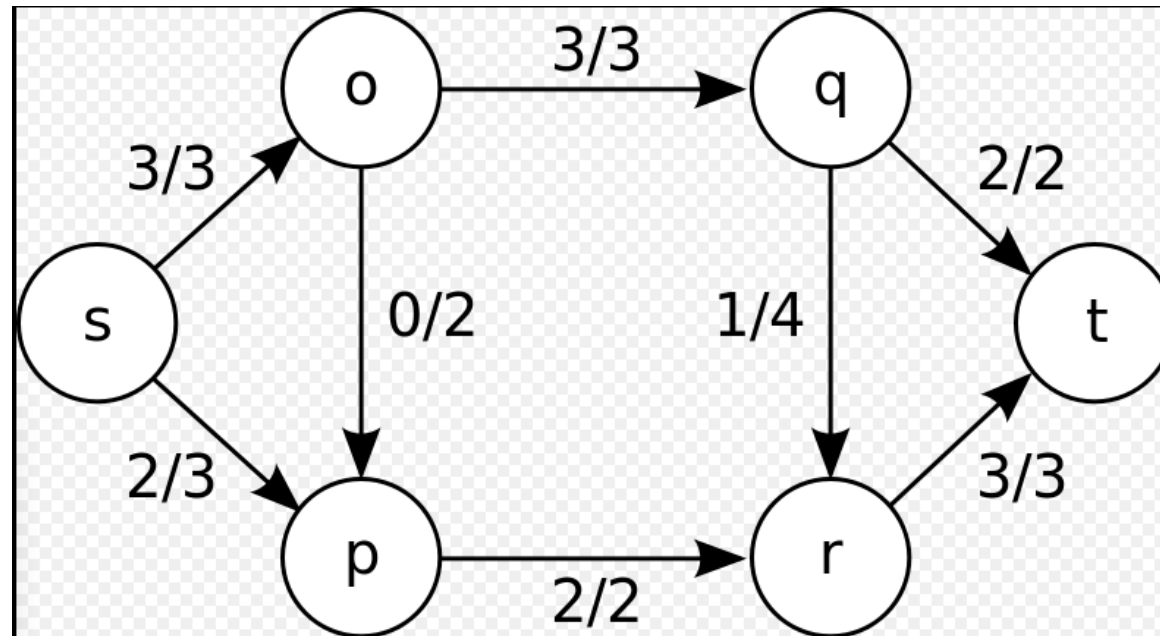


Parallel Graph Algorithms (continued)

MaxFlow



- A **flow network** $G=(V,E)$: a directed graph, where each edge $(u,v) \in E$ has a nonnegative **capacity** $c(u,v) \geq 0$.
- If $(u,v) \notin E$, we assume that $c(u,v)=0$.
- Two distinct vertices : **source s** and **sink t**.

Find $f: E \rightarrow \mathbb{R}$, such that

- **Capacity constraint:** For all $u, v \in V$,
we require $f(u, v) \leq c(u, v)$
- **Flow conservation:** For all $u \in V \setminus \{s, t\}$,
we require $\sum_{e.in.v} f(e) = \sum_{e.out.v} f(e)$
- **Maximize** $|f| = \sum_{v \in V} f(s, v)$

A Long History

Initially defined by Ford and Fulkerson (1956)

Date	Discoverer	Running time
1969	Edmonds and Karp	$O(nm^2)$
1970	Dinic	$O(n^2m)$
1974	Karzanov	$O(n^3)$
1977	Cherkasky	$O(n^2m^{1/2})$
1978	Malhotra, Pramodh Kumar, and Maheshwari	$O(n^3)$
1978	Galil	$O(n^{5/3}m^{2/3})$
1978	Galil and Naamad; Shiloach	$O(nm(\log n)^2)$
1980	Sleator and Tarjan	$O(nm \log n)$
1982	Shiloach and Vishkin	$O(n^3)$
1983	Gabow	$O(nm \log U)$
1984	Tarjan	$O(n^3)$
1985	Goldberg	$O(n^3)$
1986	Goldberg and Tarjan	$O(nm \log(n^2/m))$
1986	Ahuja and Orlin	$O(nm + n^2 \log U)$

MaxFlow for sparse digraphs with m edges and integer capacities between 1 and C

1997	length function	$O(m^{3/2} \log m \log C)$	Goldberg-Rao
2012	compact network	$O(m^2 / \log m)$	Orlin
?	?	$O(m)$?

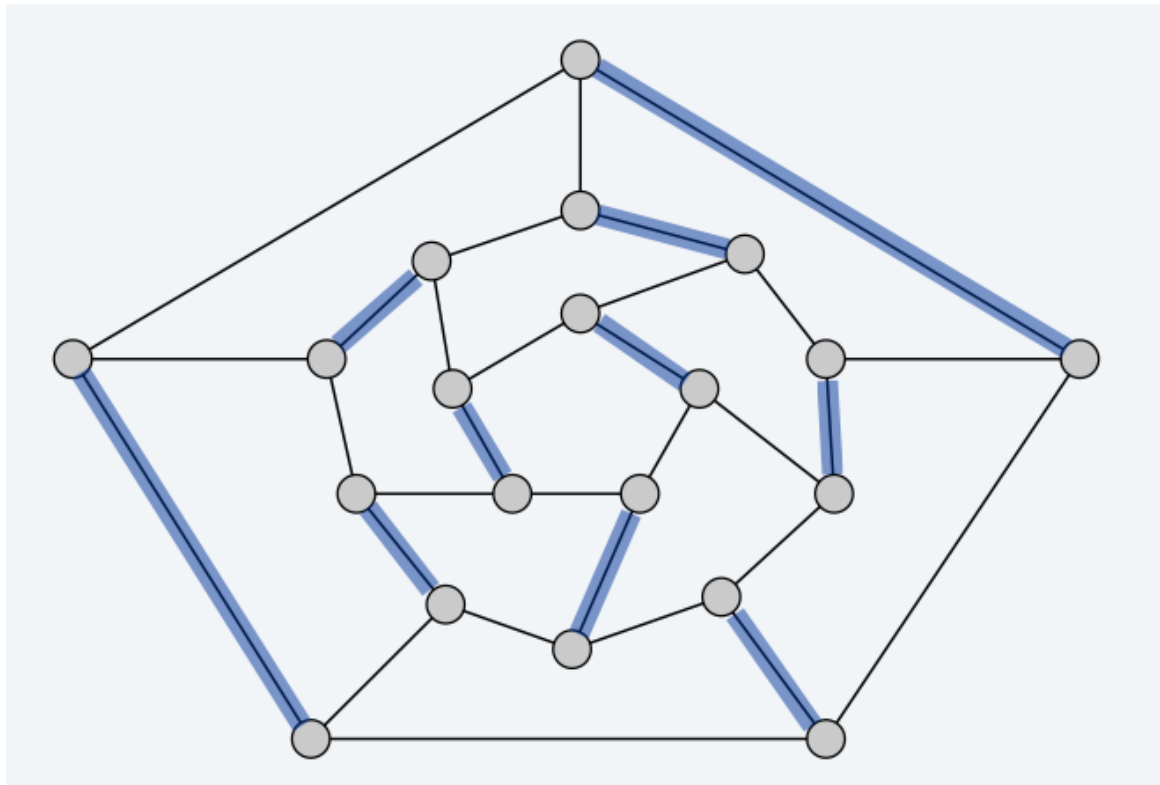
Applications

- Data mining.
- Open-pit mining.
- Bipartite matching.
- Network reliability.
- Baseball elimination.
- Image segmentation.
- Network connectivity.
- Distributed computing.
- Security of statistical data.
- Egalitarian stable matching.
- Network intrusion detection.
- Multi-camera scene reconstruction.
- Sensor placement for homeland security.
- Many, many, more.

Example: Matching

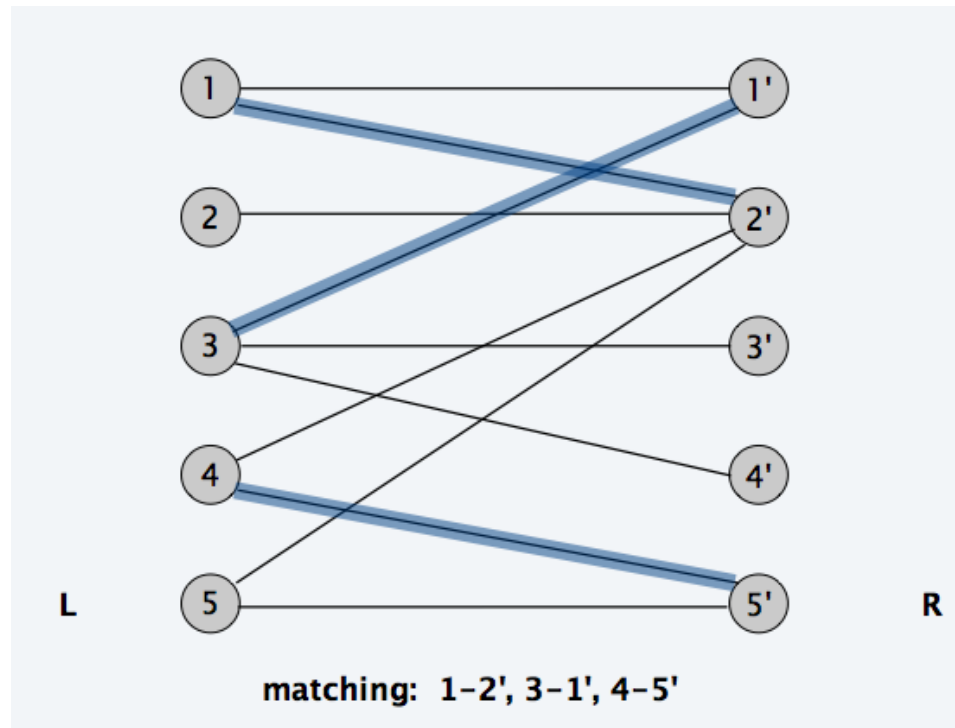
Given an undirected graph $G = (V, E)$ a subset of edges $M \subseteq E$ is a **matching** if each node appears in at most one edge in M .

Max matching: Given a graph, find a max cardinality matching.



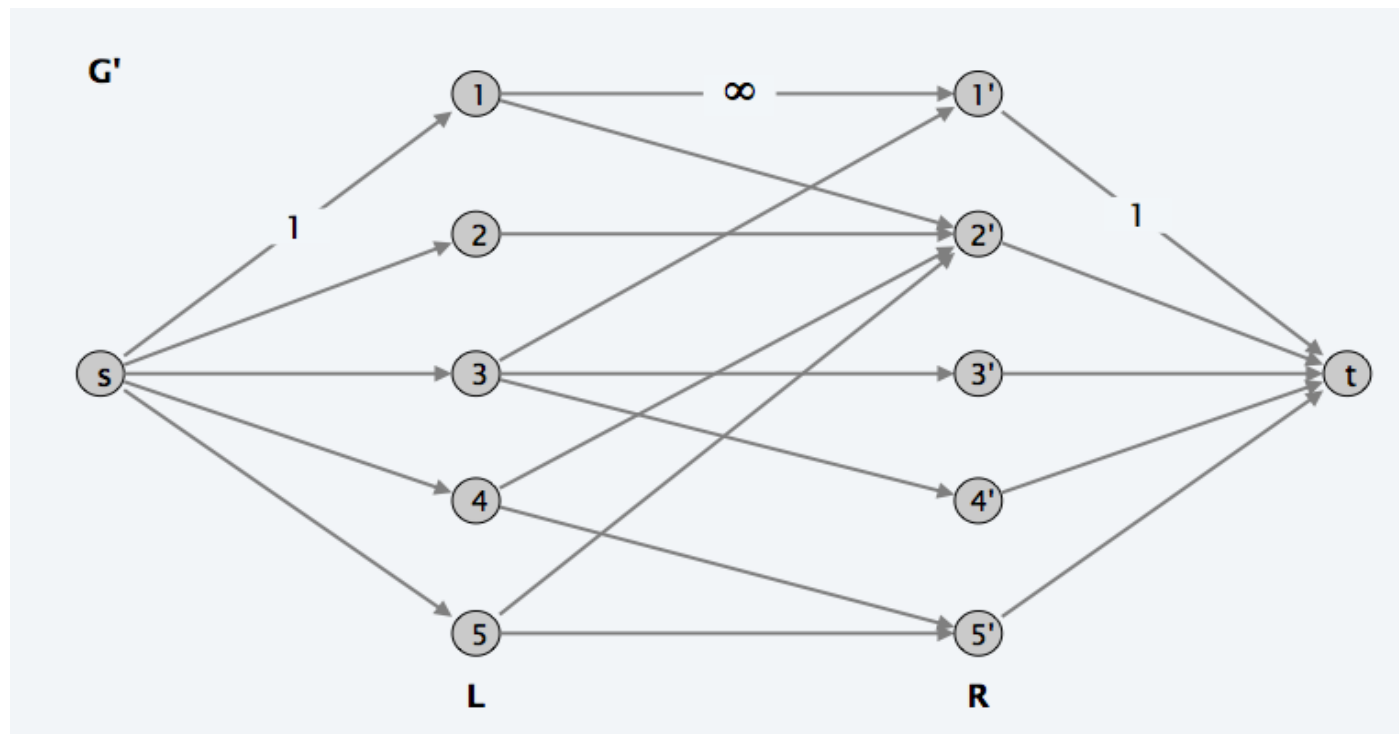
Bipartite Matching

A graph G is **bipartite** if the nodes can be partitioned into two subsets L and R such that every edge connects a node in L to one in R



Bipartite Matching: Maxflow Formulation

- Create digraph $G' = (L \cup R \cup \{s, t\}, E')$.
- Direct all edges from L to R, and assign infinite (or unit) capacity.
- Add source s , and unit capacity edges from s to each node in L.
- Add sink t , and unit capacity edges from each node in R to t .



Solving MaxFlow: The Ford-Fulkerson method

The Ford-Fulkerson method depends on three important ideas that transcend the method and are relevant to many flow algorithms and problems: **residual networks**, **augmenting paths**, and **cuts**. These ideas are essential to the important max-flow min-cut theorem, which characterizes the value of maximum flow in terms of cuts of the flow network.

FORD-FULKERSON-METHOD(G,s,t)

initialize flow f to 0

while there exists an *augmenting* path p

do *augment* flow f along p

return f

Augmenting Paths

- Given a flow network and a flow, the **residual network** consists of edges that can admit more net flow.
- The amount of additional net flow from u to v before exceeding the capacity $c(u,v)$ is the **residual capacity** of (u,v) , given by:

$$c_f(u,v) = c(u,v) - f(u,v)$$

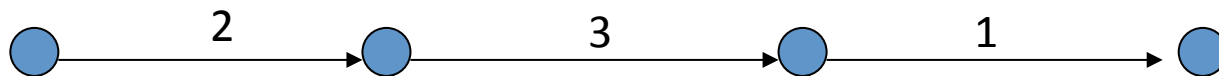
and in the other direction:

$$c_f(v,u) = c(v,u) + f(u,v).$$

- If f is a flow in G and f' is a flow in the residual network G_f then $f+f'$ is also a valid flow in G

Augmenting Paths

- Given a flow network $G=(V,E)$ and a flow f , an **augmenting path** is a simple path from s to t in the residual network G_f .
- **Residual capacity** of p : the maximum amount of net flow that we can ship along the edges of an augmenting path p , i.e.,
 $c_f(p)=\min\{c_f(u,v):(u,v) \text{ is on } p\}$.



The residual capacity is 1

The basic Ford-Fulkerson algorithm

FORD-FULKERSON(G, s, t)

for each edge $(u, v) \in E[G]$

do $f[u, v] = 0$

$f[v, u] = 0$

while there exists a **path** p from s to t in the residual network G_f

do $c_f(p) = \min\{c_f(u, v) : (u, v) \text{ is in } p\}$

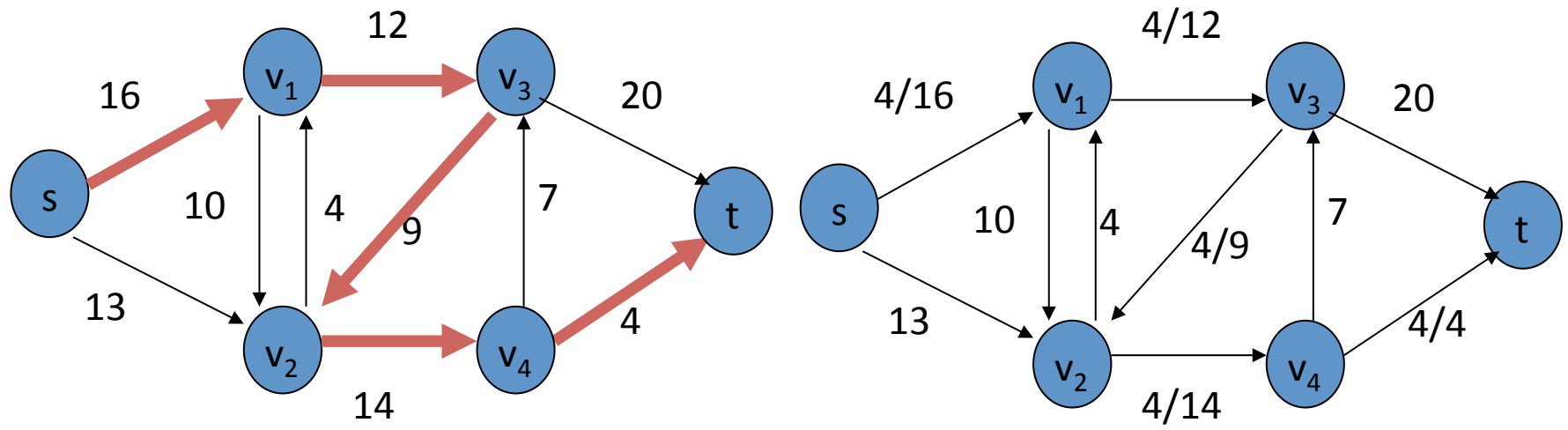
for each edge (u, v) in p

do $f[u, v] = f[u, v] + c_f(p)$

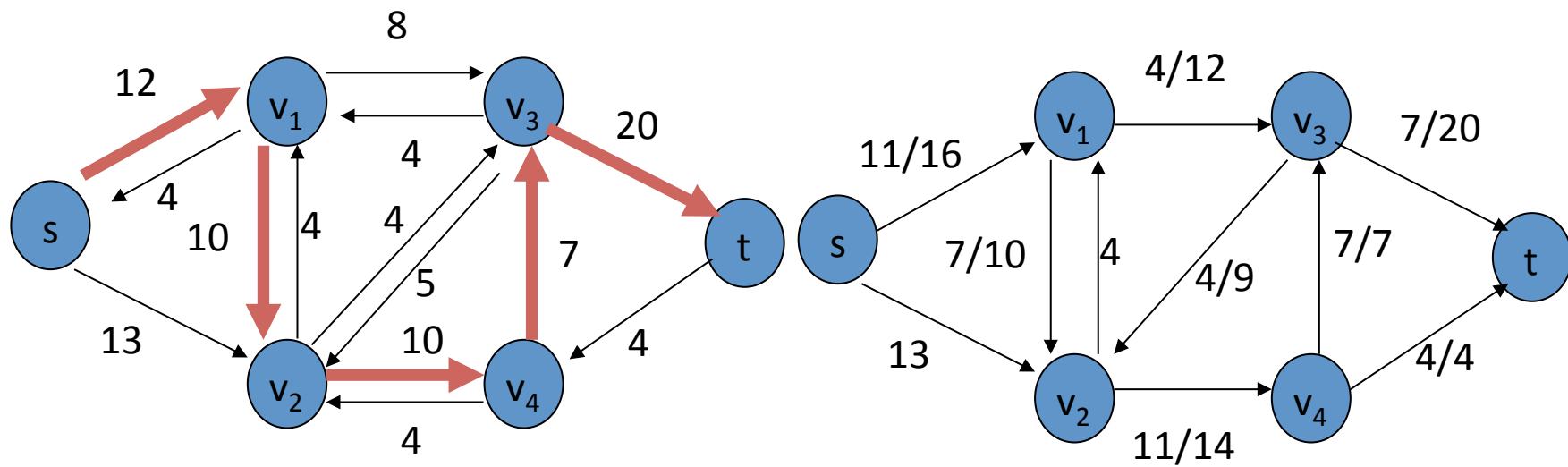
Example Execution

In the following slides:

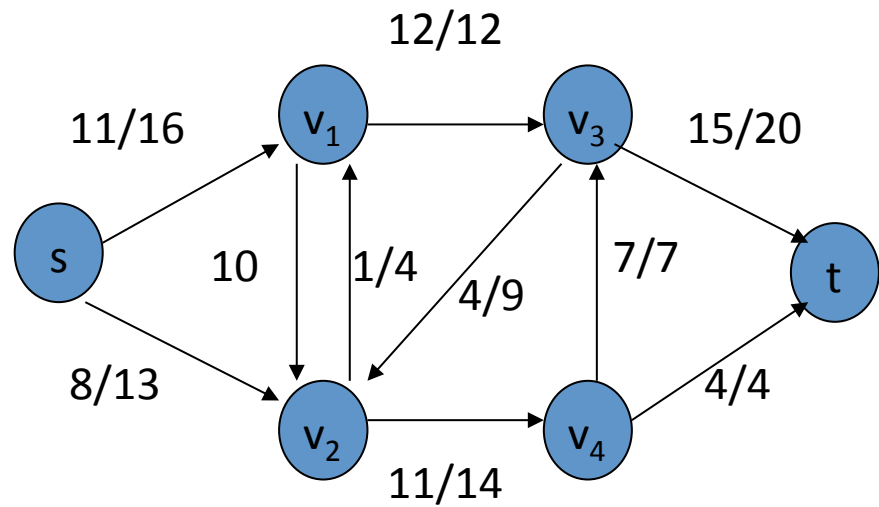
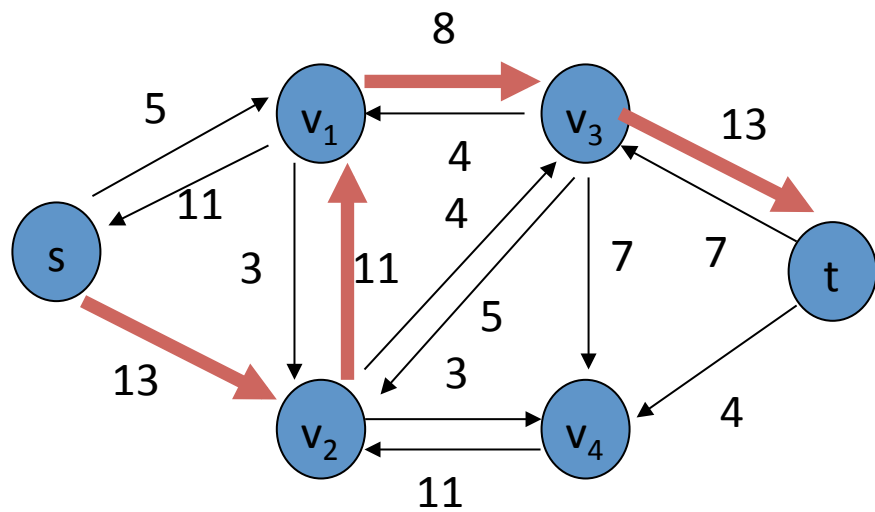
(a)-(d) Successive iterations of the **while** loop: The left side of each part shows the residual network G_f from line 4 with a shaded augmenting path p . The right side of each part shows the new flow f that results from adding f_p to f . The residual network in (a) is the input network G . (e) The residual network at the last **while** loop test. It has no augmenting paths, and the flow f shown in (d) is therefore a maximum flow.



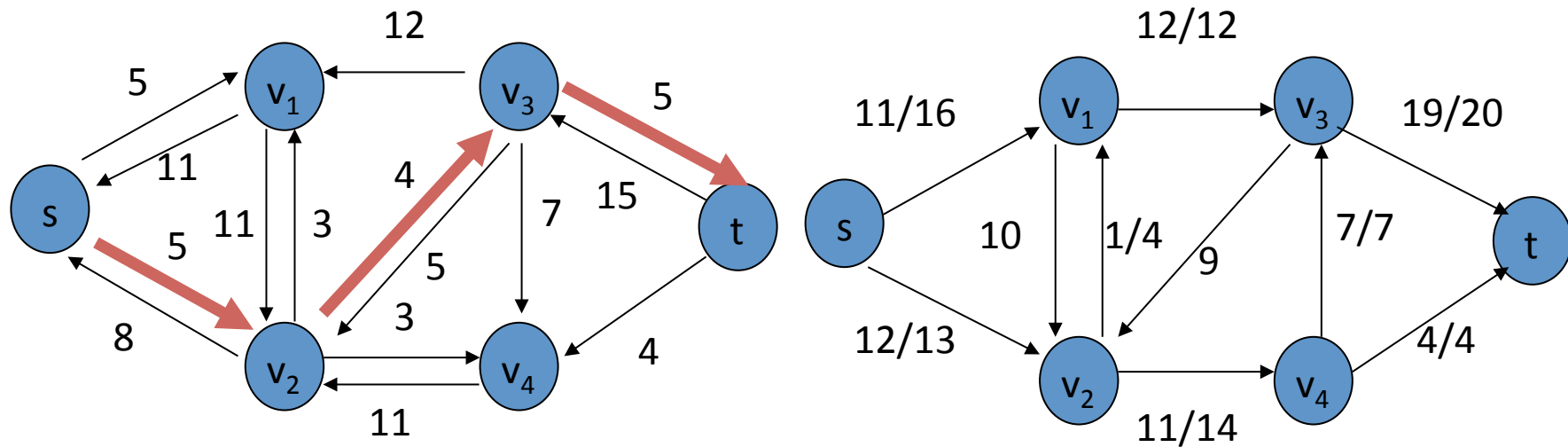
(a)



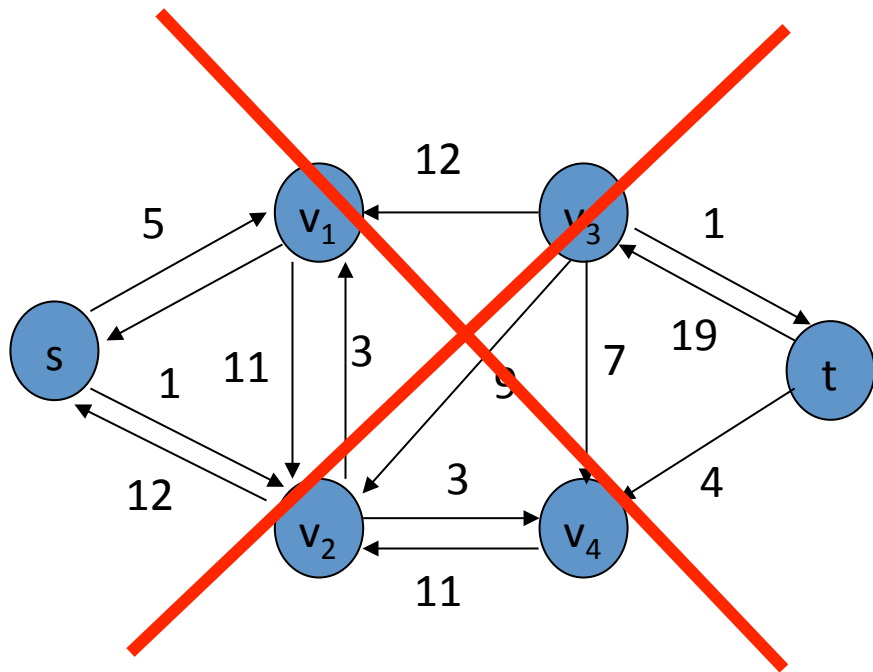
(b)



(c)



(d)



(e)

The Edmonds-Karp algorithm

A practical implementation of Ford Fulkerson

- Find the augmenting path using **breadth-first search**.
- Breadth-first search gives the shortest path for graphs. (Assuming the length of each edge is 1.)
- Time complexity of Edmonds-Karp algorithm is $O(VE^2)$.
- The proof is very hard and is not required here.

Relationship with Cut Sets

A **cut in a network** with source s and sink t is a subset $X \subset V$, such that

$$s \in X \text{ and } t \notin X$$

$(X, V \setminus X)$ is the set of edges from a vertex in X to a vertex in $V \setminus X$

$$|f| \leq C(X)$$

The **capacity** of a cut X equals:

$$C(X) = \sum_{x \in (X, V \setminus X)} c(x)$$

→ For every flow $f: E \rightarrow \mathbb{R}$ and cut X ,

$$|f| \leq C(X)$$

Max Flow == Min Cut

Theorem 1: A flow in a network G is maximal iff there exists no augmenting path in G

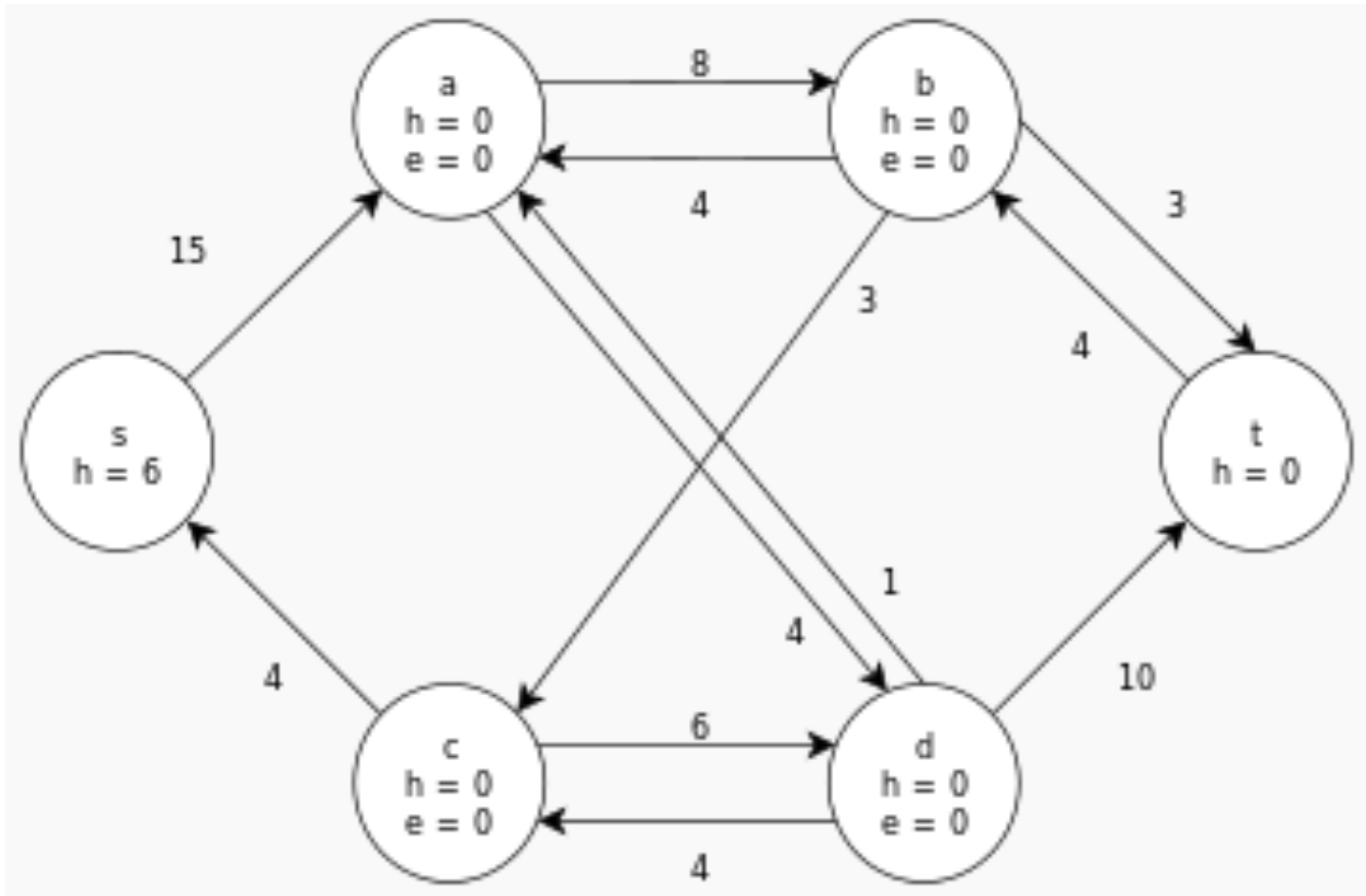
Theorem 2: The maximal flow in a network G equals the minimal capacity cut set of G

Proof (sketch) Given that f is a maximal flow in G . Construct X such that $s \in X$, and for all v for which there exists an augmenting path from s to v : $v \in X$. Then t cannot belong to X , because there is no augmenting path anymore. So X is a proper cut of G . So $C(X) = |f|$ and $|f| \leq C(Y)$ for any cut Y . So X is the minimal cut. The reverse follows trivially.

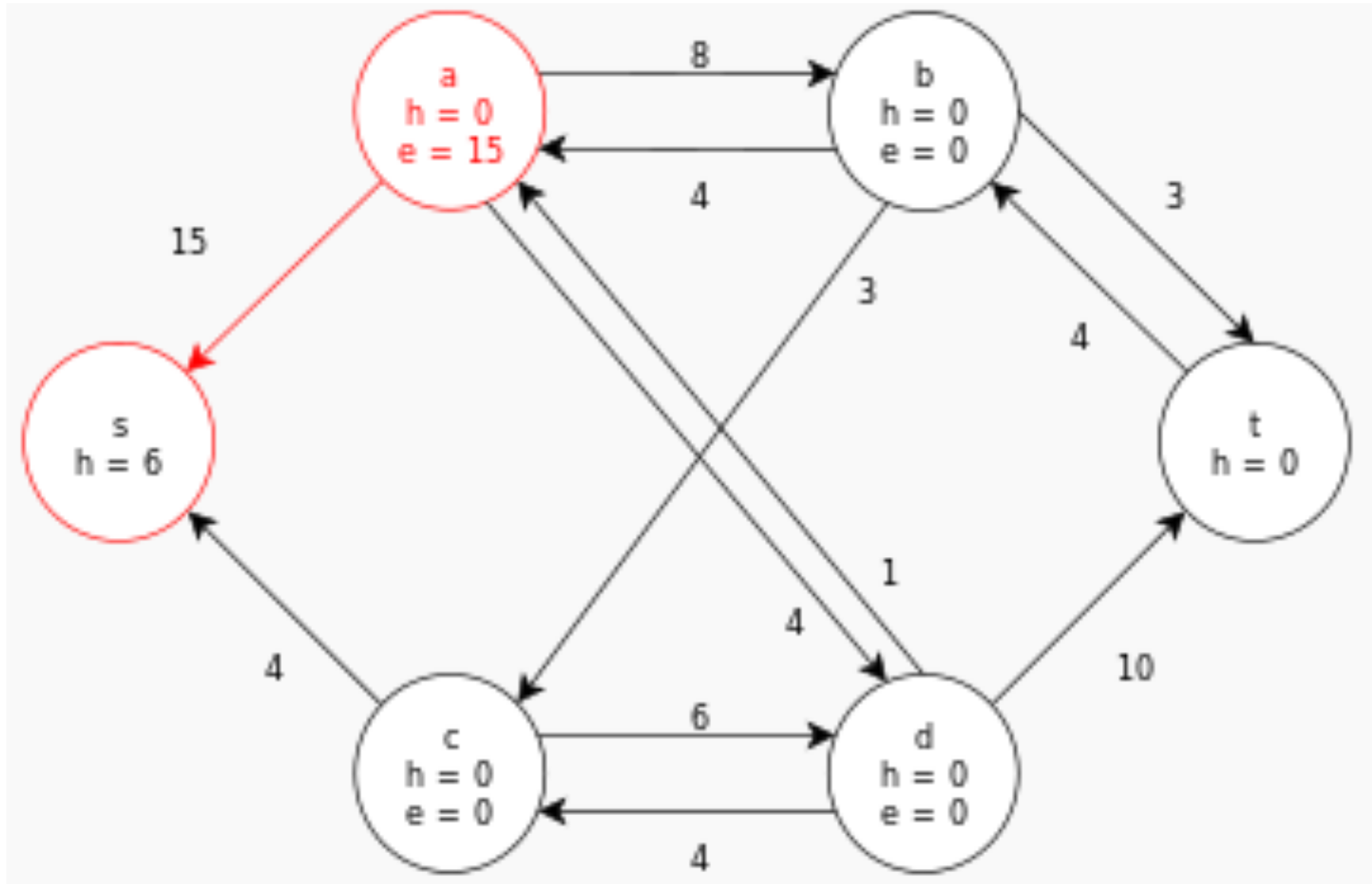
Push-Relabel Algorithm by Goldberg and Tarjan (JACM 1988)

- Input: network $(G = (V, E), s, t, c)$
- $h[s] := |V|$
- for each $v \in V - \{s\}$ do $h[v] := 0$
- for each $(s, v) \in E$ do $f(s, v) := c(s, v)$
- while f is not a feasible flow
 - let $c'(u, v) = c(u, v) + f(u, v) - f(v, u)$ be the capacities of the residual network
 - if there is a vertex $v \in V - \{s, t\}$ and a vertex $w \in V$ such that $e_f(v) > 0$, $h(v) > h(w)$, and $c'(v, w) > 0$ then
 - * push $\min\{c'(v, w), e_f(v)\}$ units of flow on the edge (v, w)
 - else, let v be a vertex such that $e_f(v) > 0$, and set $h[v] := h[v] + 1$
- output f

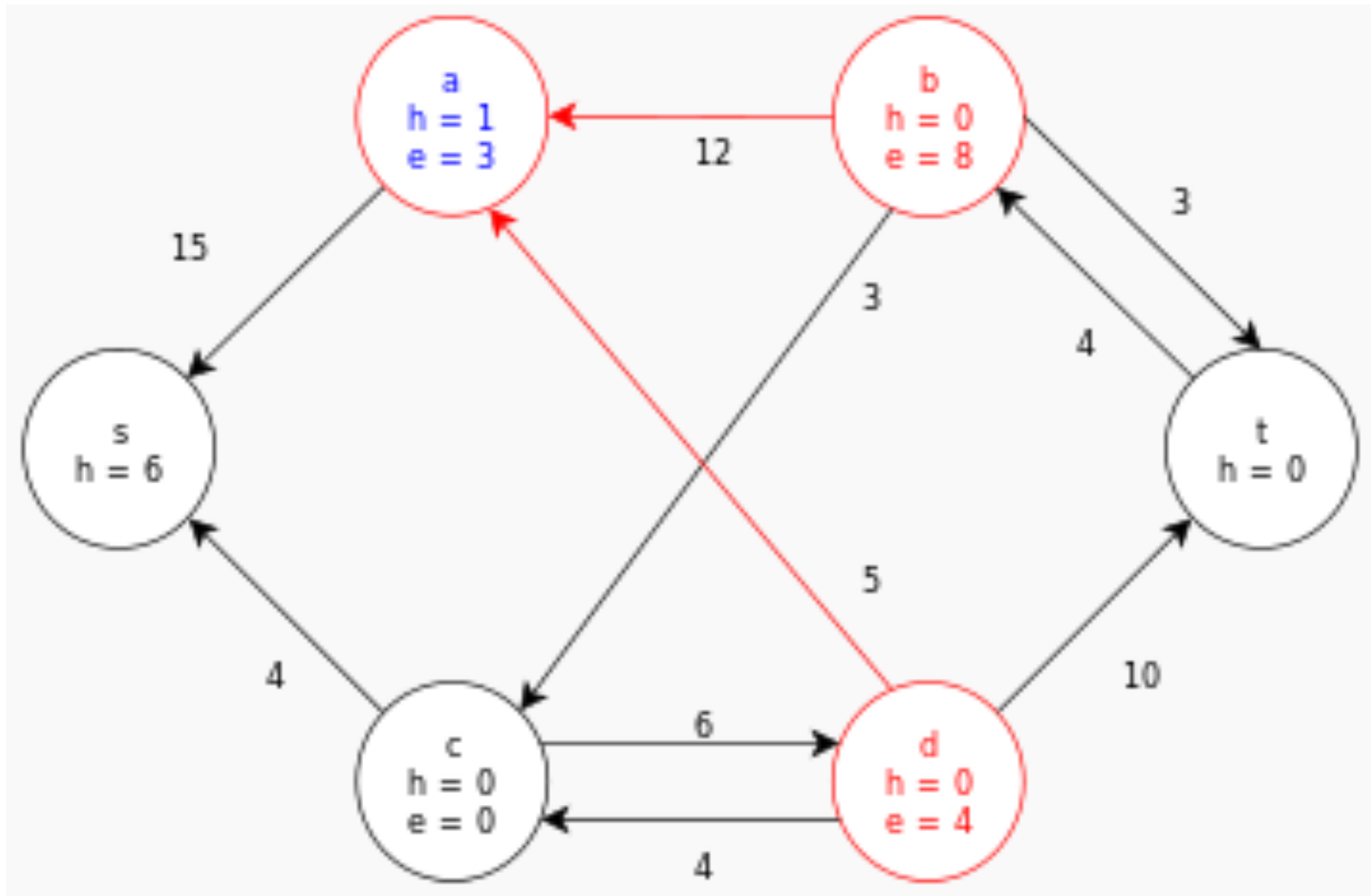
Example



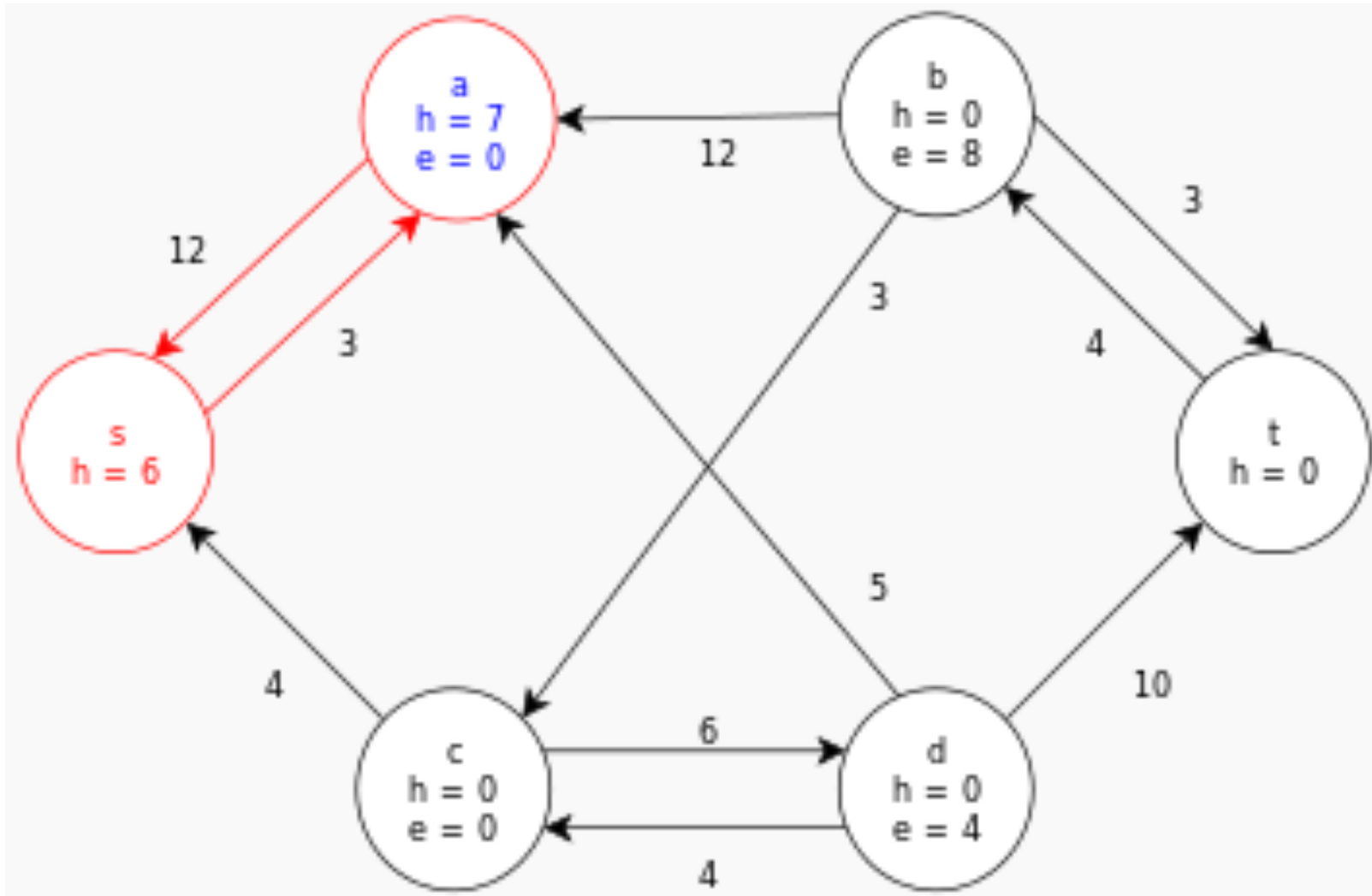
Excess flow is pushed to a



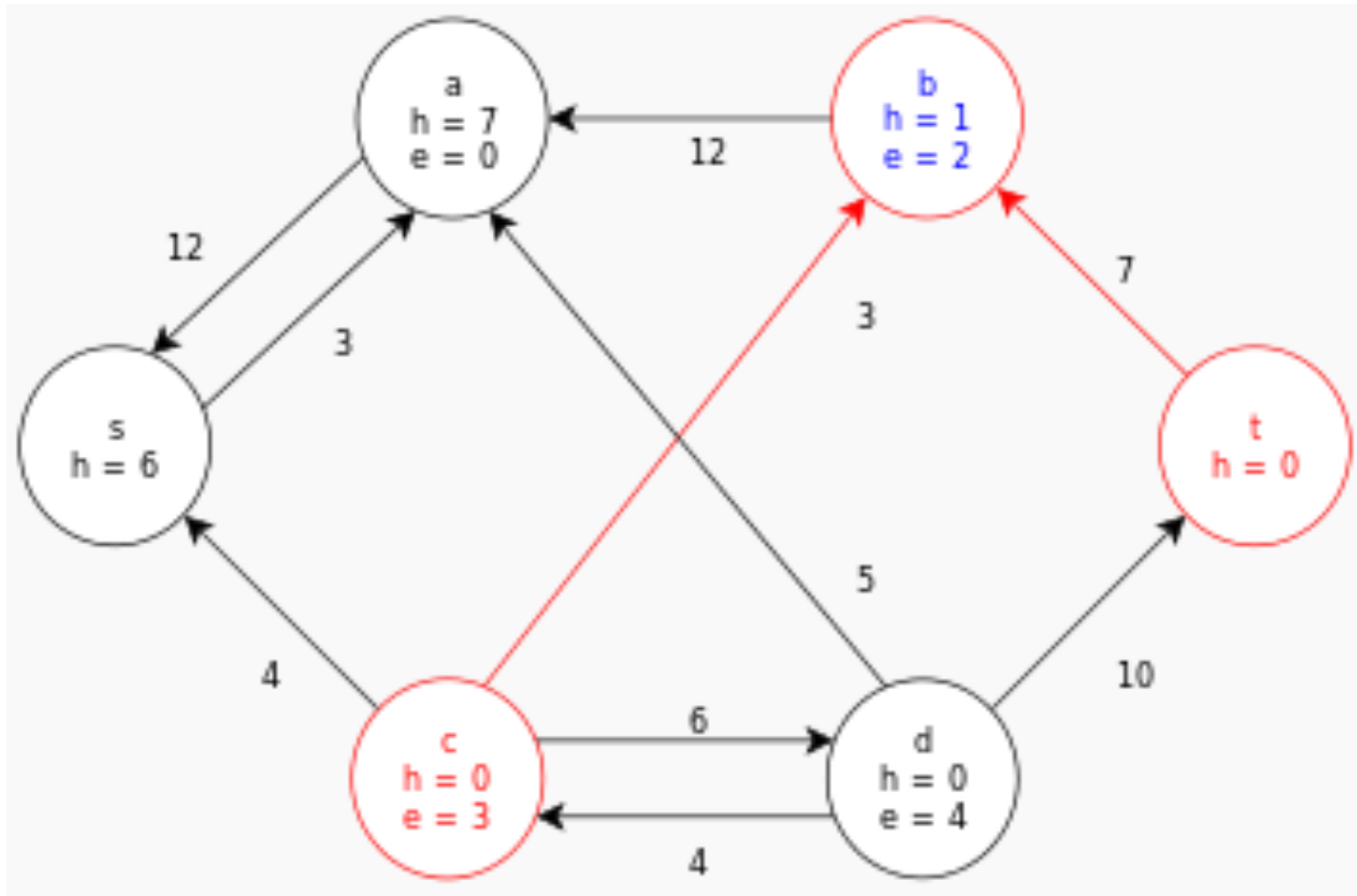
First $h[a]$ is incremented to 1 and then excess flow is pushed from a to b and d



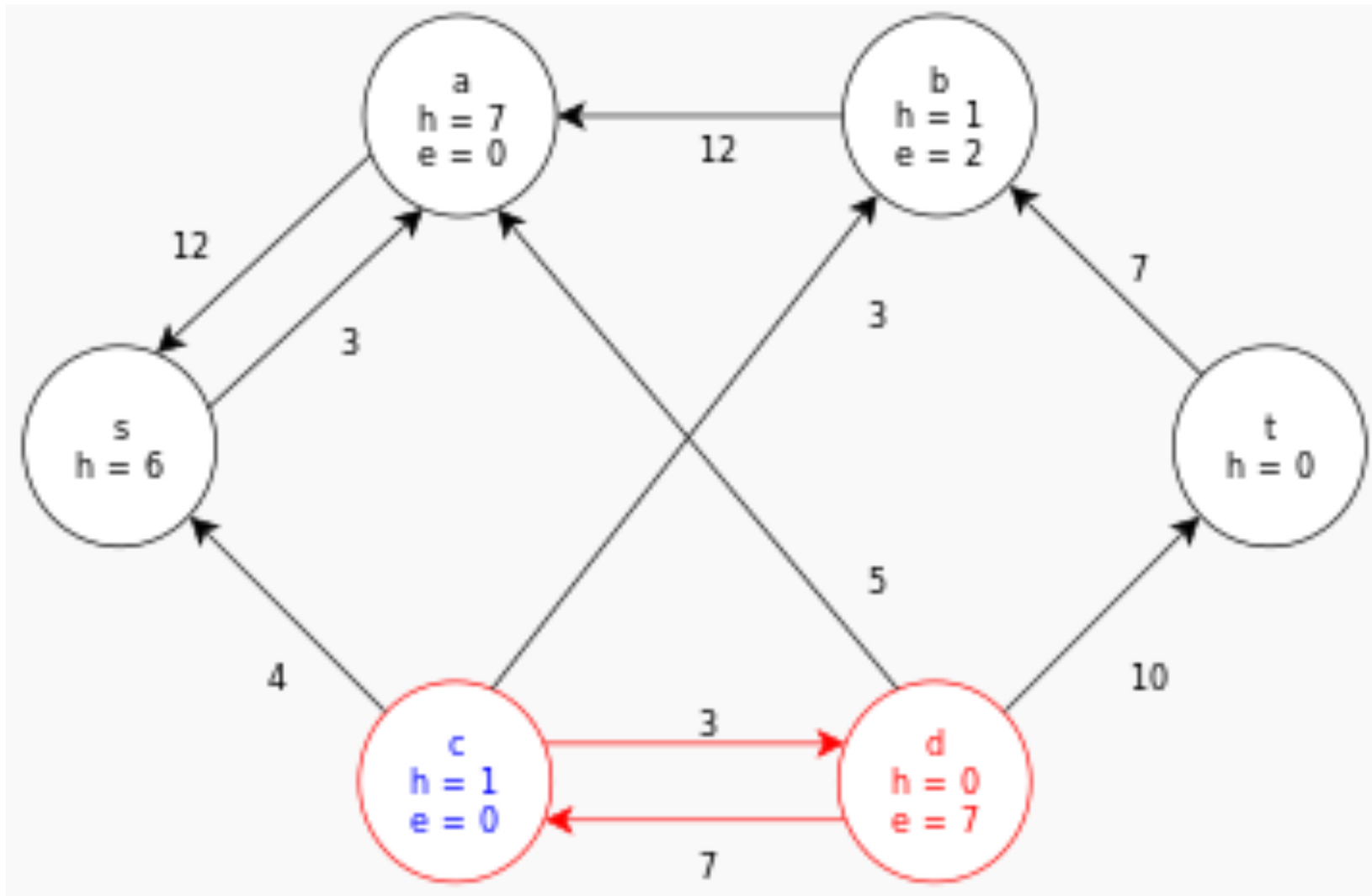
$h[a]$ is incremented to 7!! then excess flow is pushed (back) from a to s



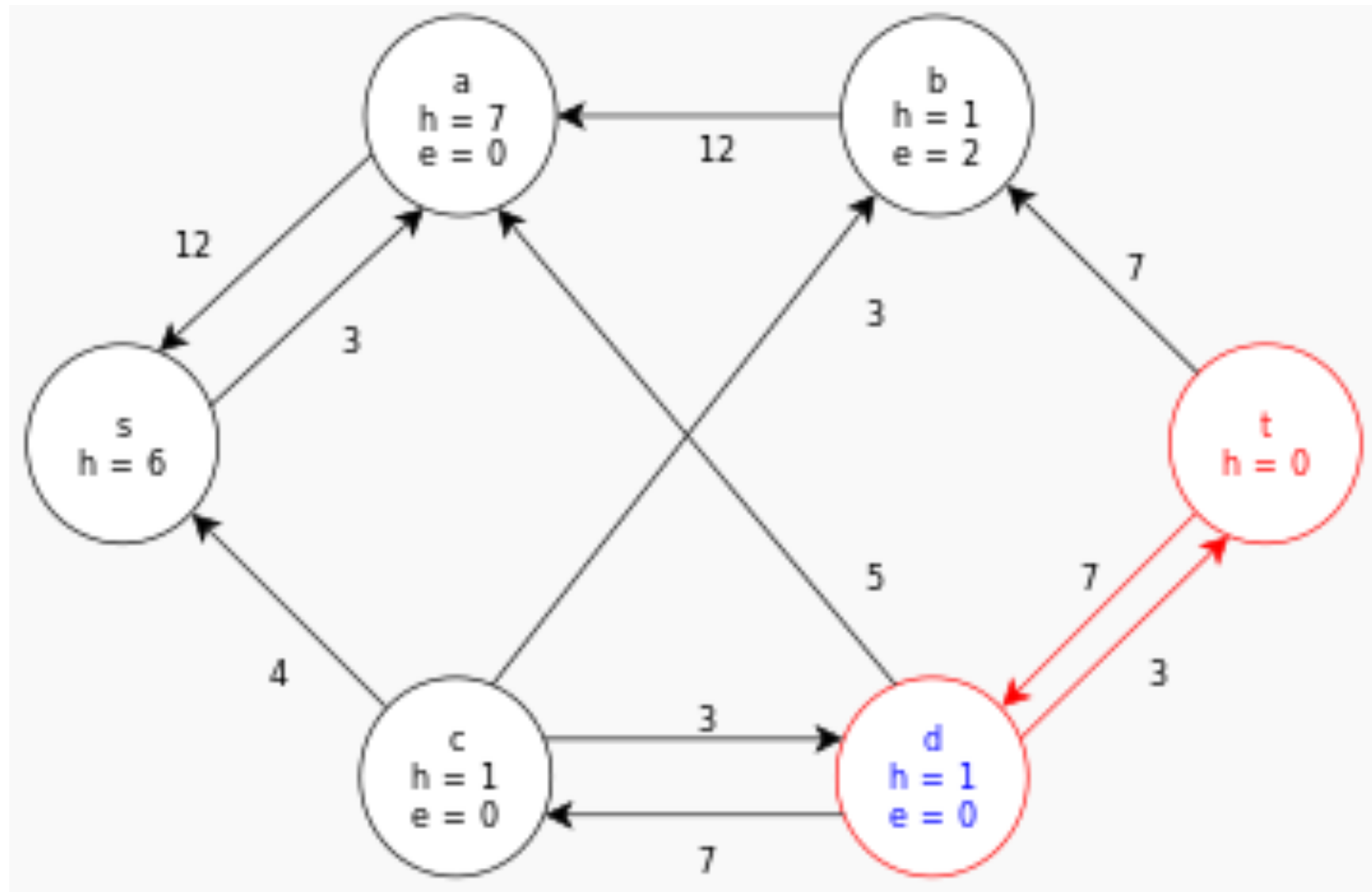
First $h[b]$ is incremented to 1, then excess flow is pushed from b to c and t



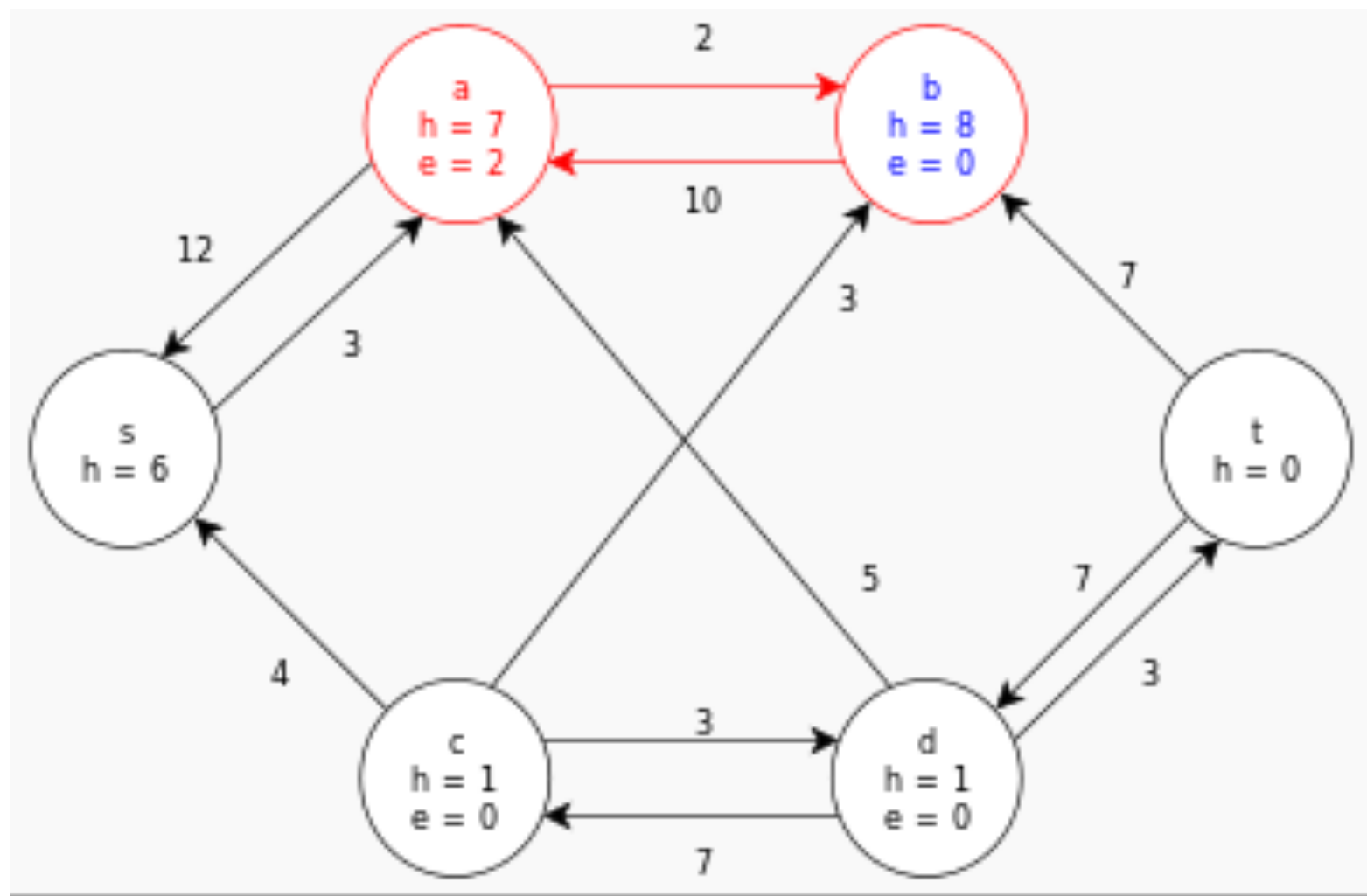
First $h[c]$ is incremented to 1, then excess flow is pushed from c to d



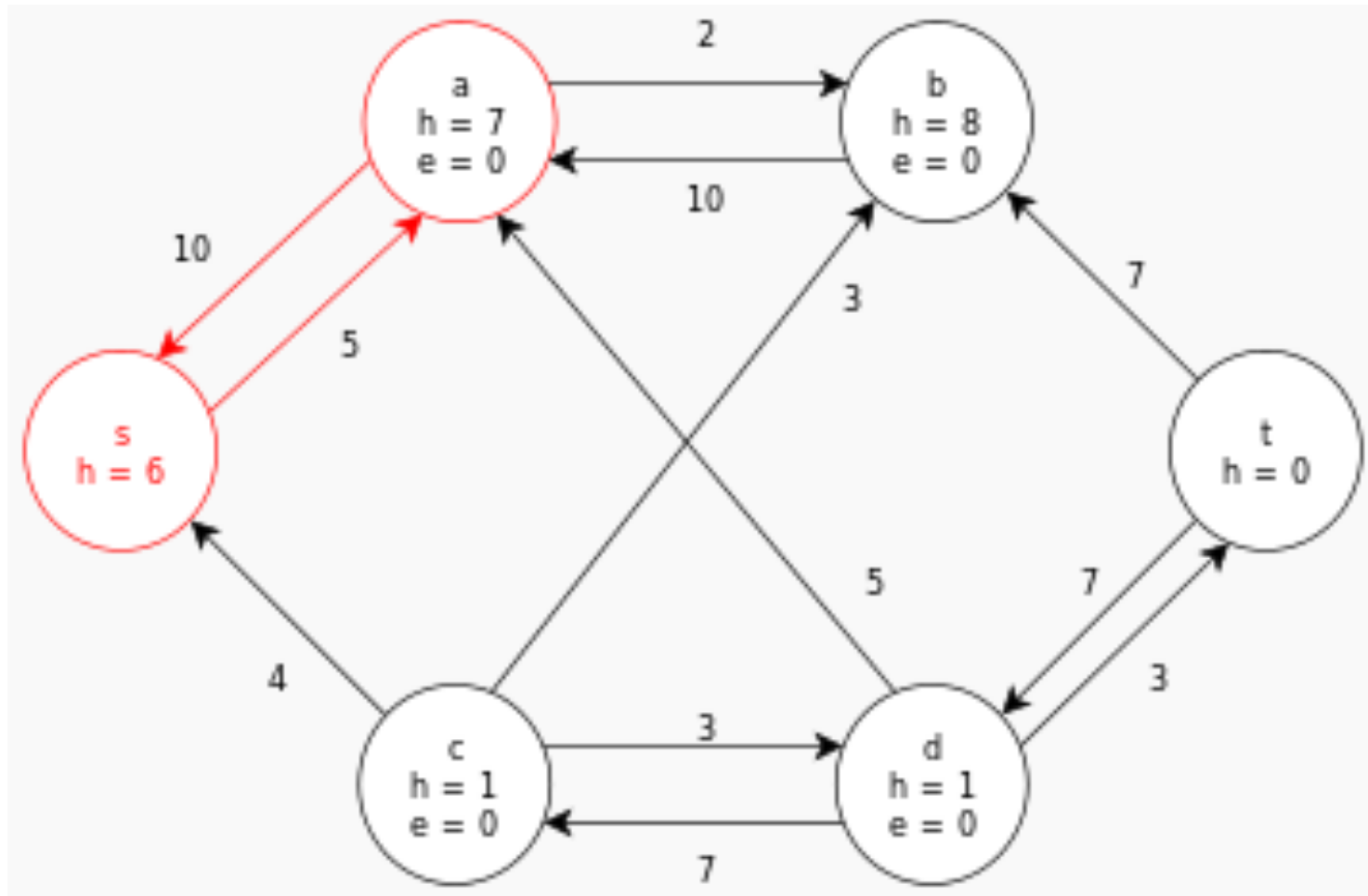
First $h[d]$ is incremented to 1, then excess flow is pushed from d to t



b is the only node with excess > 0, b has no outgoing residual edges, so $h[b]$ is incremented to 8 and b will push **back** excess flow to a



node a is the only active node with excess flow > 0 and will push flow back to s



A parallel version of push relabel

