Relationship between flows and cuts

- A ***minimum cut*** is a cut whose capacity is minimum among all possible cuts
- To prove the "if" part of the Augmenting Path Theorem, the idea is to define a cut $C = \{(u,v) \in E : u \in A \text{ and } v \in B\}$ such that $val(f) = cap(C)$
- Then f must be a maximum flow (not proved here), because no additional flow can get from a vertex in A to a vertex in B
- Let $G = (V,E)$ be a network with source s, sink t, capacity function c and flow f, and suppose that f admits no augmenting path
- Let $A \subseteq V$ be the set of vertices that we can reach along a "partial augmenting path" from s, and let $B = V \setminus A$ (so the absence of a complete augmenting path implies that $s \in A$ and $t \in B$)

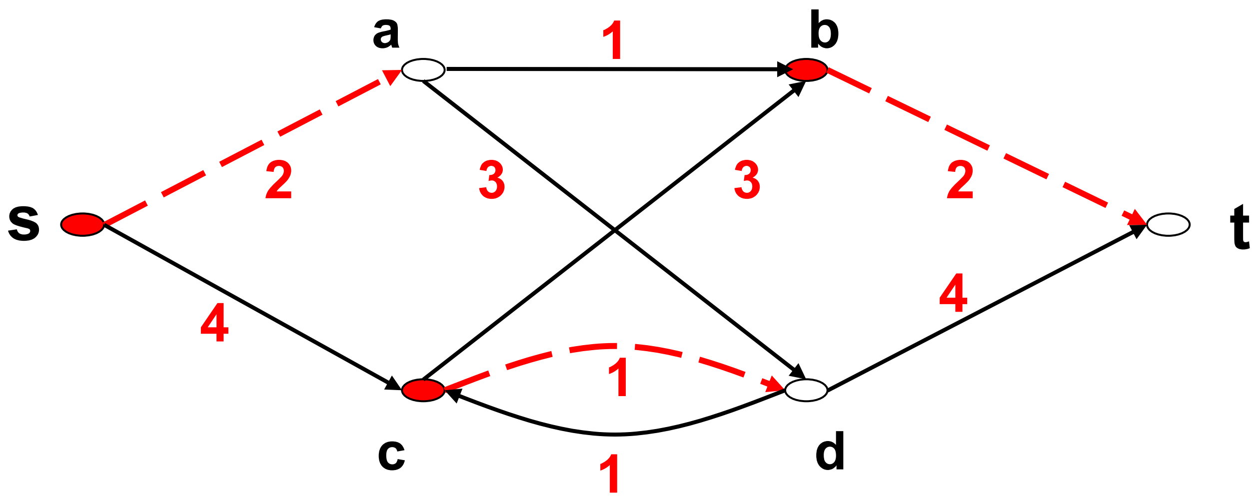
Relationship between flows and cuts (continued)

- Define $C = \{(u,v) \in E : u \in A \text{ and } v \in B\}$
- Then it must be the case that $f(u,v)=c(u,v)$ for each $(u,v) \in C$, and $f(v,u)=0$ for each $(v,u) \in E$ such that $v \in B$ and $u \in A$
  - otherwise we could extend some partial augmenting path to reach a vertex in B
- It follows that $val(f)=cap(C)$ (not proved here), and therefore that f is a maximum flow
- This is the essence of the so-called **Max Flow – Min Cut Theorem:**
- **Theorem:** The value of a maximum flow is equal to the capacity of a minimum cut

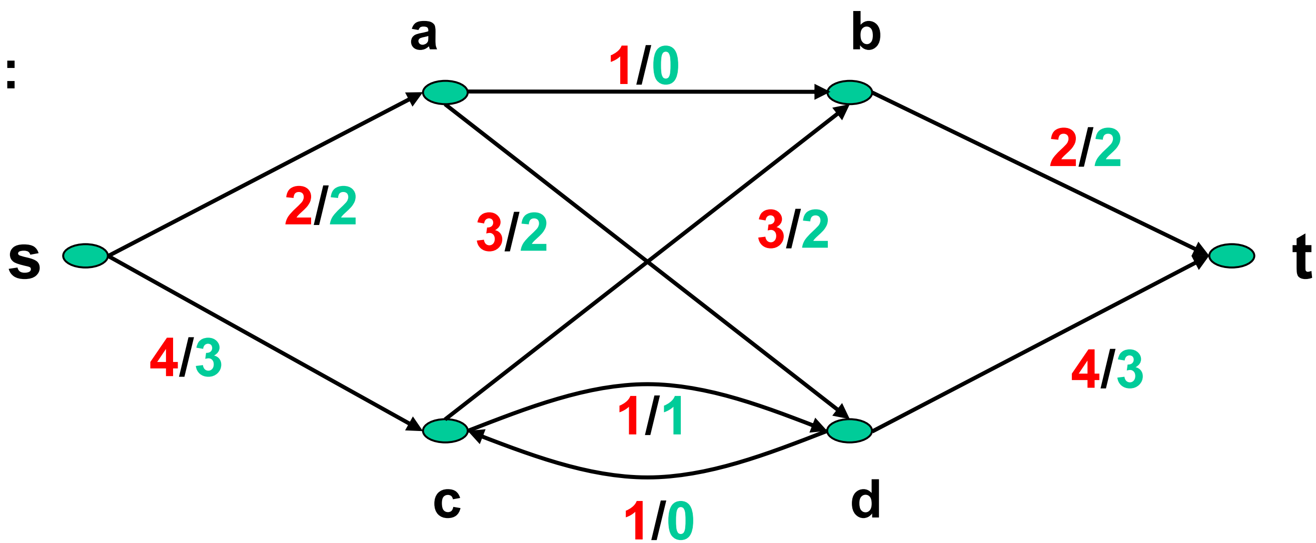
Example application of the Theorem

Example cut C:

cap(C)=5



Example flow g:



Here, val(g)=cap(C)

By the Max Flow – Min Cut Theorem, g is a maximum flow

Finding a maximum flow

- Start with a flow of zero on all edges
- Repeatedly search for an augmenting path
  - and augment the flow along this path
- Until no such path exists

Searching for augmenting paths – the residual graph

- Let $G=(V,E)$ be a network with capacity function c, and let f be a flow in G

- The *residual graph* $G'=(V',E')$ with respect to G and f is a directed graph with capacity function c' defined as follows:

The residual graph

- **$V'=V$**
  - **G' has the same vertex set as G**

- **$(u,v)\in E'$ if and only if:**

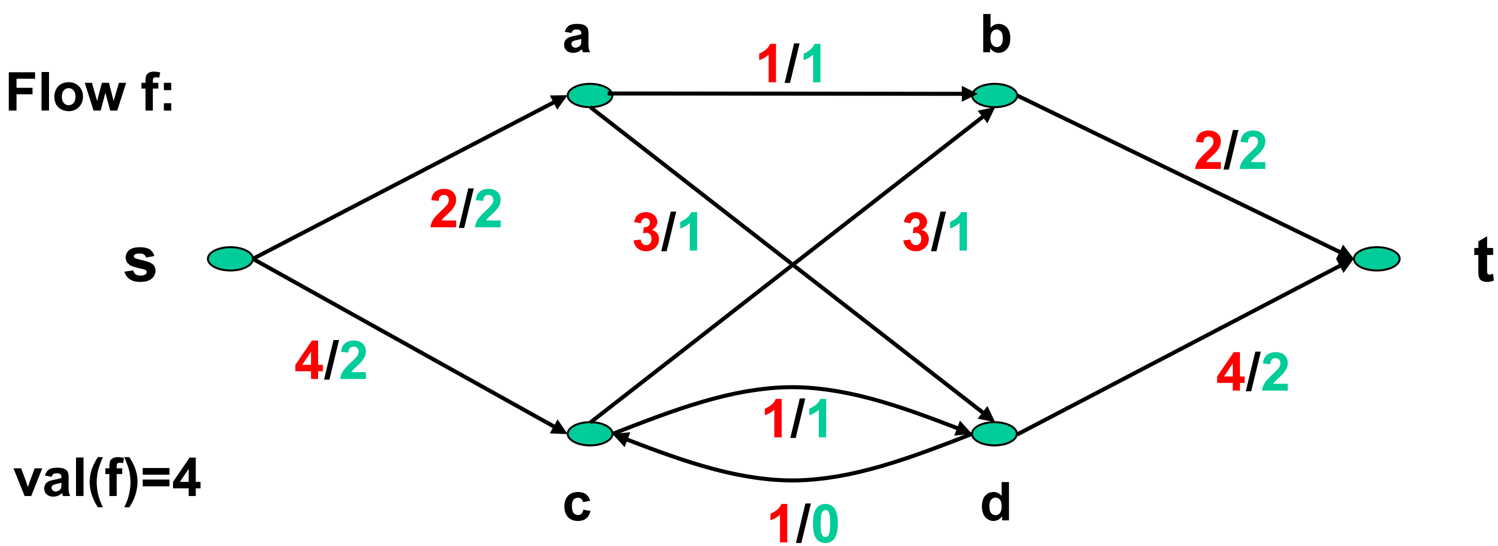
  - **$(u,v)\in E$ and $f(u,v)<c(u,v)$**
    - **so (u,v) can be a forward edge in an augmenting path**
    - **in this case define $c'(u,v) = c(u,v) - f(u,v)$**

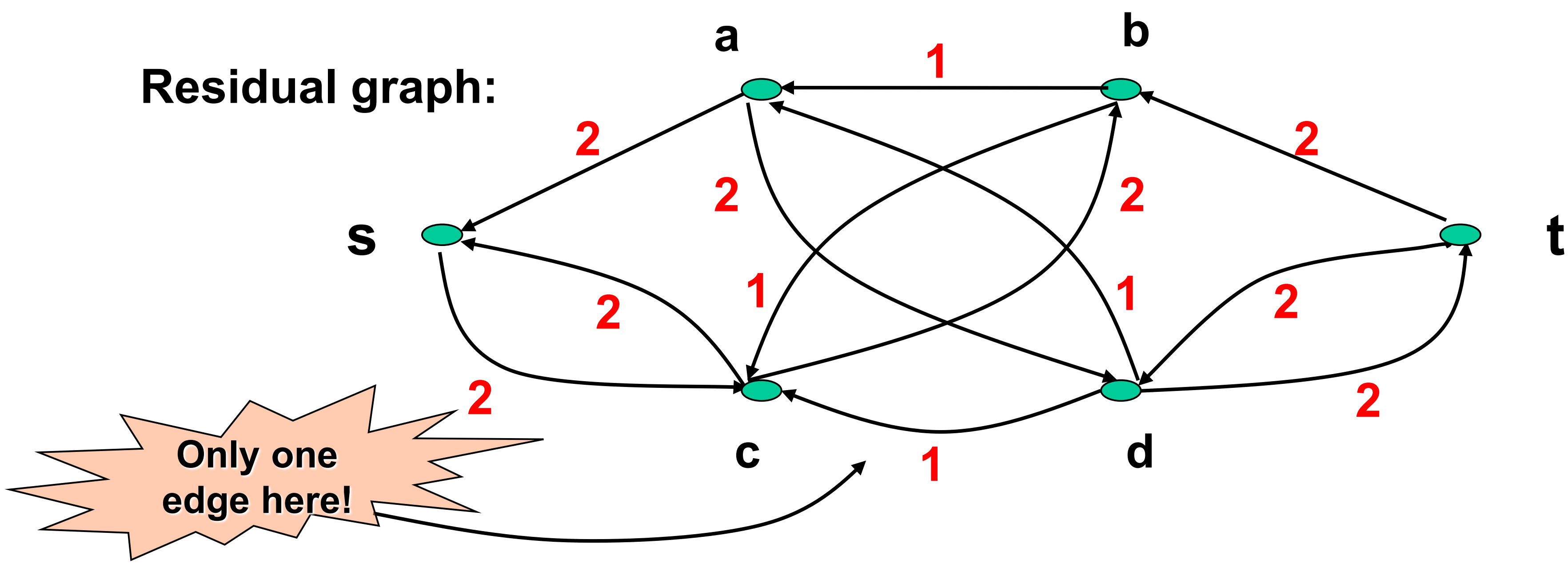
or

  - **$(v,u)\in E$ and $f(v,u)>0$**
    - **so (u,v) can be a backward edge in an augmenting path**
    - **in this case define $c'(u,v) = f(v,u)$**

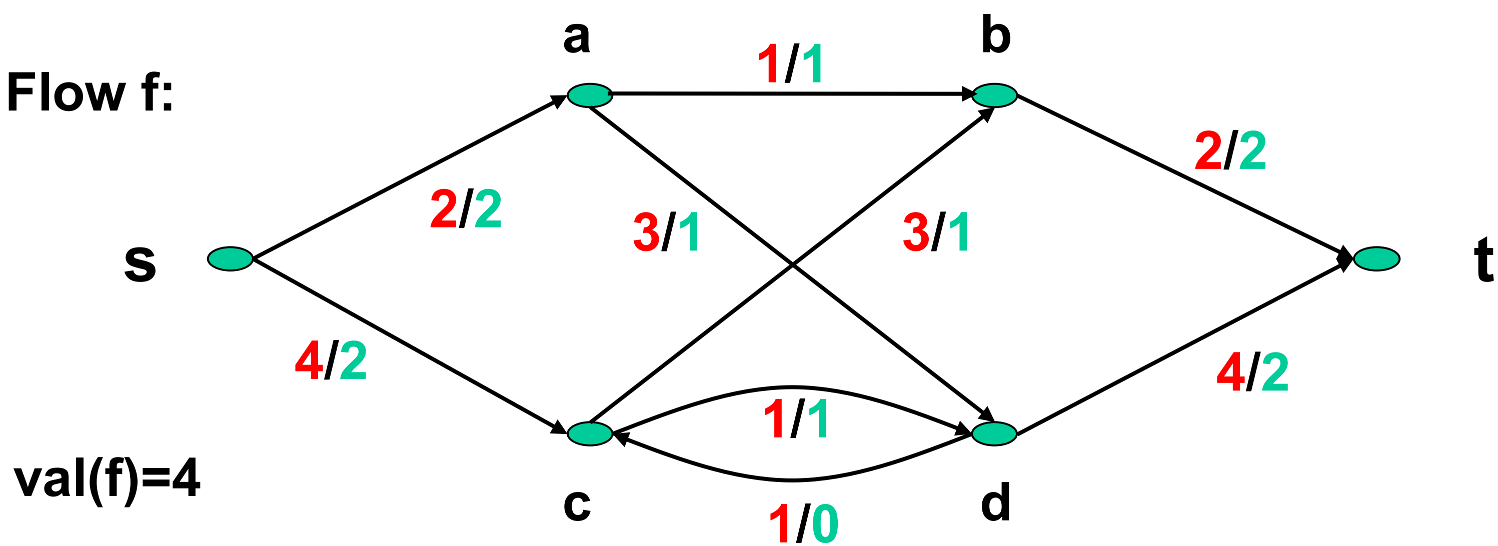
A directed path from s to t in the residual graph G' corresponds to an augmenting path with respect to f in G

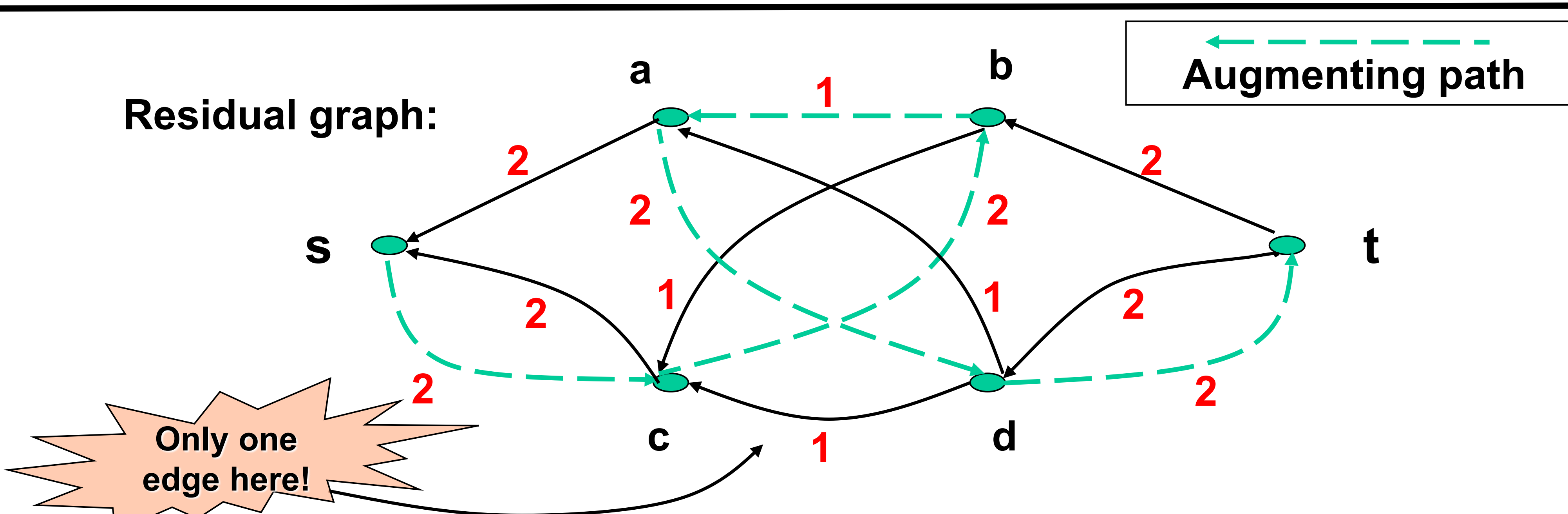
Example residual graph





Example residual graph





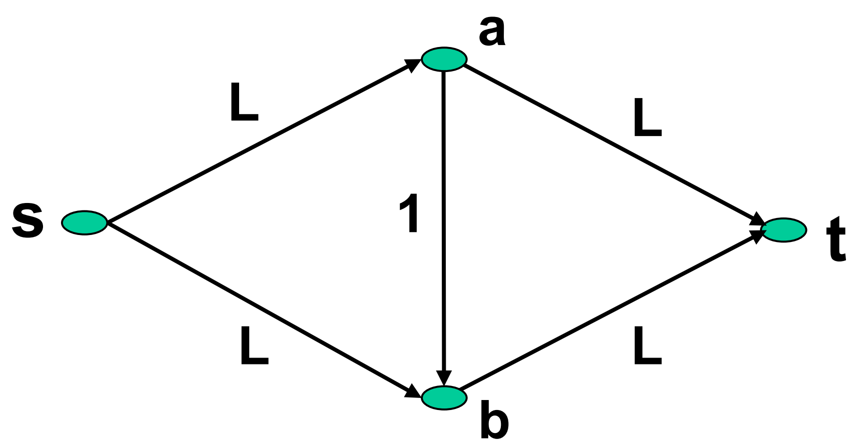
Ford-Fulkerson Algorithm

```
/** Calculates maximum flow f in network
    G=(V,E) with capacity function c */
for ((u,v) : E)
   f(u,v)=0;
while (true)
{  build residual graph G'=(V',E'), capacity function c';
   search for path P in G' from s to t; // (†)
   if (such a path P found)      // augment f
   {  m = Math.min{c'(u,v) : (u,v) ∈ P}; // residual capacity
      for ((u,v) : P)
         if ((u,v)∈E && f(u,v) + m ≤ c(u,v))
            f(u,v) += m; // (u,v) is a forward edge
         else
            f(v,u) -= m; // (u,v) is a backward edge
   }
   else
      break;   // f is a maximum flow
}
```

Complexity of the Ford-Fulkerson Algorithm

- Initialisation is $O(|E|)$
- During a loop iteration:
  - Build residual graph – $O(|V|+|E|)$
  - Search for a directed path from s to t
    - $O(|V|+|E|)$ using breadth-first or depth-first search
  - Augment along a path (if found) – $O(|E|)$
  - Every vertex is on a directed path from s to t, therefore $|V|=O(|E|)$
- Number of loop iterations is $\leq$ value of max flow
  - In the worst case $m=1$ during every loop iteration, so that flow only increases by 1 every time round the loop
- Overall complexity is $O(|E|.\text{max flow})$

A worst-case example



Max flow = 2L

Algorithm might choose augmenting paths:

(s,a) (a,b) (b,t), then
(s,b) (b,a) (a,t), then
(s,a) (a,b) (b,t), then
(s,b) (b,a) (a,t)...

Total 2L iterations

On the other hand, algorithm might choose

(s,a) (a,t), then
(s,b) (b,t)

Total 2 iterations

L can be arbitrarily large

Improving the worst-case

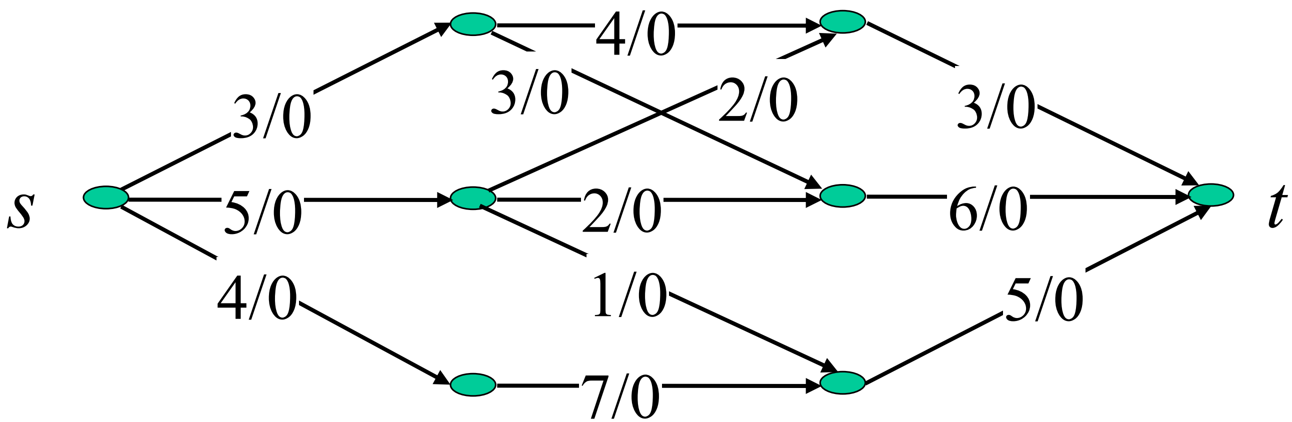
- At point (†) in the Ford-Fulkerson Algorithm, use breadth-first search to find the shortest augmenting path (i.e. with the smallest number of edges)

- Edmonds and Karp (1972): if we follow this practice, number of times algorithm searches for an augmenting path is $O(|V||E|)$

- Therefore Ford-Fulkerson algorithm can be implemented to run in $O(|V||E|^2)=O(|E|^3)$ time

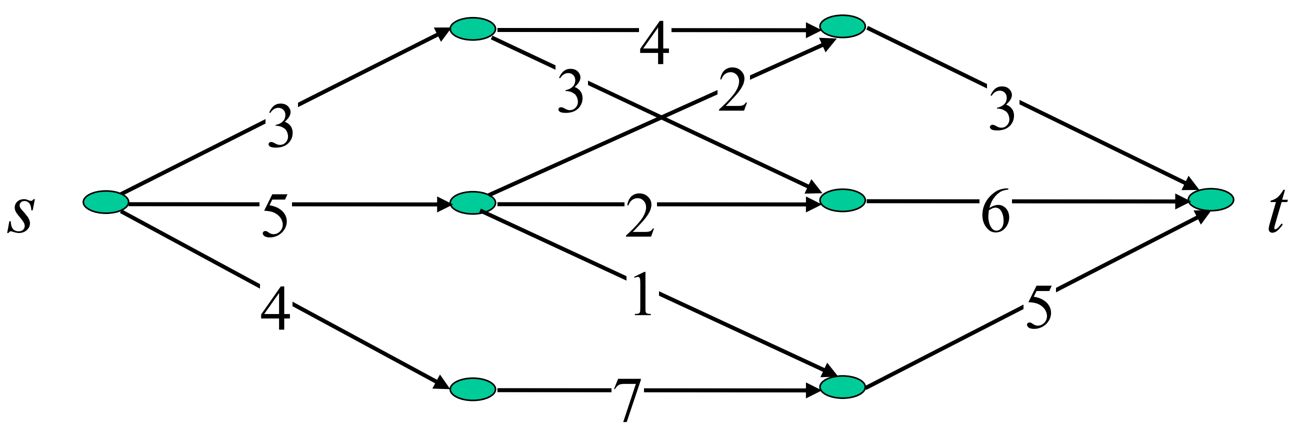
- Fastest algorithm to date: Orlin (2013): $O(|V||E|)$

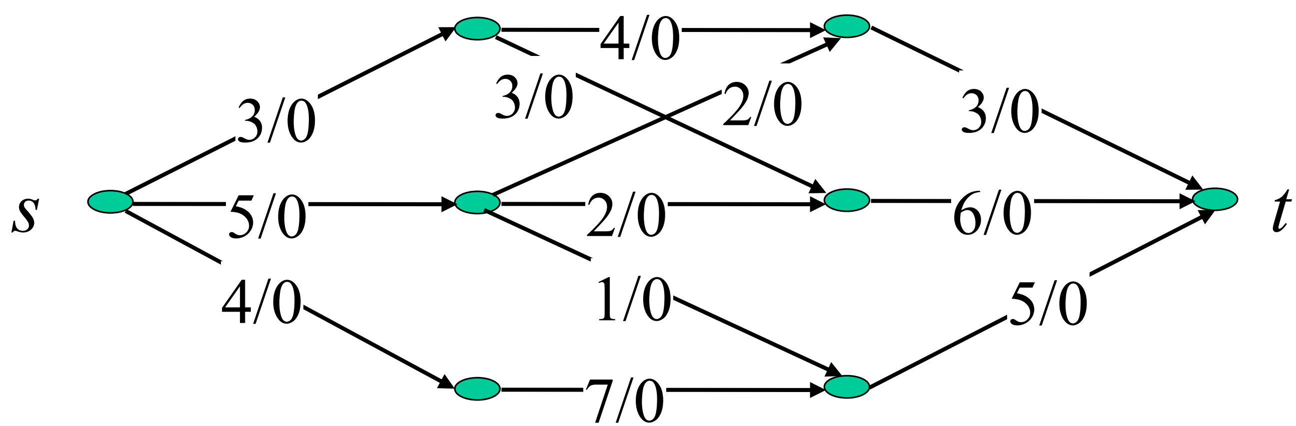
Example



Residual Graph

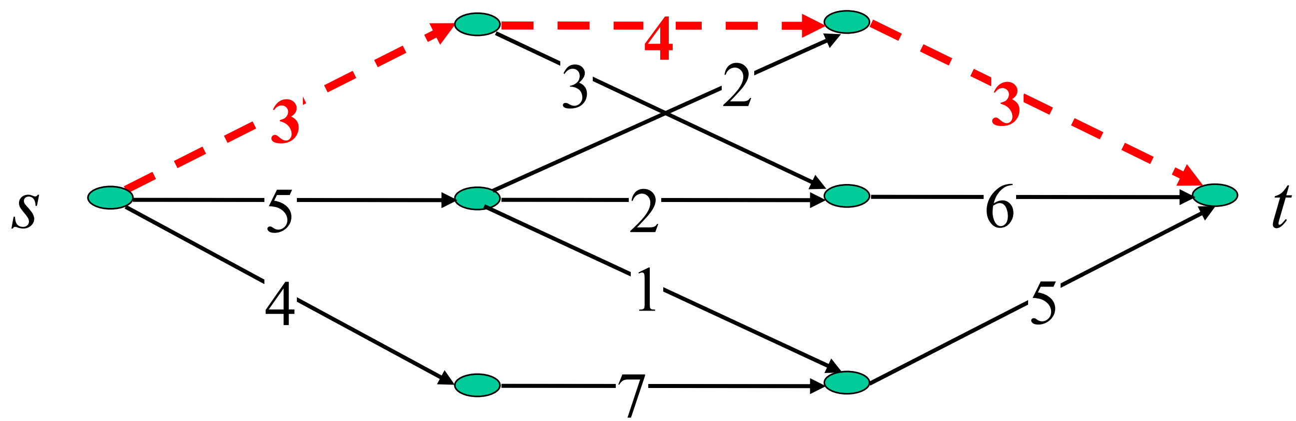
Iteration 1 of main loop



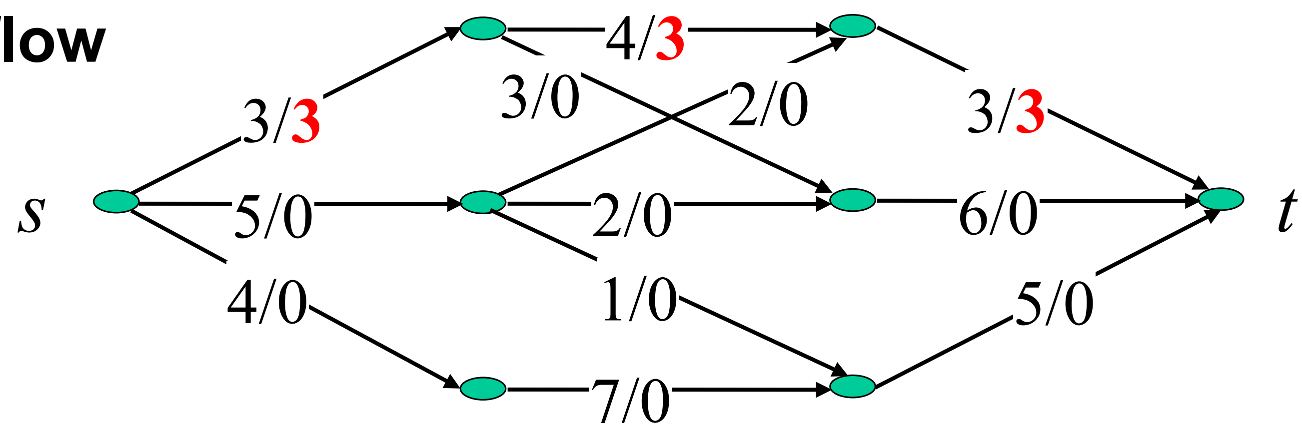


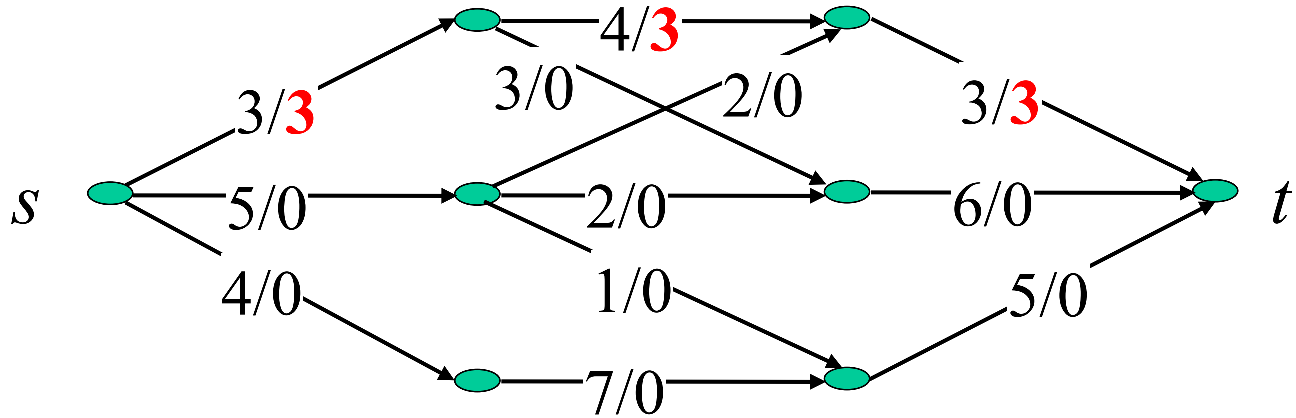
Residual Graph

Iteration 1 of main loop



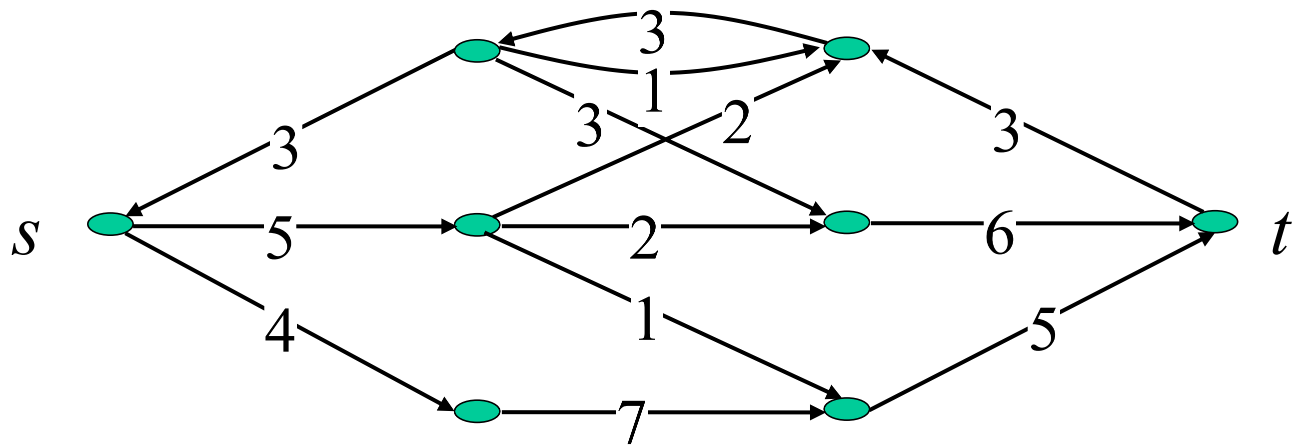
Updated flow

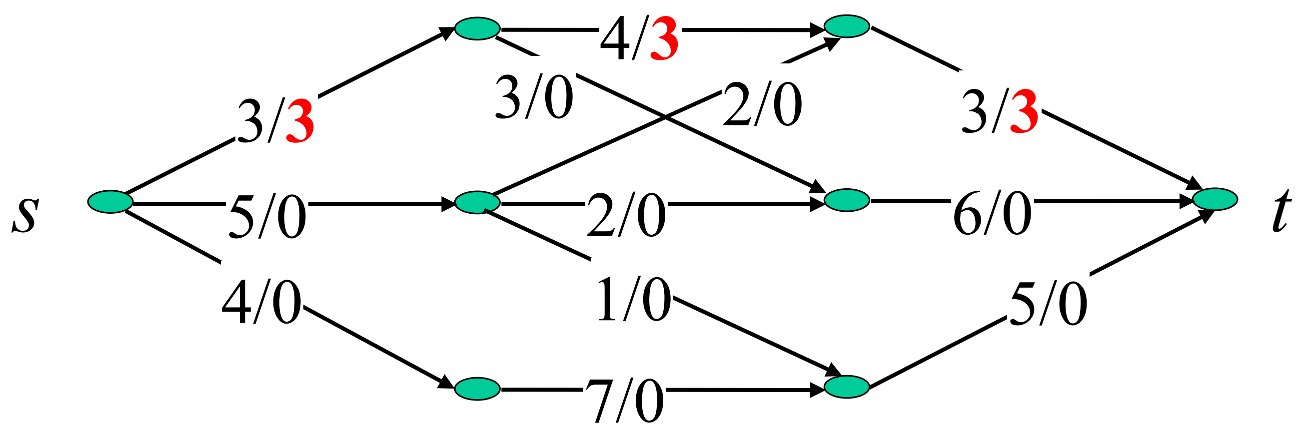




Residual Graph

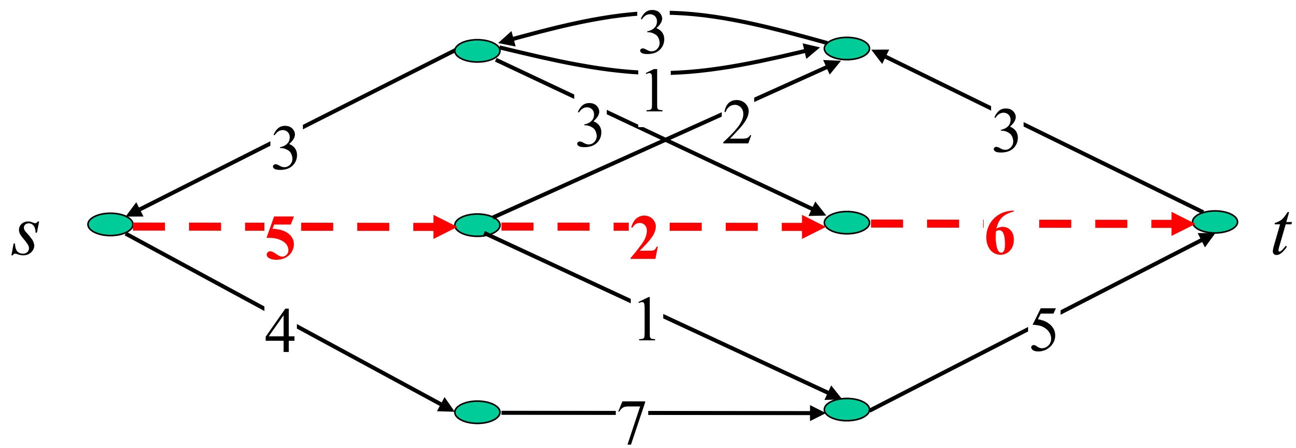
Iteration 2 of main loop



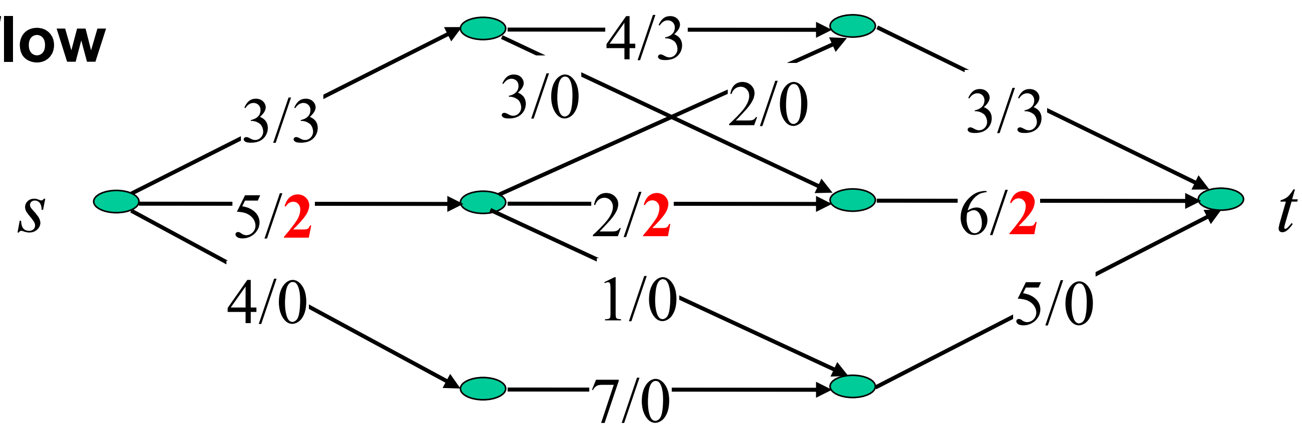


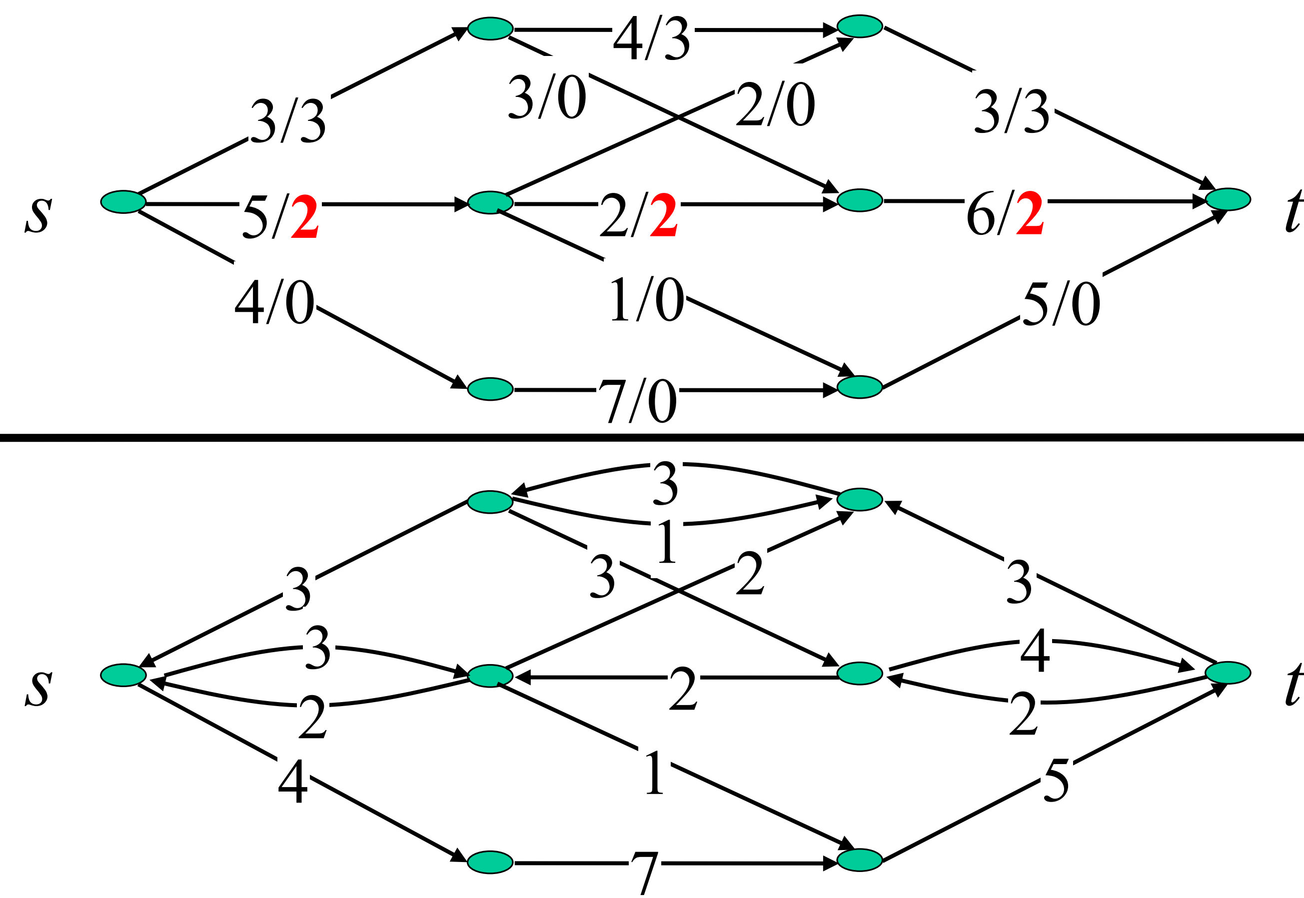
Residual Graph

Iteration 2 of main loop



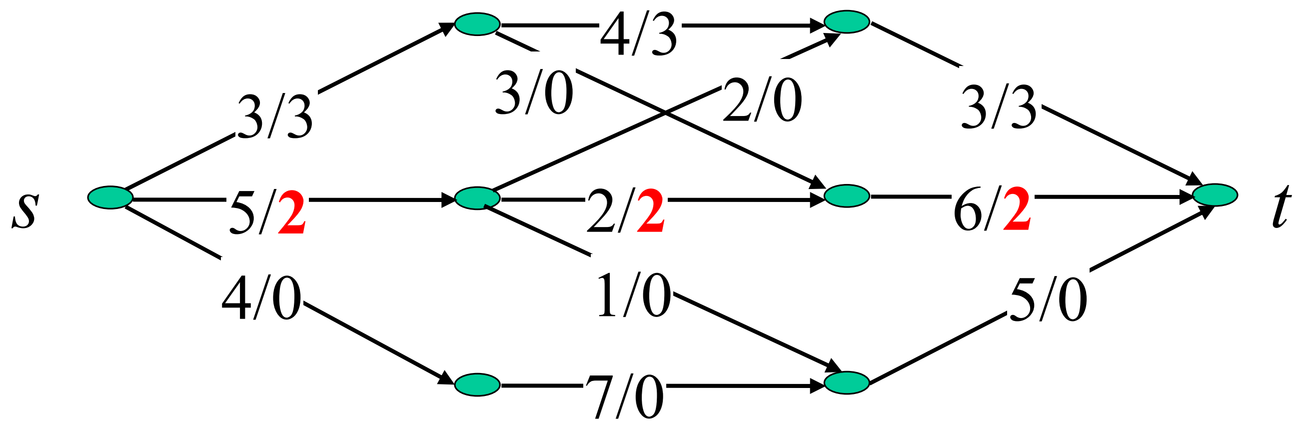
Updated flow





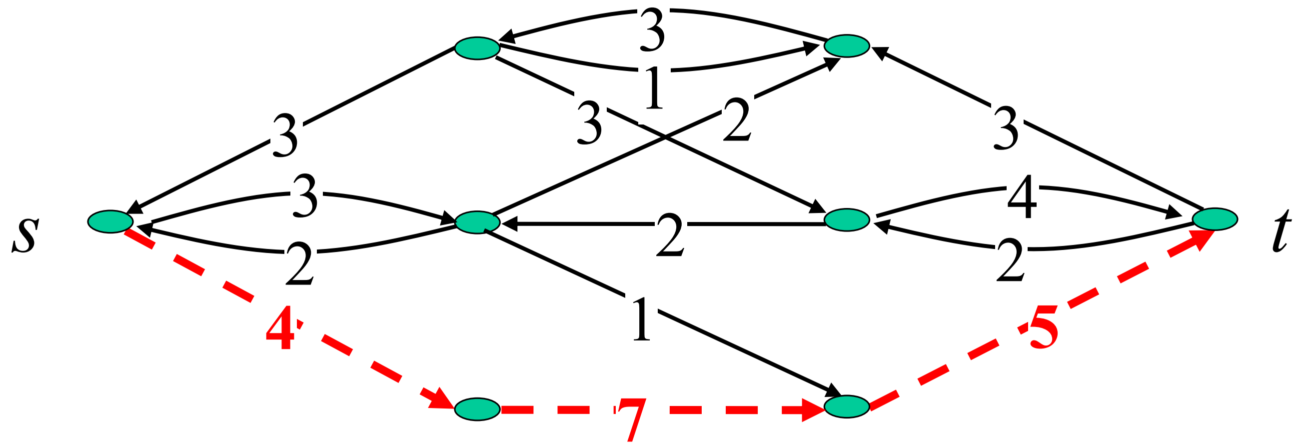
Residual Graph

Iteration 3 of main loop

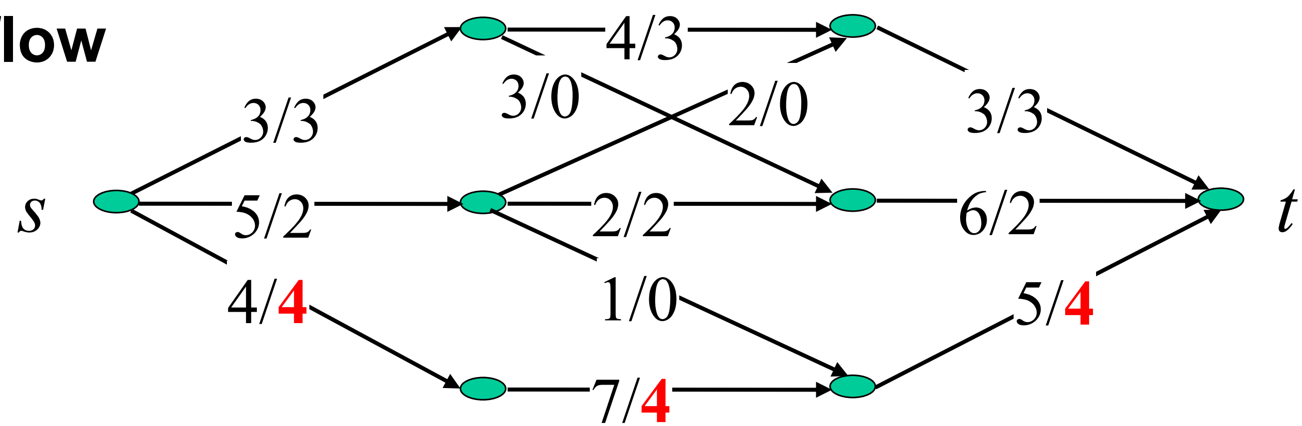


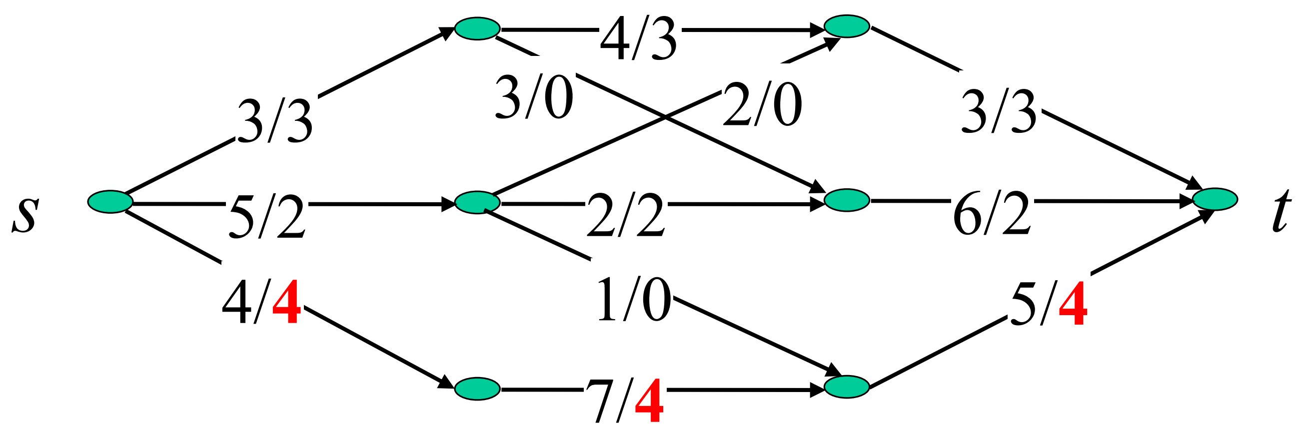
Residual Graph

Iteration 3 of main loop



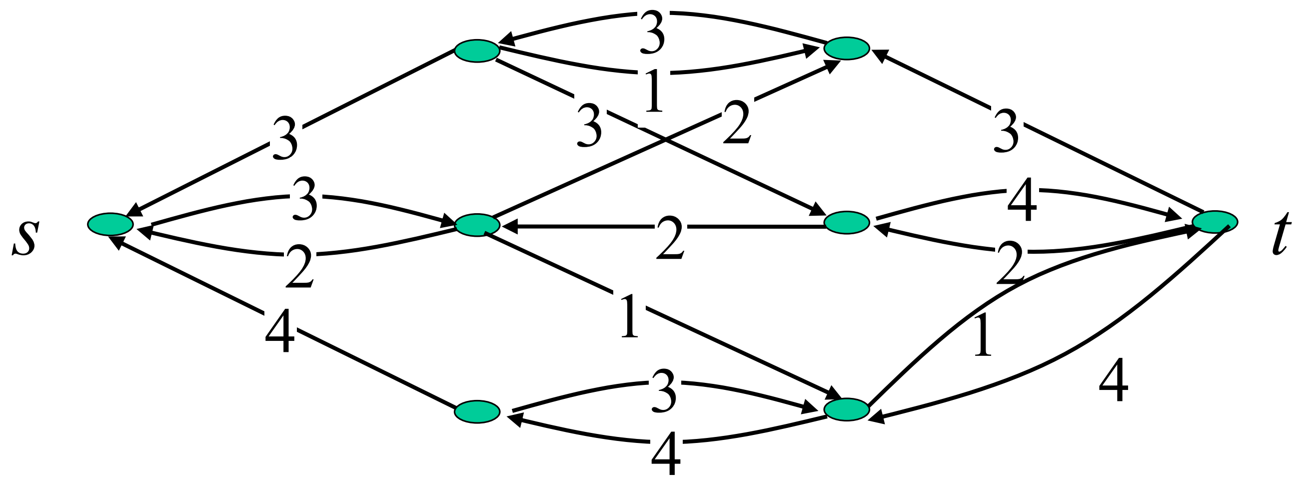
Updated flow

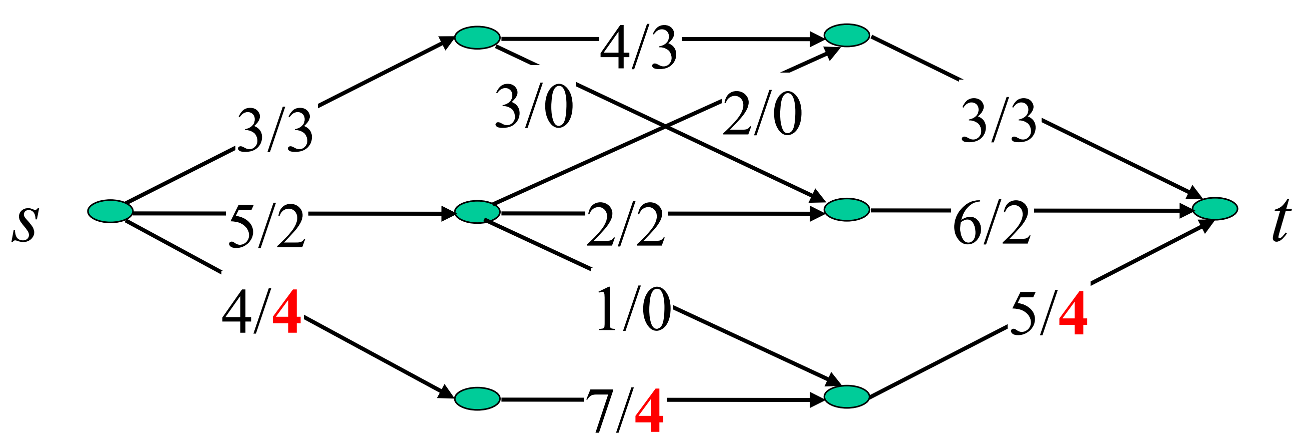




Residual Graph

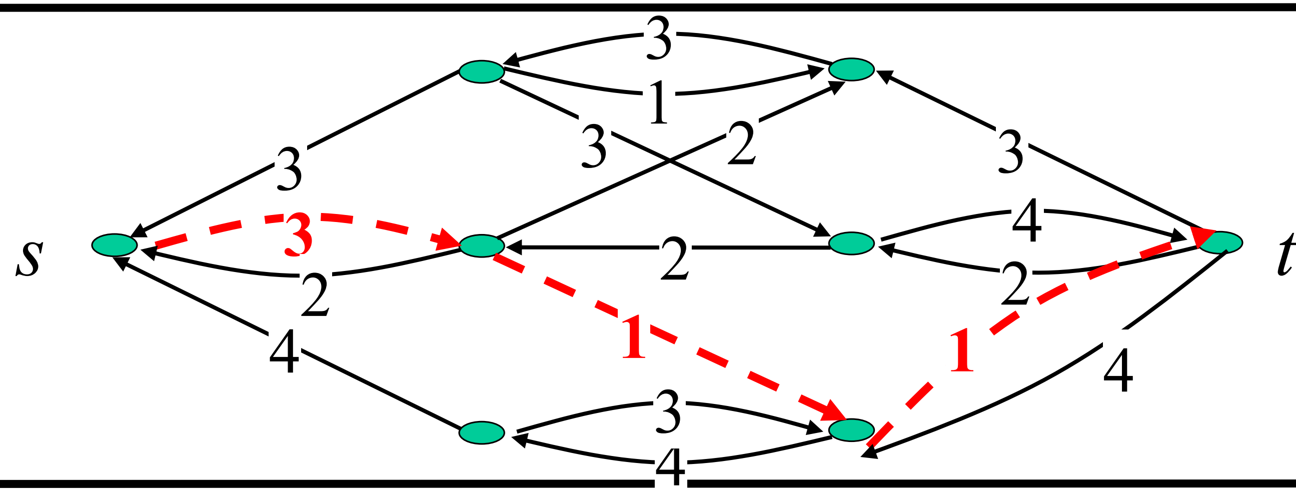
Iteration 4 of main loop



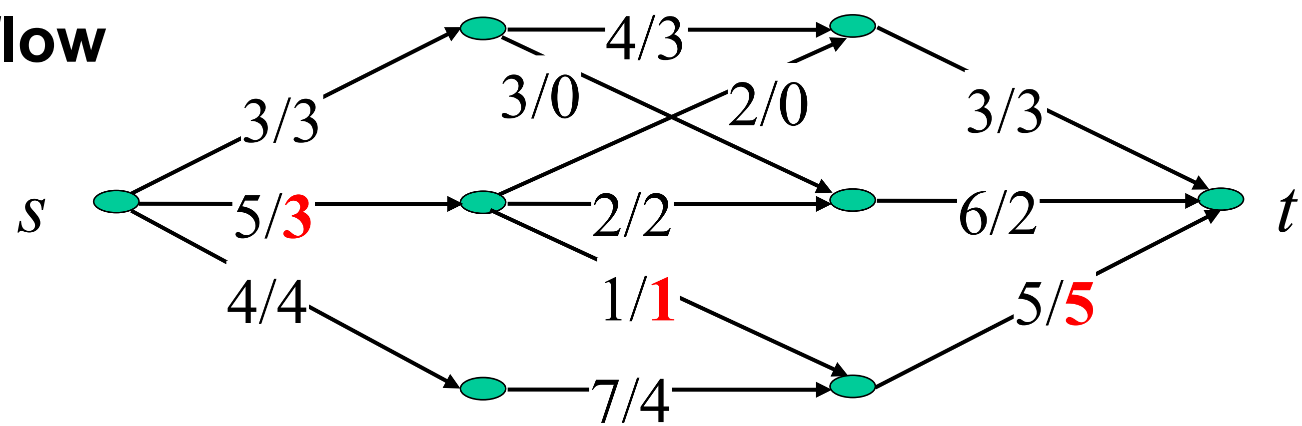


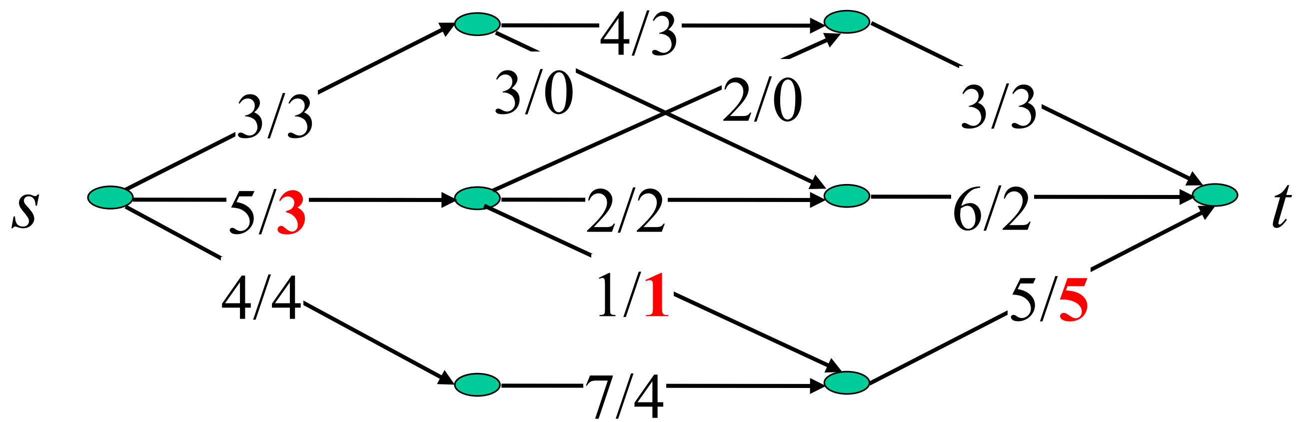
Residual Graph

Iteration 4 of main loop

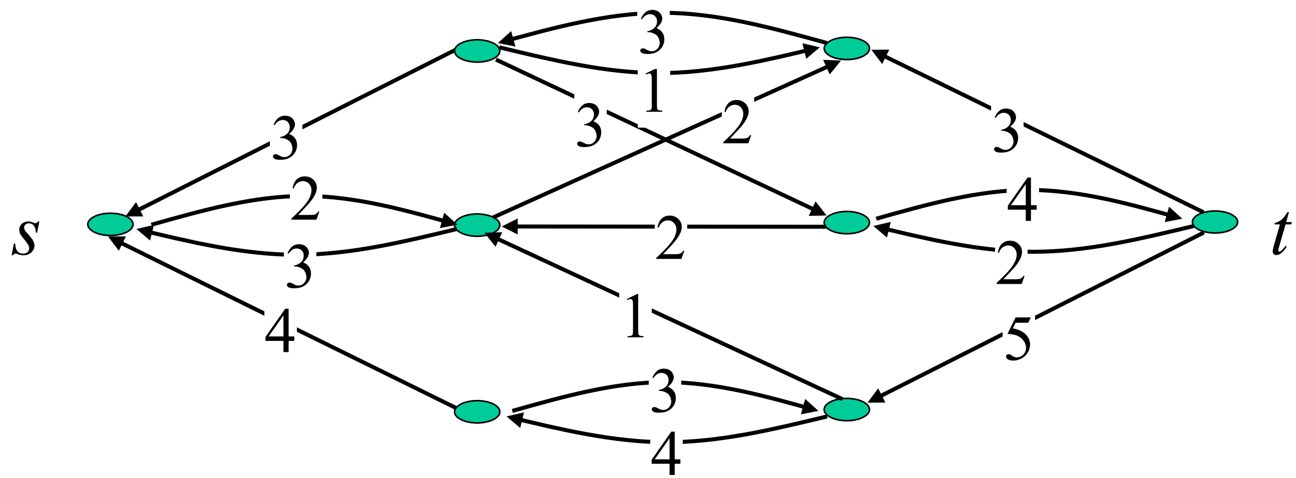


Updated flow

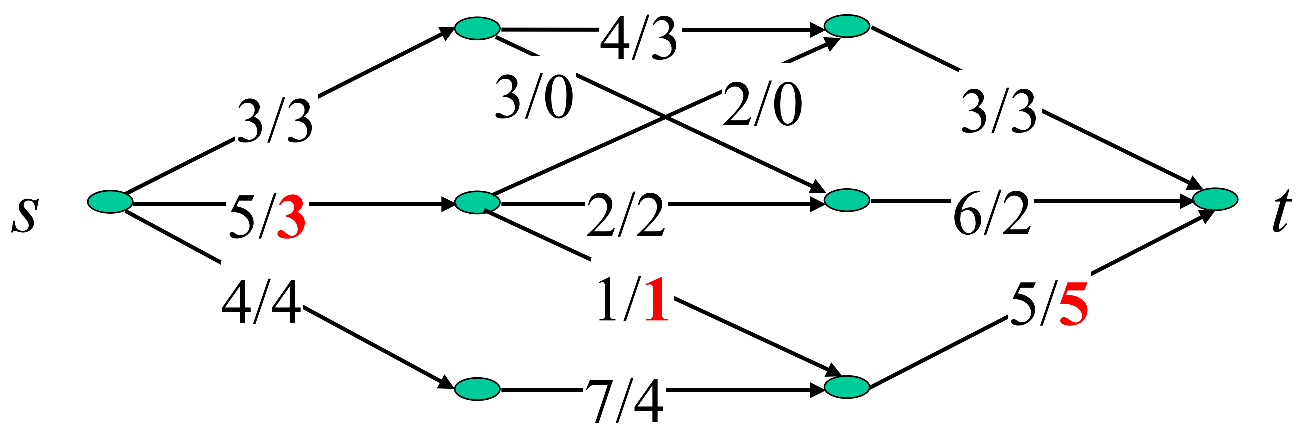




Residual Graph

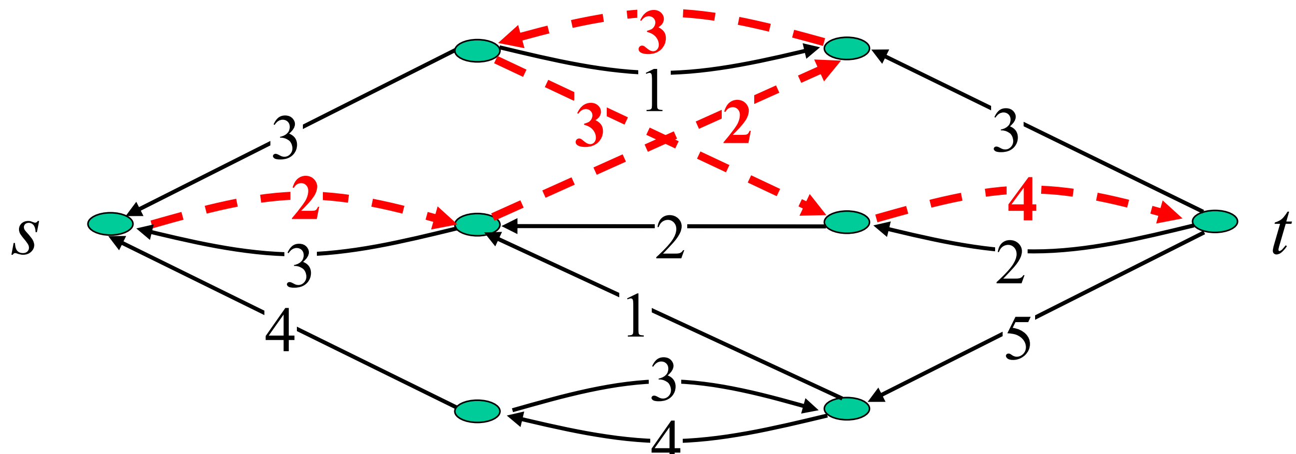


Iteration 5 of main loop



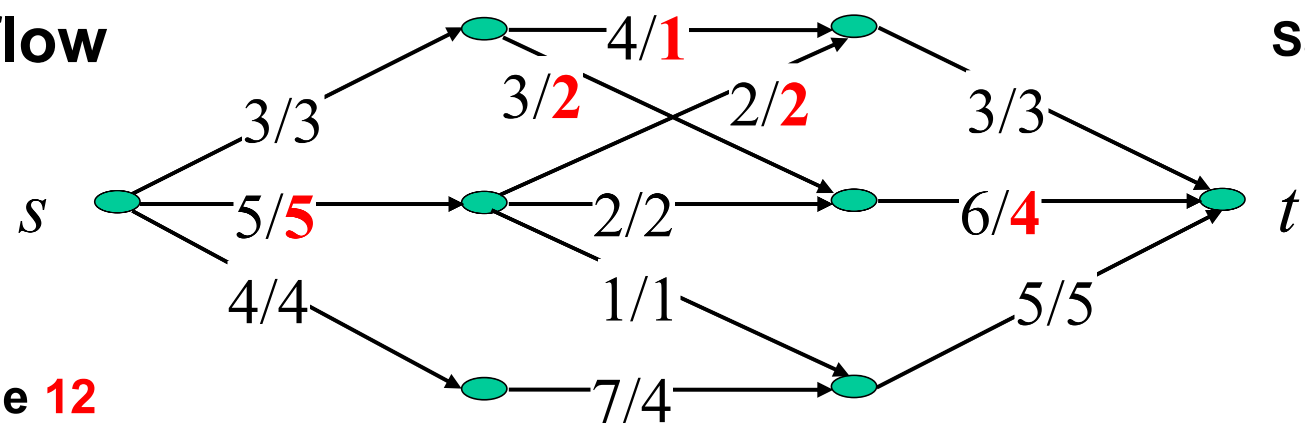
Residual Graph

Iteration 5 of main loop



Updated flow

Saturating flow



Flow has value 12

A faster algorithm for maximum matching

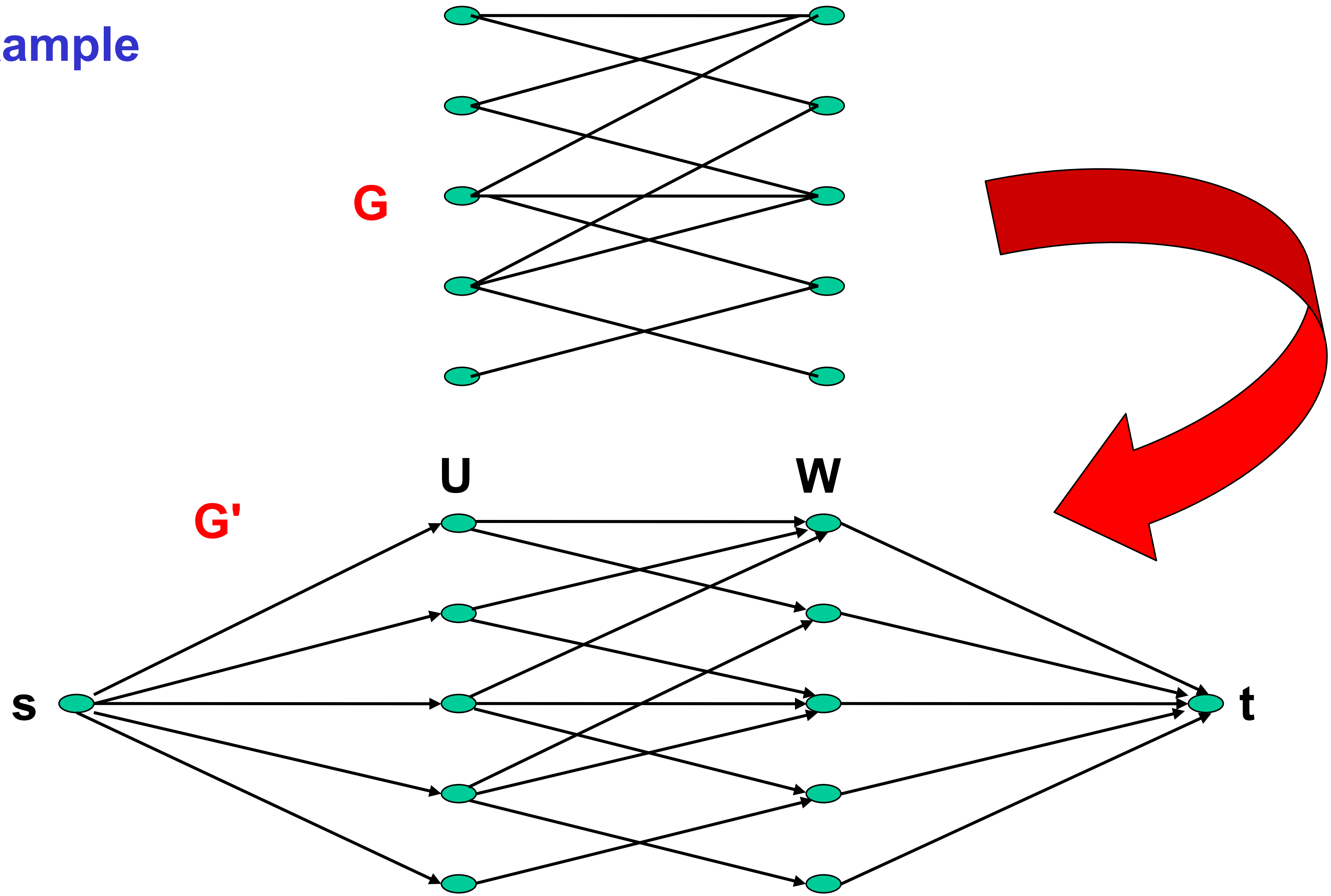
Maximum matching in bipartite graphs
Input: Bipartite graph G
Output: Maximum matching M in G

Reduce this problem to a network flow problem:

- Let $G=(V,E)$ be a bipartite graph, where G has bipartition $V=U \cup W$
- Form a directed graph $G'=(V',E')$ as follows:
- $V'=V\cup\{s,t\}$ for two new vertices s, t
- $E'=\{(s,u) : u\in U\} \cup \{(u,w) : u\in U \wedge w\in W \wedge \{u,w\}\in E\} \cup \{(w,t) : w\in W\}$
- All edges in G' have capacity 1

Claim: The cardinality of a maximum matching in G is equal to the value of a maximum flow in G'
Proof: Exercise

Example



When the network has this special form, maximum flow can be found in O(√ |V|(|V|+|E|)) time – previous matching algorithm was O(|V|(|V|+|E|))