INFO 6205
Program Structure and Algorithms

Nik Bear Brown

Preflow-Push Push–relabel maximum flow algorithm

Topics

- Preflow-Push Push–relabel maximum flow algorithm

Preflows

- At each intermediate stages we permit more flow arriving at nodes than leaving (except for s)
- i.e., $e(i)$ = *excess* at i = net excess flow into node i.
- The excess is required to be nonnegative.

Flows vs. Preflows

Augmenting Path Algorithm

Flow into i = Flow out of i

Push flow along a path from s to t

d(j) = distance from j to t in the residual network.

Preflow Algorithm

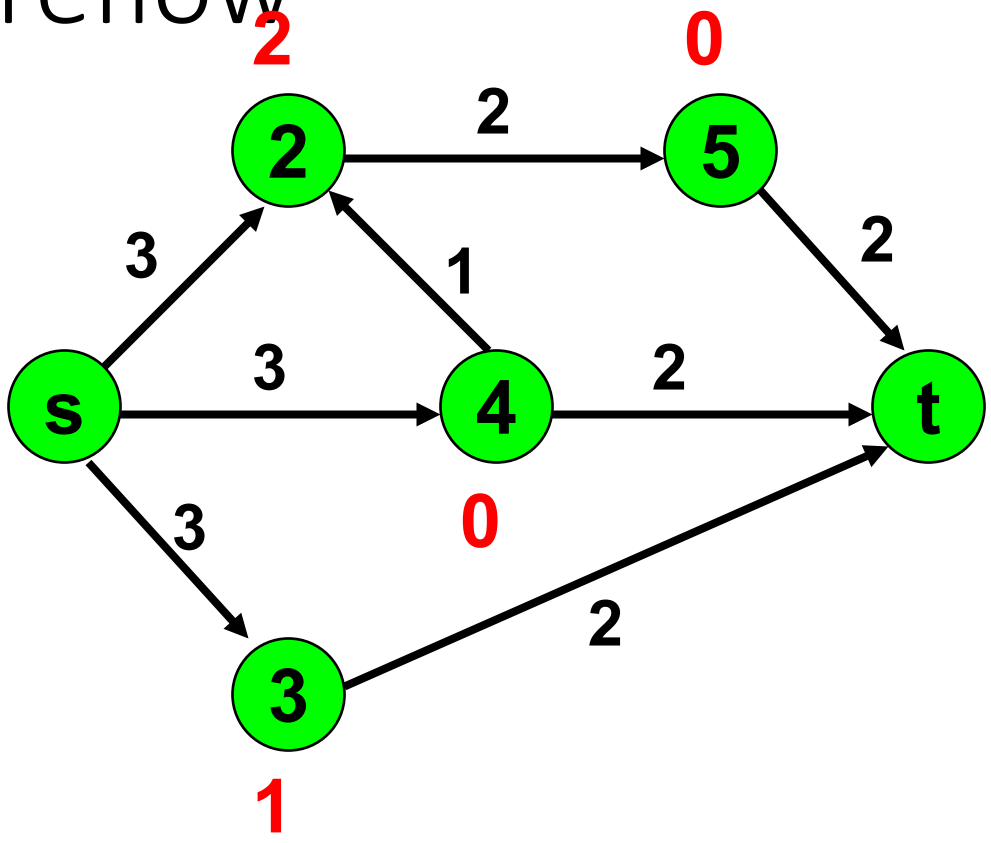
Flow into i ≥ Flow out of i
for i ≠ s.

Push flow in one arc at a time

d(j) ≤ distance from j to t in the residual network

- d(t) = 0
- d(i) ≤ d(j) + 1 for each arc (i, j) ∈ G(x),

A Feasible Preflow



The ***excess*** ***e(j)*** at each node j ≠ s, t is the flow in minus the flow out.

Note: total excess = flow out of s minus flow into t.

A Preflow

- Def. An preflow is a function that satisfies:
  - For each e ∈ E: $0 \le f(e) \le c(e)$ (capacity)
  - For each v ∈ V – {s, t}: $\sum_{e \text{ in to } v} f(e) \ge \sum_{e \text{ out of } v} f(e)$ (weak conservation)
- Def. The excess of a preflow f at node v :
  - Required to be non-negative except s,t

$$e_f(v) = \sum_{e \text{ in } s} f(e) - \sum_{e \text{ out of } s} f(e) .$$

1. Algorithm maintains preflow (not flow)
2. Each vertex is associated with a height
3. Flow is “downhill”
4. Vertices with excess are sometimes “lifted/relabeled”.

Labelling/Height function

- H: V -> N
  - Source and Sink condition
    - h(s) = n
    - h(t) = 0
  - Steepness condition
    - For all (v,w) in E_f , h(v) <= h(w)+1
    - Here E_f is the residual graph.

Invariant

Compatibility of preflows and labellings

Lemma

- If s-t preflow f is compatible with the labelling h, then there is no s-t path in the residual graph G_f.
- Cor: If s-t flow f is compatible with a labelling h, then f is a flow of maximum value.

Preflow push algorithm

- Initialize
- **while** push or lift operation possible **do**
  - Select an applicable push or lift operation and perform it

Lemmas / Invariants

- If there is an overflowing vertex (except t), then a lift or push operation is possible
- The height of a vertex never decreases
- When a lift operation is done, the height increases by at least one.
- h remains a height function during the algorithm

Another invariant and the correctness

- There is no path in G_f from *s* to *t*
  - **Proof**: the height drops by at most one across each of the at most *n*-1 edges of such a path
- When the algorithm terminates, the preflow is a maximum flow from *s* to *t*
  - *f* is a flow, as no vertex except *t* has excess
  - As G_f has no path from *s* to *t*, *f* is a maximum flow

Number of lifts

- For all u: h[u] < 2n
  - h[s] remains n. When vertex is lifted, it has excess, hence path to s, with at most n – 1 edges, each allowing a step in height of at most one up.
- Each vertex is lifted less than 2n times
- Number of lift operations is less than $2n^2$

Counting pushes

- Saturating pushes and not saturating pushes
  - Saturating: sends $c_f(u,v)$ across (u,v)
  - Non-saturating: sends $e[u] < c_f(u,v)$
- Number of saturating pushes
  - After saturating push across (u,v), edge (u,v) disappears from G_f.
  - Before next push across (u,v), it must be created by push across (v,u)
  - Push across (v,u) means that a lift of v must happen
  - At most $2n$ lifts per vertex: $O(n)$ sat. pushes across edge
  - $O(nm)$ saturating pushes

Non-saturating pushes

- Look at $$\Phi = \sum_{e(v)>0} h[v]$$
- Initially $\Phi = 0$.
- Φ increases by lifts in total at most $2n^2$
- Φ increases by saturating pushes at most by $2n$ per push, in total $O(n^2 m)$
- Φ decreases at least one by a non-saturating push across (u,v)
    - After push, u does not overflow
    - v may overflow after push
    - $h(u) > h(v)$
- At most $O(n^2 m)$ pushes

Algorithm

- Implement
  - O(n) per lift operation
  - O(1) per push
- O(n^2m) time

Preflow

Notation from Introduction to Algorithms

- Function f: $V * V \to \mathbf{R}$
  - Skew symmetry: $f(u,v) = - f(v,u)$
  - Capacity constraints: $f(u,v) \leq c(u,v)$
  - Notation: $f(V,u)$
  - For all u, except s: $f(V,u) \geq 0$ (***excess flow***)
  - u is ***overflowing*** when $f(V,u) > 0$.
  - Maintain: $e(u) = f(V,u)$.

Height function

- h: V $\rightarrow$ **N**:
  - h(s) = n
  - h(t) = 0
  - For all (u,v) $\in$ E_f (residual network):
    h(u) $\leq$ h(v)+1

Initialize

- Set height function h
  - $h(s) = n$
  - $h(t) = 0$
  - $h(v) = 0$ for all v except s

Do not change

- **for** each edge (s,u) **do**
  - $f(s,u) = c(s,u)$; $f(u,s) = -c(s,u)$
  - $e[u] = c(s,u)$;

Initial preflow

Basic operation 1: Push

- Suppose $e(u) > 0$, $c_f(u,v) > 0$, and $h[u] = h[v]+1$
- Push as much flow across (u,v) as possible

  $r = \min\{e[u], c_f(u,v)\}$

  $f(u,v) = f(u,v) + r;$

  $f(v,u) = -f(u,v);$

  $e[u] = e[u] - r;$

  $e[v] = e[v] + r.$

Basic operation 2: Relabel/Lift

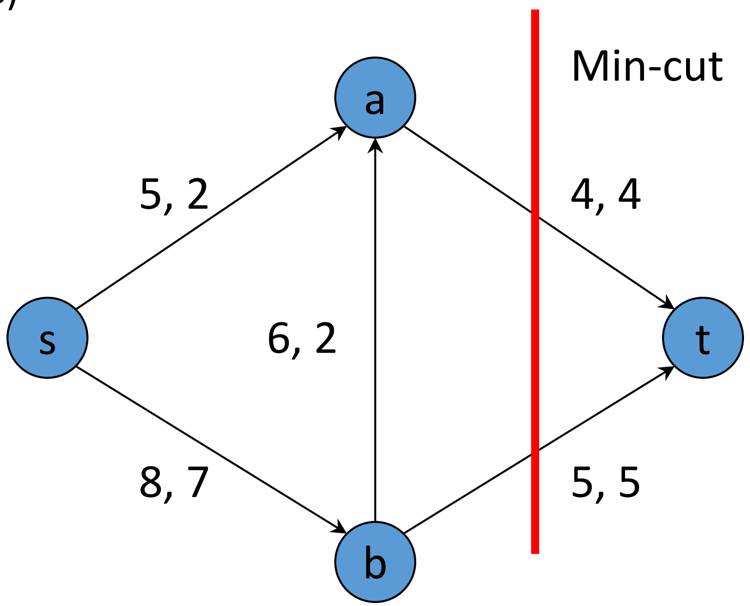
- *When no push can be done from overflowing vertex (except s,t)*
- Suppose e[u]>0, and for all $(u,v) \in E_f$: h[u] $\leq$ h[v], $u \neq s$, $u \neq t$
- Set h[u] = 1 + min {h[v] | $(u,v) \in E_f$}

Preflow push algorithm

- Initialize
- **while** push or lift operation possible **do**
  - Select an applicable push or lift operation and perform it

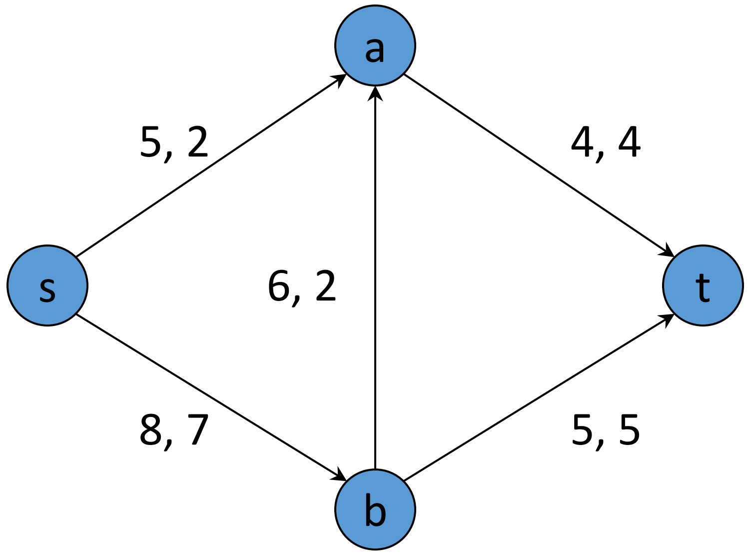
Max Flow Problem

- Given a graph with “Source” and “Sink” nodes we want to compute:
  - The maximum rate at which fluid can flow from Source to Sink
  - The rate of flow through each edge of the graph
- Given a graph:
  - Edges have flow capacities
    - First value is flow capacity
    - Second value is current flow rate
  - Edges behave like pipes
  - Nodes are junctions of pipes
- History:
  - Ford and Fulkerson 1956 (Augmenting Paths)
  - Dinic 1970
  - Edmonds and Karp 1972
  - Malhotra, Kumar and Maheshwari 1978
  - Goldberg and Tarjan 1986 (Preflow Push)
  - Boykov and Kolmogorov 2006



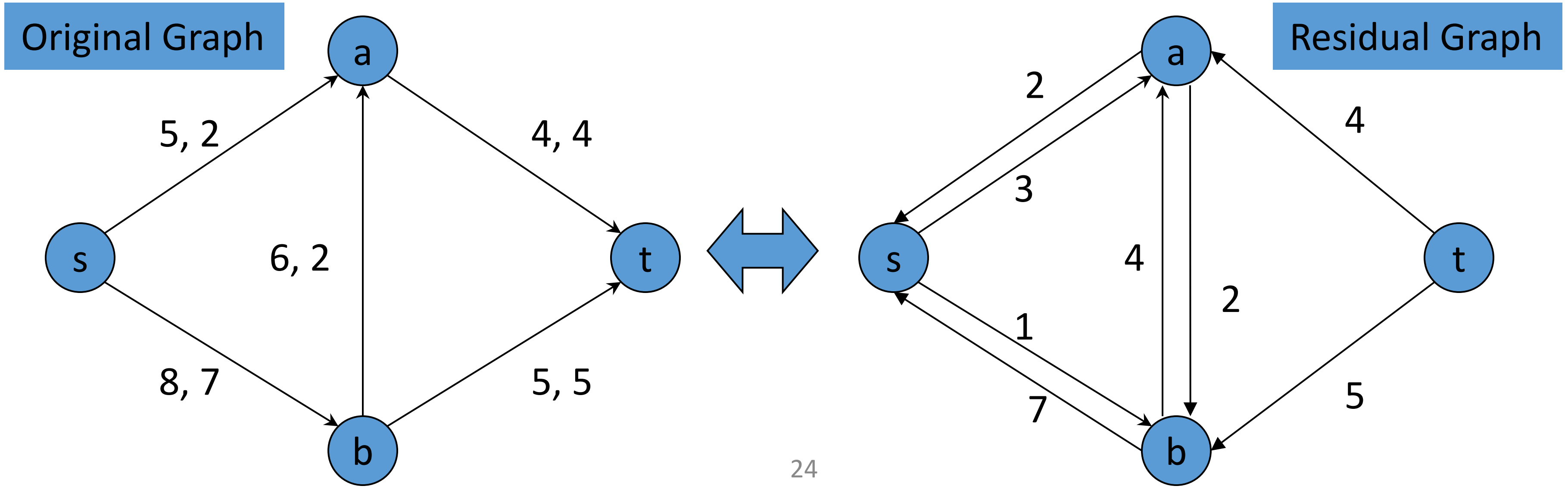
Max Flow Problem

- Flow $f(u,v)$ is a real-valued function defined for every edge (u,v) in the graph
- The flow needs to satisfy the following 3 properties:
  - $f(u,v) \leq c(u,v)$ i.e. capacity of edge (u,v)
  - $f(u,v) = -f(v,u)$
  - Flow coming into node v = Flow leaving v



Max Flow Problem

- Residual Graph:
  - A Graph that contains edges that can admit more flow
  - Define a Graph *G(V,E') for G(V,E)*
  - We define residual flow $r(u,v) = c(u,v) - f(u,v)$
  - If $r(u,v) > 0$ then (u,v) is in E'



Preflow Push Algorithm for Maxflow problem

- Relaxation algorithm:
  - Performs local updates repeatedly until global constraint is satisfied
  - Similar to Stencil computations
- Fluid flows from a higher point to a lower point
- In the beginning
  - “Source” is the highest point
  - “Sink” and all other nodes are at the lowest point
- Source sends maximum flow on its outgoing edges
  - Sending flow to out-neighbors is called “Push” operation
- The height of Source’s neighbors is then increased so that fluid can flow out
  - Increasing height of a node is called “Relabel” operation

Preflow Push Algorithm for Maxflow problem

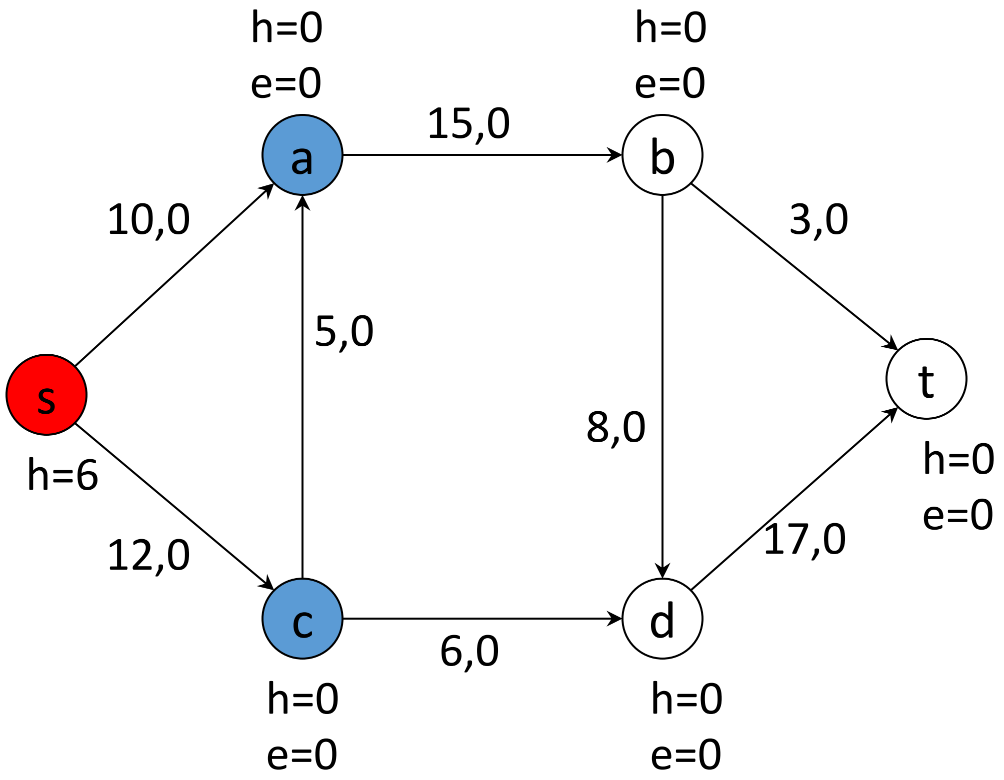
- A node is allowed (temporarily) to have more flow coming into it than flow going out
  - i.e. a node can have “excess flow”
  - But the edges must respect the capacity condition
  - Source can have arbitrary amount of excess flow
- A node is said to be “active” if it has excess flow in it

Preflow Push Algorithm for Maxflow problem

- We increase the height of the “active” node with a “Relabel” operation
  - If h is the minimum height among neighbors that can accept flow
  - Then height is relabeled to (h+1)
- Then we “push” the excess flow to the neighbors
  - That are lower than the active node and can admit flow
  - Thus make them active
- Algorithm terminates when there are no “active” nodes left

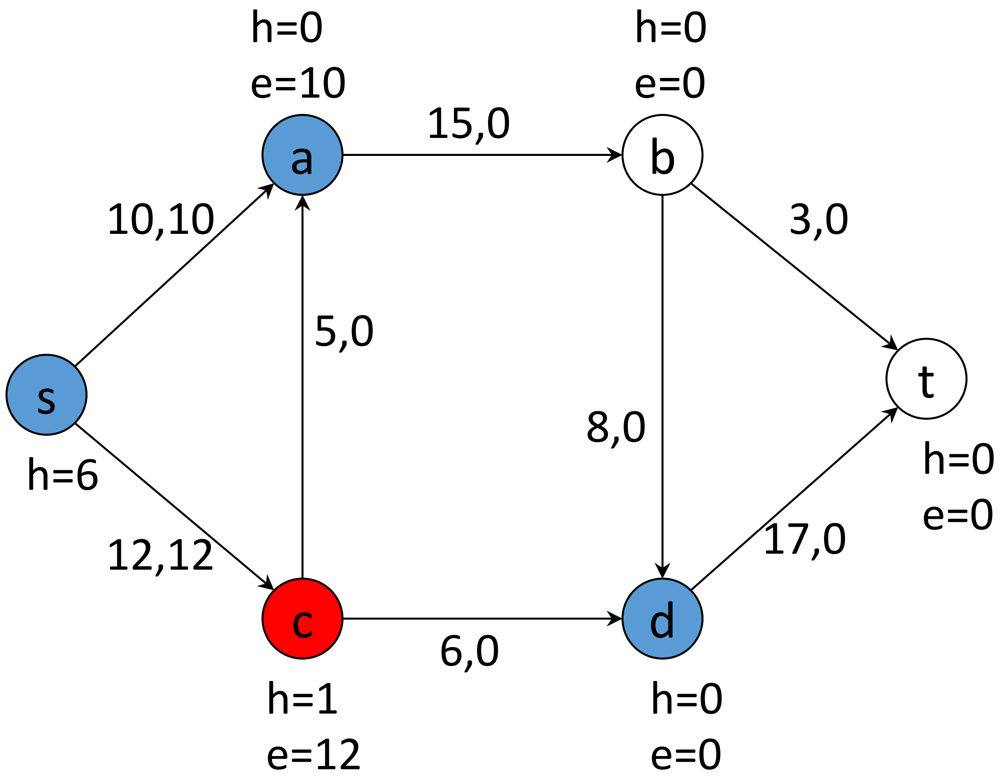
Example

- A simple graph:
  - Nodes have two attributes:
    - Height ‘h’
    - Excess flow ‘e’
  - Edges have pairs:
    - First value is edge capacity
    - Second value is flow
- Initialize:
  - s has h=6 i.e. number of nodes
  - Push 10 along (s,a)
    - e(a) = 10
  - Push 12 along (s,c)
    - e(c) = 12



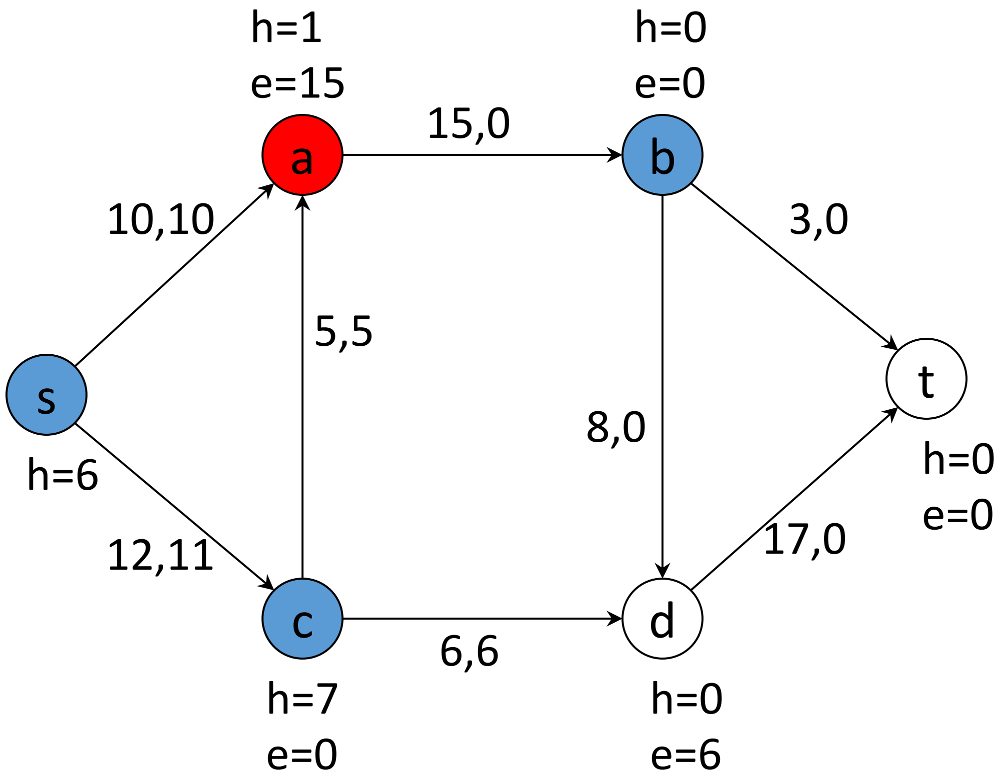
Example

- Relabel c with h=1
  - Push 6 along (c,d)
    - e(d) = 6
  - Push 5 along (c,a)
    - e(a) = 5
- Relabel c with h=7
  - Push -1 along (s,c)



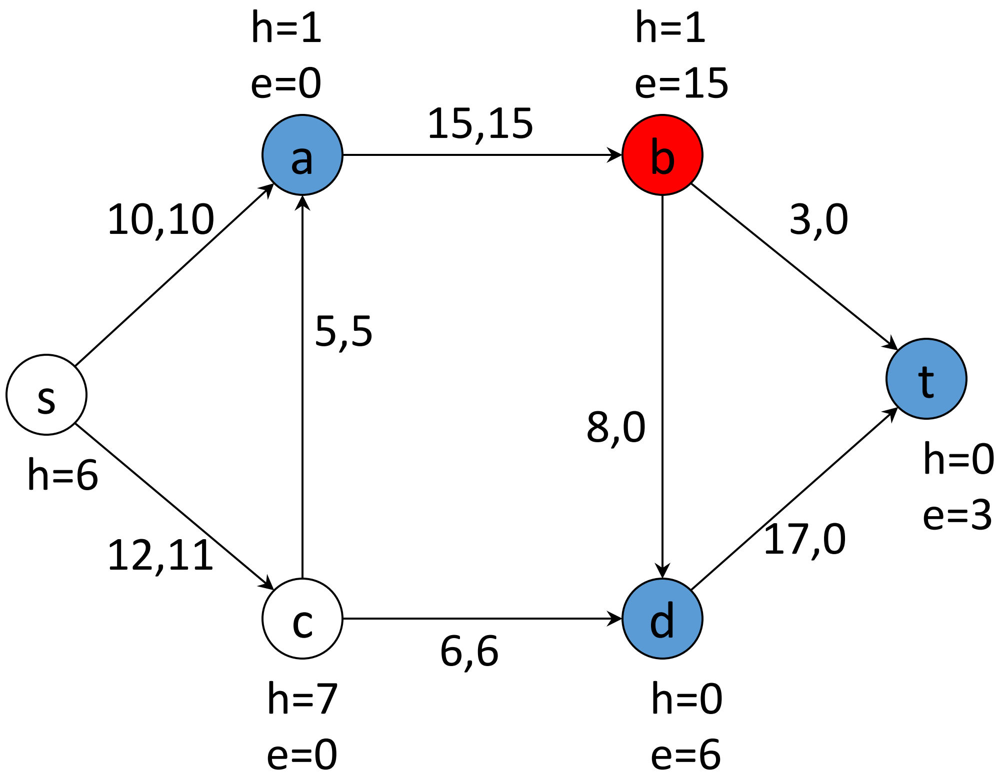
Example

- Relabel a with h=1
  - Push 15 along (a,b)
    - e(b) = 15



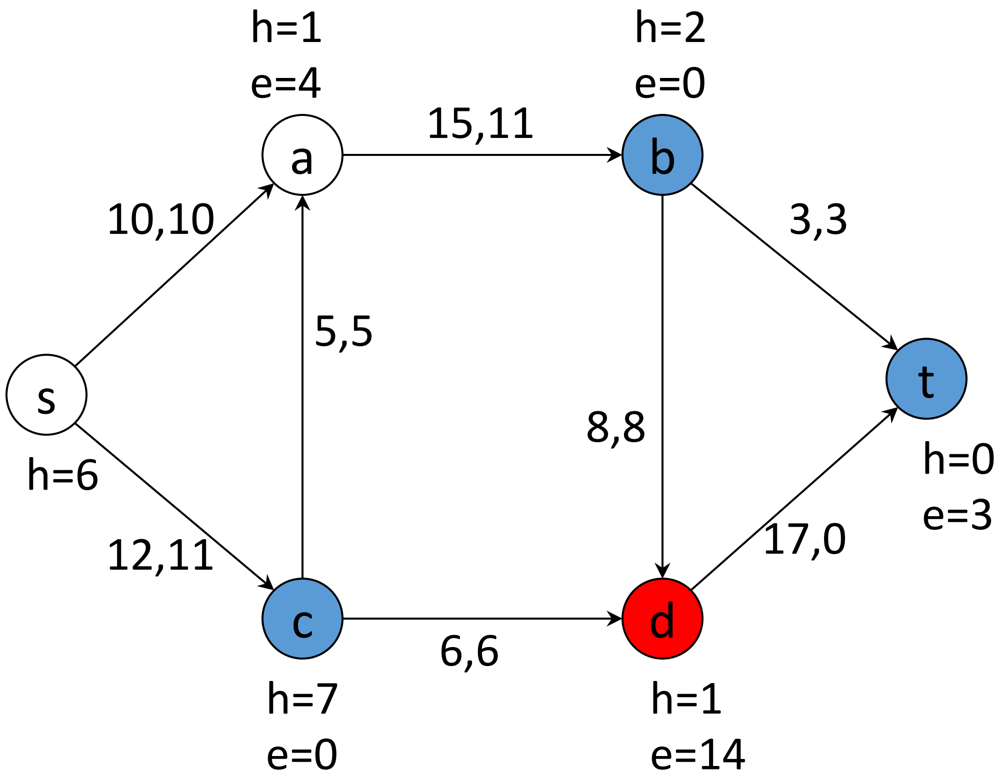
Example

- Relabel b with h=1
  - Push 3 along (b,t)
    - e(t) = 3
  - Push 8 along (b,d)
    - e(d) = 14
- Relabel b with h=2
  - Push -4 along (a,b)
    - e(a) = 4



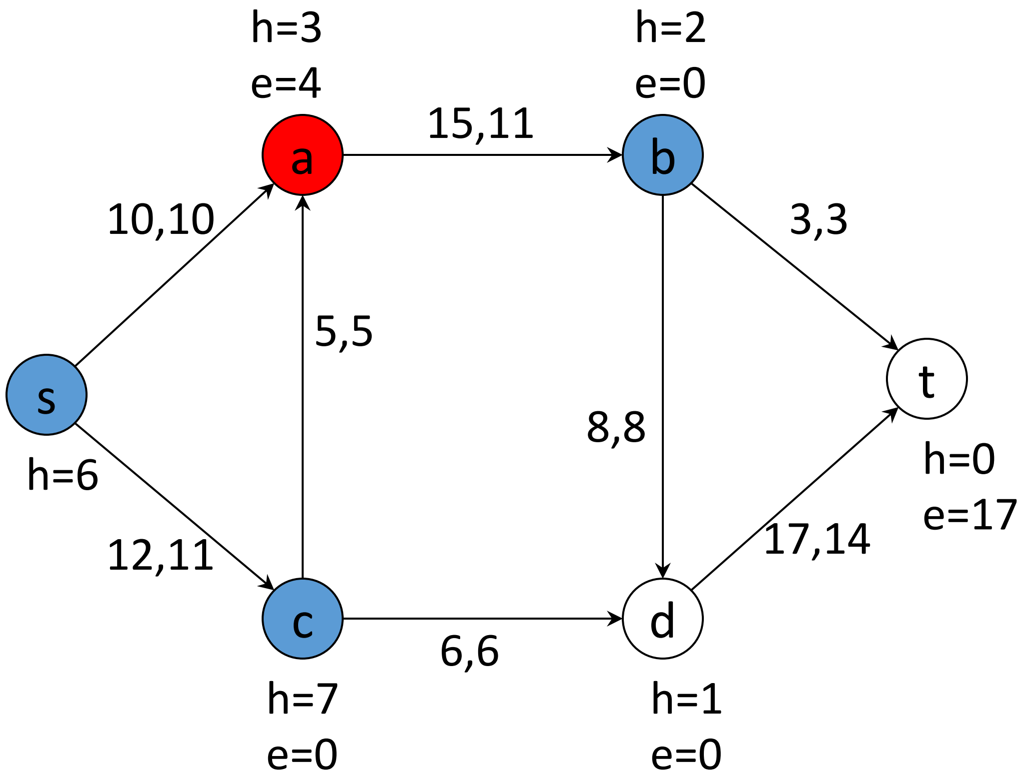
Example

- Relabel d with h=1
  - Push 14 along (d,t)
    - e(t) = 17



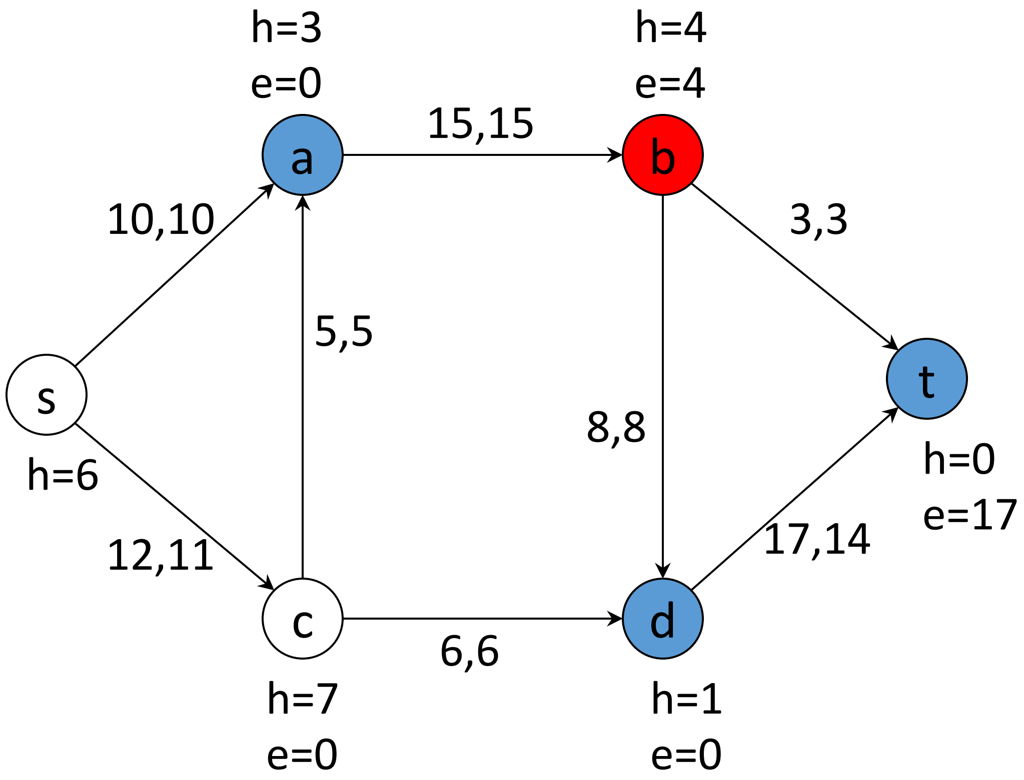
Example

- Relabel a with h=3
  - Push 4 along (a,b)
    - e(b) = 4



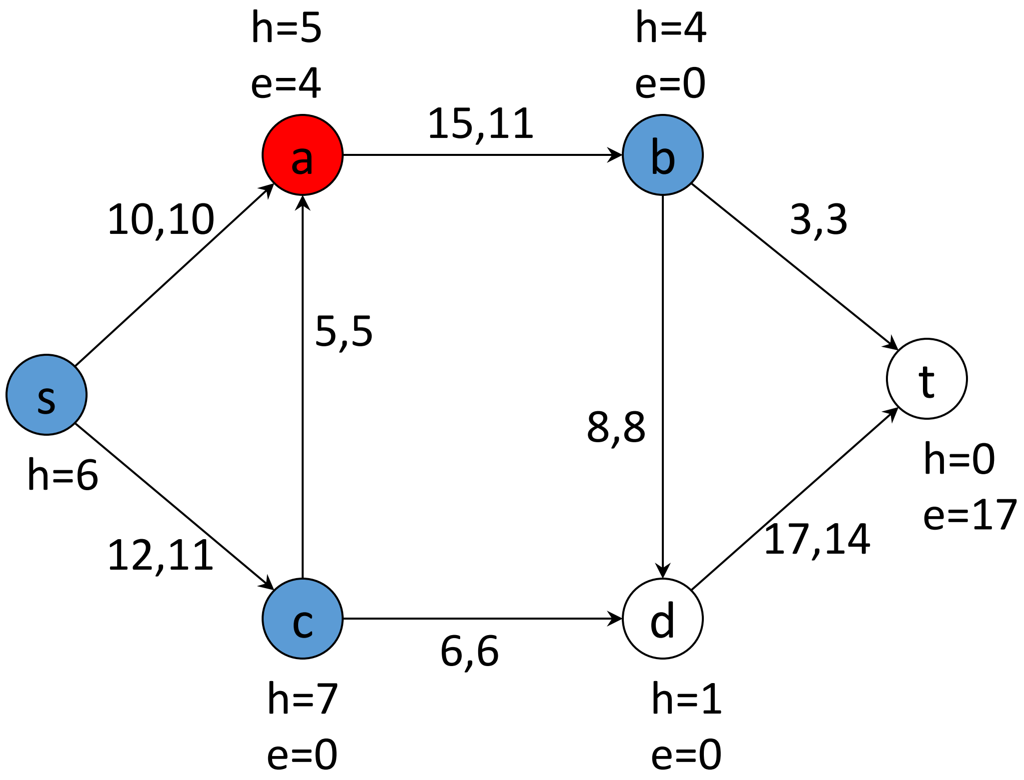
Example

- Relabel b with h=4
  - Push -4 along (a,b)
    - e(a) = 4



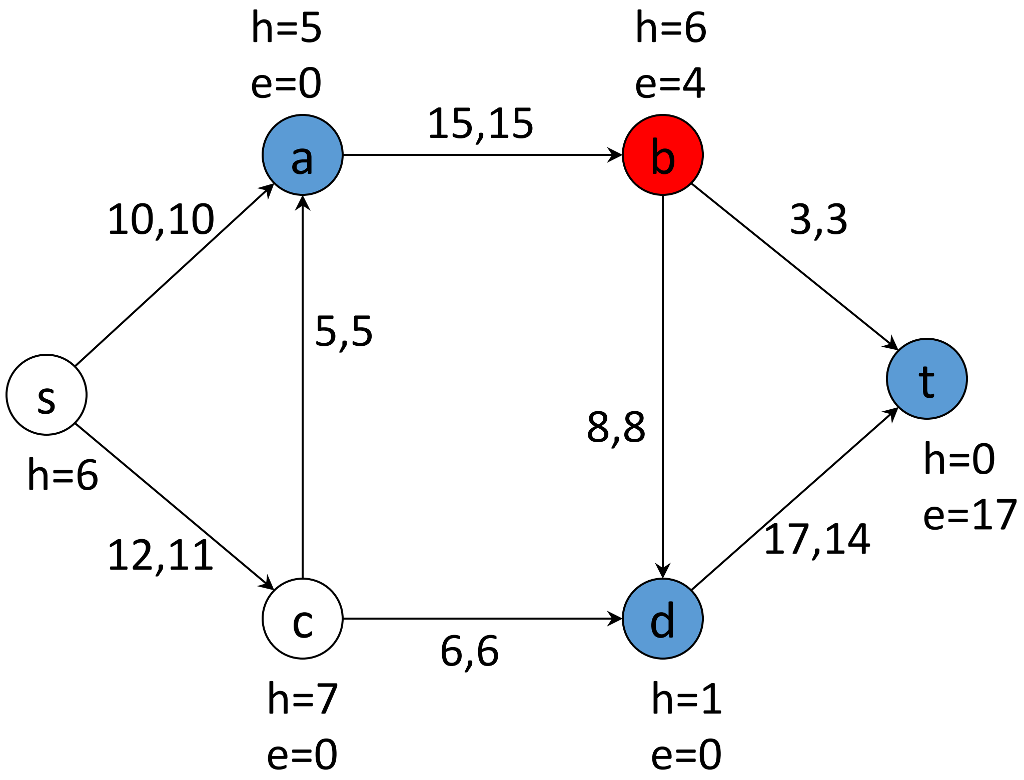
Example

- Relabel a with h=5
  - Push 4 along (a,b)
    - e(b) = 4



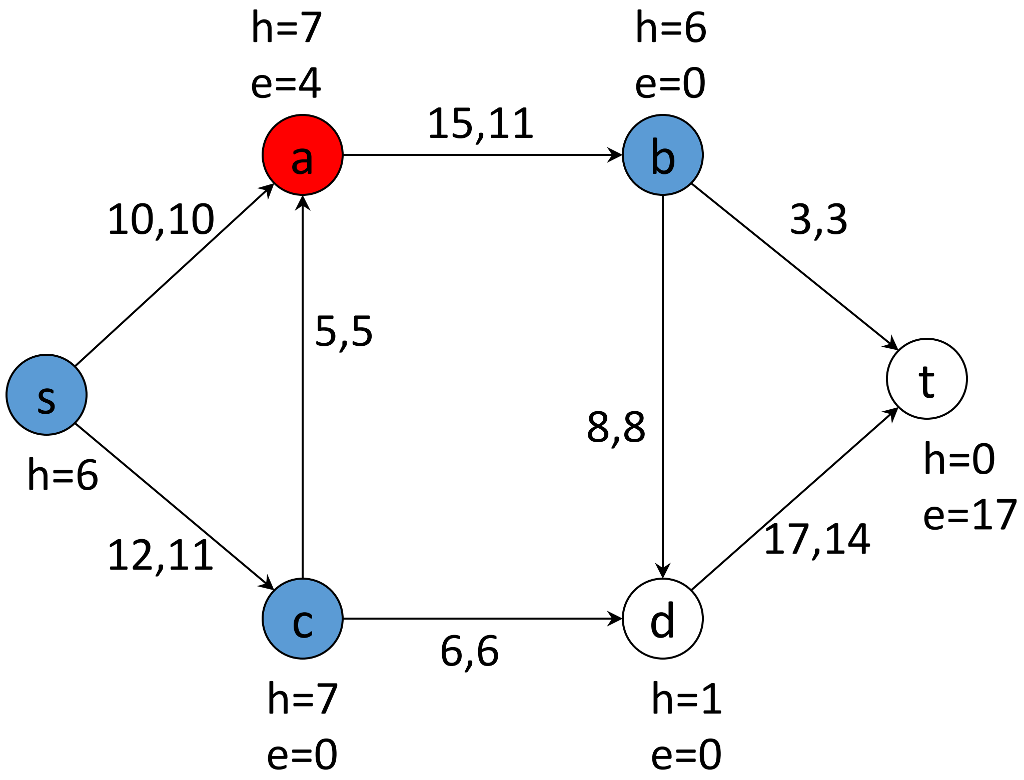
Example

- Relabel b with h=6
  - Push -4 along (a,b)
    - e(a) = 4



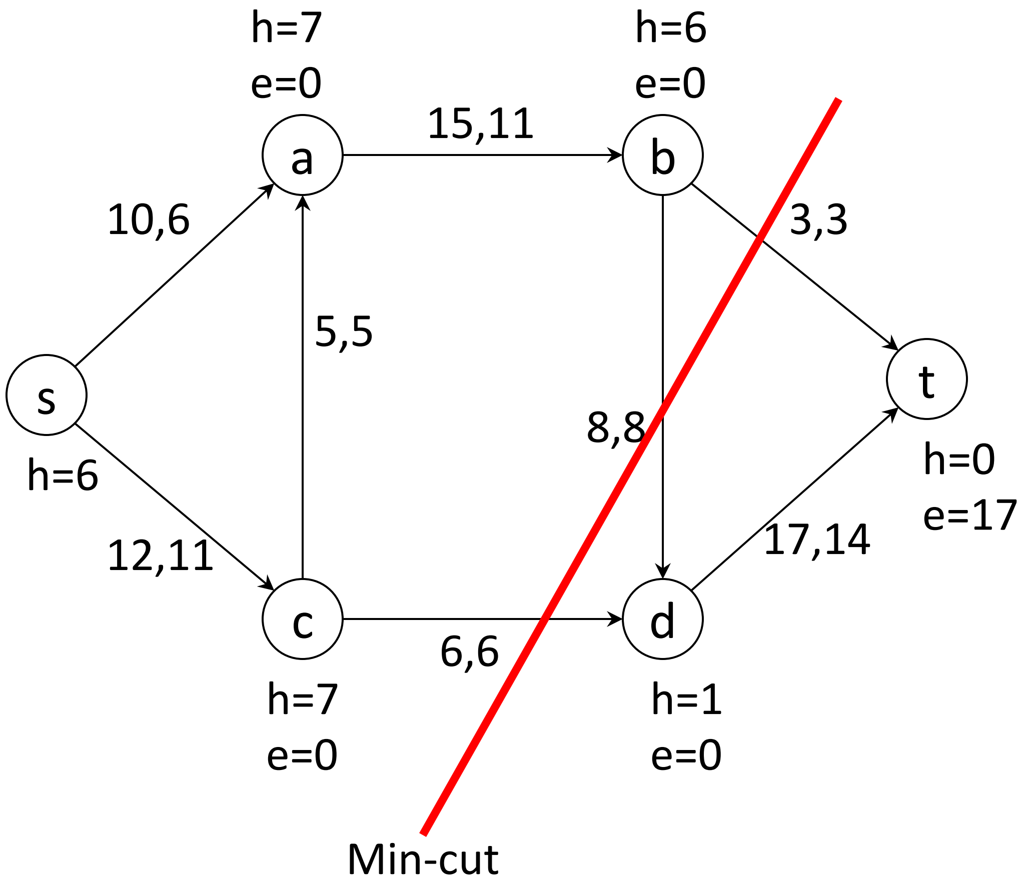
Example

- Relabel a with h=7
  - Push -4 along (s,a)
    - e(a) = 0



Example

- No active nodes left
- The algorithm terminates



Pseudo code

```
Worklist wl = initializePreflowPush();
while (!wl.isEmpty()) {
  Node n = wl.remove();
  n.relabel();
  for (Node w in n.neighbors()) {
    if (n can push flow to w) {
      pushflow(n, w);
      wl.add(w);
    }
  }
  if (n has excess flow) {
    wl.add(n);
  }
}
```