f

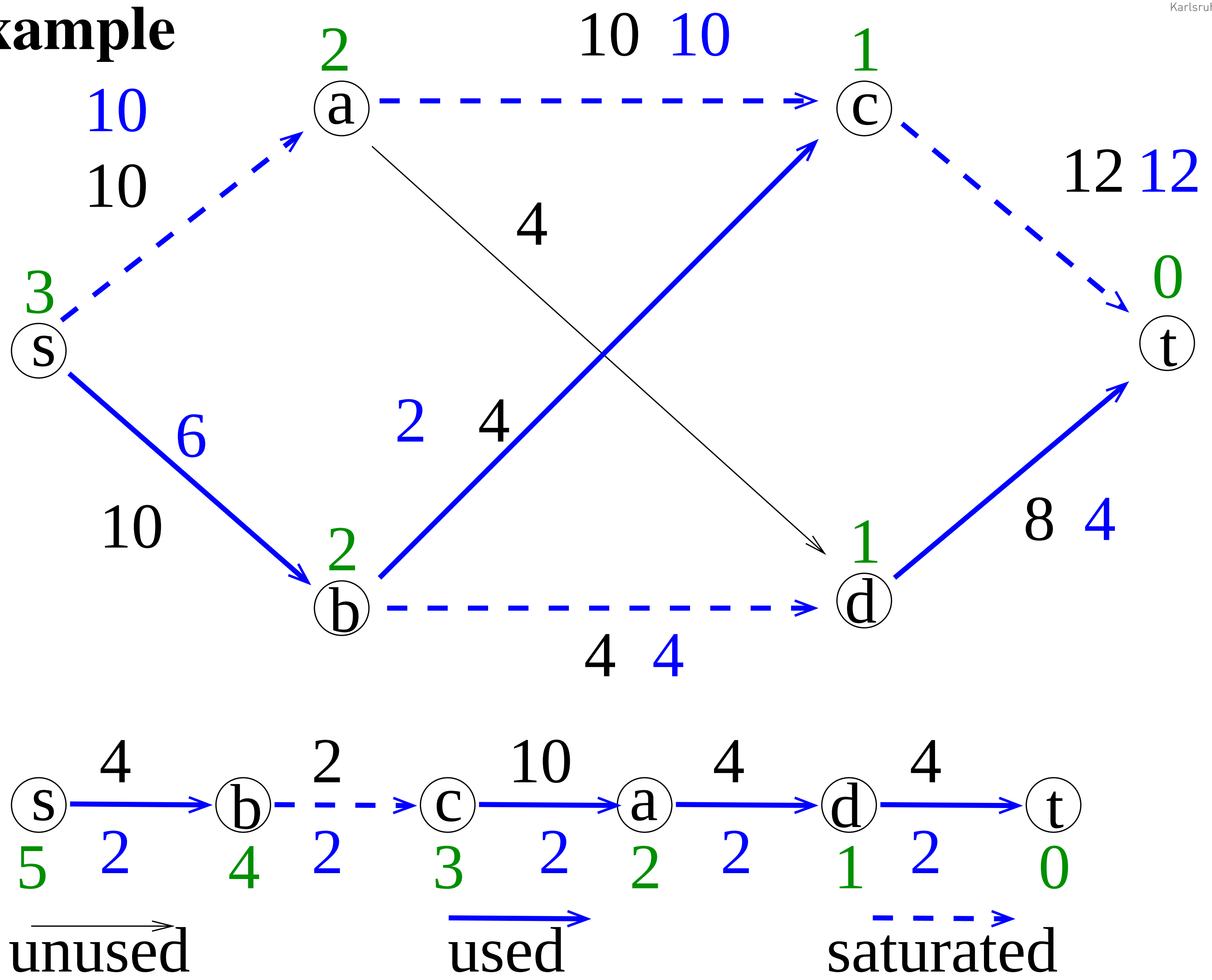
augment f+=f_b

return f

Dinitz – Correctness

analogous to Ford-Fulkerson

Example


2
a
10 10
1
c
10
10
4
12 12
3
s
0
t
6
2 4
10
2
b
1
d
8 4
4 4
s 4 b 2 c 10 a 4 d 4 t
5 2 4 2 3 2 2 2 1 2 0
unused
used
saturated

Computing Blocking Flows

Idea: repeated DFS for augmenting paths

(not using DFS algorithm schema)

Function blockingFlow($L_f = (V, E)) : E \to \mathbb{N}$

$p = \langle s \rangle$: Path; $f_b = 0$: Flow

loop // Round

$v := p$.last()

if $v = t$ **then** // breakthrough

$\delta := \min\{c(e) - f_b(e) : e \in p\}$

foreach $e \in p$ **do**

$f_b(e) += \delta$

if $f_b(e) = c(e)$ **then** remove e from E

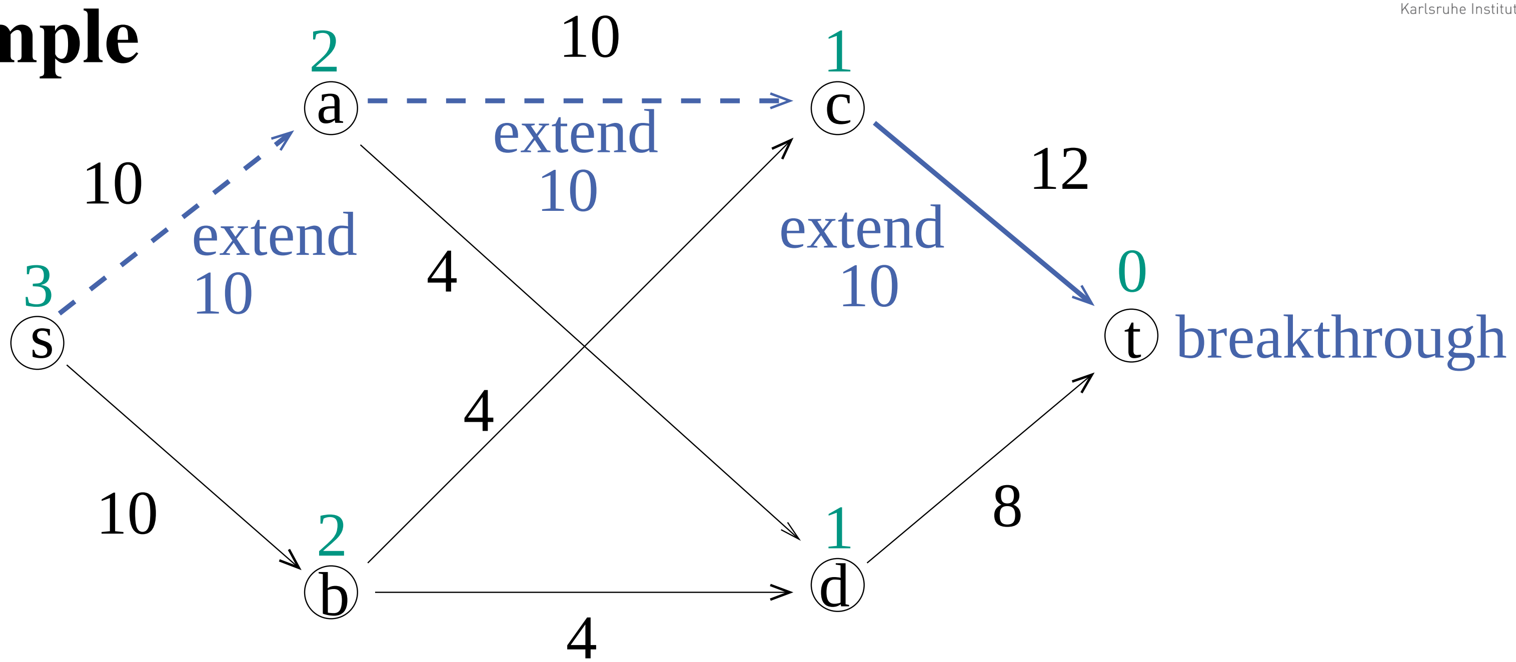
$p := \langle s \rangle$

else if $\exists e = (v, w) \in E$ **then** p.pushBack(w) // extend

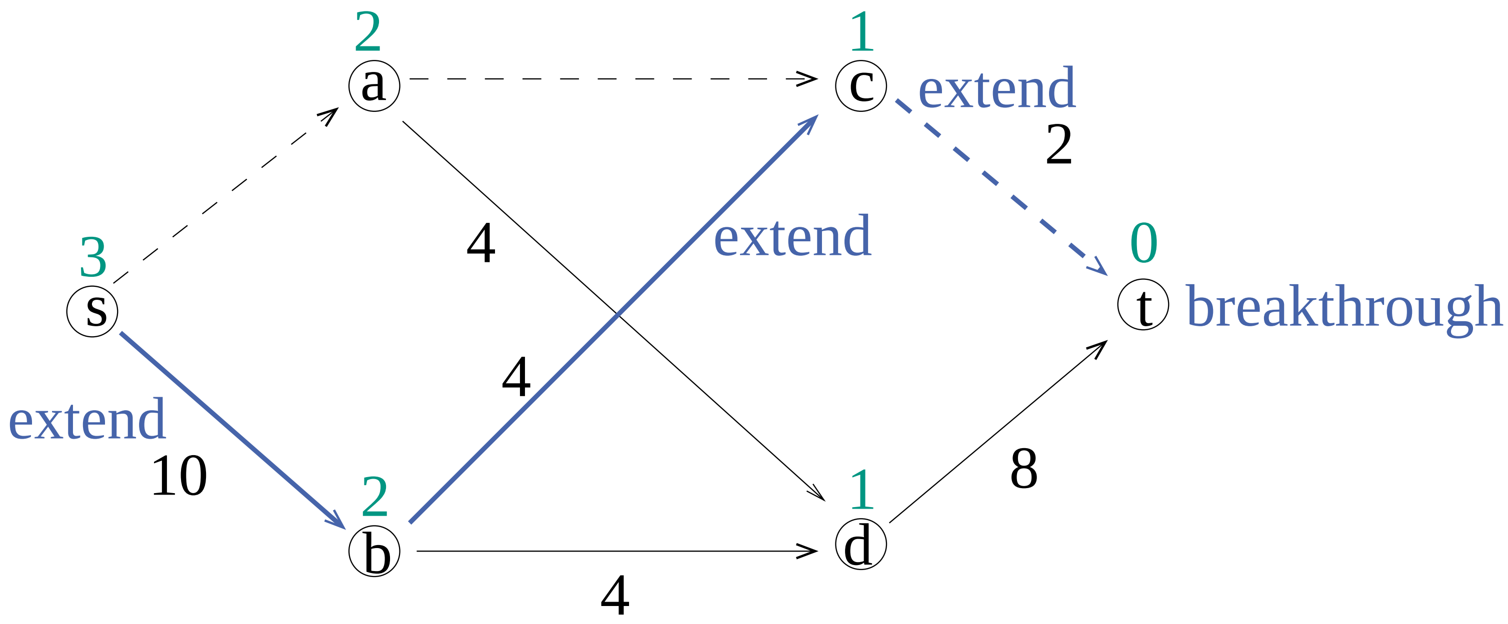
else if $v = s$ **then return** f_b // done

else delete the last edge from p in p and E // retreat

Example

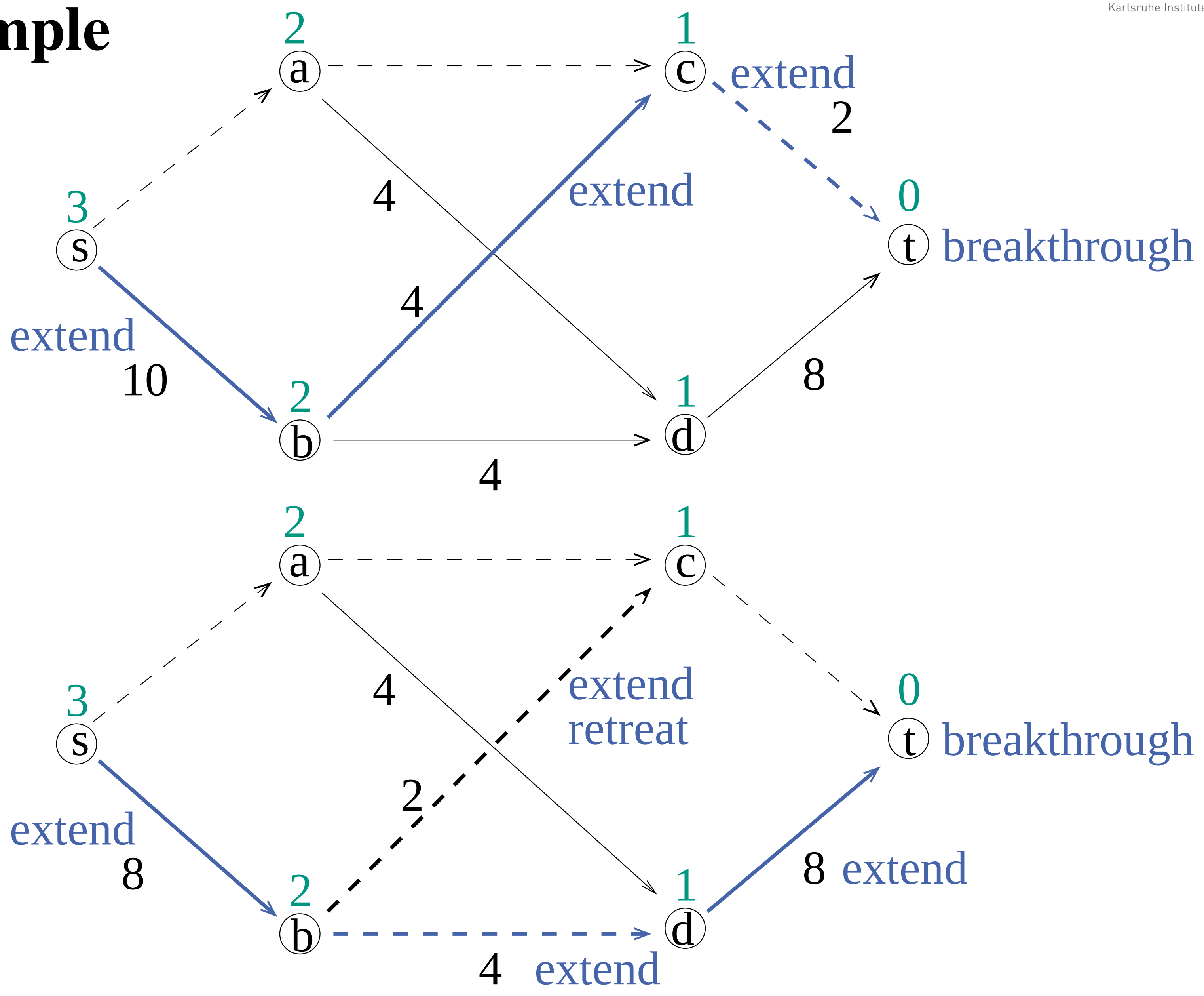


2
10
1
a
c
extend
10
10
12
extend
10
extend
10
4
3
0
s
t
breakthrough
4
10
2
1
8
b
d
4

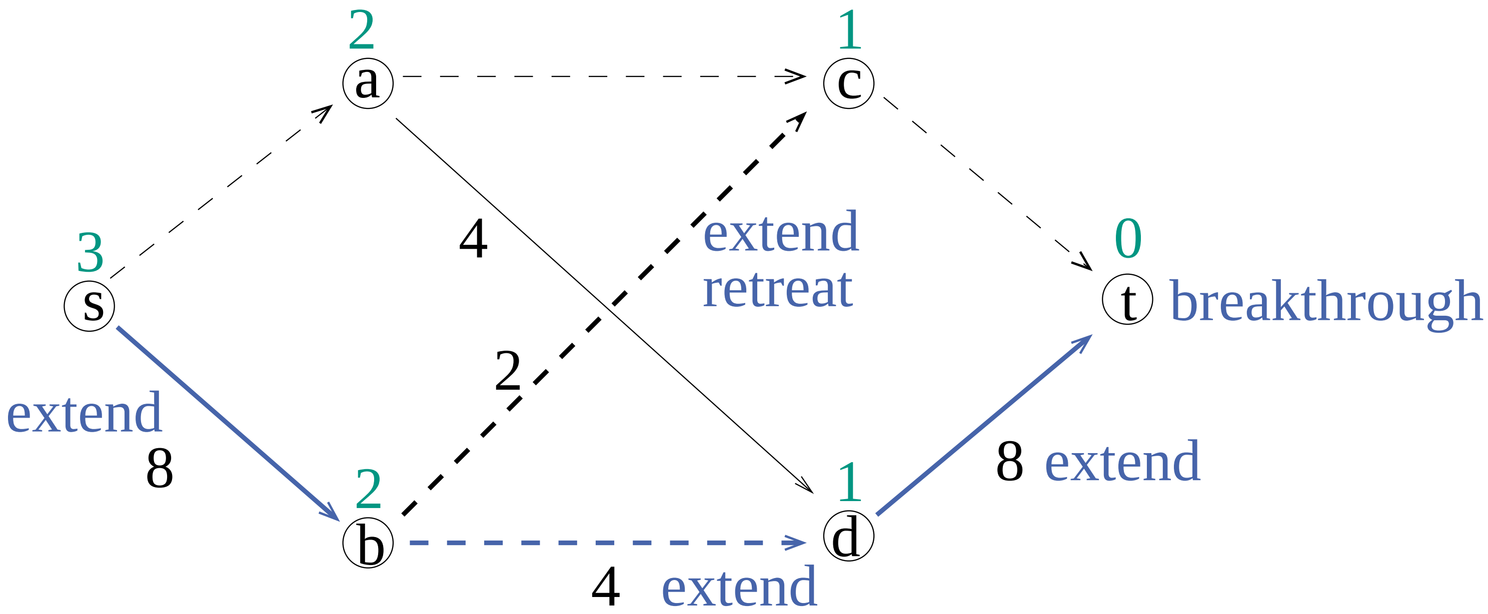


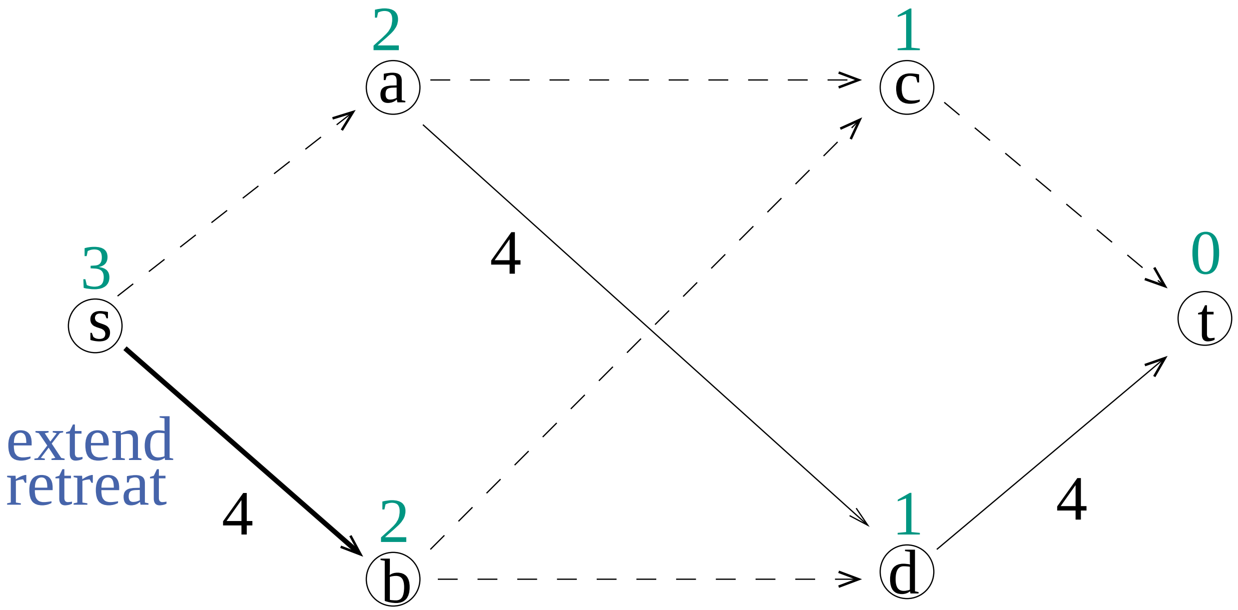
2
1
a
c
extend
2
4
extend
3
0
s
t
breakthrough
4
extend
10
2
1
8
b
d
4

Example


2
a
1
c
extend
2
4
extend
3
s
0
t
breakthrough
4
extend
10
2
b
1
d
8
4
2
a
1
c
4
extend
retreat
3
s
0
t
breakthrough
2
extend
8
2
b
1
d
8 extend
4 extend

Example


2
a
1
c
3
s
4
extend
retreat
0
t
breakthrough
extend
8
2
2
b
1
d
8 extend
4
extend


2
a
1
c
3
s
4
0
t
extend
retreat
4
2
b
1
d
4

Blocking Flows Analysis 1

- running time $\#_{extends} + \#_{retreats} + n \cdot \#_{breakthroughs}$
- $\#_{breakthroughs} \leq m$ – ≥ 1 edge is saturated
- $\#_{retreats} \leq m$ – one edge is removed
- $\#_{extends} \leq \#_{retreats} + n \cdot \#_{breakthroughs}$
  – a retreat cancels 1 extend, a breakthrough cancels $\leq n$ extends

time is $O(m + nm) = O(nm)$

Blocking Flows Analysis 2

Unit capacities:

breakthroughs saturate all edges on p,

i.e., amortized constant cost per edge.

time $O(m+n)$

Blocking Flows Analysis 3

If we use a dynamic tree data structure:

breakthrough (!), retreat, extend is possible in time $\mathrm{O}(\log n)$

$\Rightarrow$

Time $O((m+n)\log n)$

"Theory alert": In practice, this seems to be slower

(few breakthroughs, many retreat, extend ops.)

Dinitz Analysis 1

Lemma 1. *$d(s)$ increases by at least one in each round.*

Proof. not here □

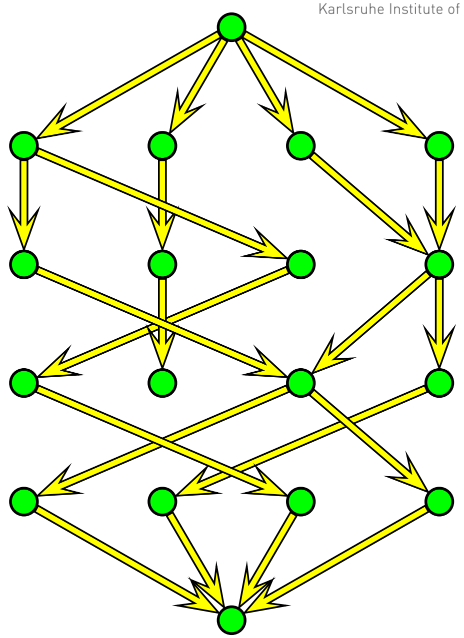
Dinitz Analysis 2

- $\leq n$ rounds
- time $\mathrm{O}(mn)$ each

time $\mathrm{O}(mn^2)$ (strongly polynomial)

time $\mathrm{O}(mn \log n)$ with dynamic trees

Dinitz Analysis 3 – Unit Capacities



Lemma 2. *At most $2\sqrt{m}$ BF computations:*

Proof. Consider iteration $k = \sqrt{m}$.
Cut in layergraph induces cut in
residual graph of capacity at most $\sqrt{m}$.
At most $\sqrt{m}$ additional phases. $\square$

Total time: $O((m+n)\sqrt{m})$

more detailed analysis: $O\left(m \min\left\{m^{1/2}, n^{2/3}\right\}\right)$

Dinitz Analysis 4 – Unit Networks

Unit capacity $+\ \forall v \in V$: $\min\{\text{indegree}(v), \text{outdegree}(v)\} = 1$:

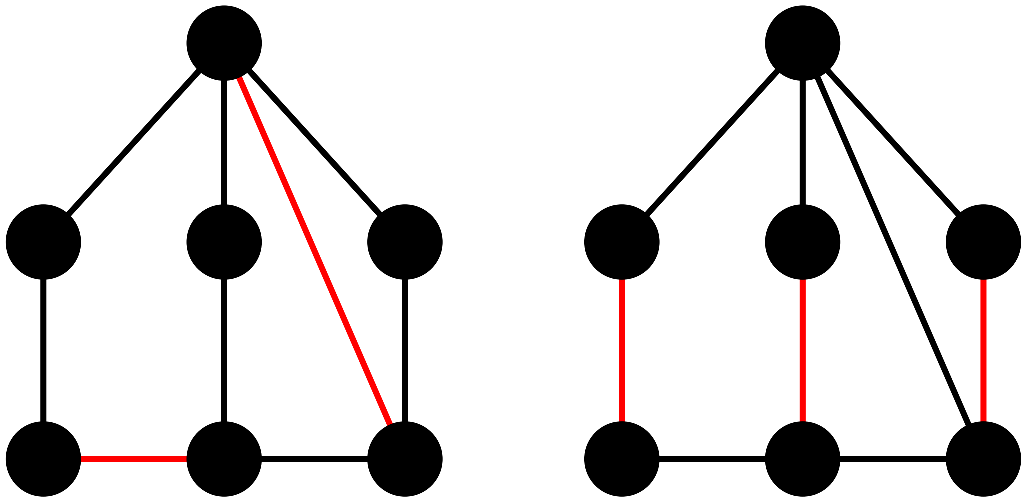
time: $\mathrm{O}((m+n)\sqrt{n})$

Matching

$M \subseteq E$ is a matching in the undirected graph $G = (V, E)$ iff (V, M) has maximum degree ≤ 1.

M is maximal if $\nexists e \in E \setminus M : M \cup \{e\}$ is a matching.

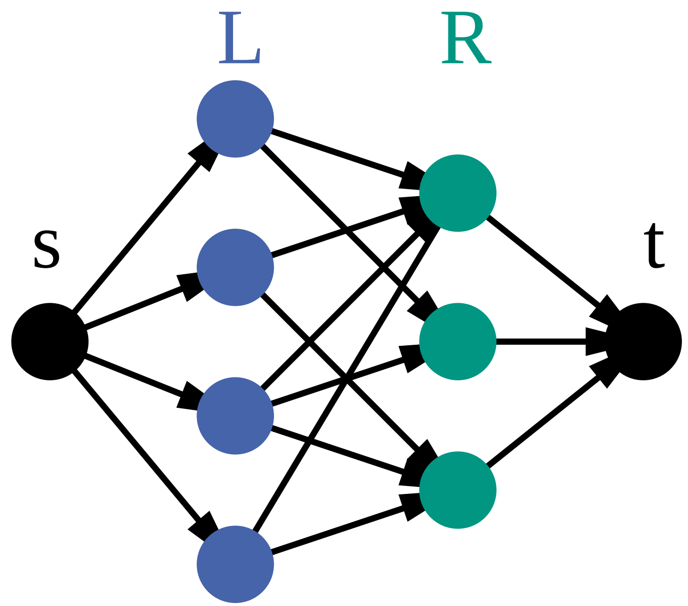
M has maximum cardinality if $\nexists$matching $M' : |M'| > |M|$



Maximum Cardinality Bipartite Matching

in $(L \cup R, E)$. Model as a unit network maximum flow problem

$$(\{s\} \cup L \cup R \cup \{t\}, \{(s,u) : u \in L\} \cup E \cup \{(v,t) : v \in R\})$$

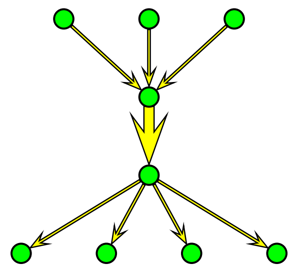


Dinitz algorithm yields $\mathrm{O}((n+m)\sqrt{n})$ algorithm

Similar Performance for Weighted Graphs?

time: $\mathrm{O}\left(m\min\left\{m^{1/2}, n^{2/3}\right\}\log C\right)$ [Goldberg Rao 97]

Problem: Fat edges between layers ruin the argument



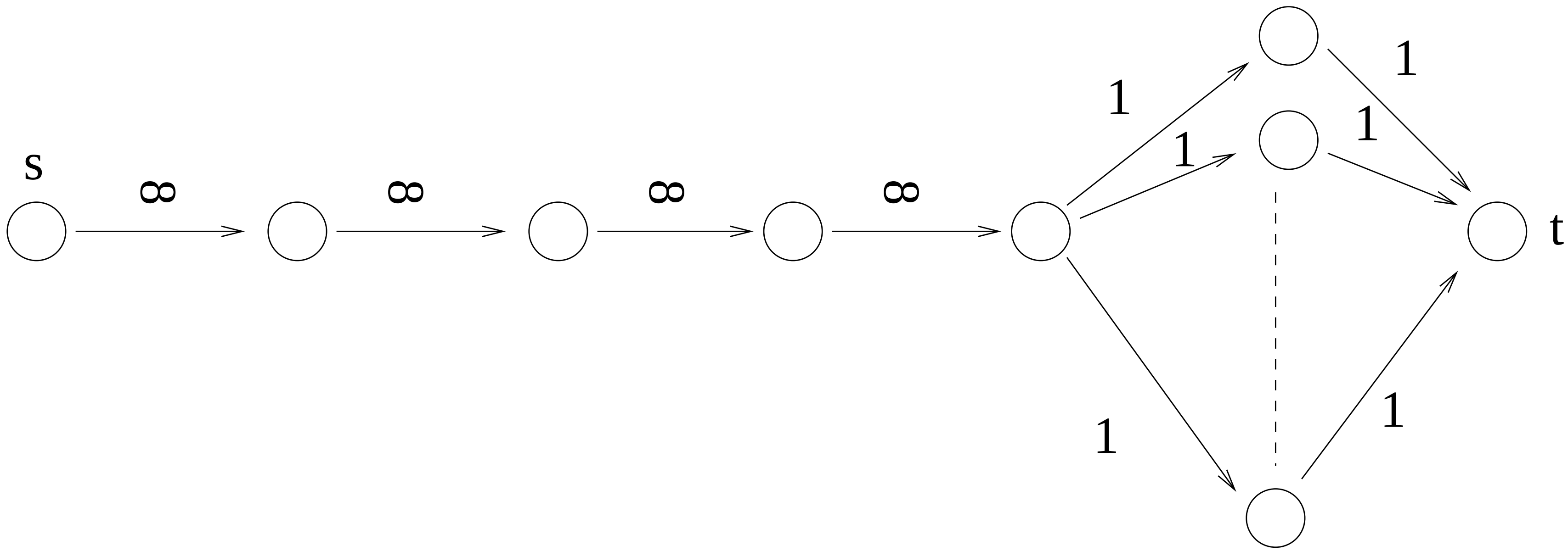
Idea: scale a parameter Δ from small to large.

Contract SCCs of fat edges (capacity $> \Delta$)

Experiments [Hagerup, Sanders Träff 98]:

Sometimes best algorithm usually slower than preflow push

Disadvantage of augmenting paths algorithms



Preflow-Push Algorithms

Preflow f: a flow where the flow conservation constraint is relaxed to

$$\text{excess}(v) := \overbrace{\sum_{(u,v)\in E} f_{(u,v)}}^{\text{inflow}} - \overbrace{\sum_{(v,w)\in E} f_{(v,w)}}^{\text{outflow}} \geq 0 \ .$$

$v \in V \setminus \{s,t\}$ is active iff $\text{excess}(v) > 0$

Procedure push$(e = (v,w), \delta)$

- **assert** $\delta > 0 \quad \wedge \quad \text{excess}(v) \geq \delta$
- **assert** residual capacity of $e \geq \delta$
- $\text{excess}(v) -= \delta$
- $\text{excess}(w) += \delta$
- **if** e is reverse edge **then** $f(\text{reverse}(e)) -= \delta$
- **else** $f(e) += \delta$

Level Function

Idea: make progress by pushing towards t

Maintain

an approximation $d(v)$ of the BFS distance from v to t in G_f.

invariant $d(t) = 0$

invariant $d(s) = n$

invariant $\forall (v,w) \in E_f : d(v) \leq d(w) + 1$ // no steep edges

Edge directions of $e = (v,w)$

steep: $d(w) < d(v) - 1$

downward: $d(w) < d(v)$

horizontal: $d(w) = d(v)$

upward: $d(w) > d(v)$

Procedure genericPreflowPush$(G=(V,E),f)$

  forall $e=(s,v)\in E$ **do** push$(e,c(e))$ // saturate

  $d(s):=n$

  $d(v):=0$ for all other nodes

  while $\exists v\in V\setminus\{s,t\}$: excess$(v)>0$ **do** // active node

    if $\exists e=(v,w)\in E_f : d(w)<d(v)$ **then** // eligible edge

      choose some $\delta\leq\min\left\{\text{excess}(v),c_e^f\right\}$

      push(e,δ) // no new steep edges

    else $d(v)$++ // relabel. No new steep edges

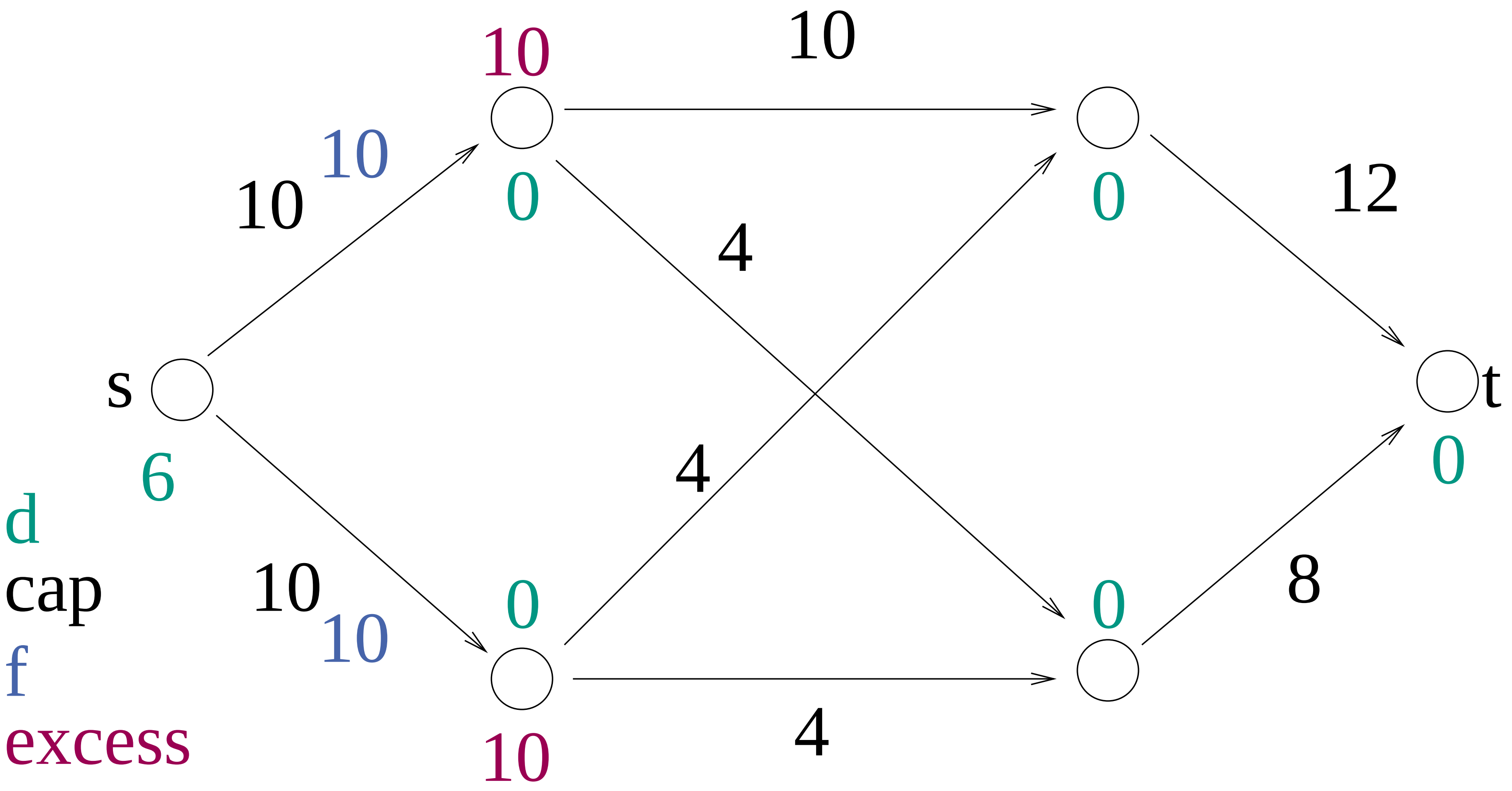
Obvious choice for δ : $\delta=\min\left\{\text{excess}(v),c_e^f\right\}$

saturating push: $\delta=c_e^f$

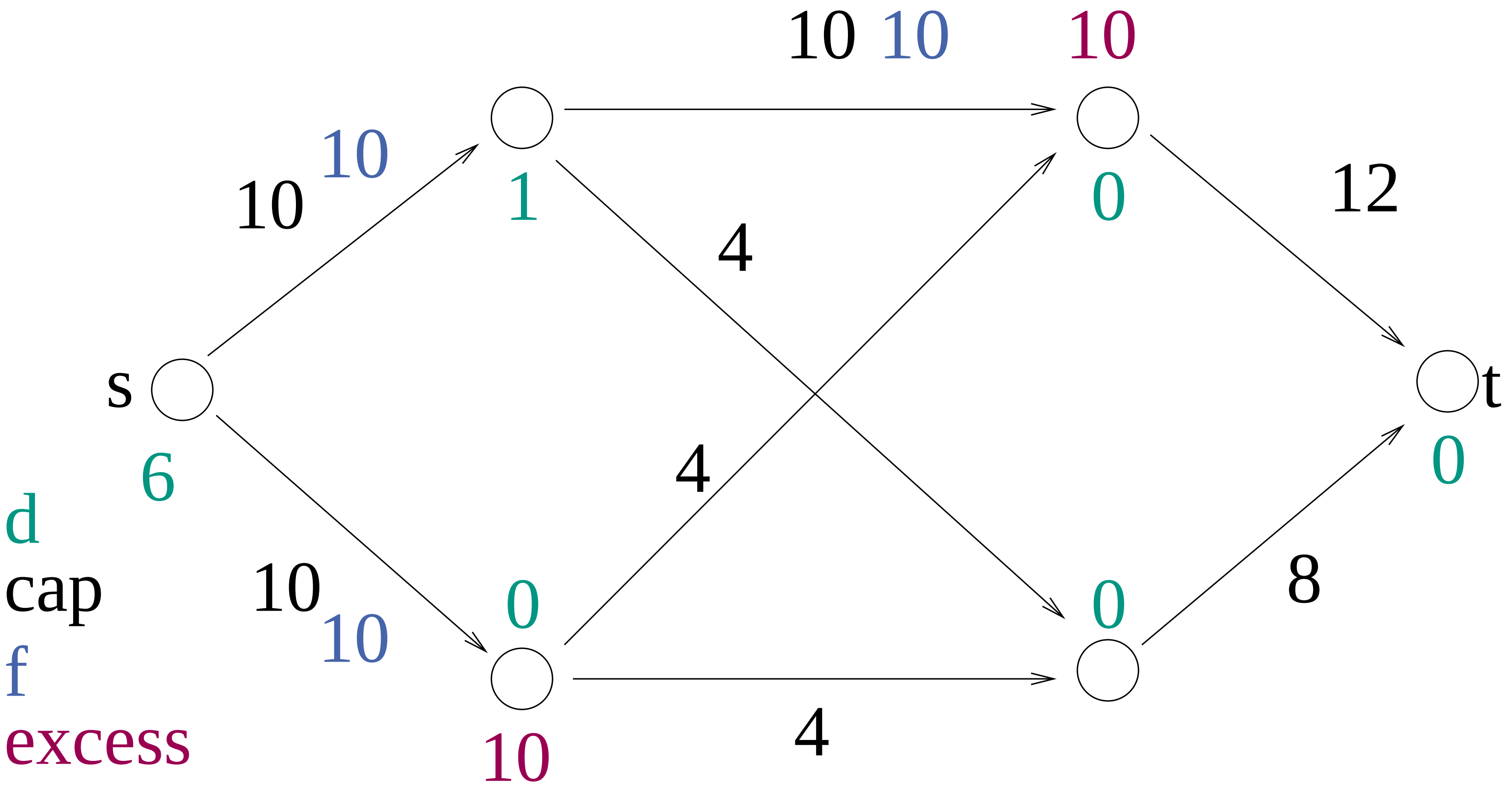
nonsaturating push: $\delta<c_e^f$

To be filled in: How to select active nodes and eligible edges?

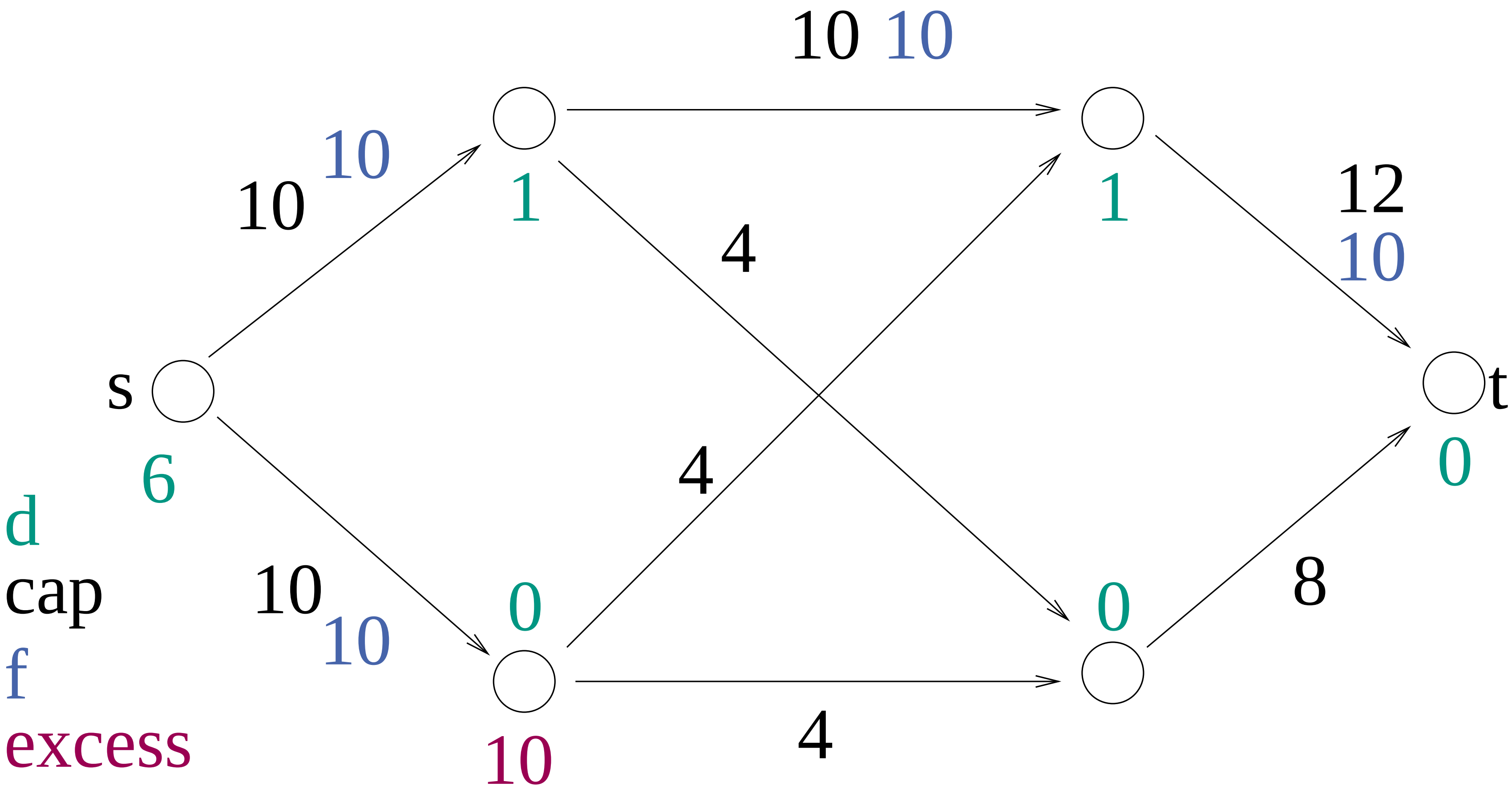
Example



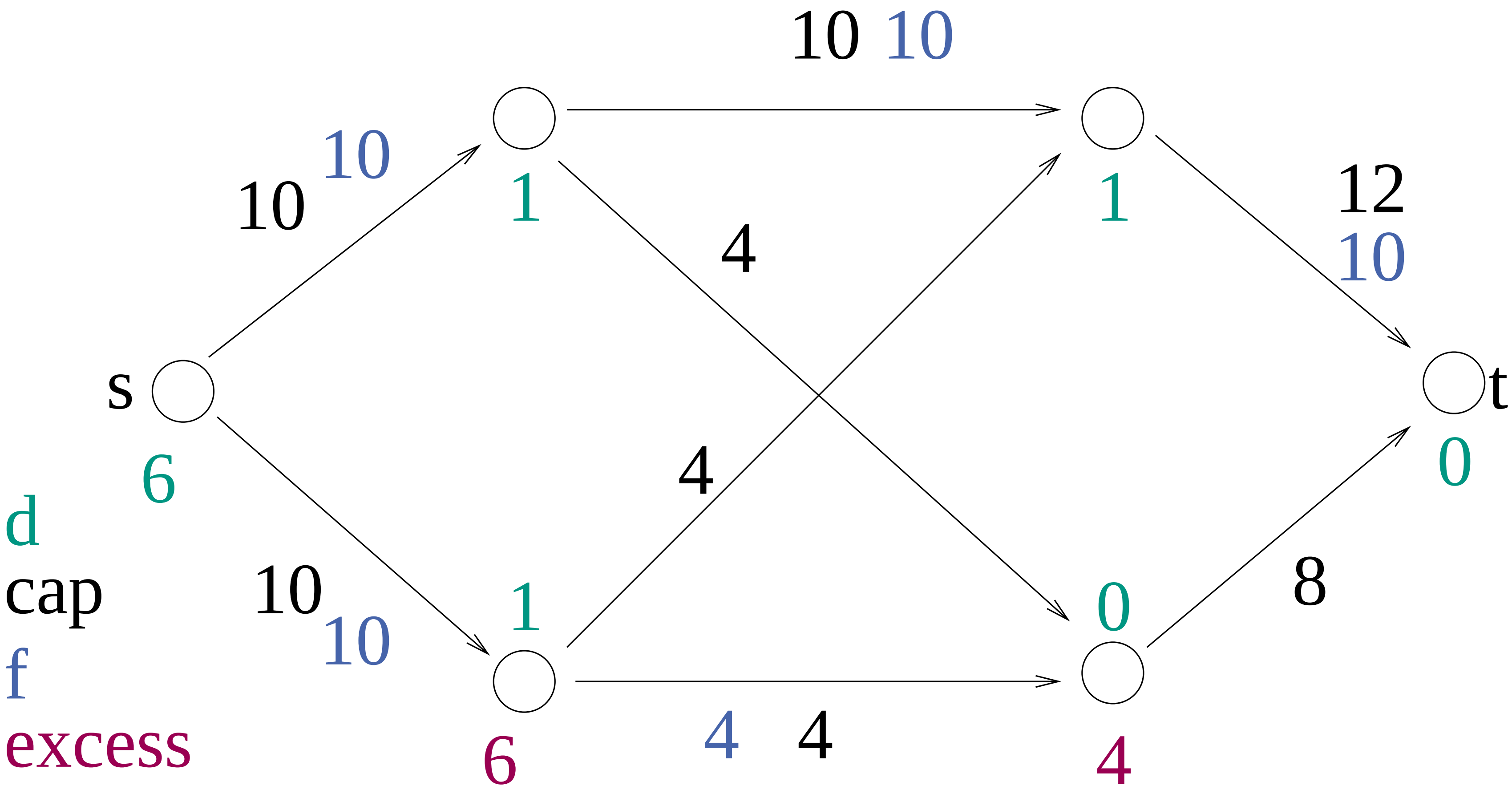
Example



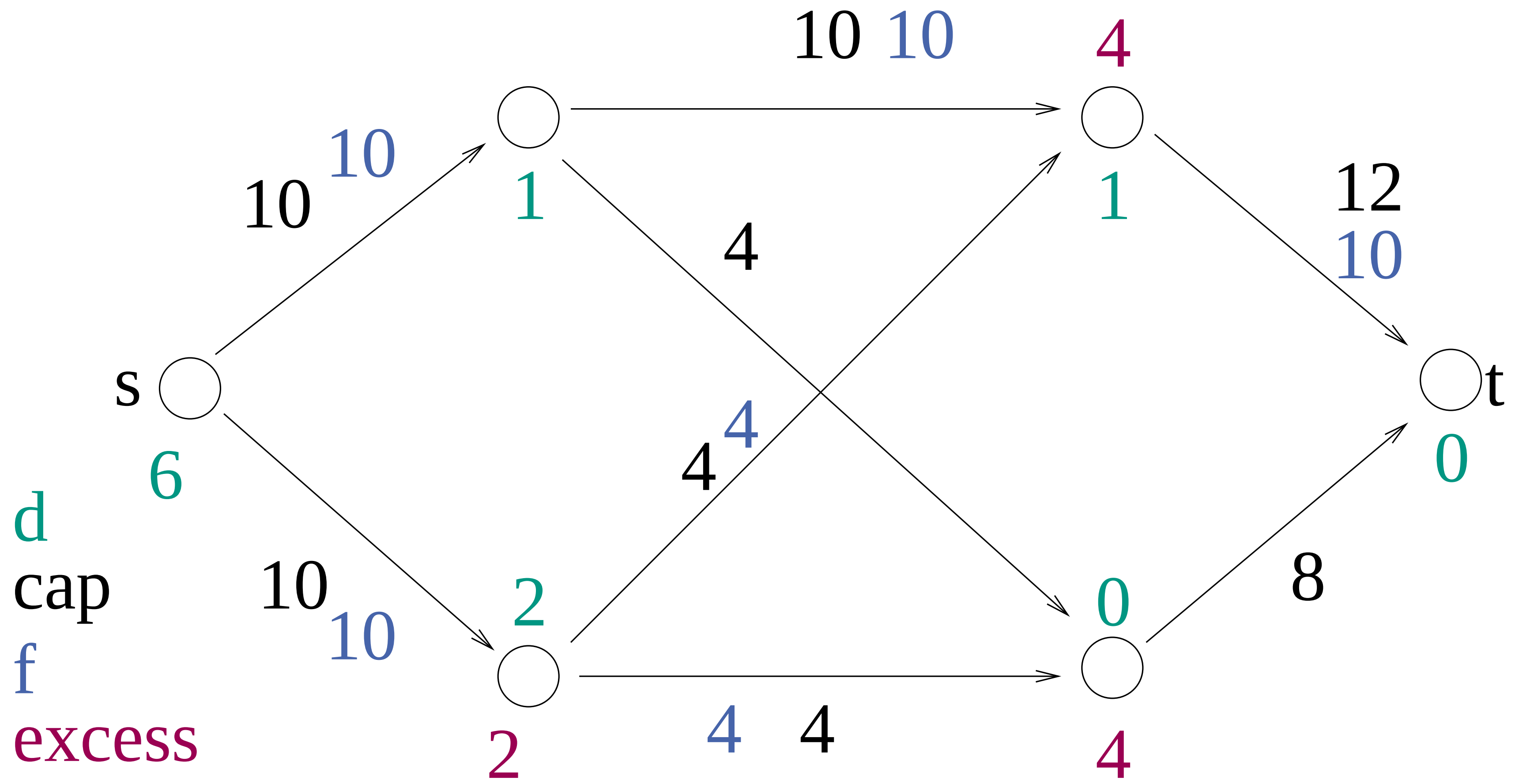
Example



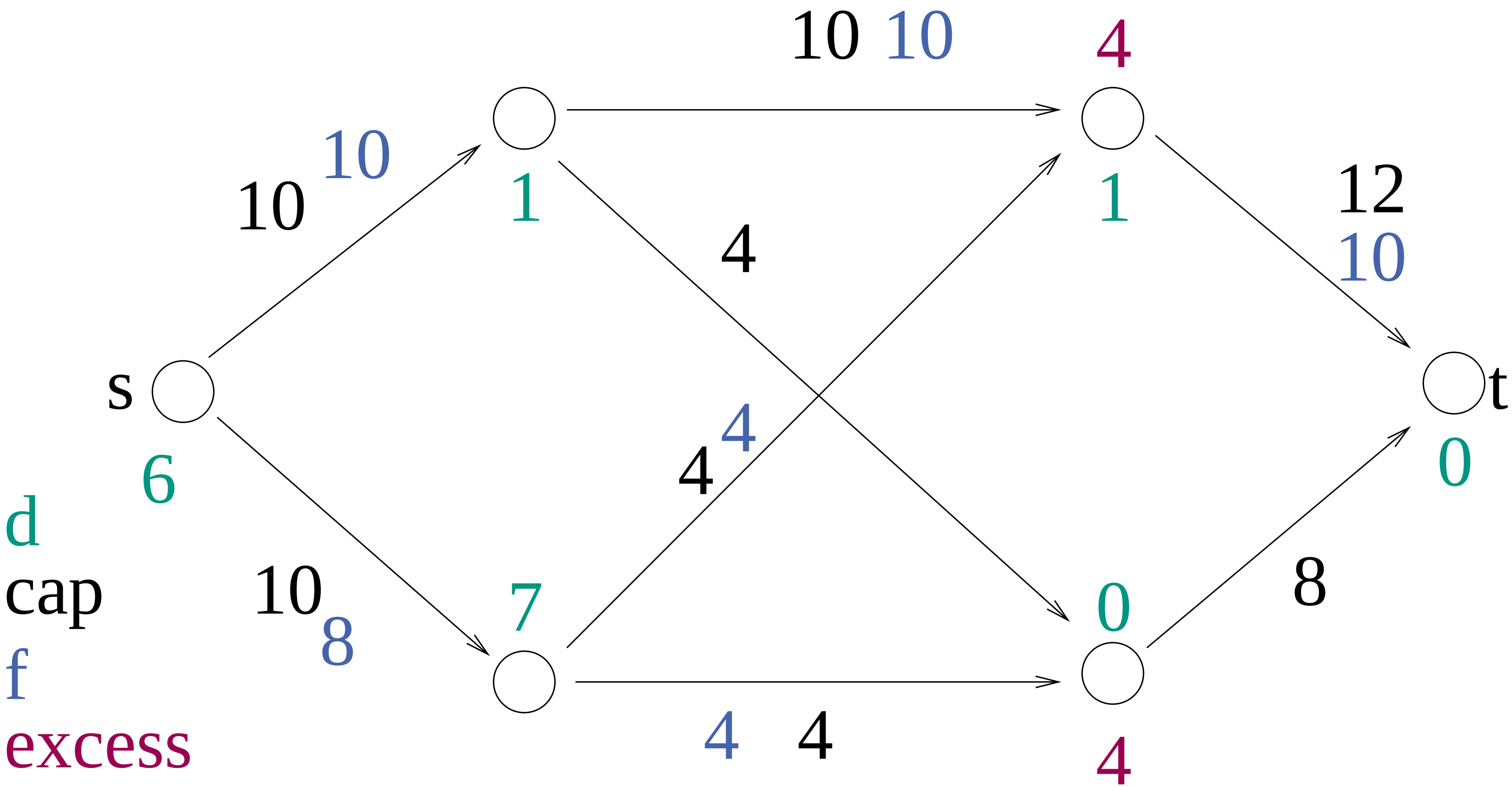
Example



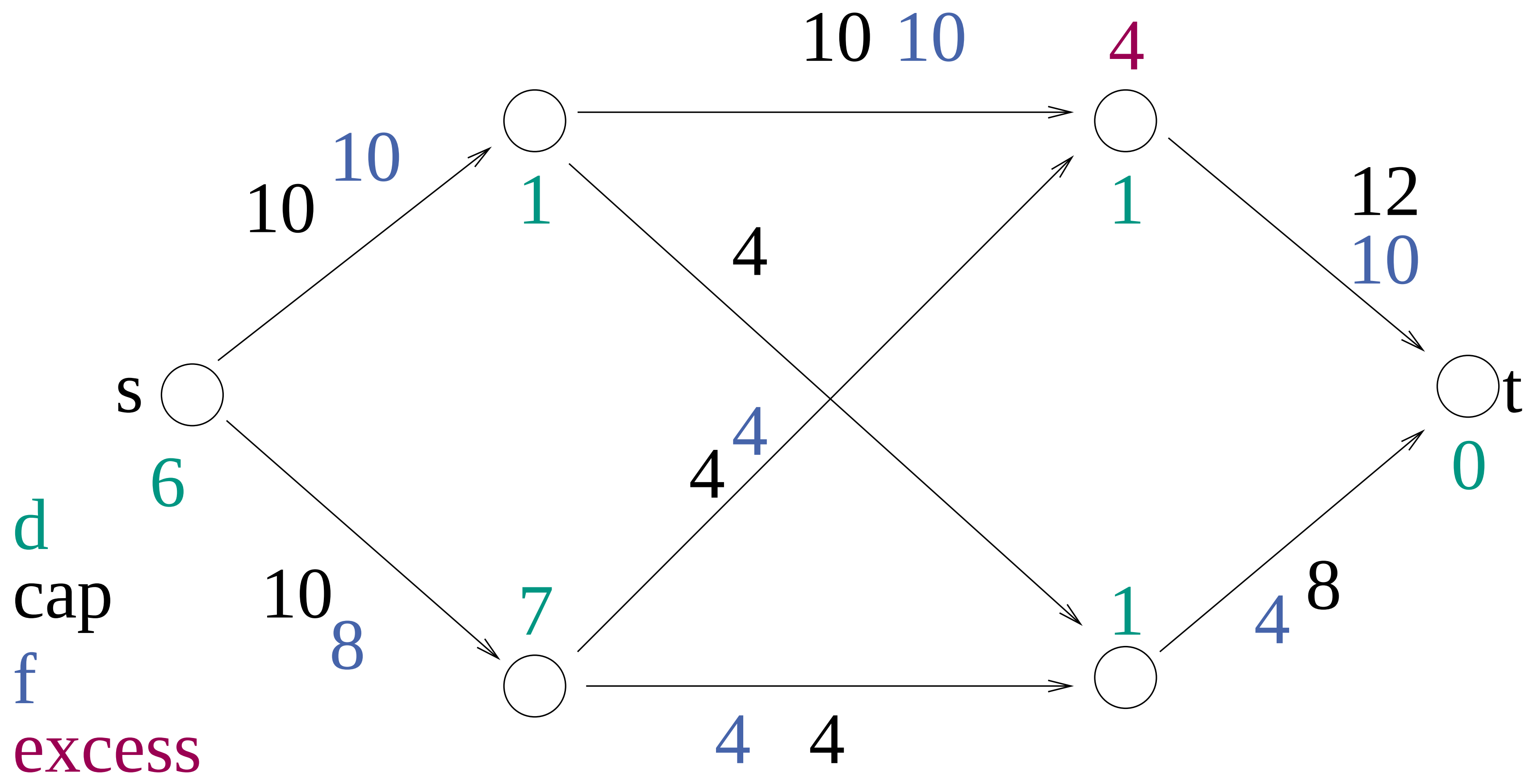
Example



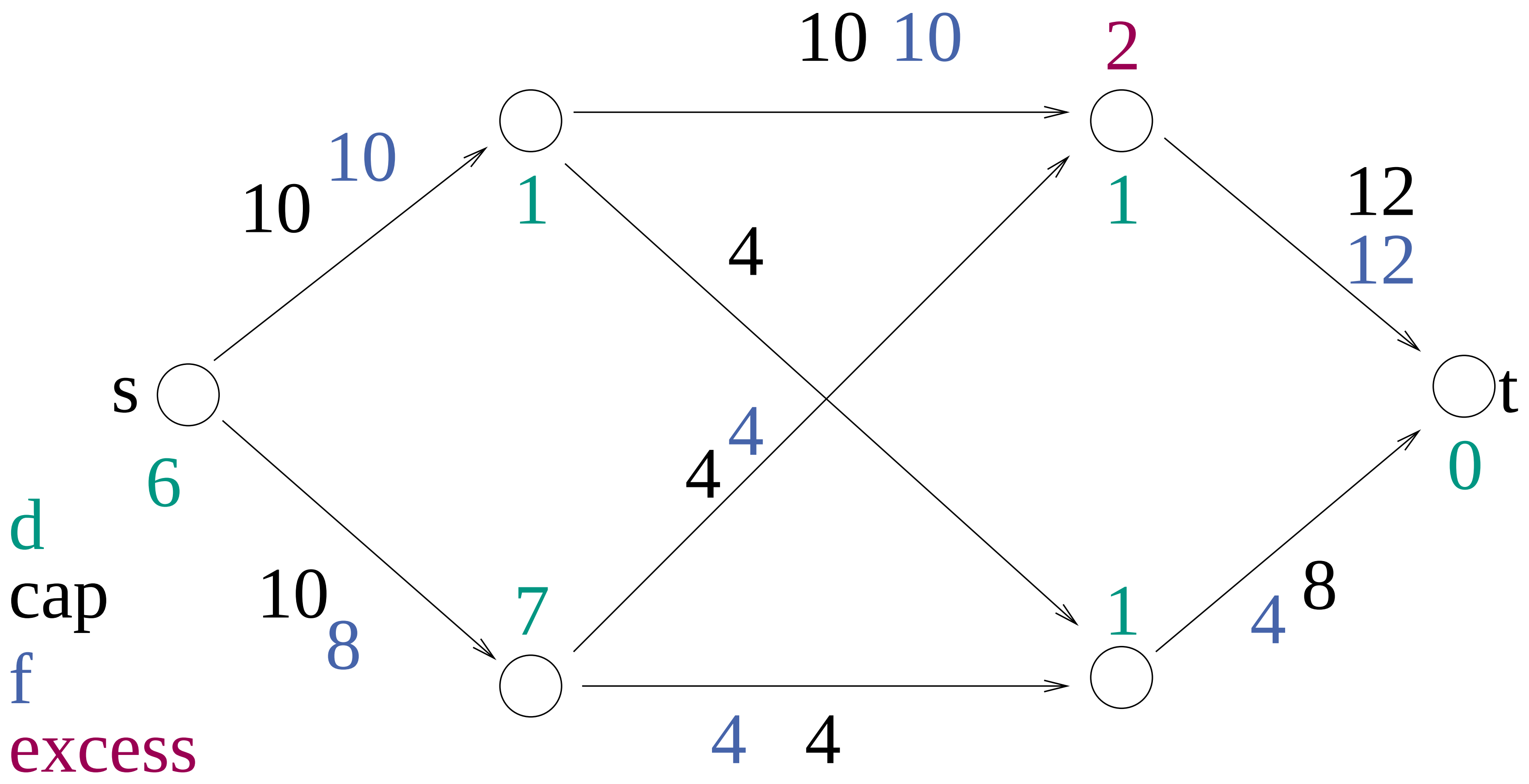
Example



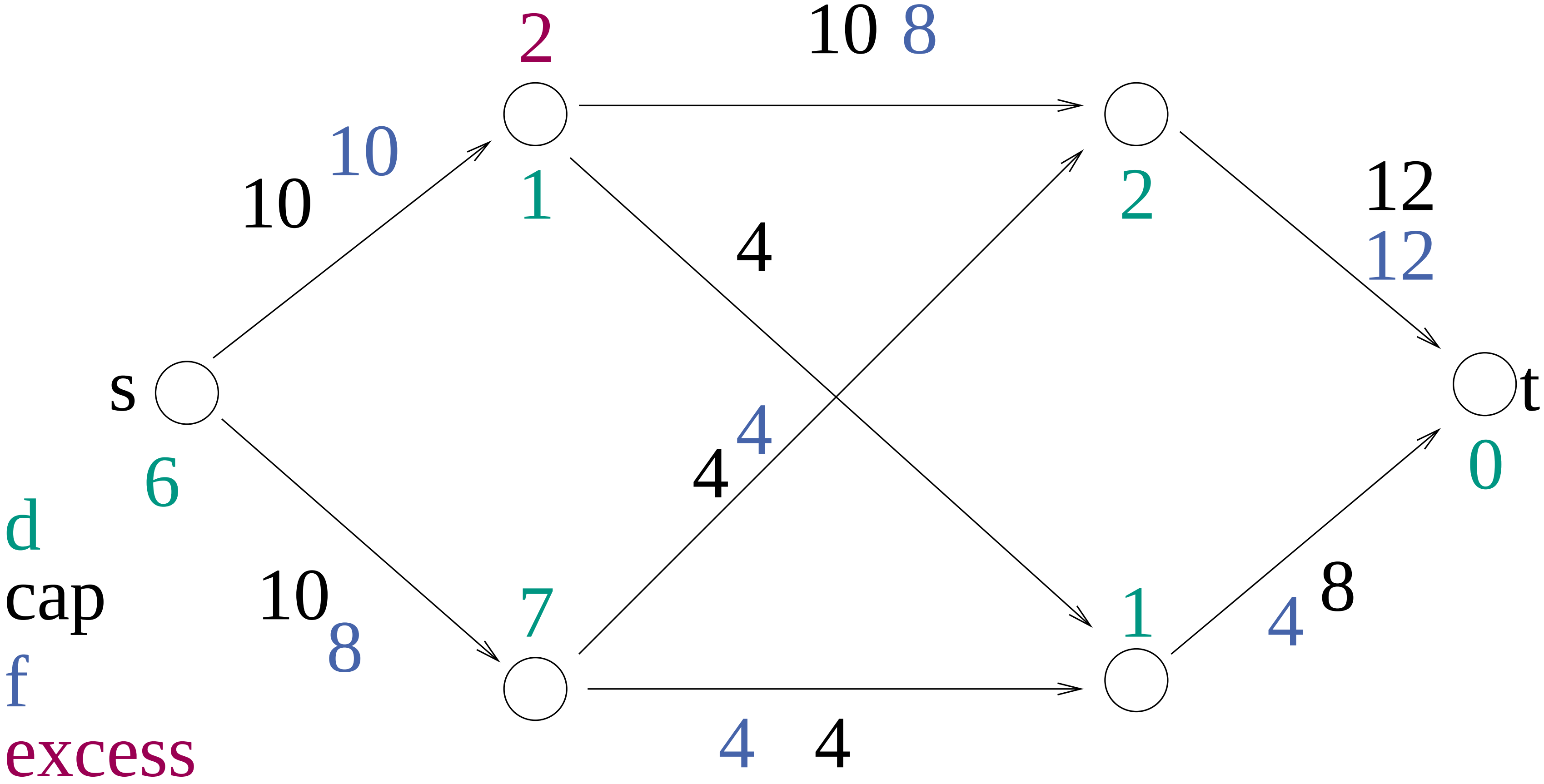
Example



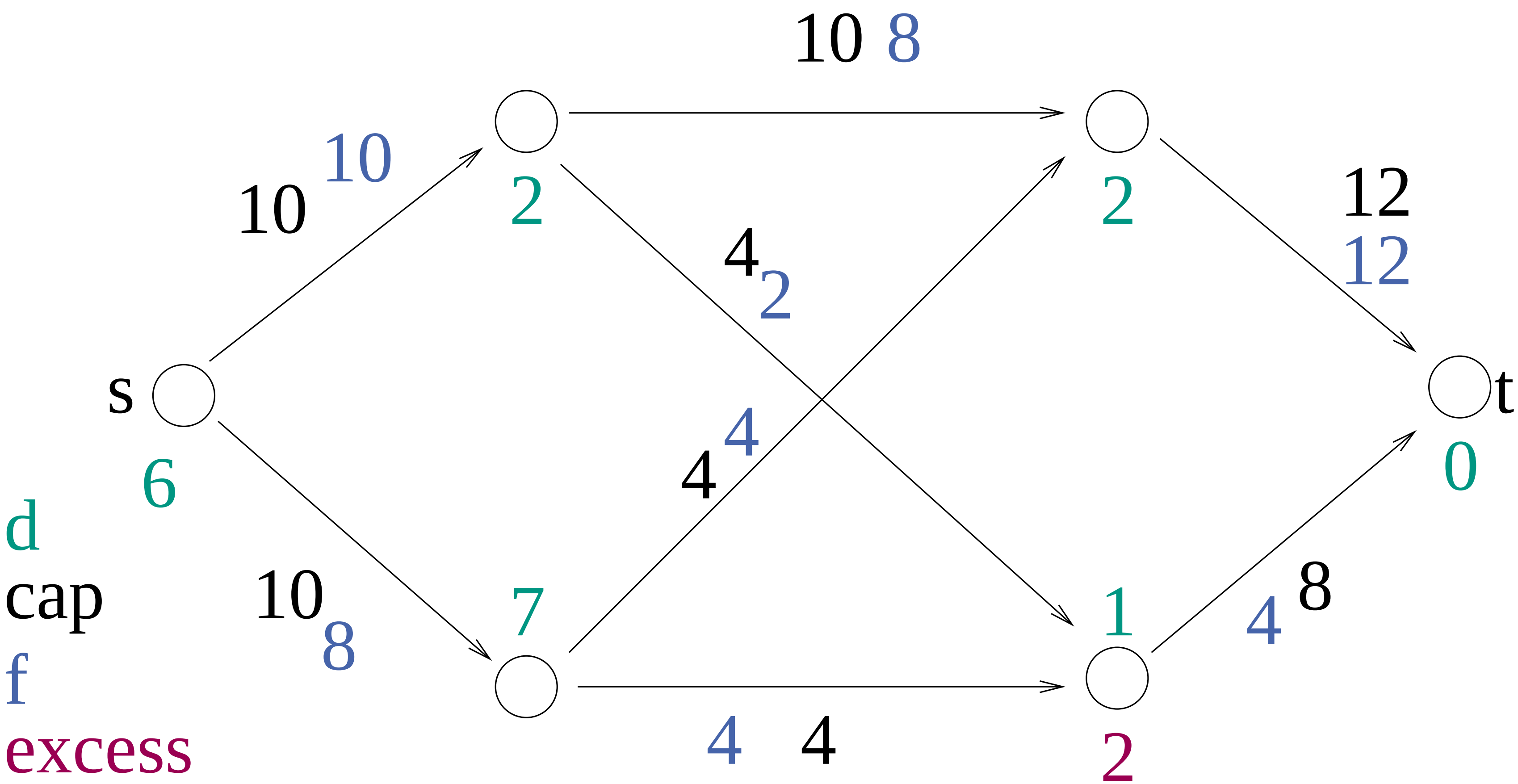
Example



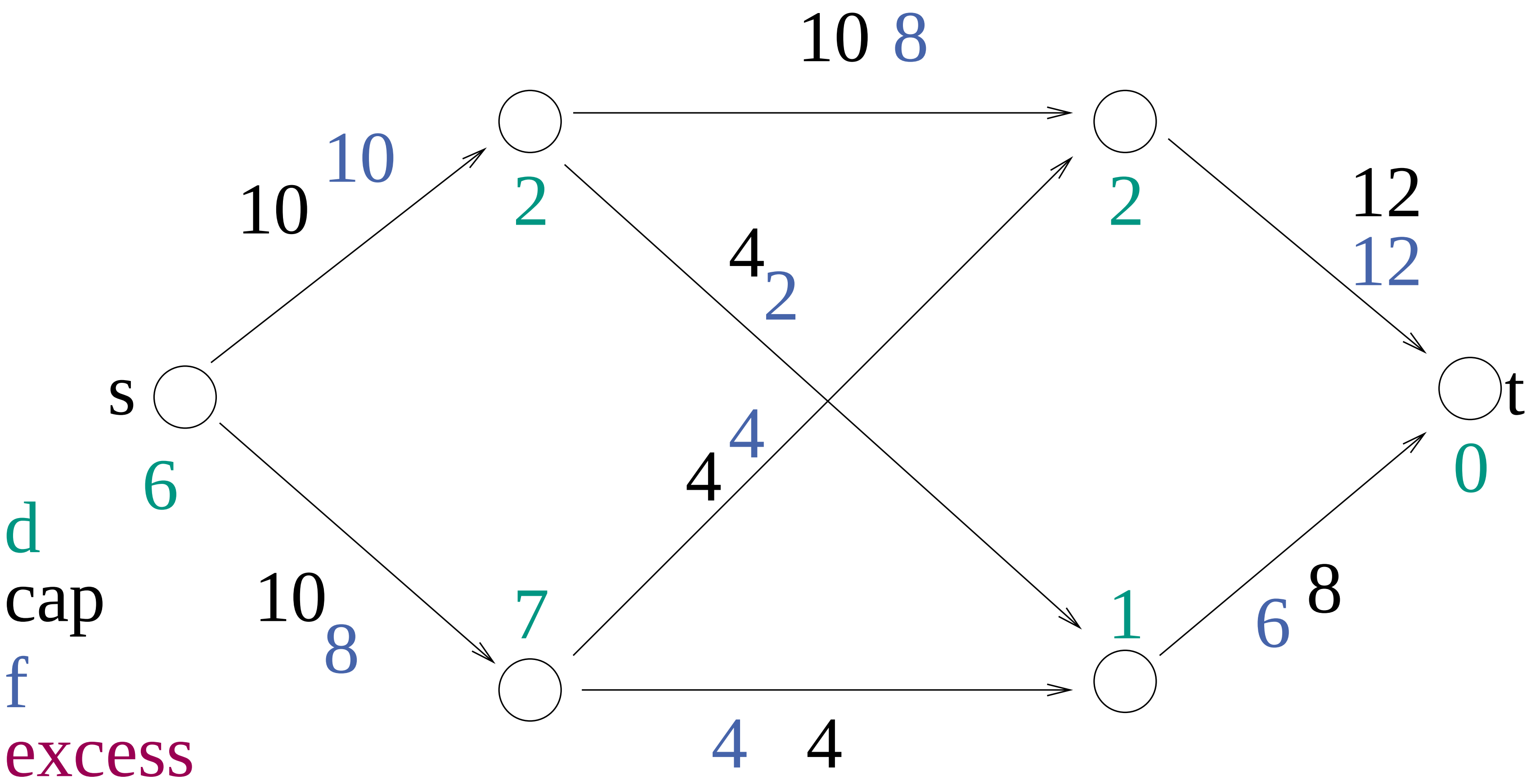
Example



Example



Example



12 pushes in total

Partial Correctness

Lemma 3. *When* genericPreflowPush *terminates*
f is a maximal flow.

Proof.
f is a flow since $\forall v \in V \setminus \{s,t\} : \mathsf{excess}(v) = 0$.

To show that f is maximal, it suffices to show that
$\nexists$ path $p = \langle s, \ldots, t \rangle \in G_f$ (Max-Flow Min-Cut Theorem):
Since $d(s) = n$, $d(t) = 0$, p would have to contain steep edges.
That contradicts the invariant. $\square$

Lemma 4. *For any cut (S,T),*

$$\sum_{u\in S} excess(u) = \sum_{e\in E\cap(T\times S)} f(e) - \sum_{e\in E\cap(S\times T)} f(e),$$

Proof:

$$\sum_{u\in S} excess(u) = \sum_{u\in S}\left(\sum_{(v,u)\in E} f((v,u)) - \sum_{(u,v)\in E} f((u,v))\right)$$

Contributions of edge e to sum:

S to T: $-f(e)$

T to S: $f(e)$

within S: $f(e)-f(e)=0$

within T: 0 ■

Lemma 5.

$\forall$ *active nodes* v : excess$(v) > 0 \Rightarrow \exists$ *path* $\langle v, \ldots, s\rangle \in G_f$

Intuition: what got there can always go back.

Proof. $S := \{u \in V : \exists \text{ path } \langle v, \ldots u\rangle \in G_f\}$, $T := V \setminus S$. Then

$$\sum_{u \in S} excess(u) = \sum_{e \in E \cap (T \times S)} f(e) - \sum_{e \in E \cap (S \times T)} f(e),$$

$\forall (u,w) \in E_f : u \in S \Rightarrow w \in S$ by Def. of G_f, S

$\Rightarrow \forall e = (u,w) \in E \cap (T \times S) : f(e) = 0$ Otherwise $(w,u) \in E_f$

Hence, $\sum_{u \in S} excess(u) \leq 0$

Only the negative excess of s can outweigh excess$(v) > 0$.

Hence $s \in S$. $\square$

Lemma 6.

$\forall v \in V : d(v) < 2n$

Proof.

Suppose v is lifted to $d(v) = 2n$.

By the Lemma 2, there is a (simple) path p to s in G_f.

p has at most $n-1$ nodes

$d(s) = n$.

Hence $d(v) < 2n$. Contradiction (no steep edges). $\square$

Lemma 7. *# Relabel operations* $\leq 2n^2$

Proof. $d(v) \leq 2n$, i.e., v is relabeled at most $2n$ times.
Hence, at most $|V| \cdot 2n = 2n^2$ relabel operations. $\square$

Lemma 8. *# saturating pushes* $\leq nm$

Proof.

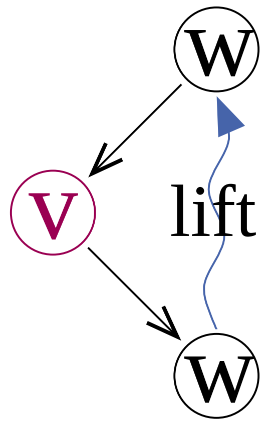
We show that there are at most n sat. pushes over any edge $e = (v, w)$.

A saturating push(e, δ) removes e from E_f.

Only a push on (w, v) can reinsert e into E_f.

For this to happen, w must be lifted at least two levels.

Hence, at most $2n/2 = n$ saturating pushes over (v, w)



$\square$

Lemma 9. *# nonsaturating pushes* $= \mathrm{O}(n^2 m)$
if $\delta = \min\left\{\mathsf{excess}(v), c_e^f\right\}$
for arbitrary node and edge selection rules.
(arbitrary-preflow-push)

Proof. $\Phi := \sum_{\{v : v \text{ is active}\}} d(v)$. (Potential)

$\Phi = 0$ initially and at the end (no active nodes left!)

Operation	$\Delta(\Phi)$	How many times?	Total effect
relabel	1	$\leq 2n^2$	$\leq 2n^2$
saturating push	$\leq 2n$	$\leq nm$	$\leq 2n^2 m$
nonsaturating push	≤ -1		

$\Phi \geq 0$ always. $\square$

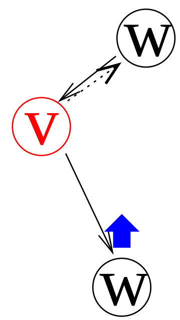
Searching for Eligible Edges

Every node v maintains a currentEdge pointer to its sequence of outgoing edges in G_f.

invariant no edge $e = (v, w)$ to the left of currentEdge is eligible

Invariant violations?

- ☐ relabel(v)? Reset currentEdge $(\leq 2n\times)$
- ☐ relabel(w)? No, no steep edges.
- ☐ push(w, v)? $\Rightarrow (v, w)$ is upward



Lemma 10.

Total cost for searching $\leq \sum_{v \in V} 2n \cdot \text{degree}(v) = 4nm = \mathrm{O}(nm)$

Theorem 11. *Arbitrary Preflow Push finds a maximum flow in time* $O(n^2 m)$.

Proof.

Lemma 3: partial correctness

Initialization in time $O(n+m)$.

Maintain set (e.g., stack, FIFO) of active nodes.

Use reverse edge pointers to implement push.

Lemma 7: $2n^2$ relabel operations

Lemma 8: nm saturating pushes

Lemma 9: $O(n^2 m)$ nonsaturating pushes

Lemma 10: $O(nm)$ search time for eligible edges

Total time $O(n^2 m)$ □

FIFO Preflow push

Examine a node: Saturating pushes until nonsaturating push or relabel.

Examine all nodes in phases (or use FIFO queue).

Theorem: time $\mathrm{O}\left(n^3\right)$

Proof: not here

Highest Level Preflow Push

Always select active nodes that maximize $d(v)$

Use bucket priority queue (insert, increaseKey, deleteMax)

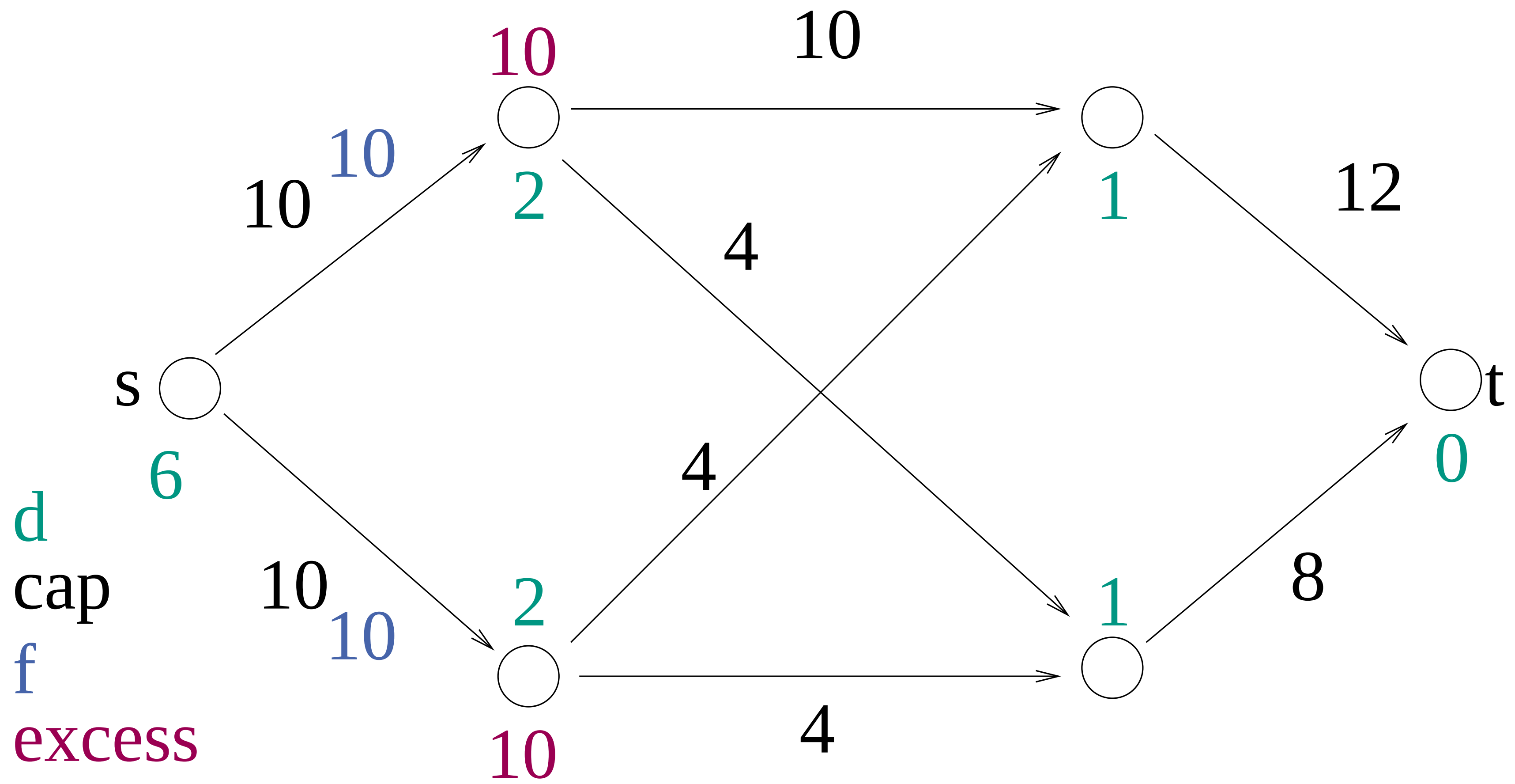
not monotone (!) but relabels “pay” for scan operations

Lemma 12. *At most $n^2\sqrt{m}$ nonsaturating pushes.*

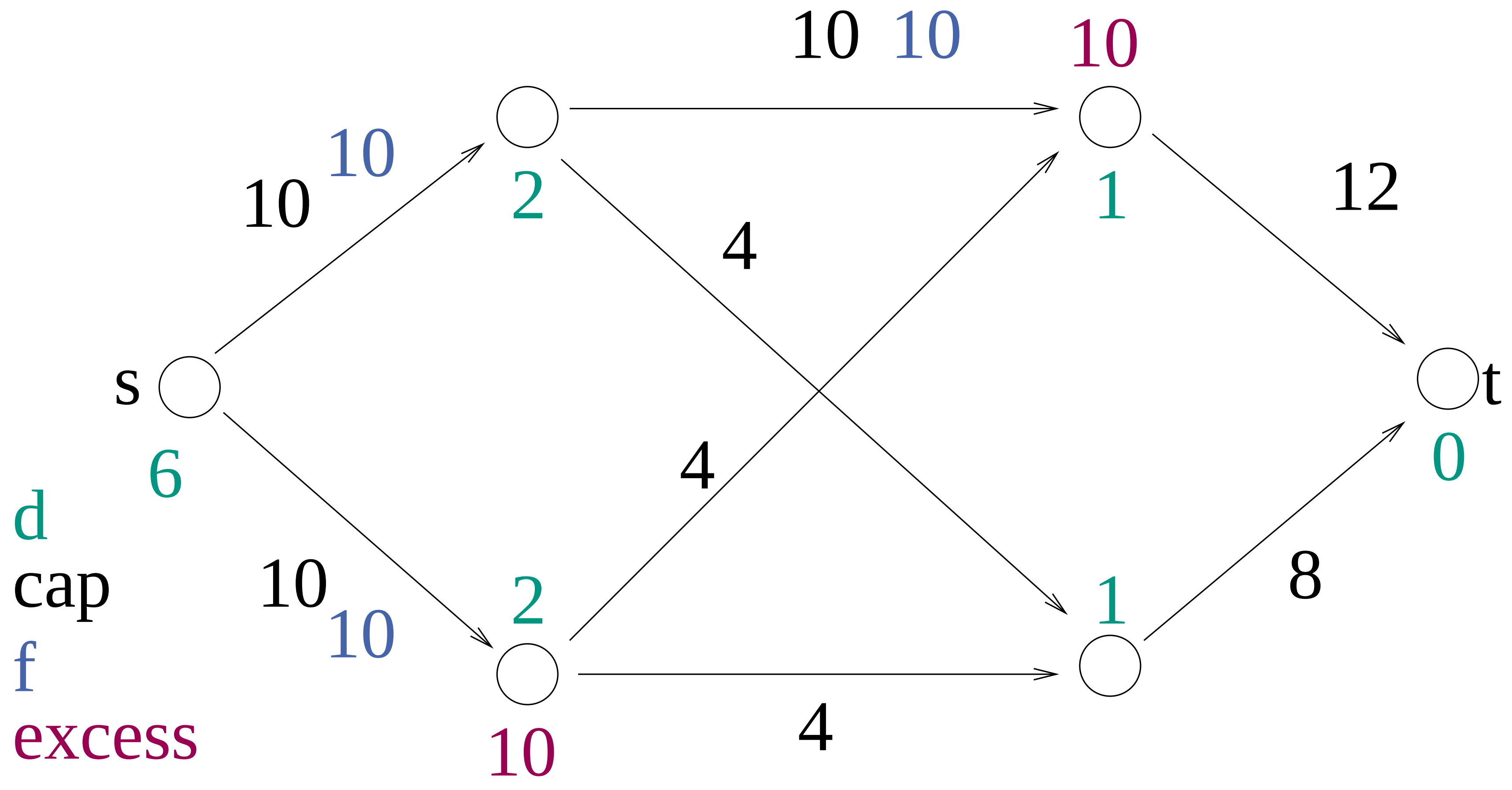
Proof. later $\square$

Theorem 13. *Highest Level Preflow Push finds a maximum flow in time* $\mathrm{O}\left(n^2\sqrt{m}\right)$.

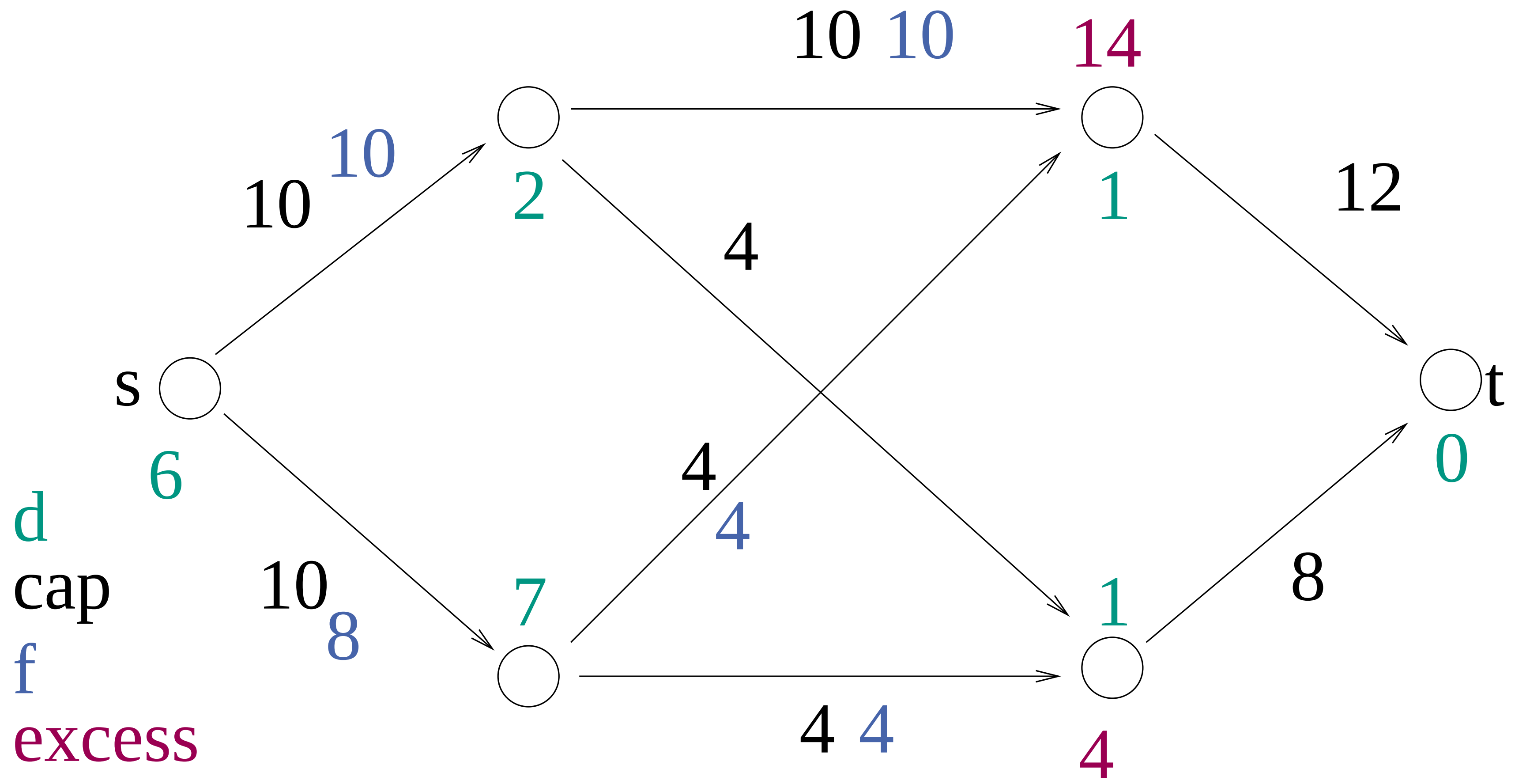
Example



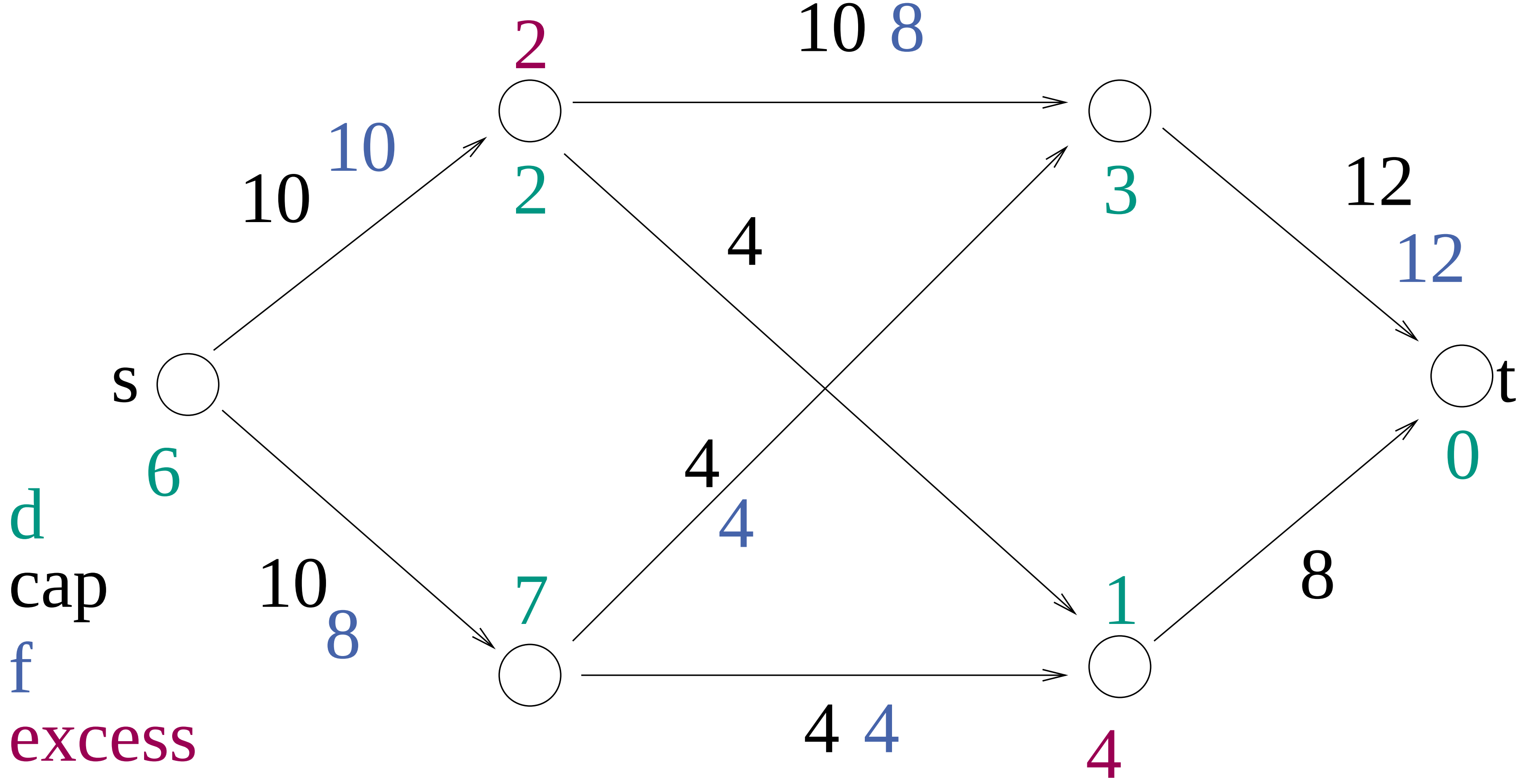
Example



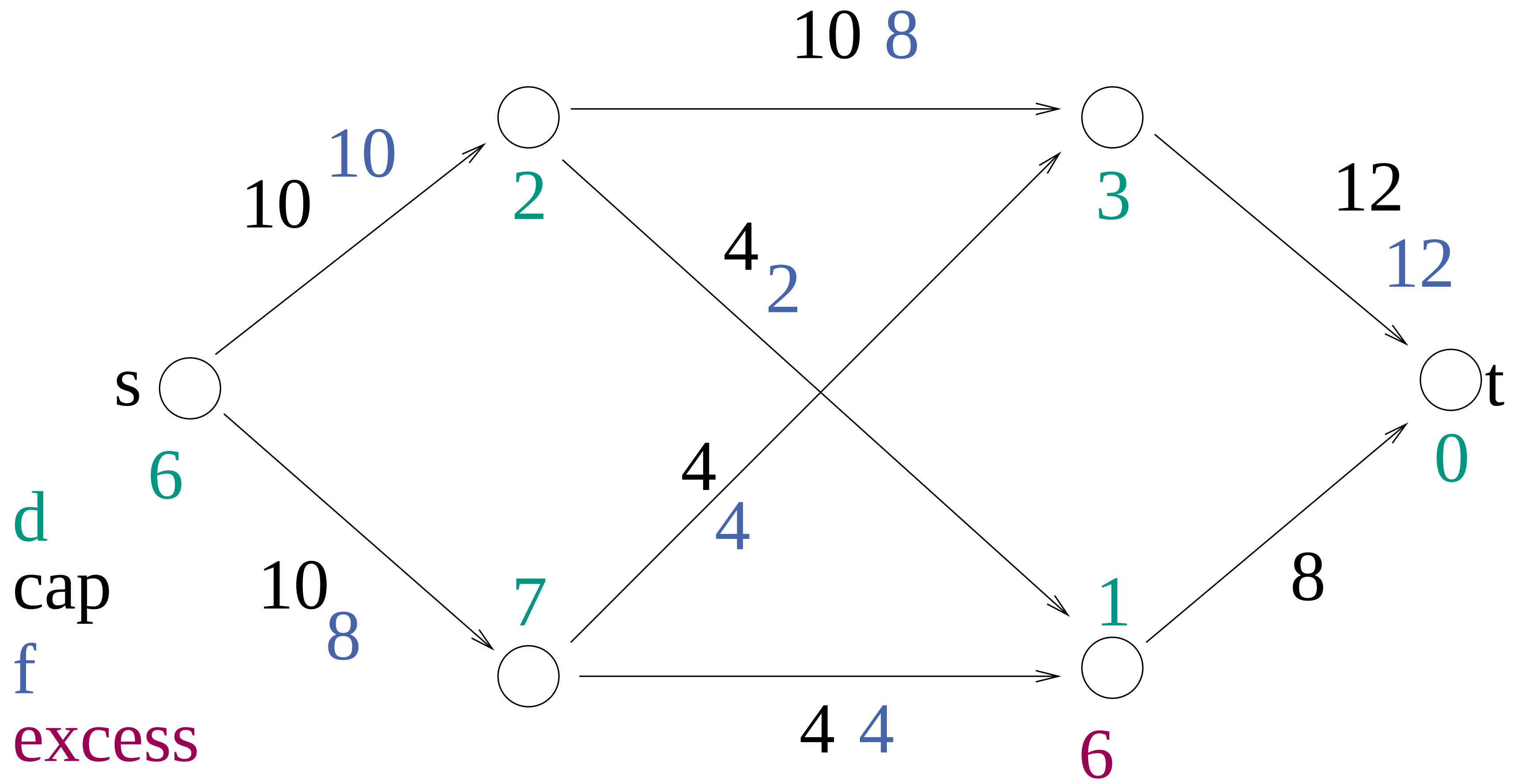
Example



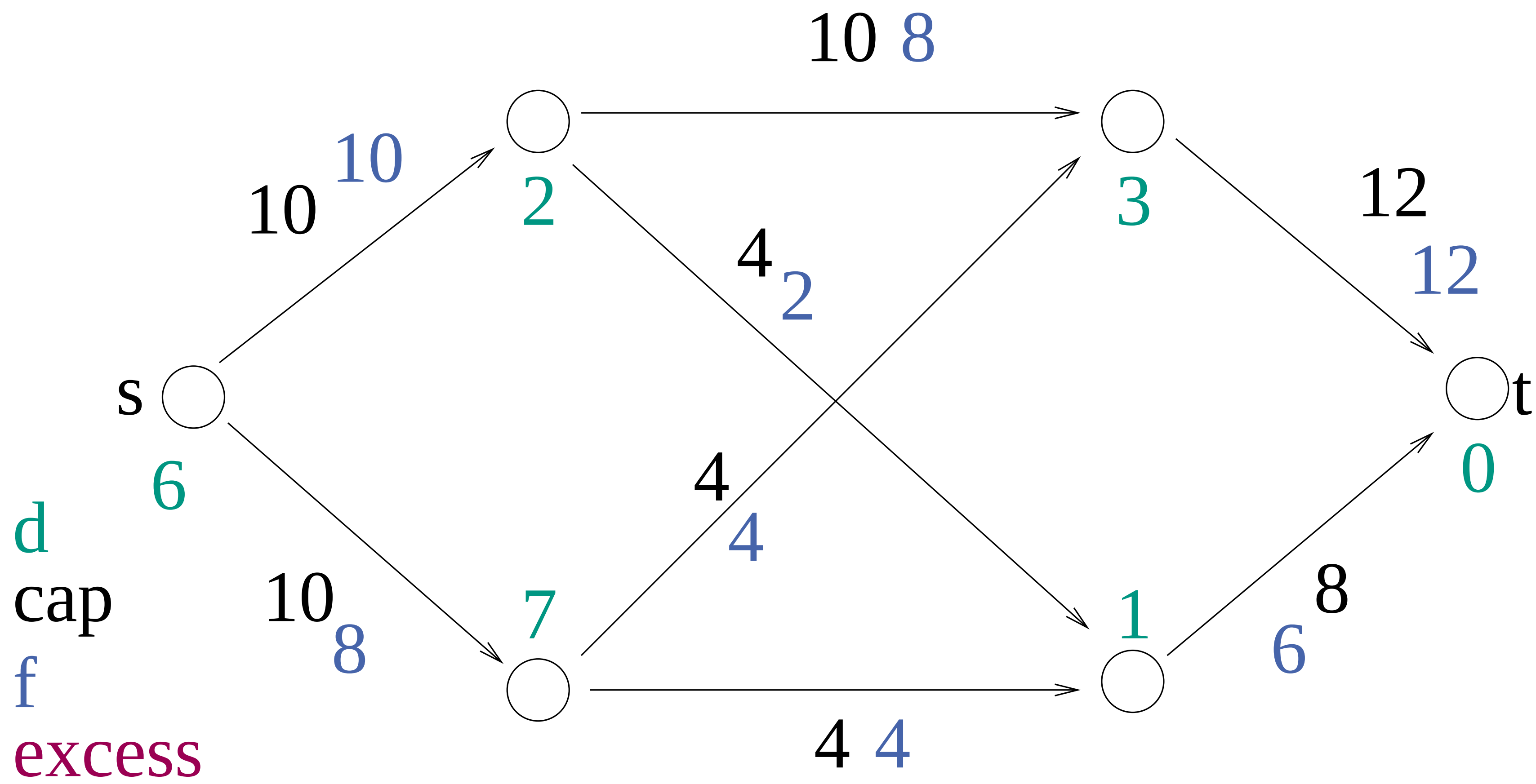
Example



Example



Example



9 pushes in total, 3 less than before

Proof of Lemma 12

$K := \sqrt{m}$ tuning parameter

$d'(v) := \frac{|\{w : d(w) \leq d(v)\}|}{K}$ scaled number of dominated nodes

$\Phi := \sum_{\{v : v \text{ is active}\}} d'(v).$ (Potential)

$d^* := \max\{d(v) : v \text{ is active}\}$ (highest level)

phase:= all pushes between two consecutive changes of d^*

expensive phase: more than K pushes

cheap phase: otherwise

Claims:

1. $\leq 4n^2K$ nonsaturating pushes in all cheap phases together
2. $\Phi \geq 0$ always, $\Phi \leq n^2/K$ initially (obvious)
3. a relabel or saturating push increases Φ by at most n/K.
4. a nonsaturating push does not increase Φ.
5. an expensive phase with $Q \geq K$ nonsaturating pushes decreases Φ by at least Q.

Operation	Amount
Relabel	$2n^2$
Sat.push	nm

Lemma 7+Lemma 8+2.+3.+4.:$\Rightarrow$

total possible decrease $\leq (2n^2 + nm)\frac{n}{K} + \frac{n^2}{K}$

This $+5.$ $:\leq \frac{2n^3+n^2+mn^2}{K}$ nonsaturating pushes in expensive phases

This $+1.$ $:\leq \frac{2n^3+n^2+mn^2}{K} + 4n^2K = \mathrm{O}\left(n^2\sqrt{m}\right)$ nonsaturating pushes overall for $K = \sqrt{m}$ $\square$

Claims:

1. $\leq 4n^2K$ nonsaturating pushes in all cheap phases together

We first show that there are at most $4n^2$ phases

(changes of $d^* = \max\{d(v) : v \text{ is active}\}$).

$d^* = 0$ initially, $d^* \geq 0$ always.

Only relabel operations increase d^*, i.e.,

$\leq 2n^2$ increases by Lemma 7 and hence

$\leq 2n^2$ decreases

$\leq 4n^2$ changes overall

By definition of a cheap phase, it has at most K pushes.

Claims:

1. $\leq 4n^2K$ nonsaturating pushes in all cheap phases together
2. $\Phi \geq 0$ always, $\Phi \leq n^2/K$ initially (obvious)
3. a relabel or saturating push increases Φ by at most n/K.

Let v denote the relabeled or activated node.

$$d'(v) := \frac{|\{w : d(w) \leq d(v)\}|}{K} \leq \frac{n}{K}$$

A relabel of v can increase only the d'-value of v.

A saturating push on (u,w) may activate only w.

Claims:

1. $\leq 4n^2K$ nonsaturating pushes in all cheap phases together
2. $\Phi \geq 0$ always, $\Phi \leq n^2/K$ initially (obvious)
3. a relabel or saturating push increases Φ by at most n/K.
4. a nonsaturating push does not increase Φ.

v is deactivated (excess(v) is now 0)

w may be activated

but $d'(w) \leq d'(v)$ (we do not push flow away from the sink)

Claims:

1. $\leq 4n^2K$ nonsaturating pushes in all cheap phases together
2. $\Phi \geq 0$ always, $\Phi \leq n^2/K$ initially (obvious)
3. a relabel or saturating push increases Φ by at most n/K.
4. a nonsaturating push does not increase Φ.
5. an expensive phase with $Q \geq K$ nonsaturating pushes decreases Φ by at least Q.

During a phase d^* remains constant

Each nonsat. push decreases the number of active nodes at level d^*

Hence, $|\{w : d(w) = d^*\}| \geq Q \geq K$ during an expensive phase

Each nonsat. push across (v,w) decreases Φ by

$\geq d'(v) - d'(w) \geq |\{w : d(w) = d^*\}|/K \geq K/K = 1$ ■

Claims:

1. $\leq 4n^2K$ nonsaturating pushes in all cheap phases together
2. $\Phi \geq 0$ always, $\Phi \leq n^2/K$ initially (obvious)
3. a relabel or saturating push increases Φ by at most n/K.
4. a nonsaturating push does not increase Φ.
5. an expensive phase with $Q \geq K$ nonsaturating pushes decreases Φ by at least Q.

Operation	Amount
Relabel	$2n^2$
Sat.push	nm

Lemma 7+Lemma 8+2.+3.+4.:$\Rightarrow$

total possible decrease $\leq (2n^2 + nm)\frac{n}{K} + \frac{n^2}{K}$

This $+5. :\leq \frac{2n^3+n^2+mn^2}{K}$ nonsaturating pushes in expensive phases

This $+1. :\leq \frac{2n^3+n^2+mn^2}{K} + 4n^2K = \mathrm{O}\left(n^2\sqrt{m}\right)$ nonsaturating pushes overall for $K = \sqrt{m}$ $\square$

MFIFO: Modified FIFO Selection Rule

pushFront after relabel.

pushBack when activated by a push

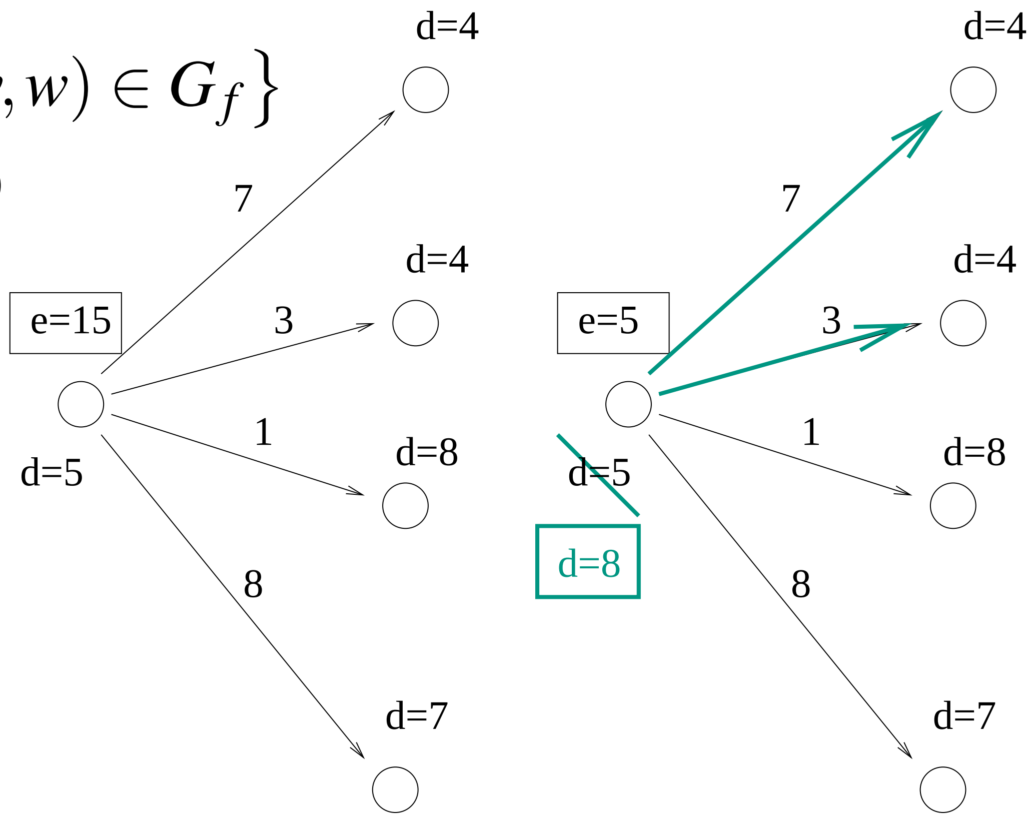
Heuristic Improvements

Naive algorithm needs $\Omega\left(n^2\right)$ relabels even on a path graph. We can do better.

aggressive local relabeling:

$$d(v) := 1 + \min\left\{d(w) : (v,w) \in G_f\right\}$$

(like a sequence of relabels)



Heuristic Improvements

Naive algorithm has best case $\Omega\left(n^2\right)$. Why? We can do better.

aggressive local relabeling: $d(v) := 1 + \min\left\{d(w) : (v,w) \in G_f\right\}$
(like a sequence of relabels)

global relabeling: (initially and every $\mathrm{O}(m)$ edge inspections):
$d(v) := G_f.\text{reverseBFS}(t)$ for nodes that can reach t in G_f.

Special treatment of nodes with $d(v) \geq n$. (Returning flow is easy)

Gap Heuristics. No node can connect to t across an empty level:

$\textbf{if}\ \{v : d(v) = i\} = \emptyset\ \textbf{then foreach}\ v$ with $d(v) > i\ \textbf{do}\ d(v) := n$

Experimental results

We use four classes of graphs:

- Random: n nodes, $2n+m$ edges; all edges (s,v) and (v,t) exist
- Cherkassky and Goldberg (1997) (two graph classes)
- Ahuja, Magnanti, Orlin (1993)

Timings: Random Graphs

Rule	BASIC	Ln	LRH	GRH	GAP	LEDA
FF	5.84	6.02	4.75	0.07	0.07	—
	33.32	33.88	26.63	0.16	0.17	—
HL	6.12	6.3	4.97	0.41	0.11	0.07
	27.03	27.61	22.22	1.14	0.22	0.16
MF	5.36	5.51	4.57	**0.06**	0.07	—
	26.35	27.16	23.65	0.19	0.16	—

$n \in \{1000, 2000\}, m = 3n$

FF=FIFO node selection, HL=hightest level, MF=modified FIFO

Ln= $d(v) \geq n$ is special,

LRH=local relabeling heuristic, GRH=global relabeling heuristics

Timings: CG1

Rule	BASIC	Ln	LRH	GRH	GAP	LEDA
FF	3.46	3.62	2.87	0.9	1.01	—
	15.44	16.08	12.63	3.64	4.07	—
HL	20.43	20.61	20.51	1.19	1.33	**0.8**
	192.8	191.5	193.7	4.87	5.34	3.28
MF	3.01	3.16	2.3	0.89	1.01	—
	12.22	12.91	9.52	3.65	4.12	—

$n \in \{1000, 2000\}, m = 3n$

FF$=$FIFO node selection, HL$=$hightest level, MF$=$modified FIFO

Ln$= d(v) \geq n$ is special,

LRH$=$local relabeling heuristic, GRH$=$global relabeling heuristics

Timings: CG2

Rule	BASIC	Ln	LRH	GRH	GAP	LEDA
FF	50.06	47.12	37.58	1.76	1.96	—
	239	222.4	177.1	7.18	8	—
HL	42.95	41.5	30.1	0.17	0.14	**0.08**
	173.9	167.9	120.5	0.36	0.28	0.18
MF	45.34	42.73	37.6	0.94	1.07	—
	198.2	186.8	165.7	4.11	4.55	—

$n \in \{1000, 2000\}, m = 3n$

FF=FIFO node selection, HL=hightest level, MF=modified FIFO

Ln= $d(v) \geq n$ is special,

LRH=local relabeling heuristic, GRH=global relabeling heuristics

Timings: AMO

Rule	BASIC	Ln	LRH	GRH	GAP	LEDA
FF	12.61	13.25	1.17	**0.06**	**0.06**	—
	55.74	58.31	5.01	0.1399	0.1301	—
HL	15.14	15.8	1.49	0.13	0.13	0.07
	62.15	65.3	6.99	0.26	0.26	0.14
MF	10.97	11.65	0.04999	**0.06**	**0.06**	—
	46.74	49.48	0.1099	0.1301	0.1399	—

$n \in \{1000, 2000\}, m = 3n$

FF=FIFO node selection, HL=hightest level, MF=modified FIFO

Ln= $d(v) \geq n$ is special,

LRH=local relabeling heuristic, GRH=global relabeling heuristics

Asymptotics, $n \in \{5000, 10000, 20000\}$

Gen	Rule	GRH			GAP			LEDA		
rand	FF	0.16	0.41	1.16	0.15	0.42	1.05	—	—	—
	HL	1.47	4.67	18.81	0.23	0.57	1.38	0.16	0.45	1.09
	MF	0.17	0.36	1.06	0.14	0.37	**0.92**	—	—	—
CG1	FF	3.6	16.06	69.3	3.62	16.97	71.29	—	—	—
	HL	4.27	20.4	77.5	4.6	20.54	80.99	2.64	12.13	**48.52**
	MF	3.55	15.97	68.45	3.66	16.5	70.23	—	—	—
CG2	FF	6.8	29.12	125.3	7.04	29.5	127.6	—	—	—
	HL	0.33	0.65	1.36	0.26	0.52	1.05	0.15	0.3	**0.63**
	MF	3.86	15.96	68.42	3.9	16.14	70.07	—	—	—
AMO	FF	0.12	0.22	0.48	0.11	0.24	0.49	—	—	—
	HL	0.25	0.48	0.99	0.24	0.48	0.99	0.12	0.24	0.52
	MF	0.11	0.24	0.5	0.11	0.24	0.48	—	—	—

Recent AE Results on Max-Flow

Faster and More Dynamic Maximum Flow by Incremental Breadth-First Search, Goldberg, Hed, Kaplan, Kohli, Tarjan, Werneck, ESA 2015

- Much faster on many (relatively easy) real world instances (image processing, graph partitioning,. . .) than preflow-push
- Worst case performance guarantee $\mathrm{O}\left(mn^2\right)$ (as in Dinitz algorithm)
- Adaptible to dynamic scenarios
- Uses pseudoflows that allow excesses and deficits.

Open problem: close gaps between theory and practice!

Zusammenfassung Flows und Matchings I

- Natürliche Verallgemeinerung von kürzesten Wegen:
  ein Pfad $\rightsquigarrow$ viele Pfade
- viele Anwendungen
- "schwierigste/allgemeinste" Graph-Probleme, die sich mit kombinatorischen Algorithmen in Polynomialzeit lösen lassen
- Beispiel für nichttriviale Algorithmenanalyse
- Manchmal sind spezielle Probleminstanzfamilien beweisbar leichter (z.B. unit capacity, matchings)

Zusammenfassung Flows und Matchings II

- Entwurfstechnik: Algorithmeninvarianten relaxieren
  (augmenting paths $\rightsquigarrow$ Preflow-Push $\rightsquigarrow$ pseudoflows
- Invarianten leiten Entwurf und Verständnis von Algorithmen
- Potentialmethode ($\neq$ Knotenpotentiale)
- Algorithm Engineering: practical case $\neq$ worst case.
  Heuristiken/Details/Eingabeeigenschaften wichtig
- Datenstrukturen: bucket queues, graph representation,
  (dynamic trees)