COS 423 Lecture 21
Maximum Flows

© Robert E. Tarjan 2011

Maximum Flow Problem

In a directed graph with source vertex *s*, sink vertex *t*, and non-negative arc capaicities, find a *maximum flow* from *s* to *t*.

Let $G = (V, E)$ be a directed graph with source vertex s, sink vertex t, arc capacities $c(v, w) \geq 0$

Assume G is *symmetric*: $(v, w) \in E$ iff $(w, v) \in E$

(Symmetrize by adding reverse arcs with capacity 0 as necessary)

pseudoflow f: antisymmetric function on arcs that is bounded by arc capacities:

$$f(v, w) = -f(w, v) \leq c(v, w)$$

(antisymmetry simplifies some formulas)

excess $e(v)$ of vertex $v = \Sigma\{f(u, v) \mid (u, v) \in E\}$

f is a *preflow iff* $e(v) \geq 0$ for $v \neq s$

f is a *flow* if $e(v) = 0$ for $v \notin \{s, t\}$

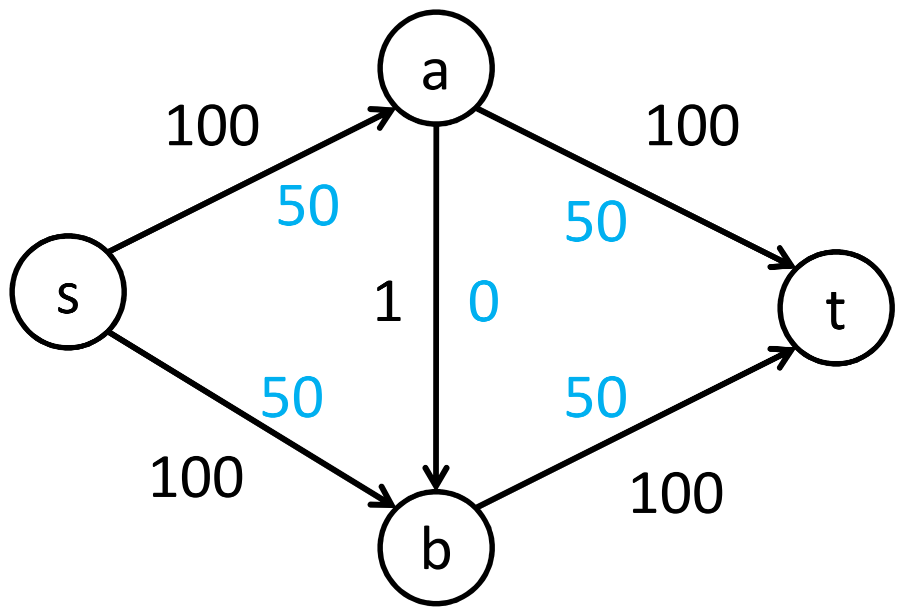
value of $f = e(t)$ ($= -e(s)$ if f is a flow)

f is *maximum* if $e(t)$ is maximum

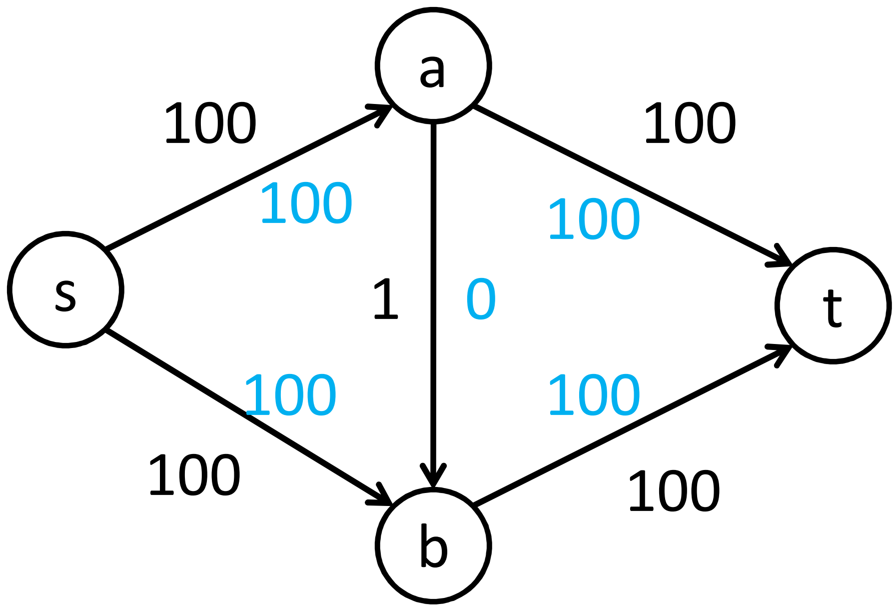
Goal: find a maximum flow

A capacitated graph with a flow

(0-capacity symmetric arcs omitted)

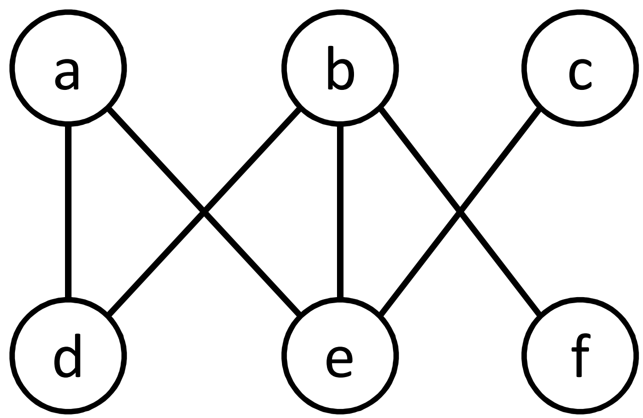


Maximum flow

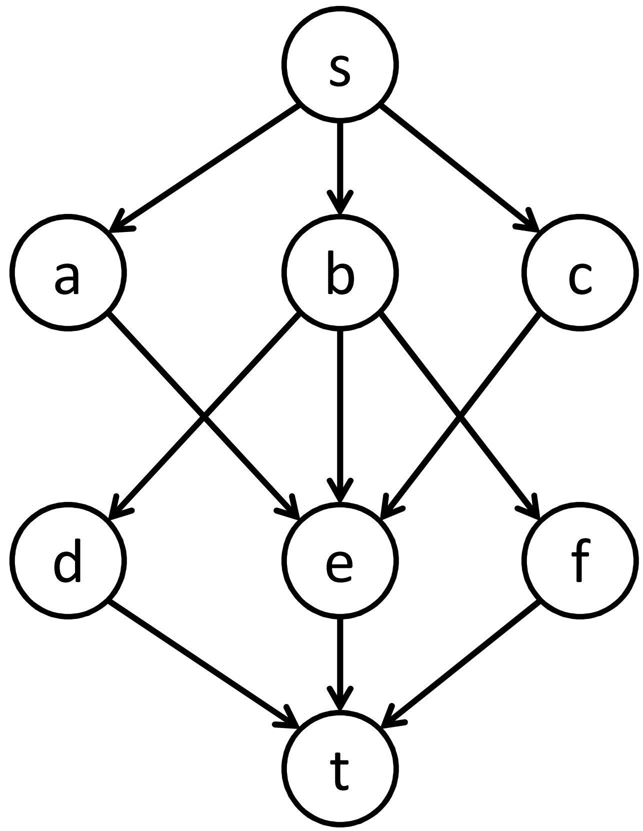


Bipartite matching via maximum flow

Find a matching of maximum size



Direct edges from X to Y, add source s sink t, arcs from s to all v in S, arcs from all w in Y to t, all capacities 1



Needs an integer solution

Augmenting path method (Ford & Fulkerson)

(v, w) is *saturated* if $f(v, w) = c(v, w)$, otherwise *residual*

residual capacity of (v, w):

$r(v, w) = c(v, w) – f(v, w)$

augmenting path: path of residual arcs from s to t

residual capacity of an augmenting path: minimum residual capacity of arcs on path

$f \leftarrow 0$;

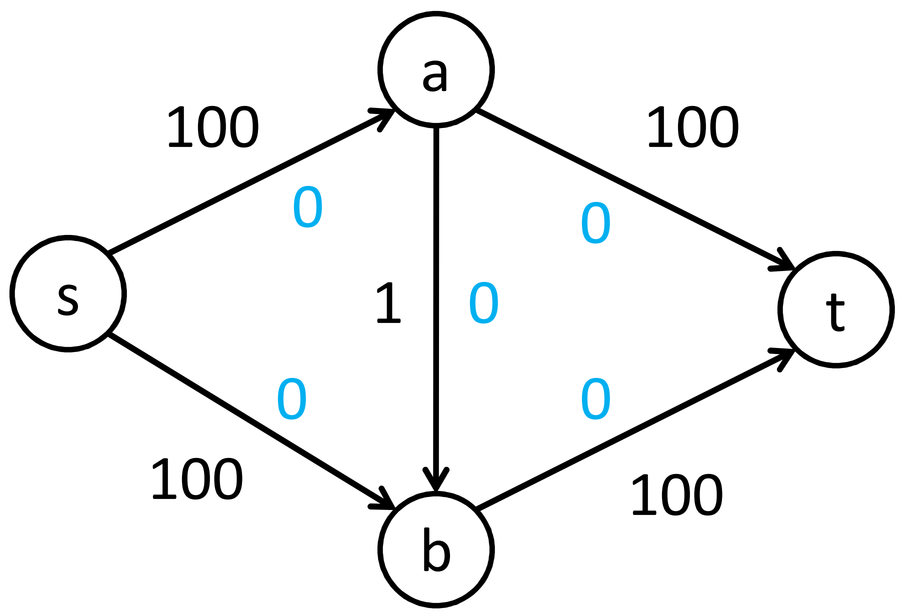
while $\exists$augmenting path P **do**

{$\Delta \leftarrow$ residual capacity of P;

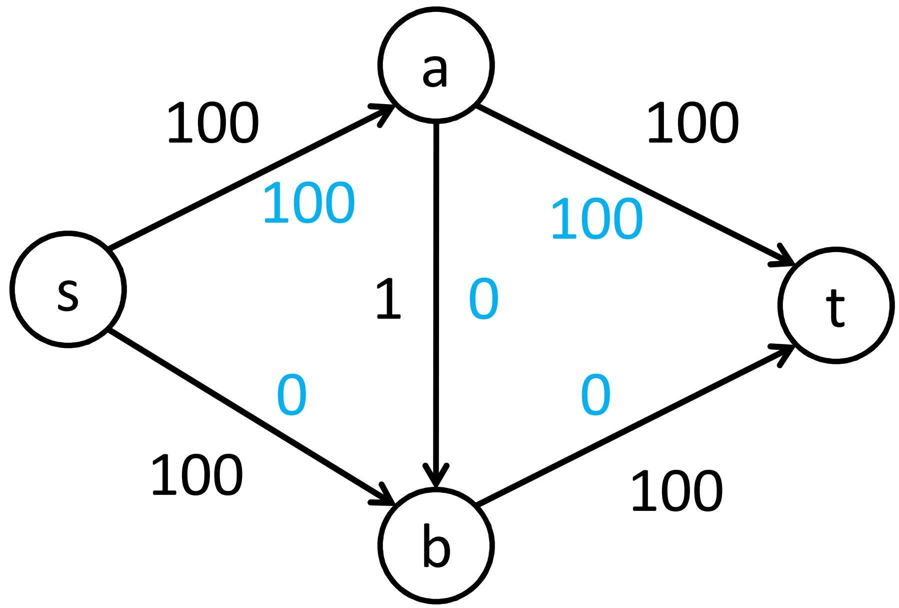
for (v, w) on P **do**

{$f(v, w) \leftarrow f(v, w) + \Delta$; $f(w, v) \leftarrow \mathrm{f(w,} \ v) - \Delta$}}

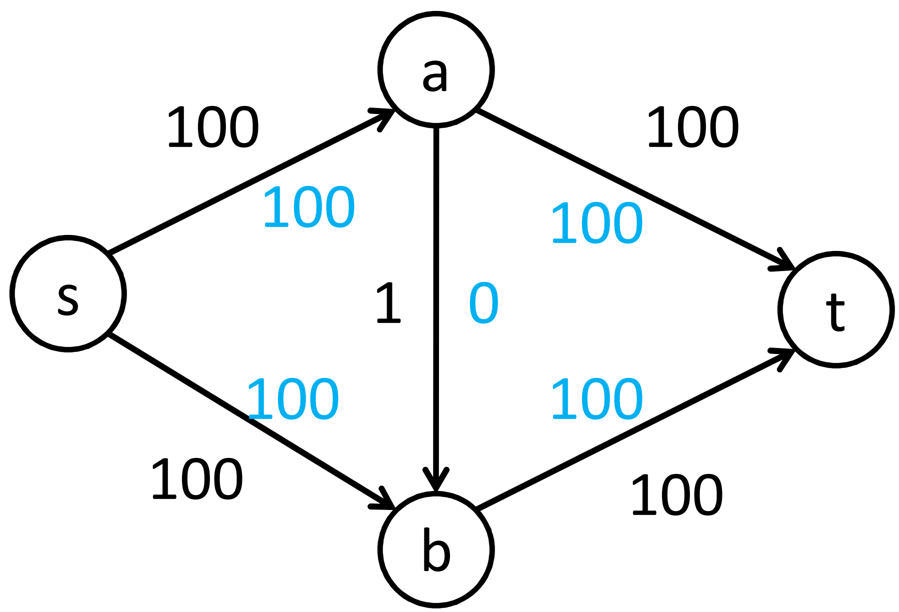
Augmenting path s, a, t



Augmenting path s, b, t



No augmenting path: flow is maximum



Correctness via duality

cut: a Partition of the vertices into two parts, X containing s and Y containing t

capacity of cut:

$$c(X, Y) = \Sigma\{c(x, y) \mid (x, y) \in E \;\&\; x \in X \;\&\; y \in Y\}$$

net flow across cut:

$$f(X, Y) = \Sigma\{f(x, y) \mid (x, y) \in E \;\&\; x \in X \;\&\; y \in Y\}$$

$$\leq \mathrm{c}(X, Y)$$

minimum cut: a cut of minimum capacity

Lemma: If X, Y *is* any cut and f is any flow, $f(X, Y) = e(t)$.

Proof: Exercise

Corollary: The maximum flow value is at most the minimum cut capacity

Max Flow, Min Cut Theorem: The maximum flow value equals the minimum cut capacity

Proof: Run the augmenting path algorithm until there is no augmenting path. Let X be the set of vertices reachable from s by a path of residual arcs, Y the rest. Then $y \in Y$, so X, Y is a cut. Also, if $(x, y) \in E$ with $x \in X$ & $y \in Y$, then $c(x, y) = f(x, y)$, so $c(X, Y) = f(X, Y)$.

Termination?

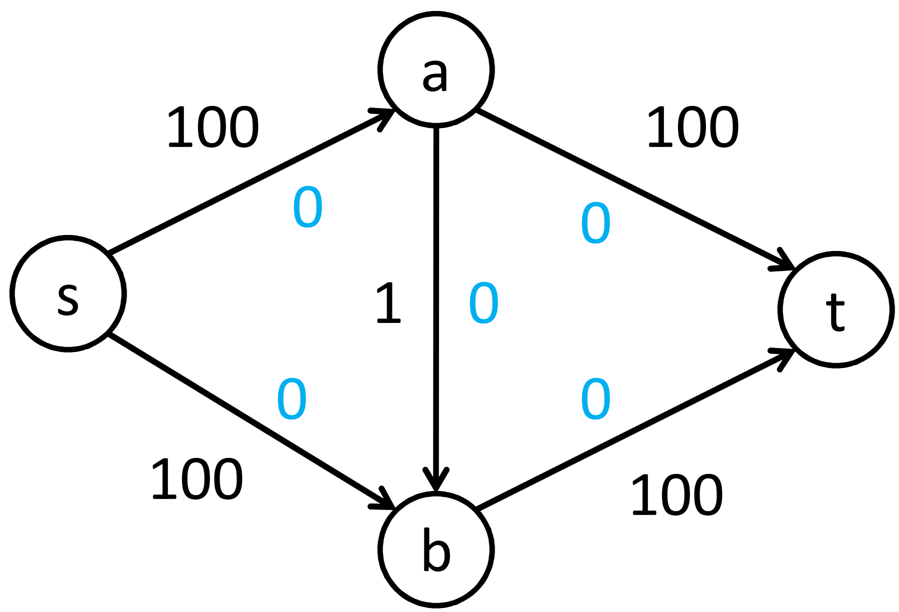
Proof of max-flow, min-cut theorem requires that the augmenting path algorithm terminates.

Ford & Fulkerson: If arc capacities are integers, each augmentation increases the flow value by at least 1, so algorithm must terminate: sum of arc capacities is an upper bound on #augmentations. This argument extends to fractional capacities. Also, if arc capacities are integers, there is an integral maximum flow.

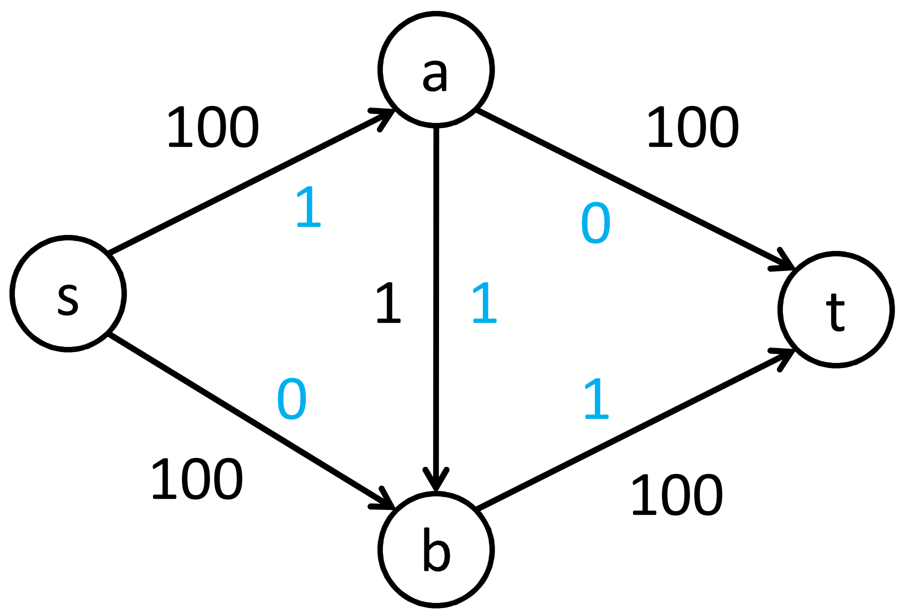
What if capacities are irrational?

How many augmentations?

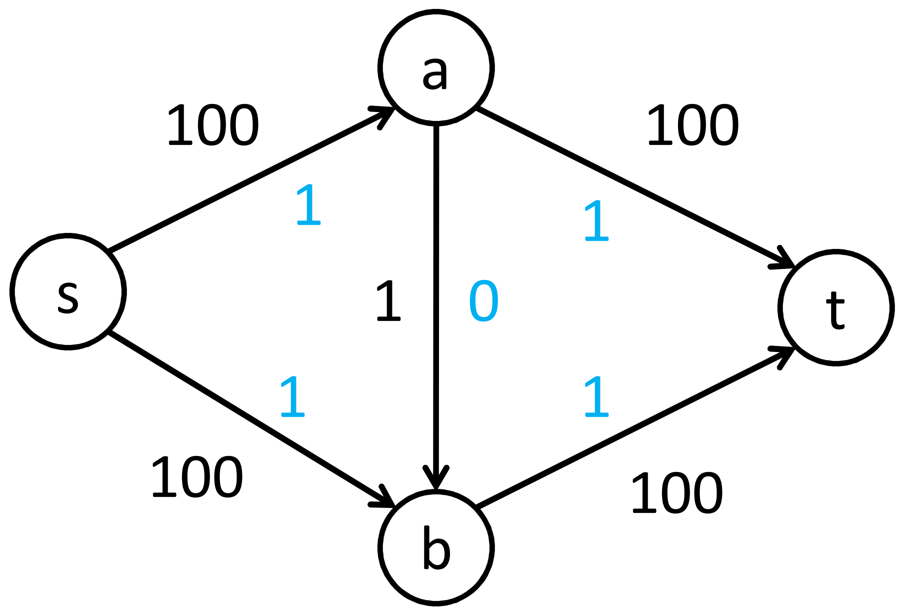
Augmenting path s, a, b, t



Augmenting path s, b, a, t



Augmenting path s, b, a, t



Maximum flow after 198 more augmentations

#augmentations *not* polynomial in graph size and #bits needed to represent capacities

If capacities are irrational, algorithm need not terminate, flow value need not converge to maximum (even though it will converge).

Efficiency requires a good choice of augmenting paths

Edmonds &Karp: Choose augmenting path with fewest arcs: O(nm) augmentations, O(nm^2) time.

Dinic: In each phase, find all augmenting paths with k arcs but no fewer: reduces amortized time per augmentation from O(m) to O(n), total time to O(n^2m) (just like Hopcroft-Karp bipartite matching algorithm)

Faster, simpler algorithms

Break computation into smaller parts: change flow on one arc at a time, move flow along estimated shortest path to sink

Allow (temporary) excess flow at a vertex: *preflow* ($e(v) \geq 0$ for $v \neq s$)

Vertex $v \notin \{s, t\}$ is *active* if $e(v) > 0$

valid vertex labeling d

$d(v)$ is a non-negative integer,

$d(t) = 0$, $d(s) = n$,

$d(v) \le d(w) + 1$ if (v, w) is residual

→ $d(v)$ is at most the number of arcs on a residual path from v to t, if there is such a path

Preflow push algorithm

$d \leftarrow 0$; $d(s) \leftarrow n$; $f \leftarrow 0$;

for $(s, v) \in E$ **do** $f(s, v) \leftarrow c(s, v)$;

while $\exists$ active v **do**

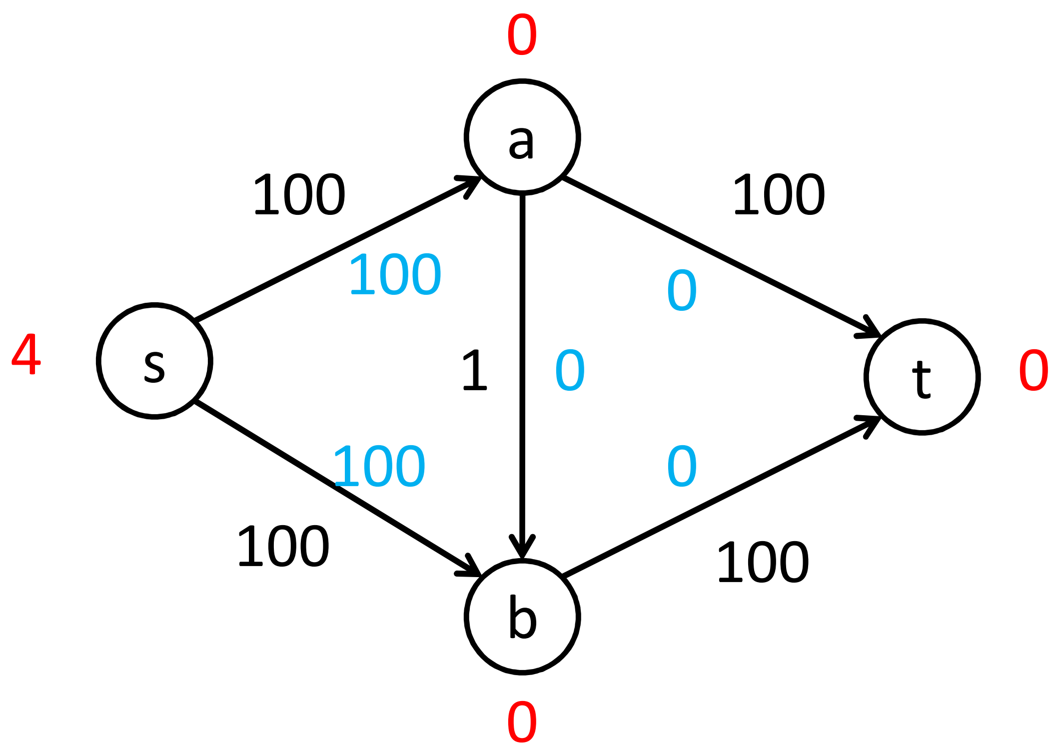
  if $\exists$ residual $(v, w) \ni d(v) > d(w)$ **then**

    $f(v, w) \leftarrow f(v, w) + \min\{e(v), r(v, w)\}$

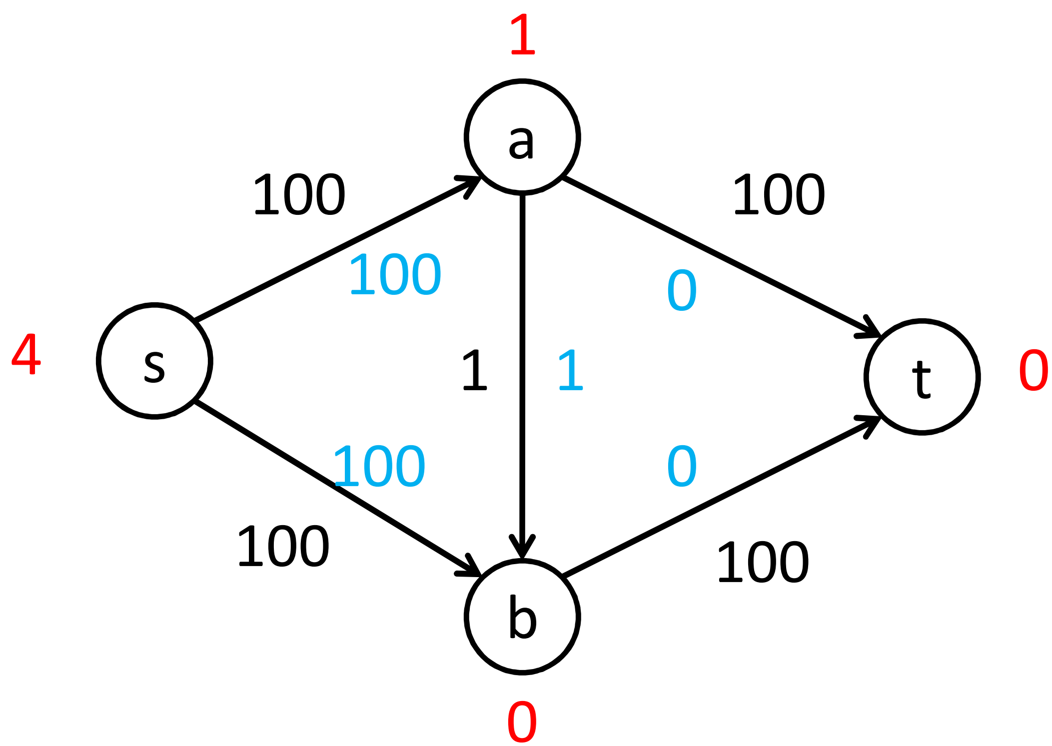
      [*push*: *saturating* if it saturates (v, w), *non-saturating* otherwise]

  else $d(v) \leftarrow 1 + \min\{d(w) \mid (v, w) \text{ residual}\}$ [*label*]

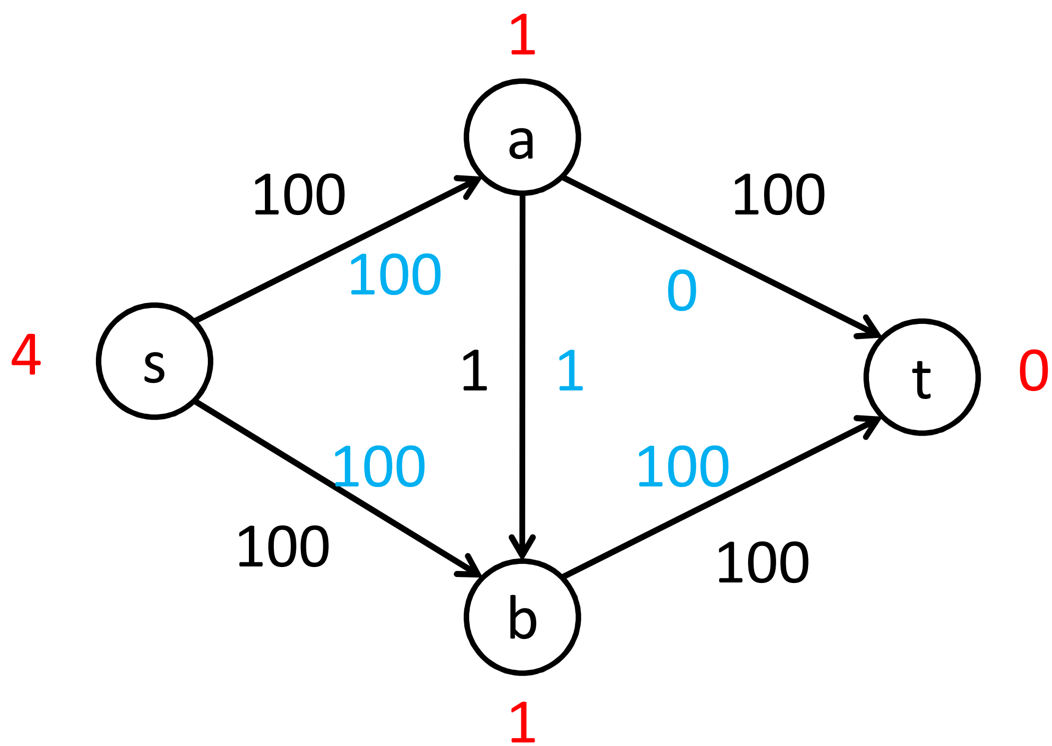
After initialization: flows, labels



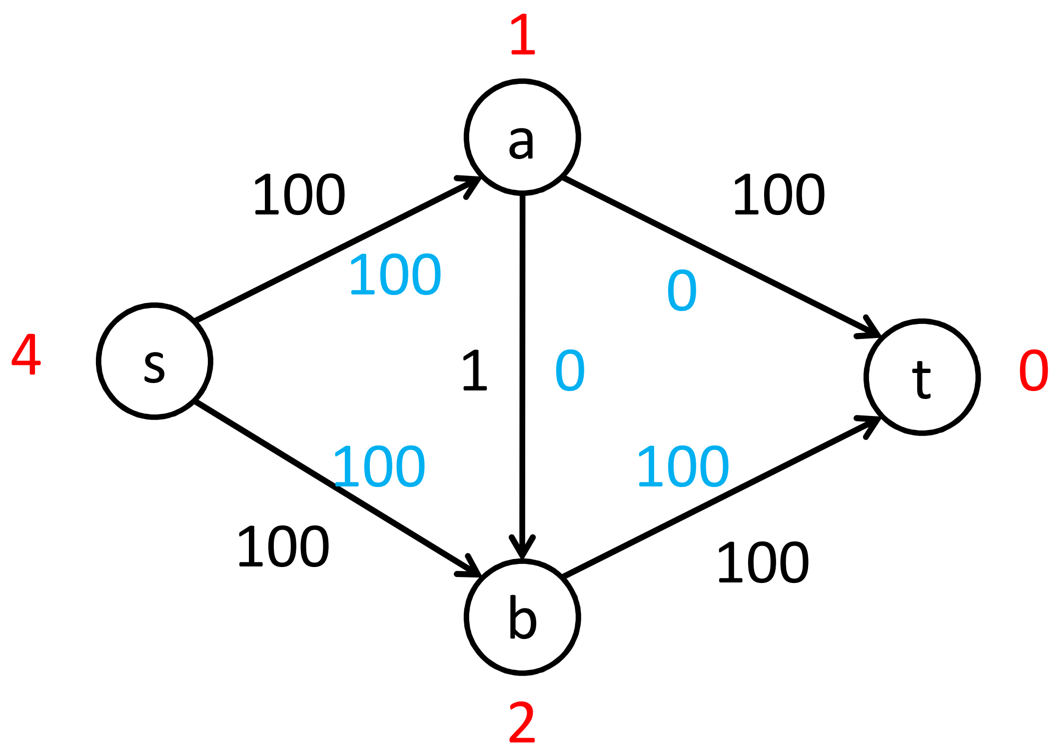
Process a: label, push to b



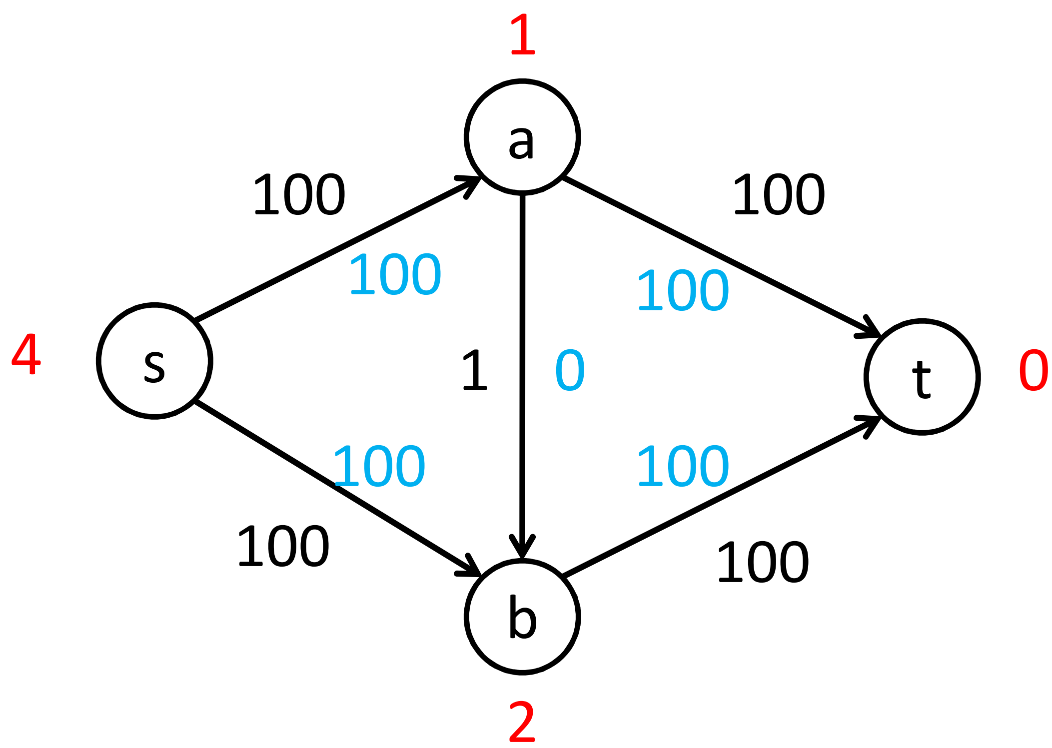
Process b: label, push to t



Process b: label, push to a



Process a: push to t, no active vertices



Basic properties of algorithm

After initialization, labeling is valid; while loop maintains validity

f is always a preflow.

$e(t)$ at any later time $\leq \Sigma\{e(v) \mid d(v) < n\}$

If $e(v) > 0$, there is a simple path of positive flow from s to v (proof: exercise)

If $e(v) > 0$, $d(v) < 2n - 1$: there is a residual path from v to s along which d can decrease by at most 1 per arc

At most $2n - 1$ labelings per vertex. Time to label a vertex is O(degree). Time for all labelings is O(nm).

Implementation of pushes: For each vertex v, maintain a *current arc* (v, w). To process v in while loop, do a push on current arc if allowed; if not, replace current arc by next arc on arc list; if current arc is last on list, label v

At most one saturating push per incident arc between labelings of $v \rightarrow$ O(nm) saturating pushes, O(1) time per push + O(nm) overhead

How many non-saturating pushes?

How to choose vertices for processing?

Variants

Label-tightening: Periodically, set all labels equal to their maximum possible values (via BFS backward from t followed by BFS backward from s). Takes $O(m)$ time

Lazy return of excess: do no steps at vertices of label $\geq n$. Once done, return excess flow to s by finding paths of positive flow from s to vertices of label $\geq n$ and reducing the flow on such paths. Simple implementation takes $O(nm)$ time

Running time by choice of vertices

Any order: O(n^2m) time

Queue of active vertices(FIFO): O(n^3) time

Highest label: O($n^2m^{½}$) time

Large excess: O($n^2 \lg U + nm$) time, if capacities are integers $\leq U$

Analysis: charge non-saturating pushes against other steps via a potential function

Generic method

$$\Phi = \Sigma\{d(v) \mid v \text{ active}\}$$

$$0 \le \Phi \le 2n^2$$

A non-saturating push decreases Φ by at least one → amortized cost ≤ 0

Amortized cost of a label of $v = \Delta d(v)$

Amortized cost of a saturating push $\le 2n$

→ #non-saturating pushes = $O(n^2m)$

FIFO method

Define *passes*:

pass 0 = processing of initially active vertices

pass $k + 1$ = processing of vertices added to queue during pass k

$$\Phi = \max\{nd(v) \mid v \text{ active}\}$$

$$0 \le \Phi \le 2n^2$$

A pass does at most n non-saturating pushes, at most one per vertex, has actual cost at most n

A pass that increases labels by a total of k increases Φ by at most $(k - 1)n$ → amortized cost ≤ kn → total amortized cost ≤ $2n^3$

→ $O(n^3)$ running time

Large-excess method

Assume capacities are integral, at most *U*

Δ is a scale factor, initially the smallest power of two no less than *U*

An excess is *big* if it is at least *Δ*/2

Process a vertex with big excess; break a tie in favor of smallest label. Never allow an excess to exceed *Δ*. When all active vertices have small excess, divide *Δ* by 2

To maintain bound on excesses during a push: increase $f(v, w)$ by $\min\{e(v), r(v, w), \Delta - e(w)\}$

If e(v) is big, e(w) is not big, and the push is non-saturating, then it increases e(w) by at least $\Delta/2$

Overhead to implement big-excess rule is $O(nm)$

A *phase* is the time between changes in Δ.
During the last phase, $\Delta = 1 \rightarrow$ #phases $\leq \lg U$

$$\Phi = \Sigma\{2e(v)d(v)/\Delta \mid v \text{ active}\}$$

$$0 \leq \Phi \leq 4n^2$$

Each non-saturating push decreases Φ by at least 1 $\rightarrow$ amortized cost of a non-saturating push is at most 0

Label increase of k increases Φ by at most $2k$, $\leq 4n^2$ over all relabelings

Decrease in Δ doubles Φ, increasing Φ by at most $2n^2$ per change in Δ, at most $2n^2 \lg U$ over all phases

→ $O(n^2 \lg U + nm)$ running time