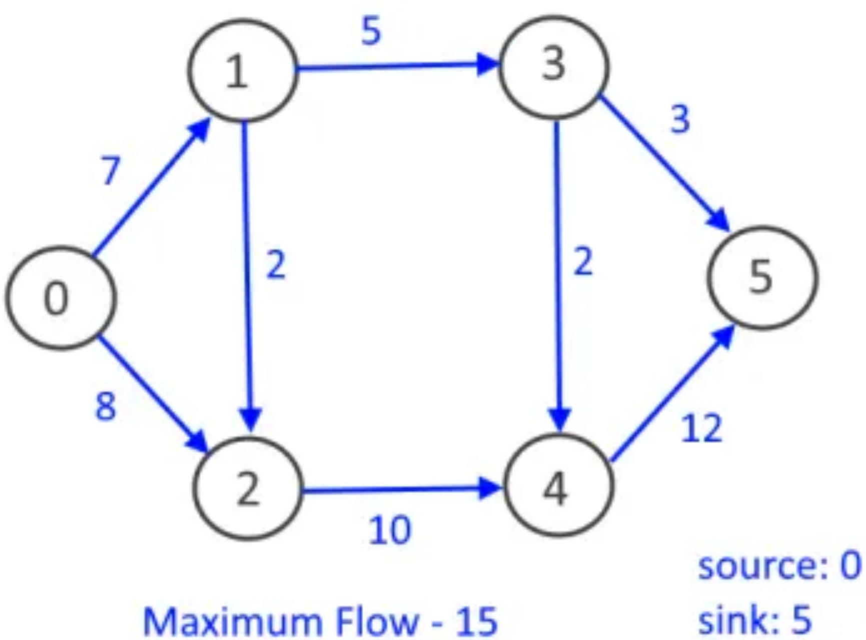
Lecture 9: Graph Algorithms (III)

Dr. Tsung-Wei Huang
Department of Electrical and Computer Engineering
University of Utah, Salt Lake City, UT

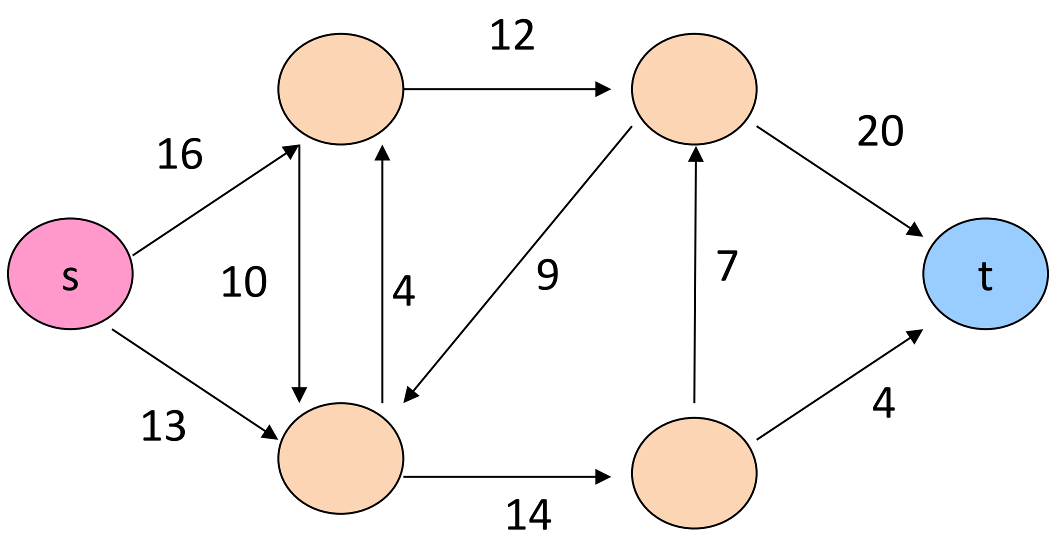
Maximum Flow

- **Find a maximum feasible s-t flow in a graph**
  - *s* is a source node and *t* is a target (sink) node
  - Each edge is associated with a capacity
  - Flow at each edge cannot exceed its capacity



Problem Formulation

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

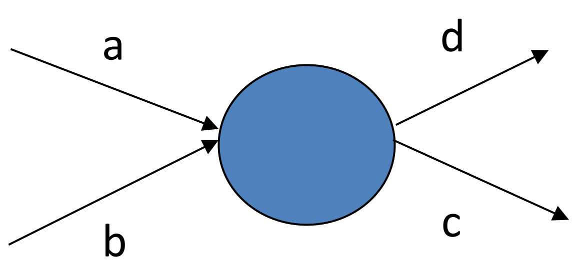


Flow Constraint

- **G=(V,E): a flow network with capacity function c.**
- ***s* -- the source and *t* -- the sink.**
- **A flow f(u, v) in G must satisfy**

1. Capacity constraint
   - For all u,v $\in$V, we require f(u, v) $\leq$ c(u, v).
2. Flow conservation
   - For all u $\in$V-{s, t}, we require

$$\sum_{e.in.v} f(e) = \sum_{e.out.v} f(e)$$



a+b = d+c

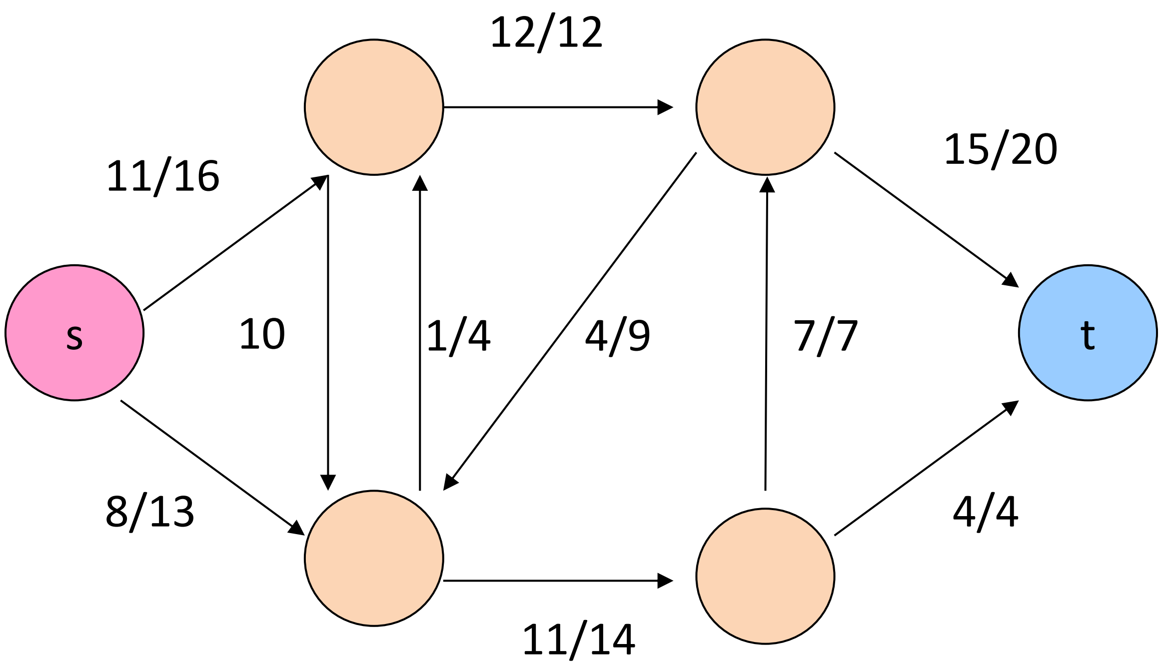
Objective

- **The quantity f (u, v) is called the net flow from vertex u to vertex v.**
- **The value of a flow is defined as**

$$|f| = \sum_{v \in V} f(s, v)$$

  - The total flow from source to any other vertices.
  - The same as the total flow from any vertices to **the sink.**

Example



A flow f in G with value $|f| = 19$

So …

- **Given a flow network G with source s and sink t**
- **Find a flow of maximum flow value from s to t.**
- **How to solve it efficiently ?**
  - Brute force …?

Ford-Fulkerson Framework

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

initialize flow *f* to *0*

while there exists an *augmenting* path *p*

do *augment* flow *f* along *p*

return *f*

Why Framework not Algorithm?

- **The framework is iterative**
  - Try to find a flow if it exists
  - Iterates the procedure until no more exists
- **Augmenting flow has different implementations**
  - Each implementation is a different algorithm
    - Edmonds-Karp, Dinic's blocking algorithm, Push-relabel, etc.
- **Augmenting flow is equivalent to finding a path**
  - u→v is connected if there remains capacity (non-zero)
  - u→v is disconnected if the capacity is zero

What is the Time Complexity?

- **Assume inner loop applies DFS**
  - Each DFS iteration contributes O(V+E)
  - Need “max-flow” iterations

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

initialize flow *f* to *0*

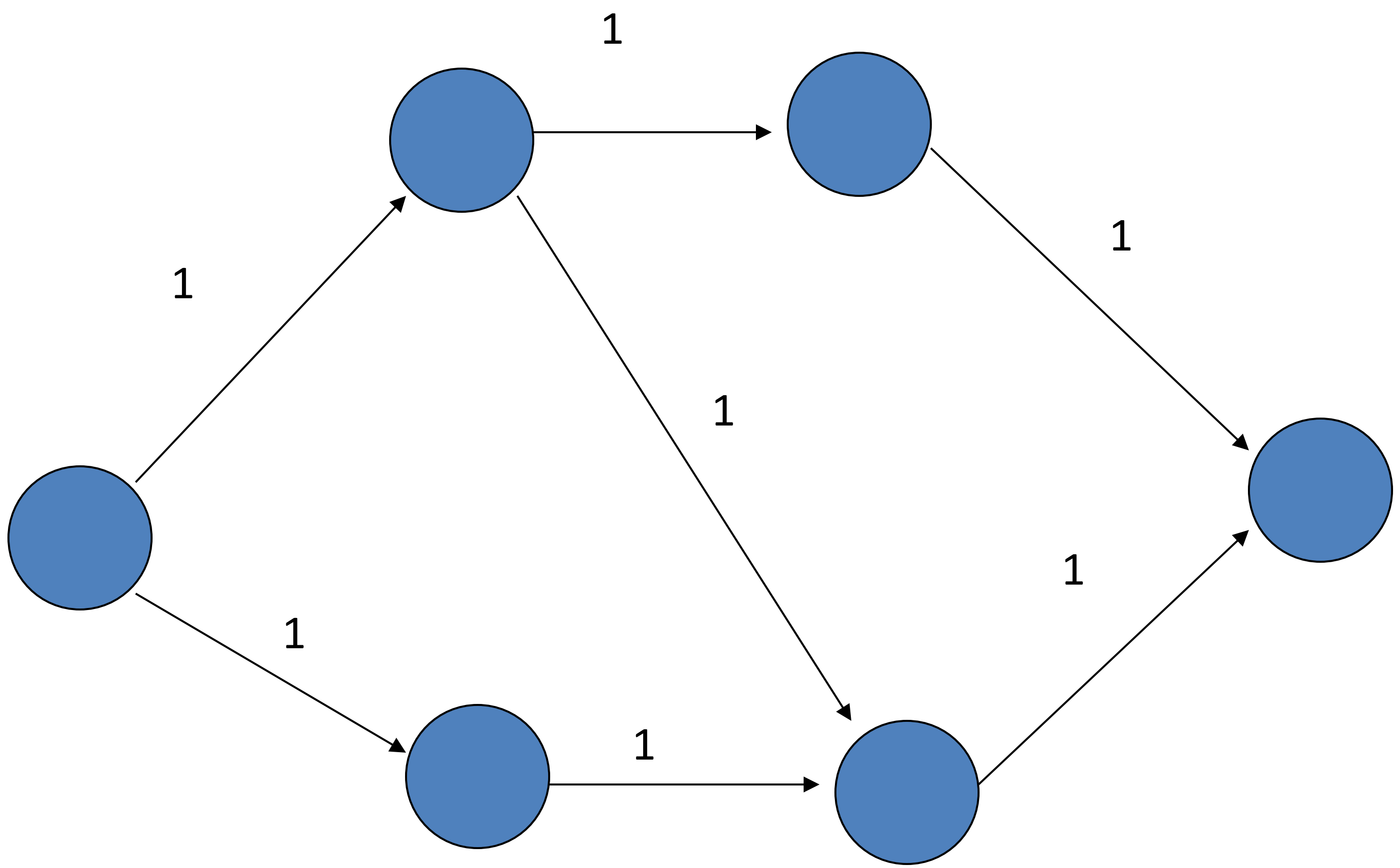
while there exists an *augmenting* path *p*

do *augment* flow *f* along *p*

return *f*

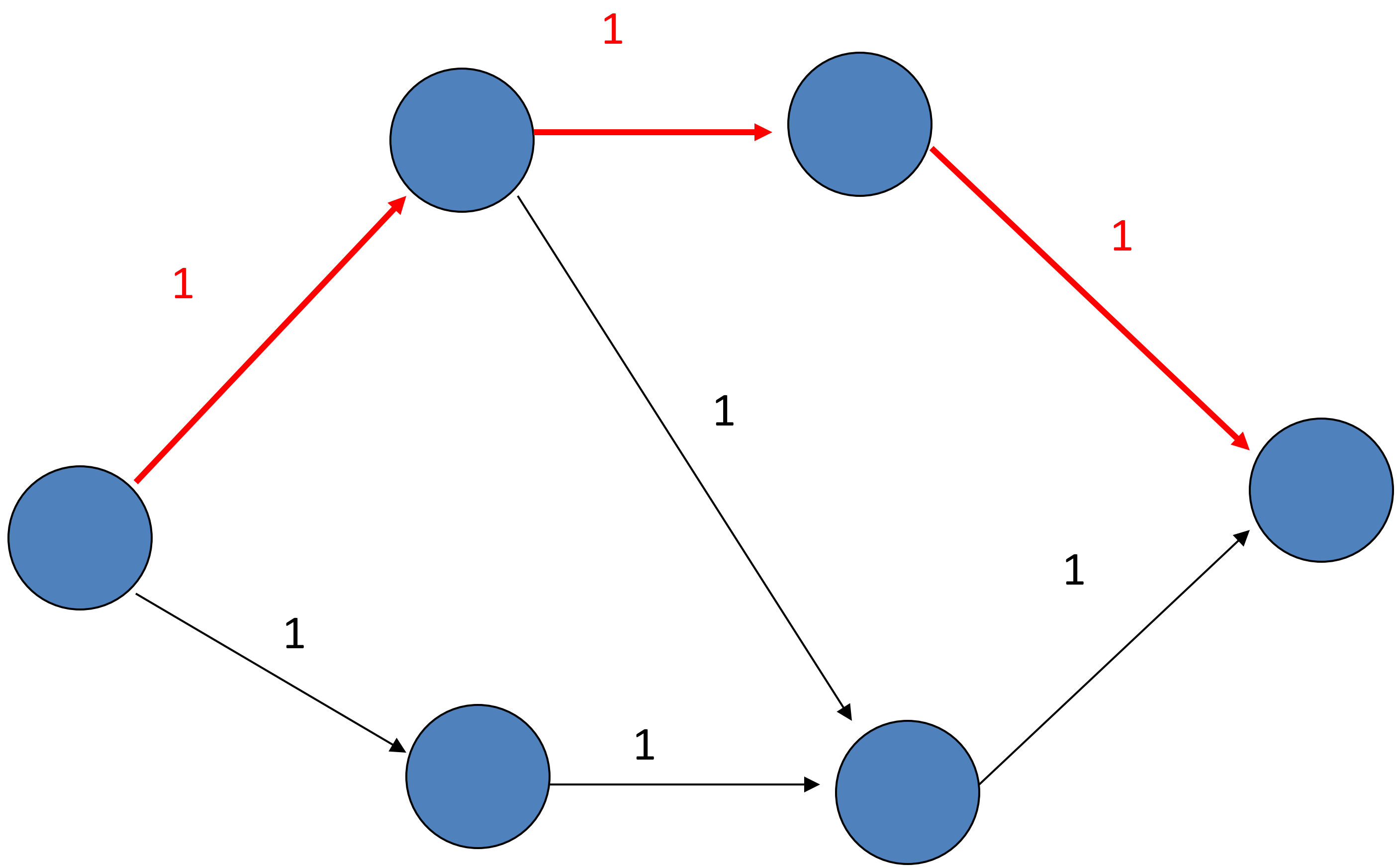
Example

- What is the maximum flow?



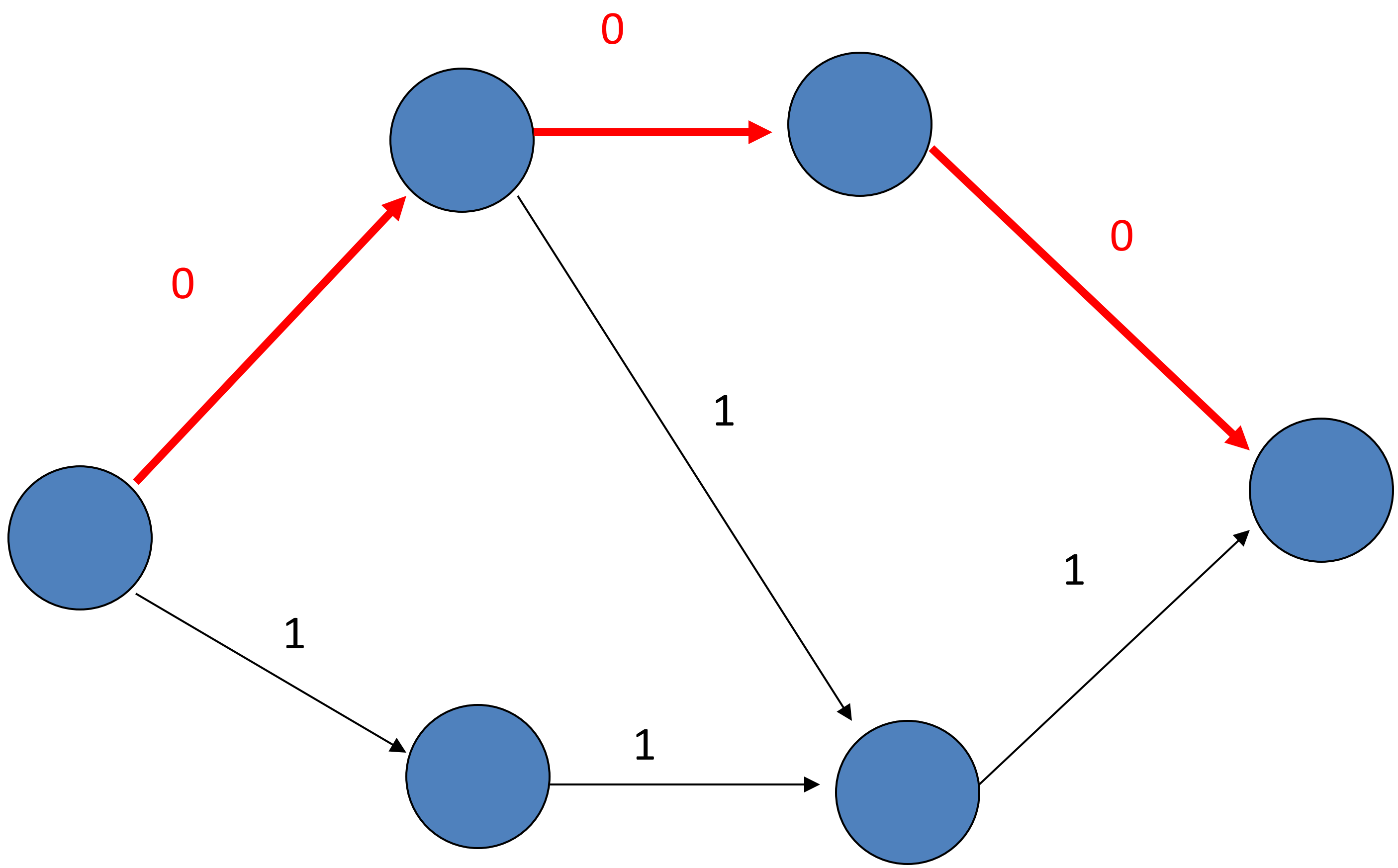
Example

- Let's do DFS to augment flow: iter1 finds flow 1



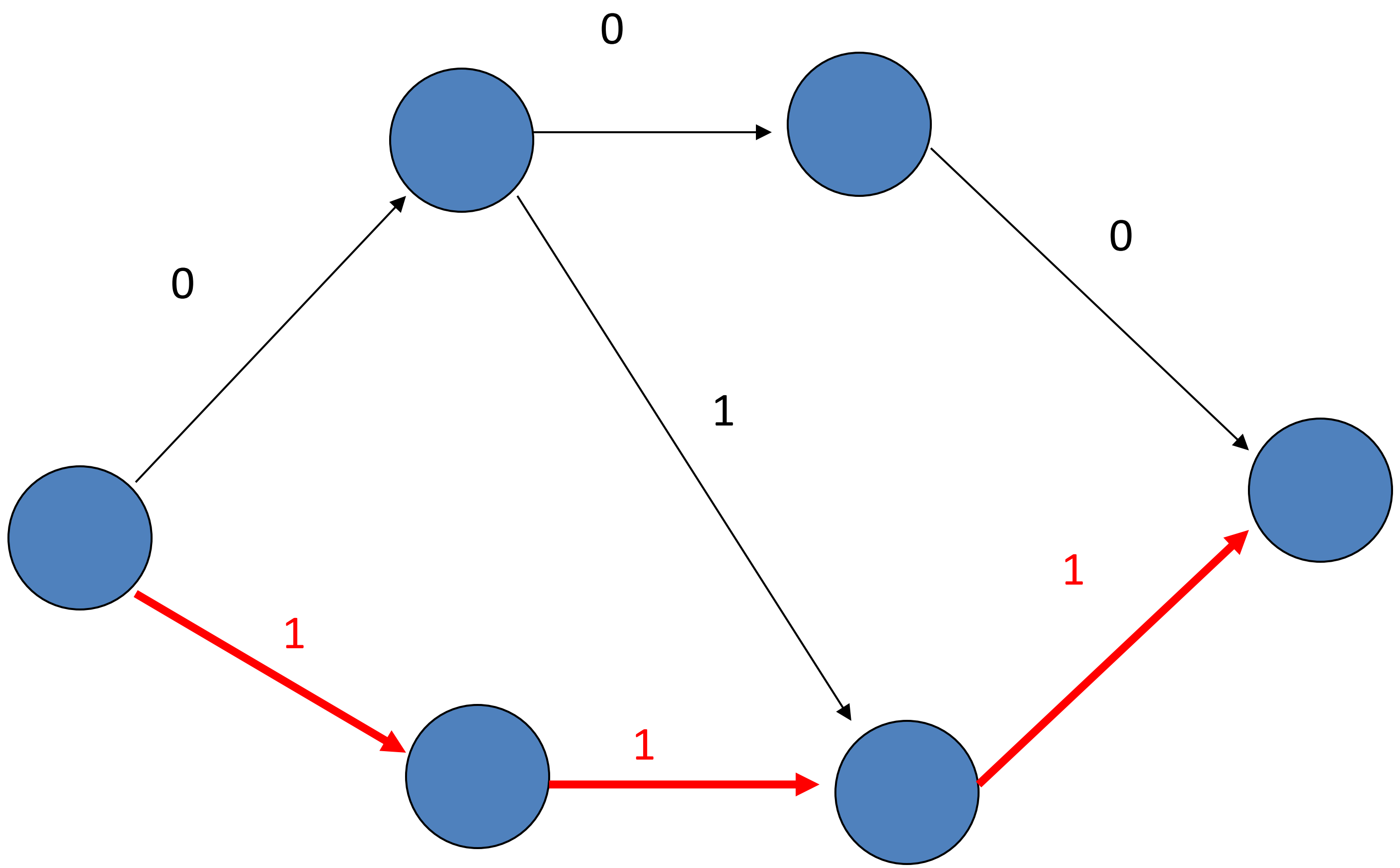
Example

- ❑ **Update remaining capacity**



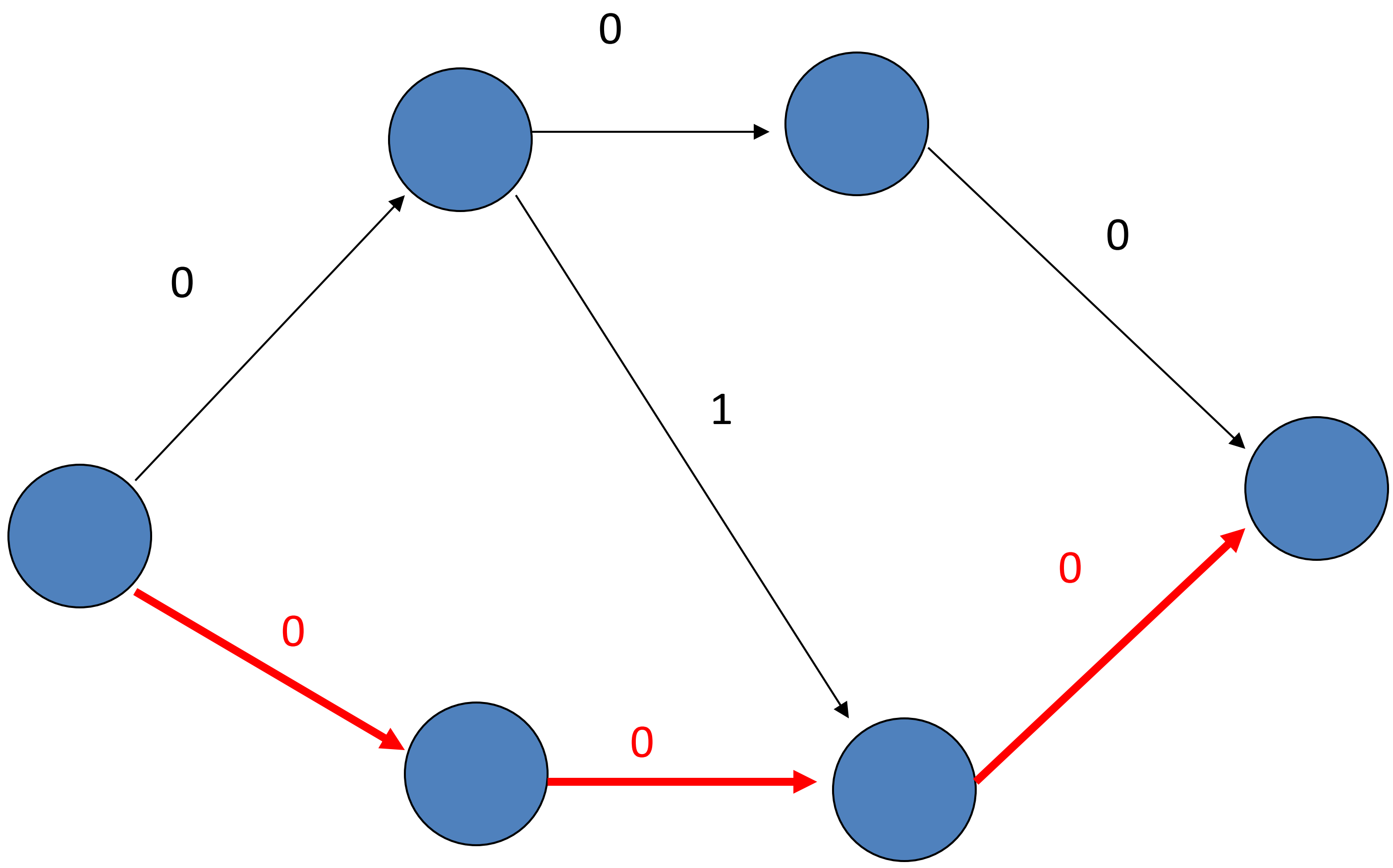
Example

- Let's do DFS to augment flow: iter2 finds flow 1



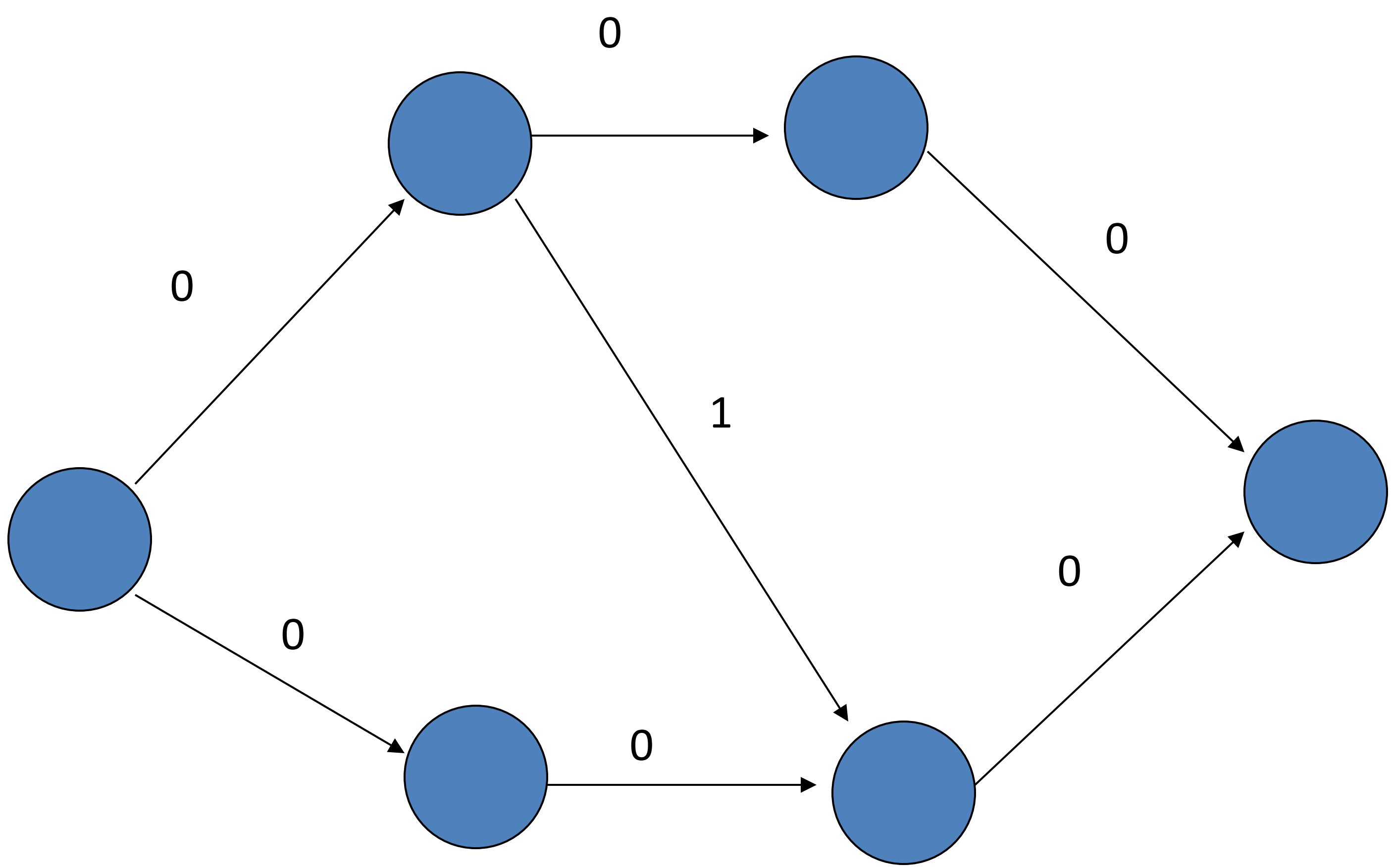
Example

- ❑ **Update remaining capacity**



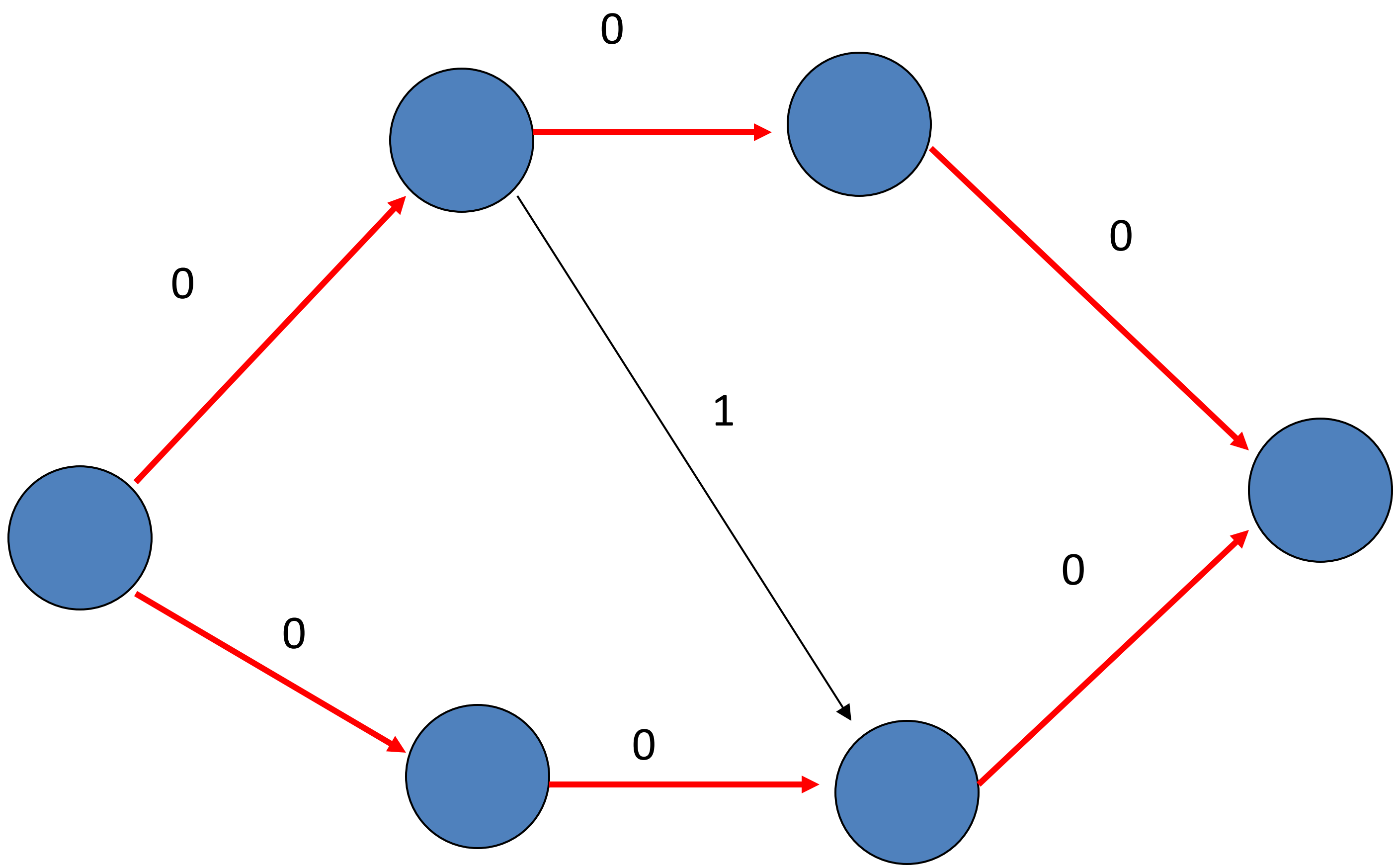
Example

- ❑ **Can we augment any flow through DFS?**



Example

- Maximum flow: 2



What is the Problem?

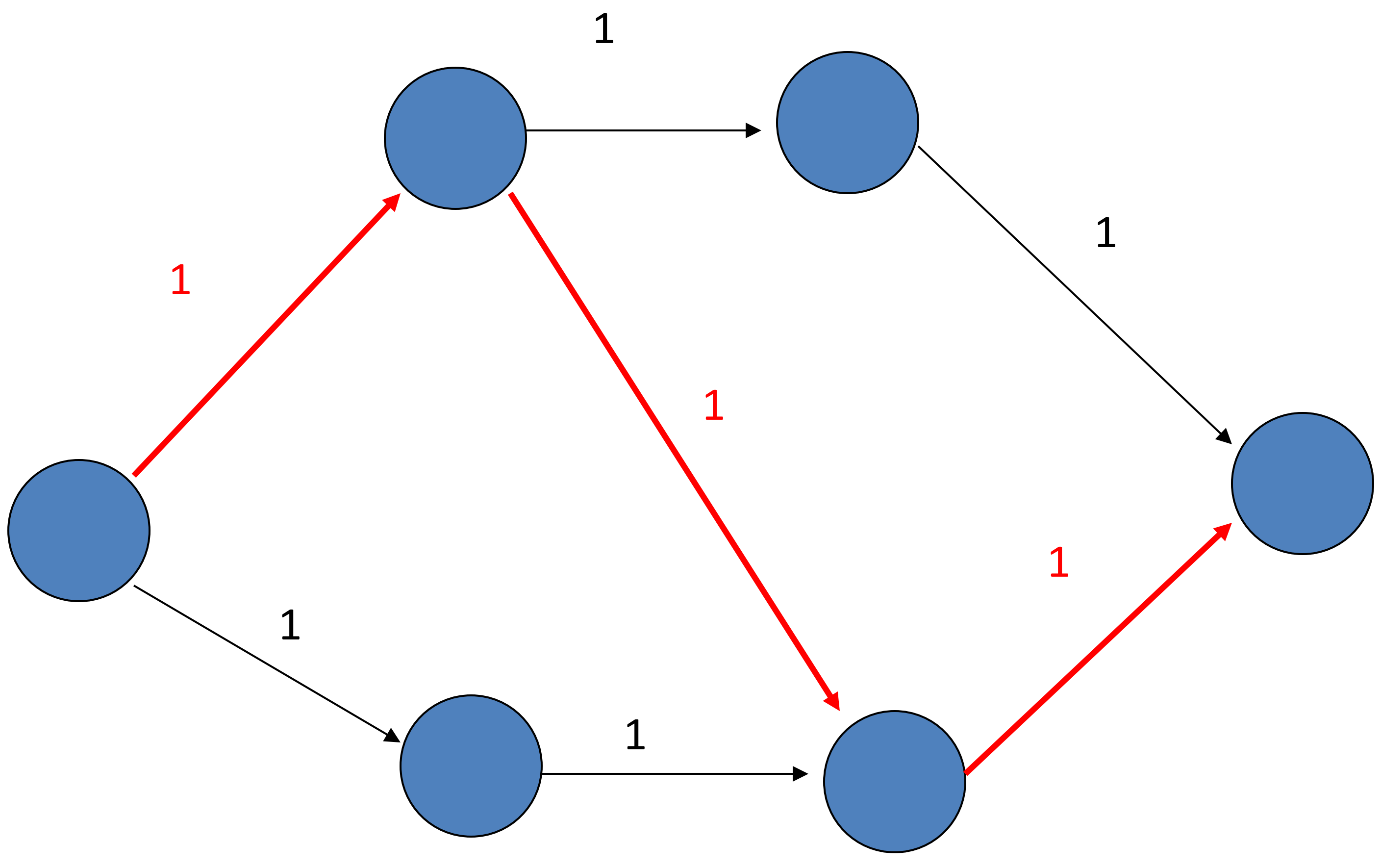
- ❑ **DFS has no order guarantee!**
  - ❑ Order you visit vertices is up to the graph data structure
  - ❑ Different orders may update capacity differently
    - In turn affect the solution

```
procedure DFS(G, v) is
    label v as discovered
    for all directed edges from v to w that are in G.adjacentEdges(v) do
        if vertex w is not labeled as discovered then
            recursively call DFS(G, w)
```

The order each vertex visited by DFS depends on the graph data structure!

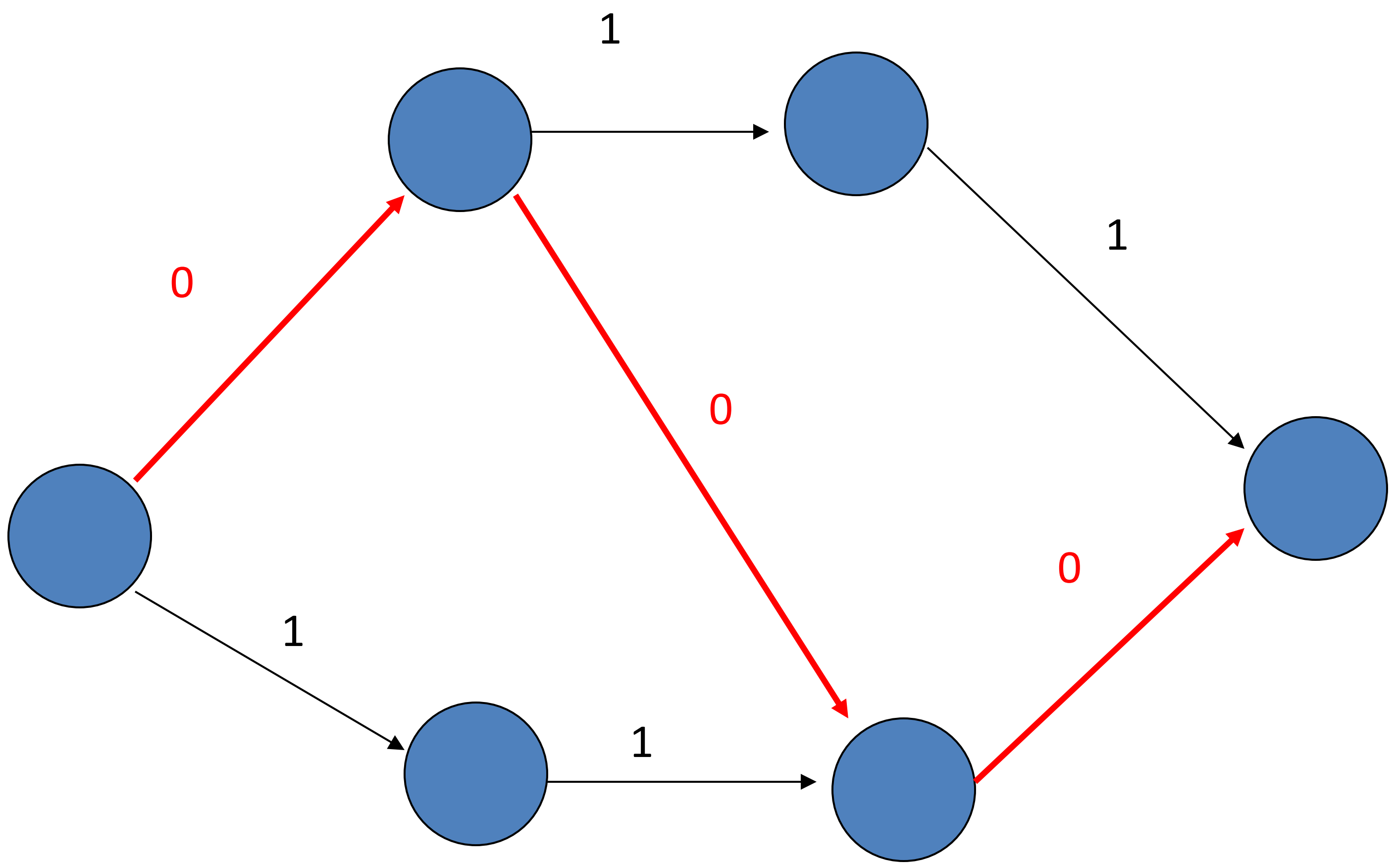
Example

- DFS finds another route in the first iteration



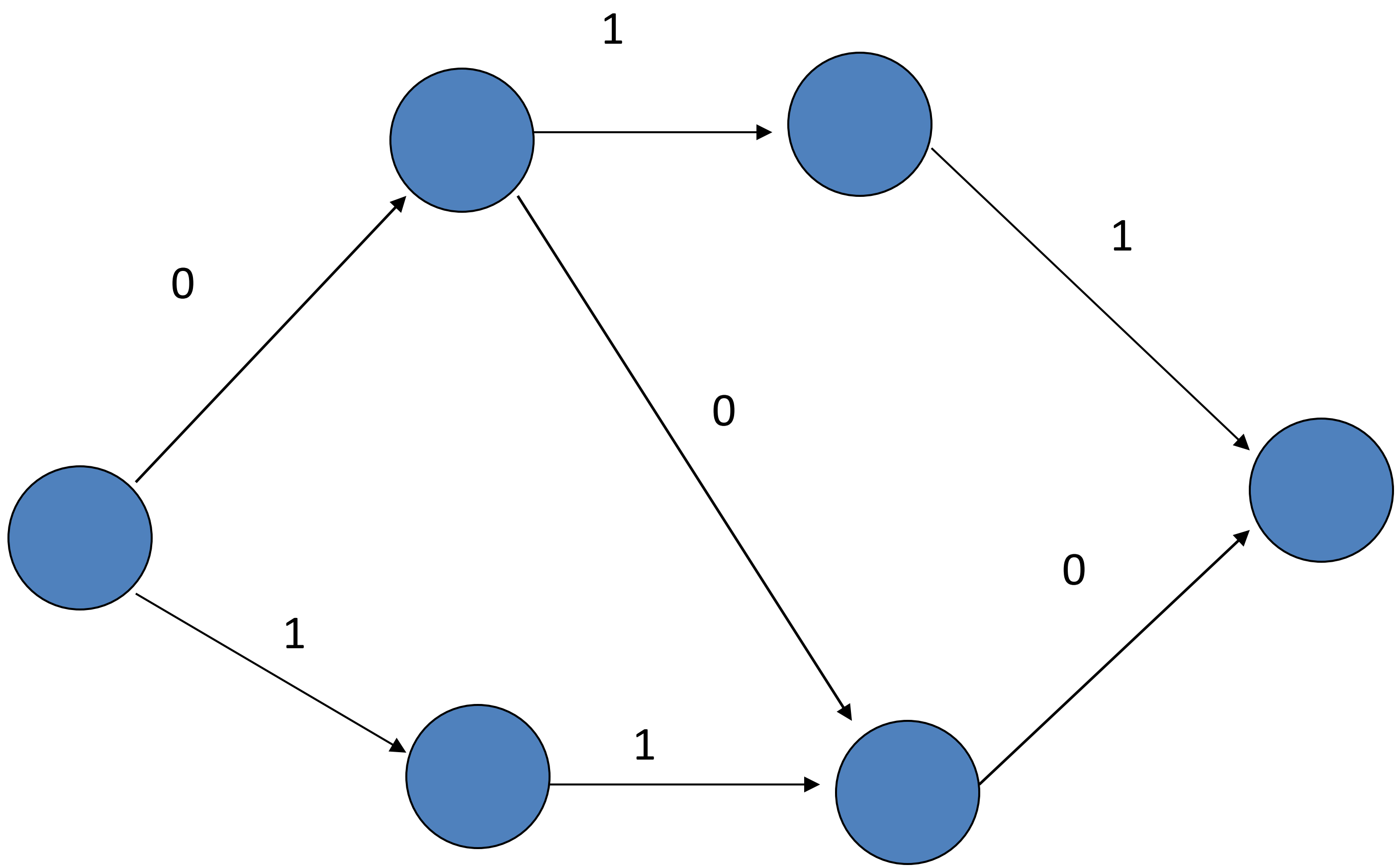
Example

- Update remaining capacity



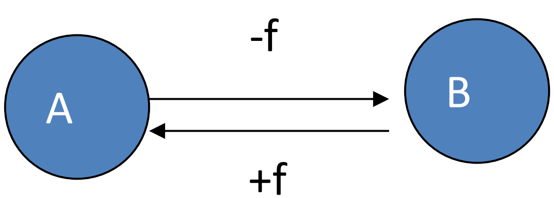
Example

- DFS to augment the flow? Maximum flow = 1?



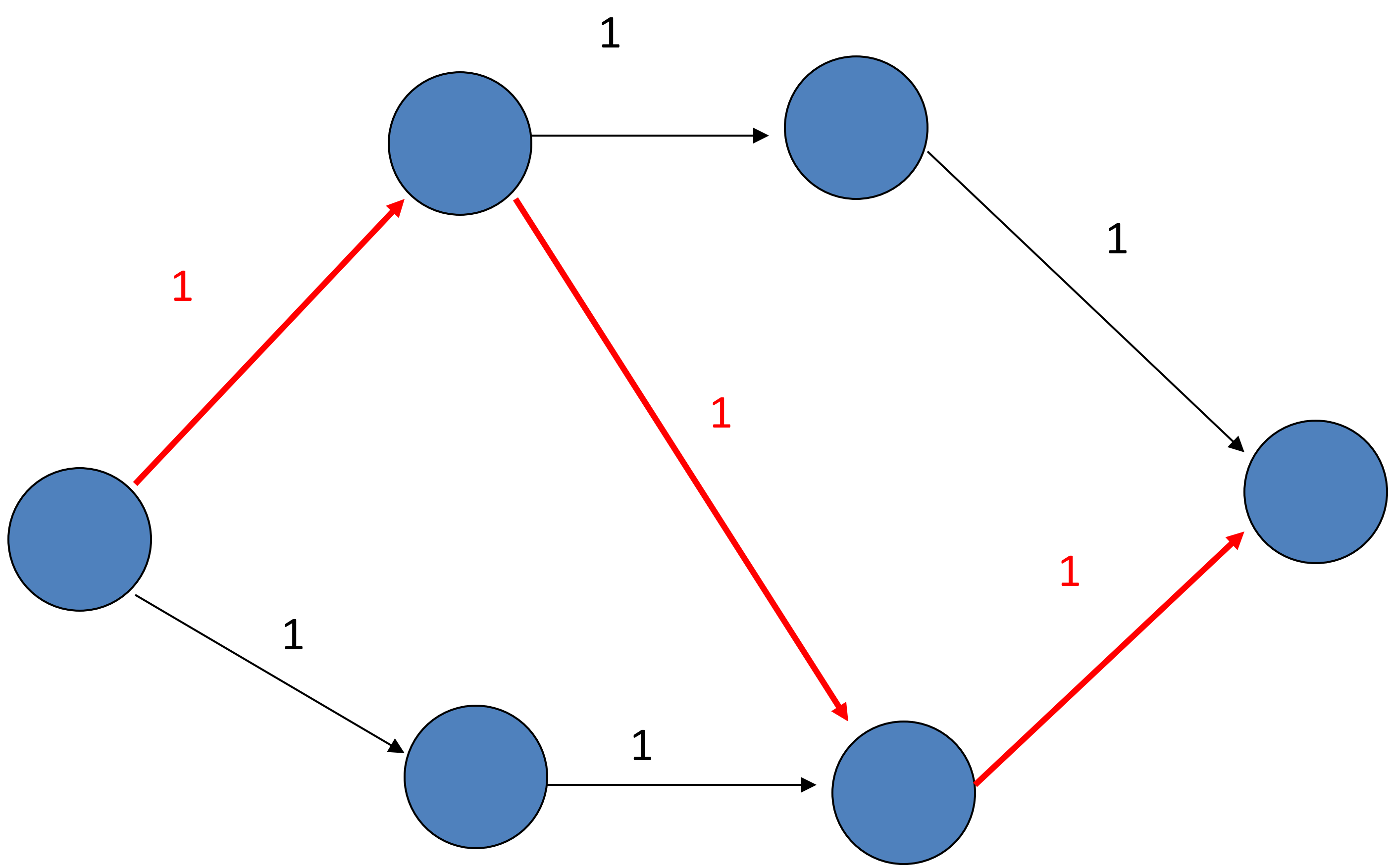
Residual Network

- **Residual network defines edges to admit 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:
    - In the regular direction: $c_f(u,v)=c(u,v)-f(u,v)$
    - In the opposite direction: $c_f(v, u)=c(v, u)+f(u, v)$.
- **If you flow f from A to B**
  - Subtract the regular direction capacity from f
  - Add f to the opposite direction capacity



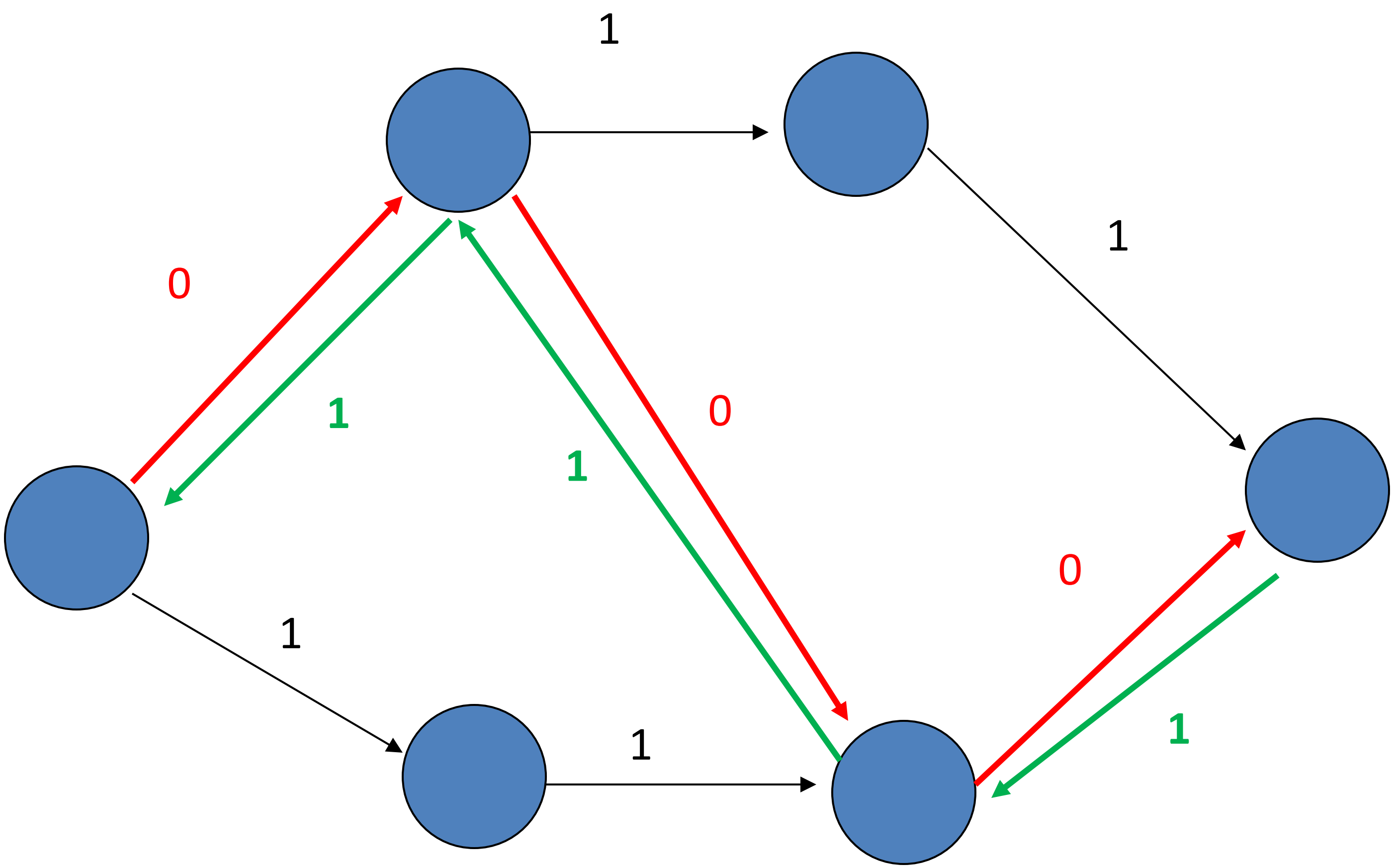
Example

- ❑ **DFS augments a unit flow in the first iteration**



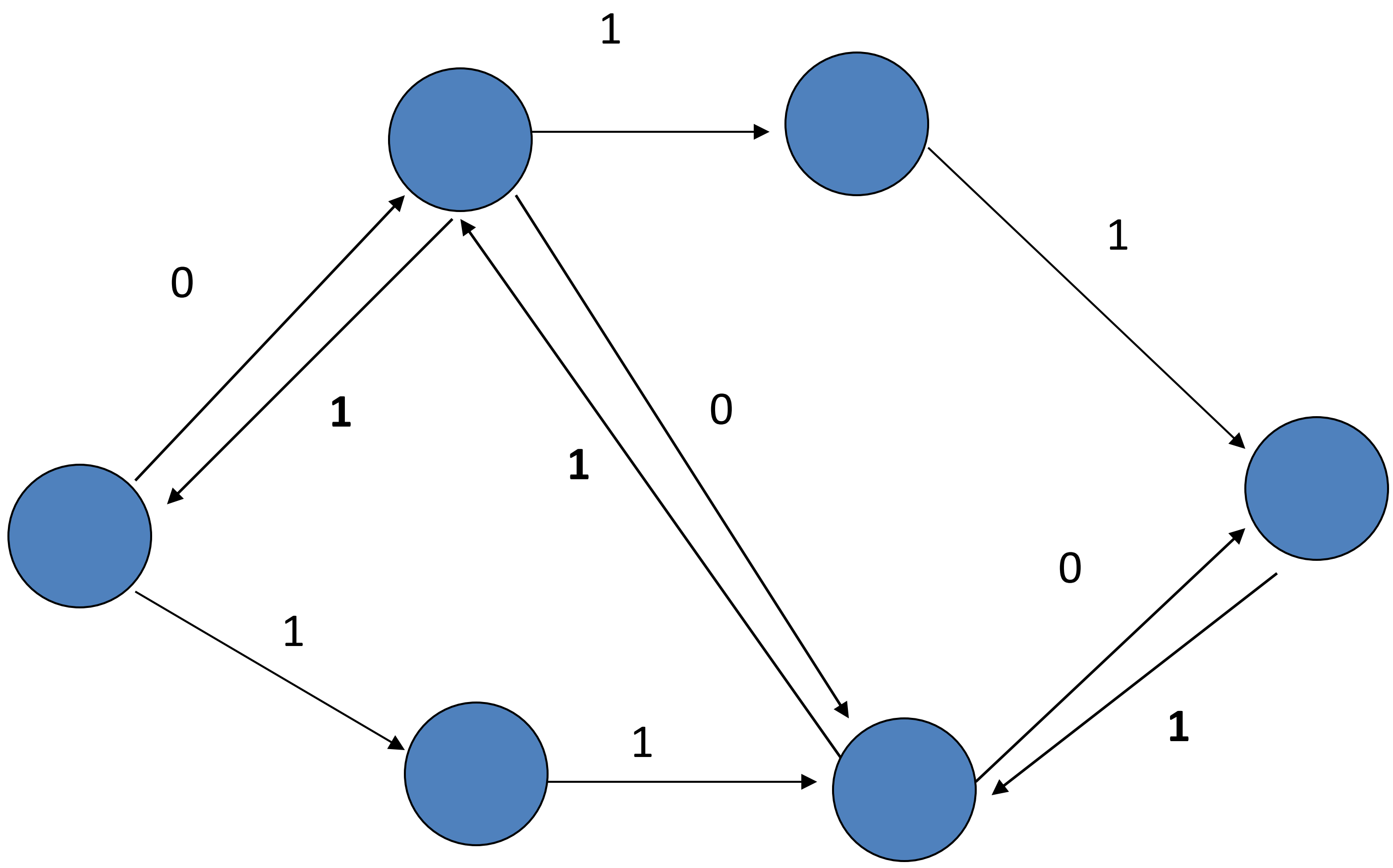
Example

- ❑ **Update the residual network**



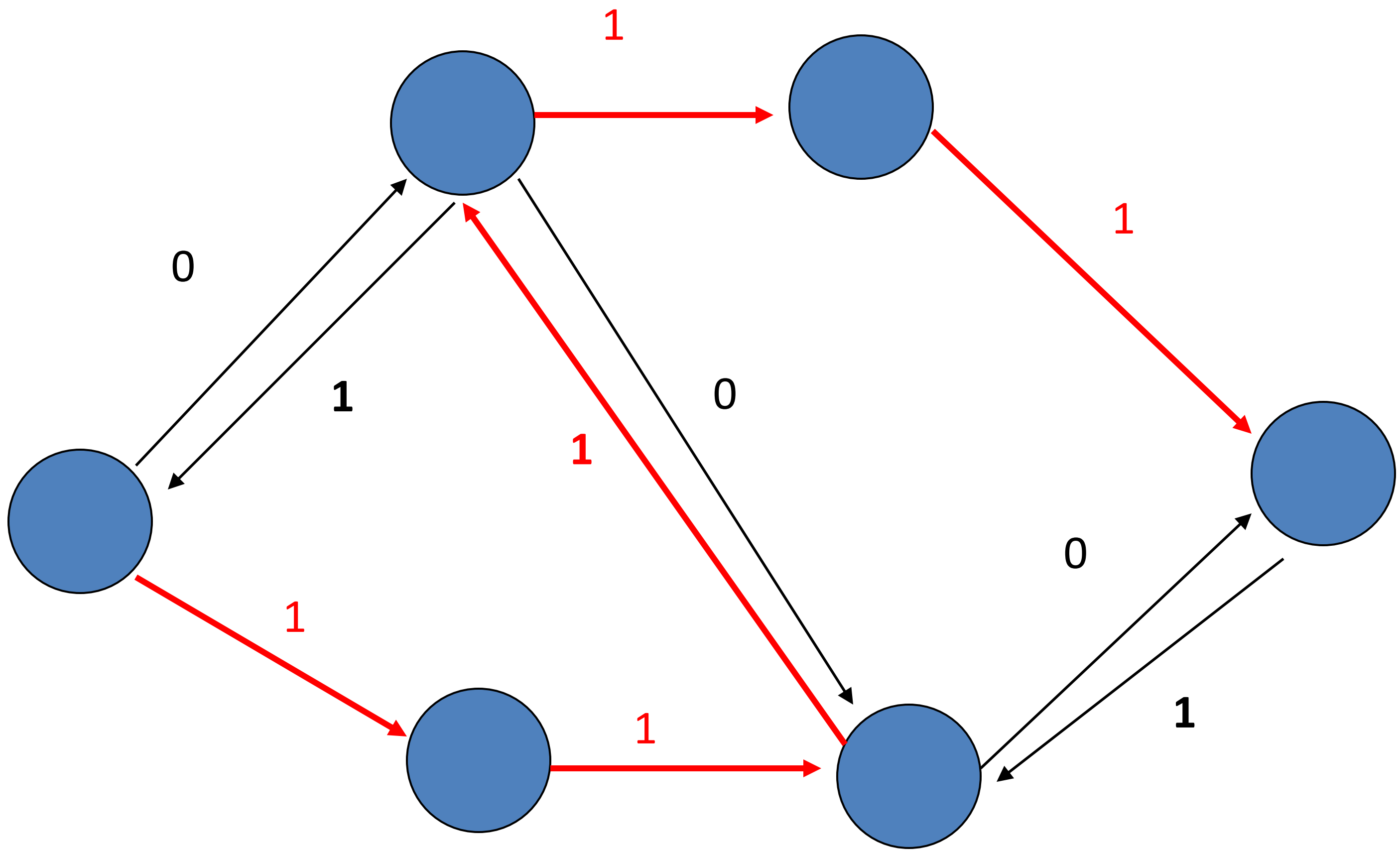
Example

- ❑ **Residual network gives a chance to “circle back”**



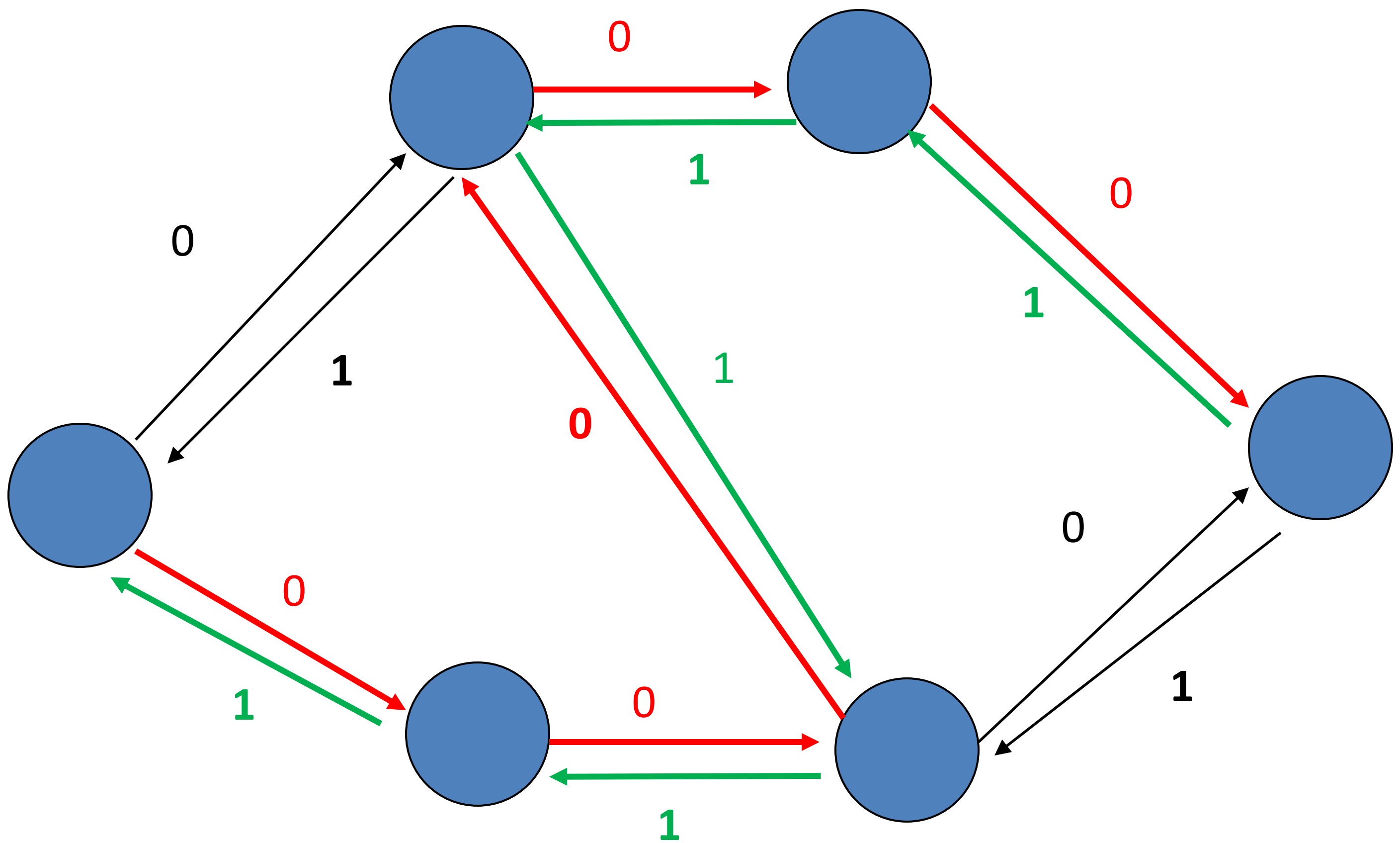
Example

- DFS augments another unit flow in the second iter



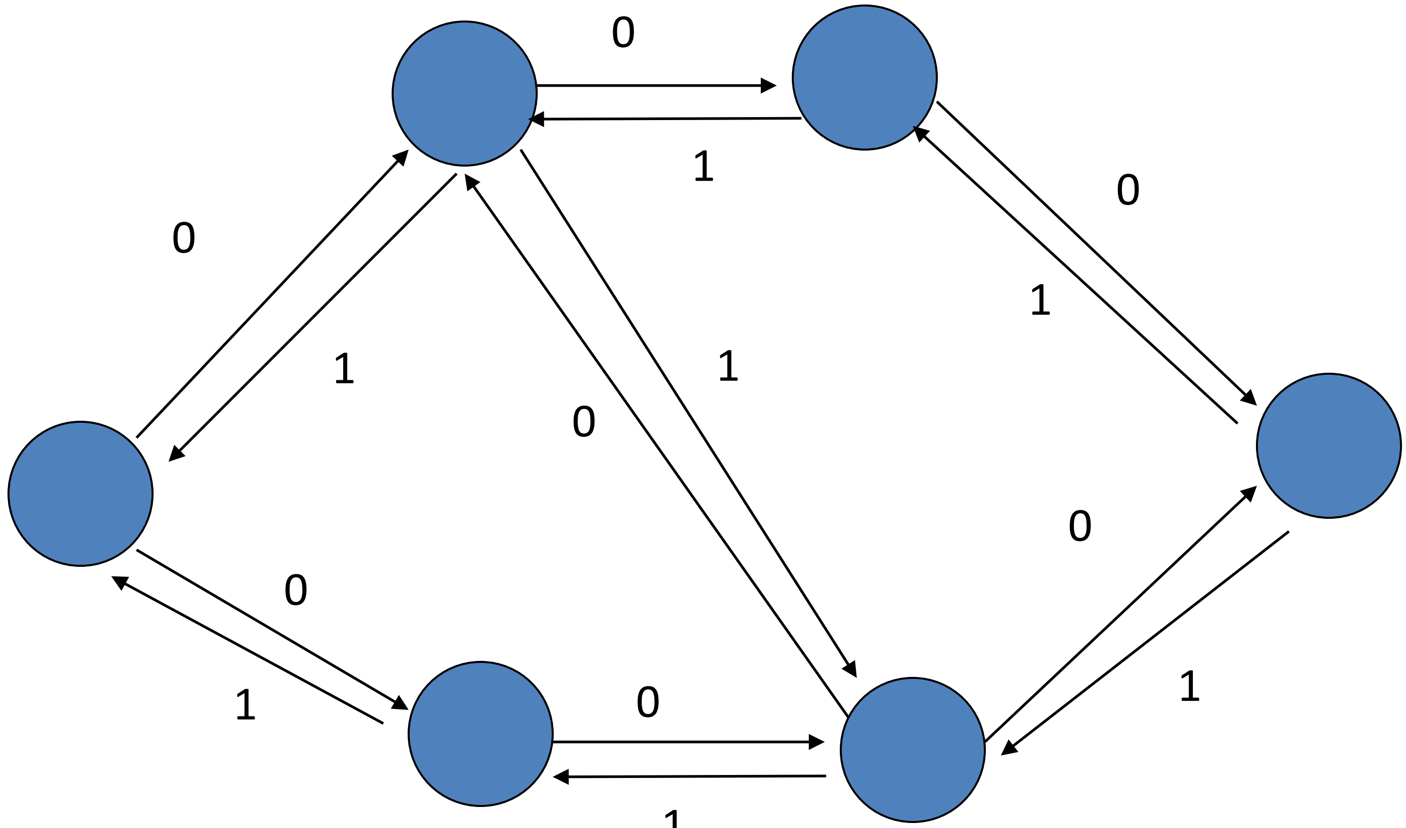
Example

- ❑ **Update residual network**



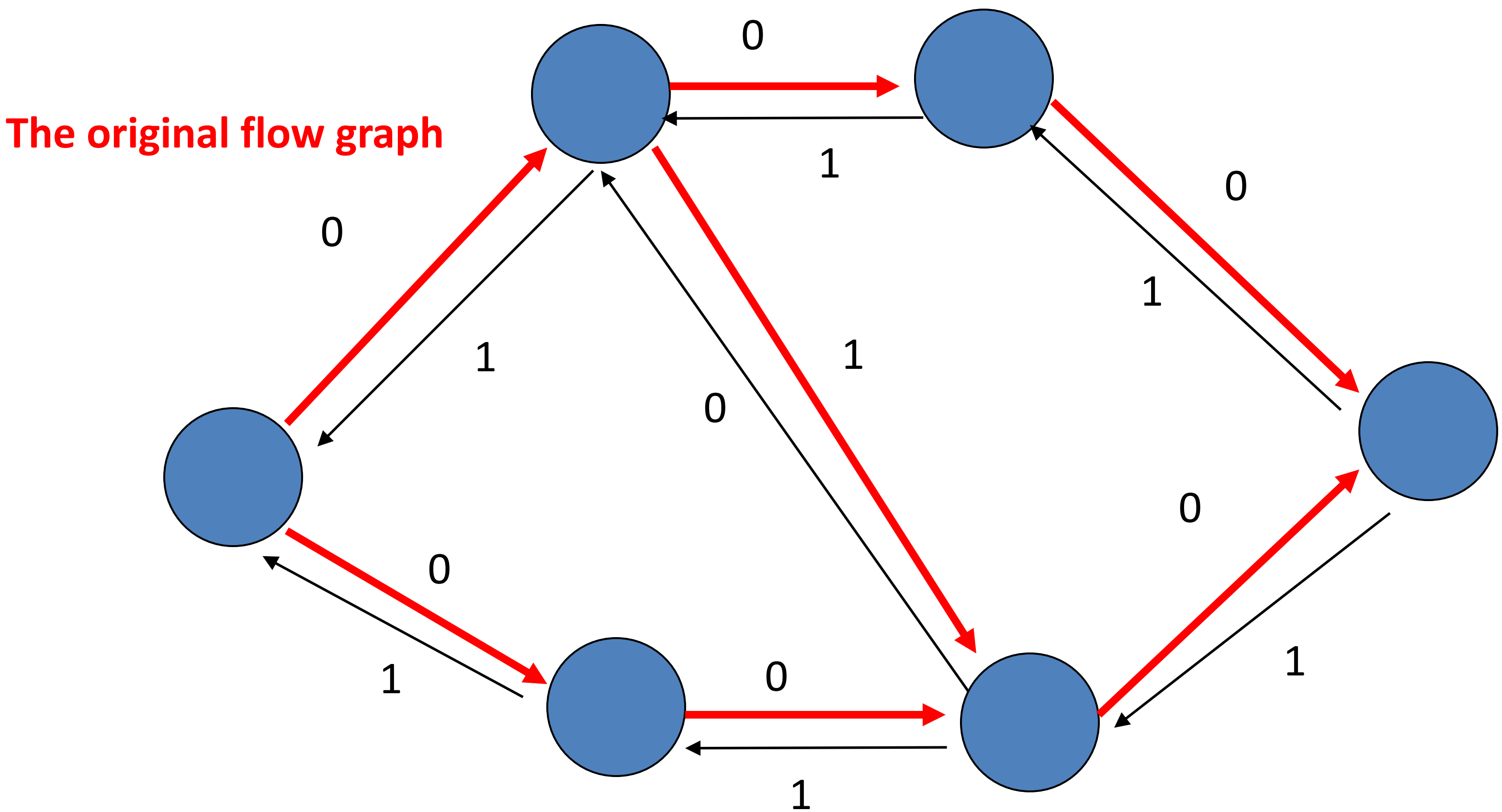
Example

- Maximum flow: 2

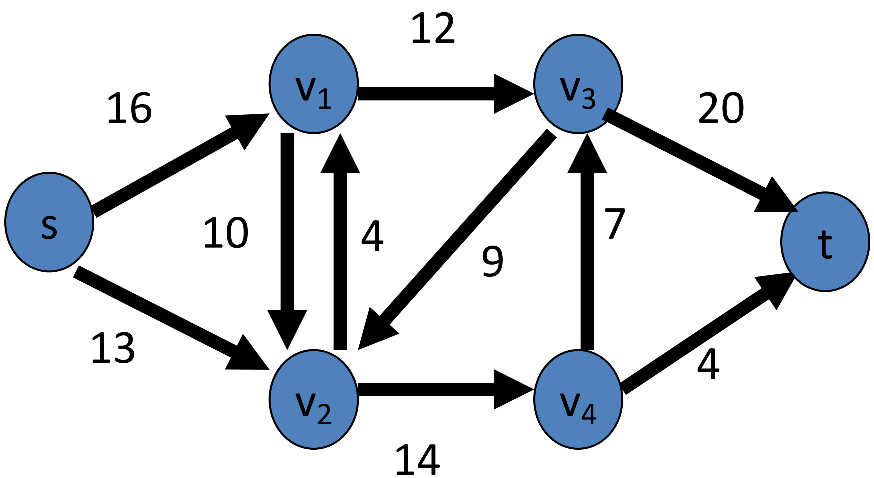


Example

- Residual network gives us a way to circle flow back

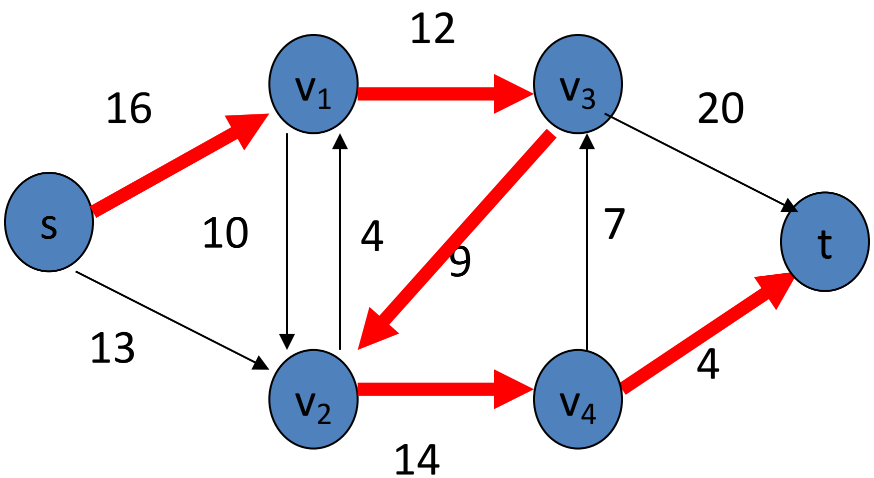


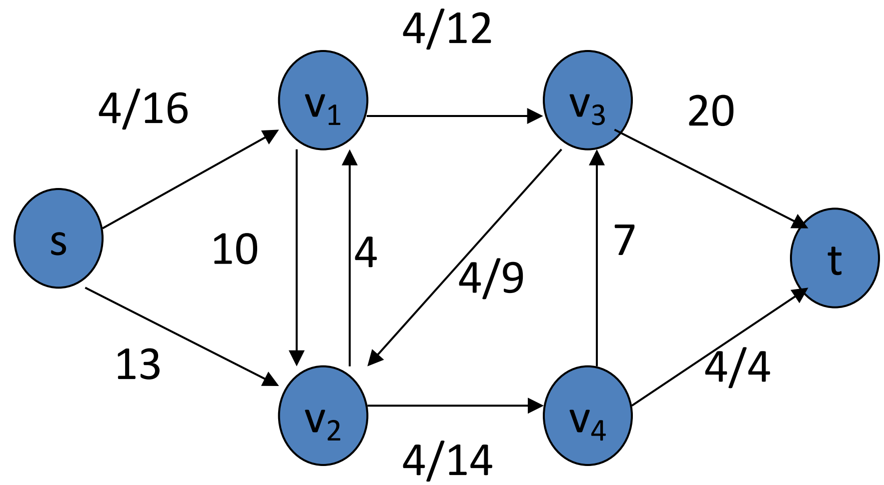
Slightly Complicated Example



Initial

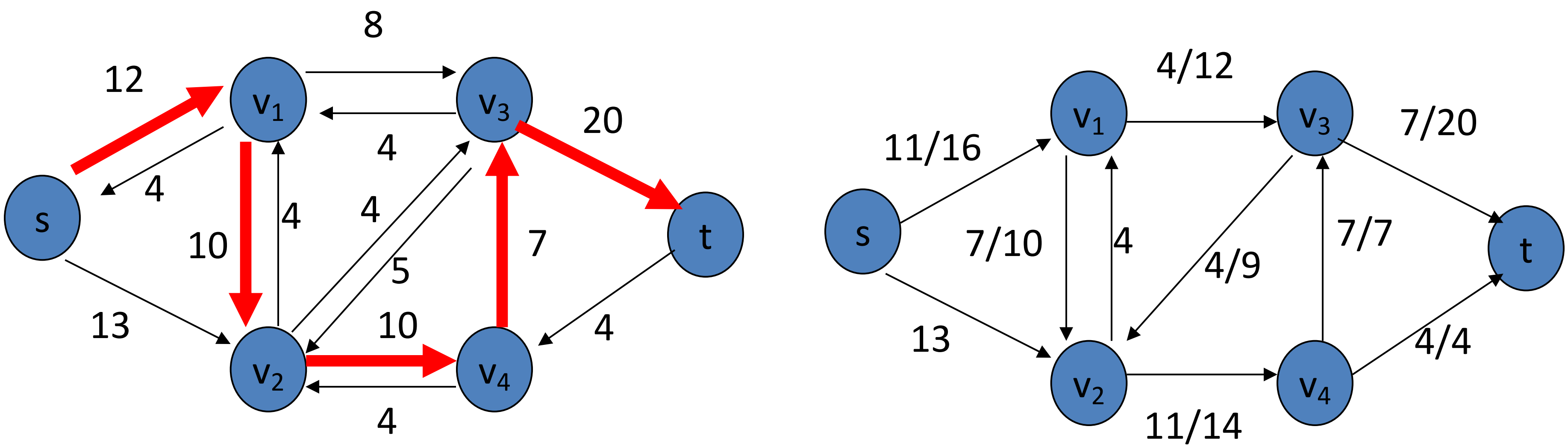
Slightly Complicated Example





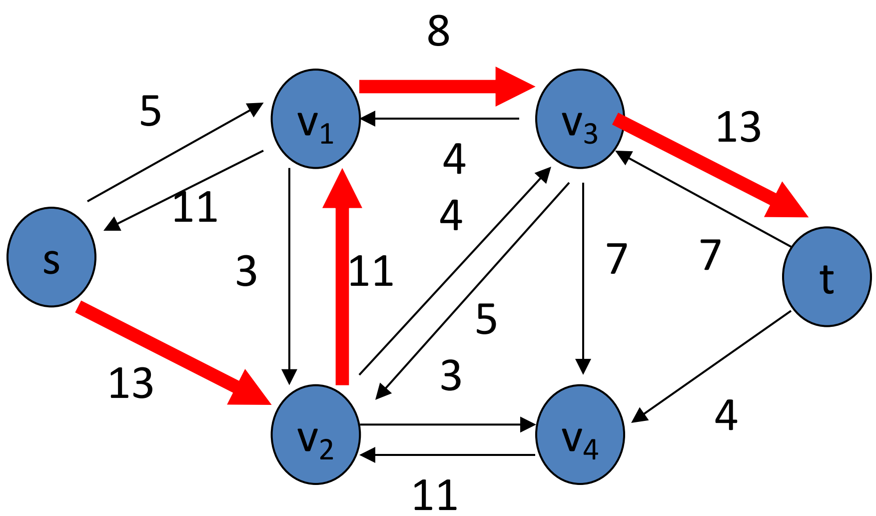
(a) Maximum flow: 4

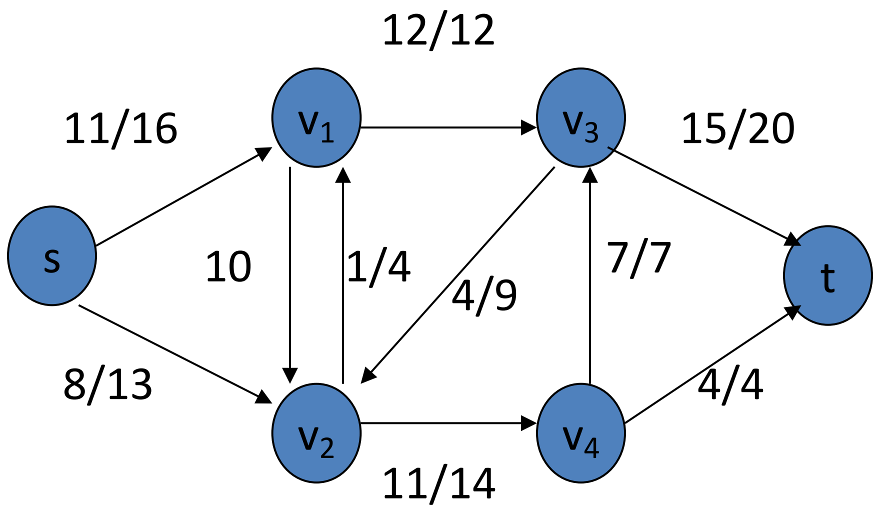
Slightly Complicated Example



(b) Maximum flow: 4 + 7

Slightly Complicated Example





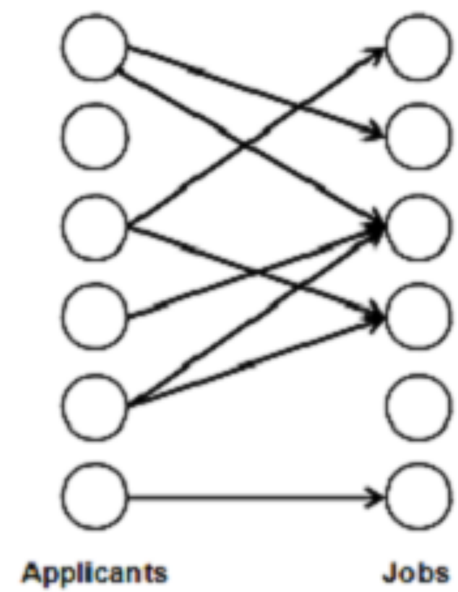
(c) Maximum flow: 4 + 7 + 8

Code Snippet

```
while(1) {
   // … initialize BFS storage

   while(!BFS.empty()) {
      now = BFS.front();
      for(next=0; next<n; next++) {
         if(visited[next])continue;
         if(mat[now][next]-flow[now][next]>0)  {    // Positive direction
            p[next] = now, visited[next] = true;
            BFS.push(next);
          }
          else if(flow[next][now]>0)  {         // Opposite direction
              p[next] = -now, visited[next] = true;
              BFS.push(next);
          }
      }
      BFS.pop();
   }

  if(!visited[sink]) break;     //If not find the augmenting path.

   for(minf=INF, i=sink; i!=source; i=abs(p[i])) {
      if(p[i]>=0) minf = min(minf, mat[p[i]][i]-flow[p[i]][i]);
      else minf = min(minf, flow[i][-p[i]]);

      for(i=sink; i!=source; i=abs(p[i])) {
         if(p[i]>=0)  flow[p[i]][i]  += minf;
         else flow[i][-p[i]] -= minf;
      }
   }
   for(i=0; i<n; i++)   MAX_FLOW += flow[source][i];
   return MAX_FLOW;
}
```

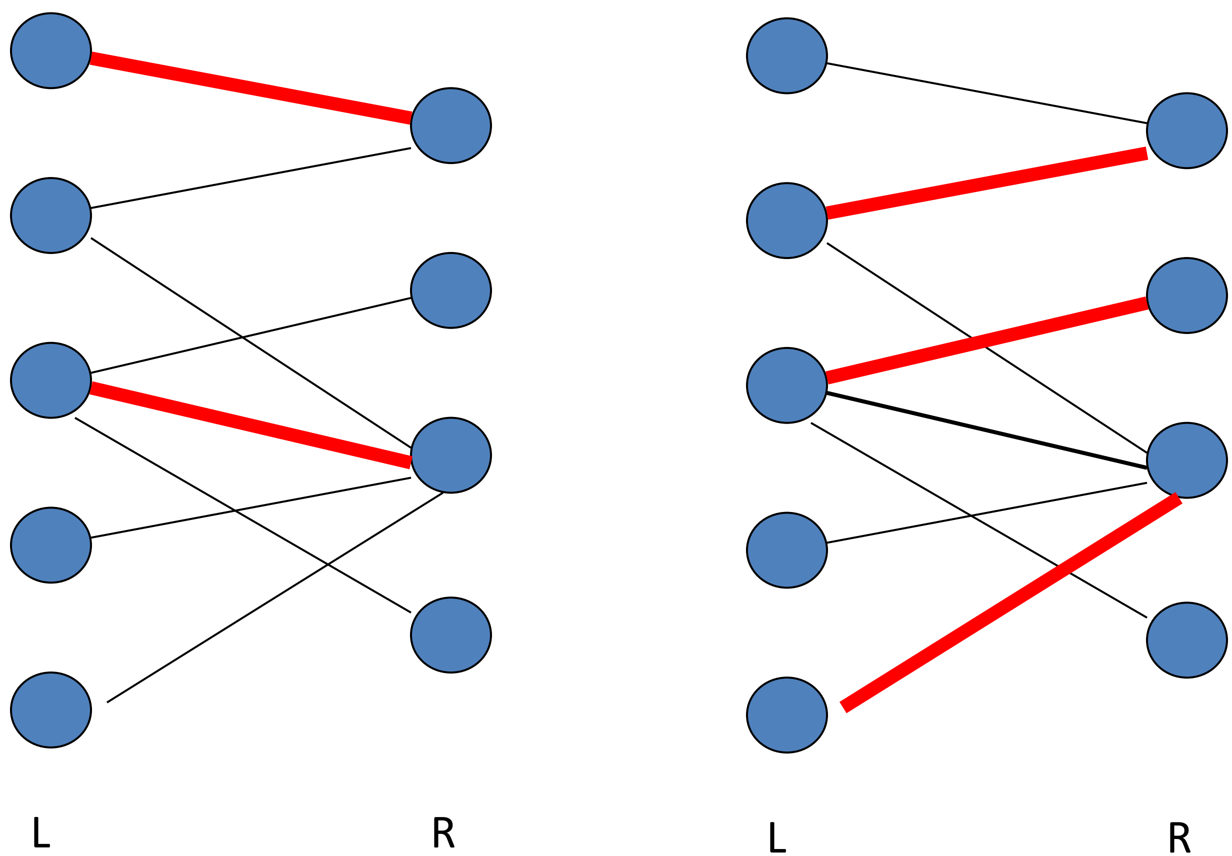
Bipartite Matching

- A matching in a *Bipartite Graph* is a set of the edges chosen in such a way that no two edges share an endpoint.
- A maximum matching is a matching of maximum size (maximum number of edges).
- In a maximum matching, if any edge is added to it, it is no longer a matching. There can be more than one maximum matchings for a given Bipartite Graph.



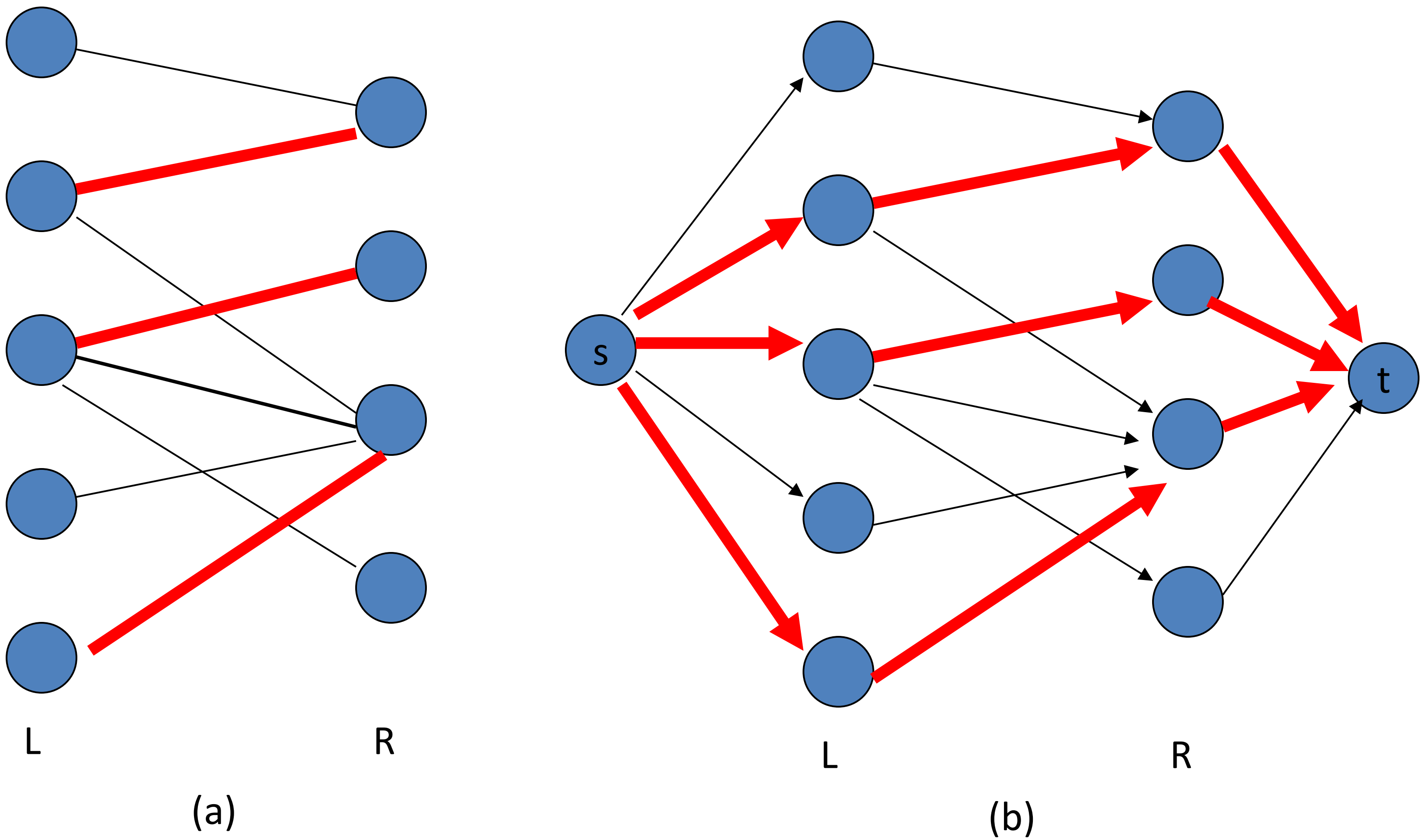
Tremendous real applications …

Maximum Bipartite Matching



(a) (b)

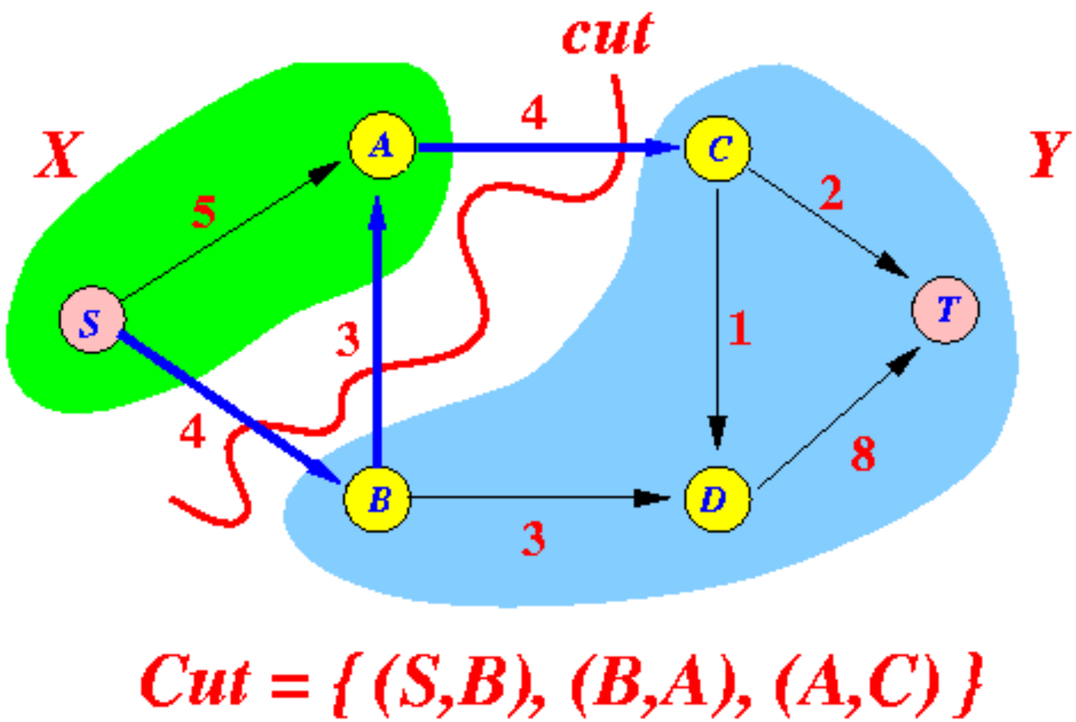
Maximum Bipartite Matching



We can solve this by running maximum flow!

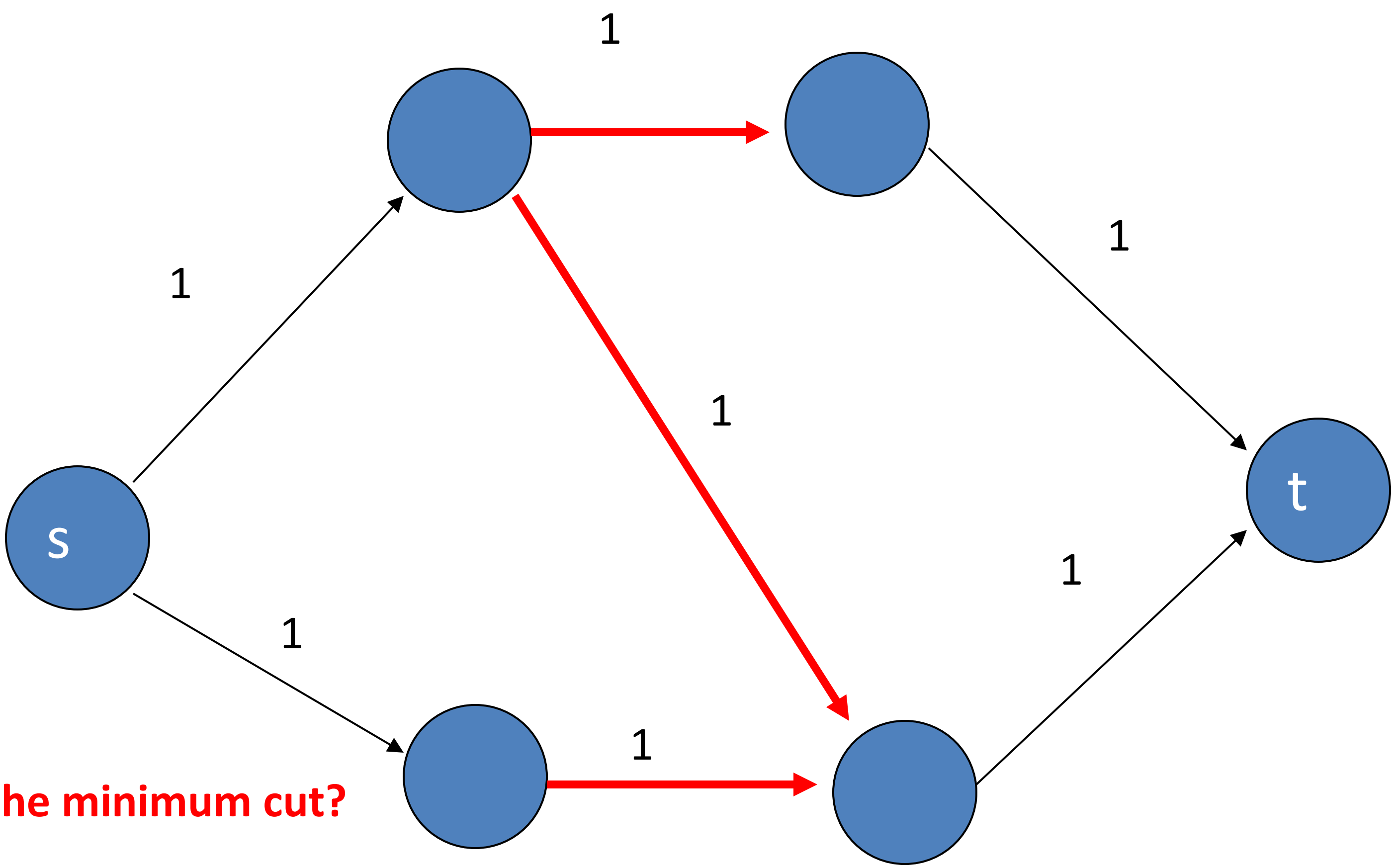
s-t Cut

- A s-t cut of a graph G consists of an edge set E such that G - E separate s and t in two components



Minimum s-t Cut

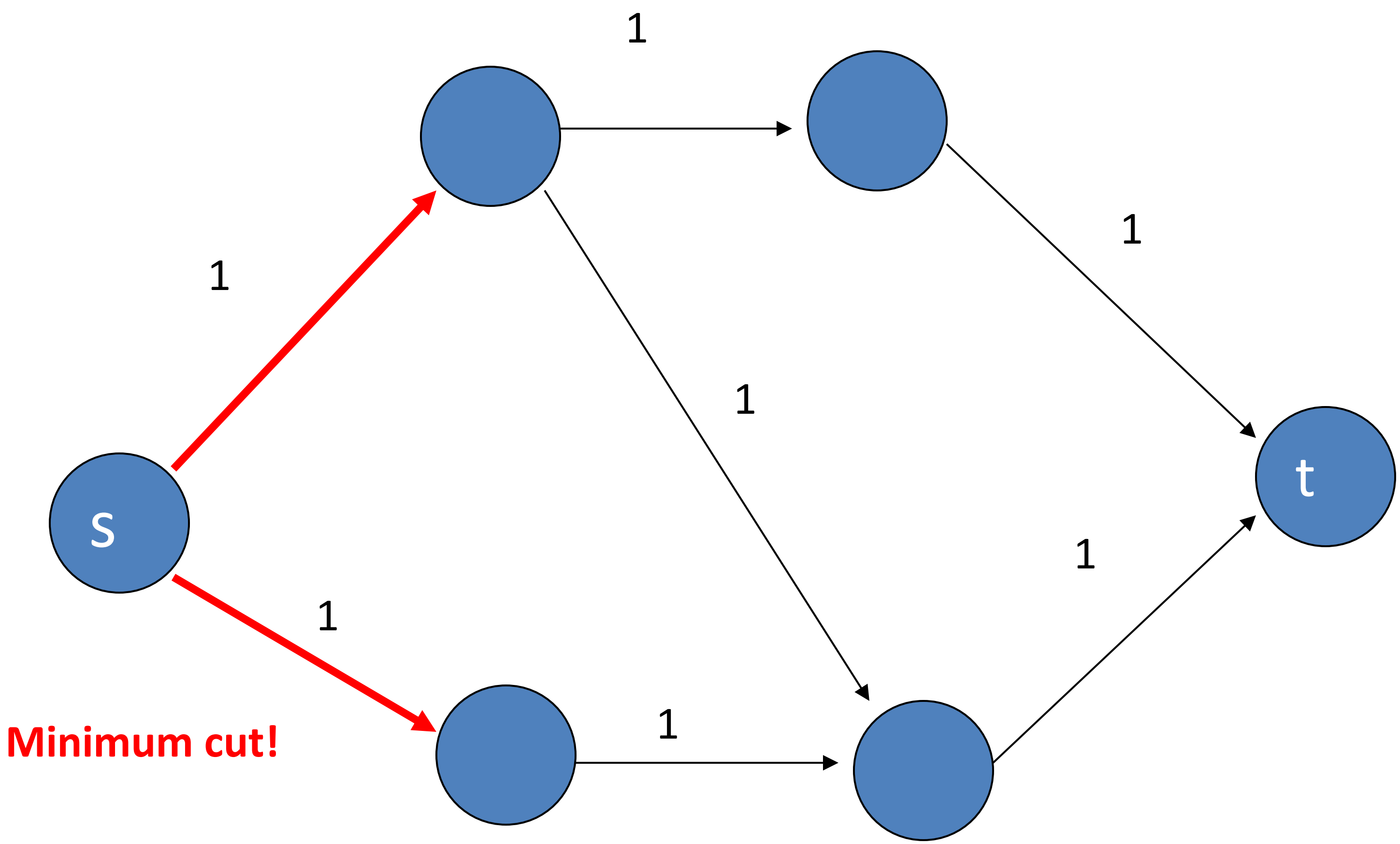
- The s-t cut set with the minimum cut weight



Is this the minimum cut?

Minimum s-t Cut

- The s-t cut set with the minimum weight



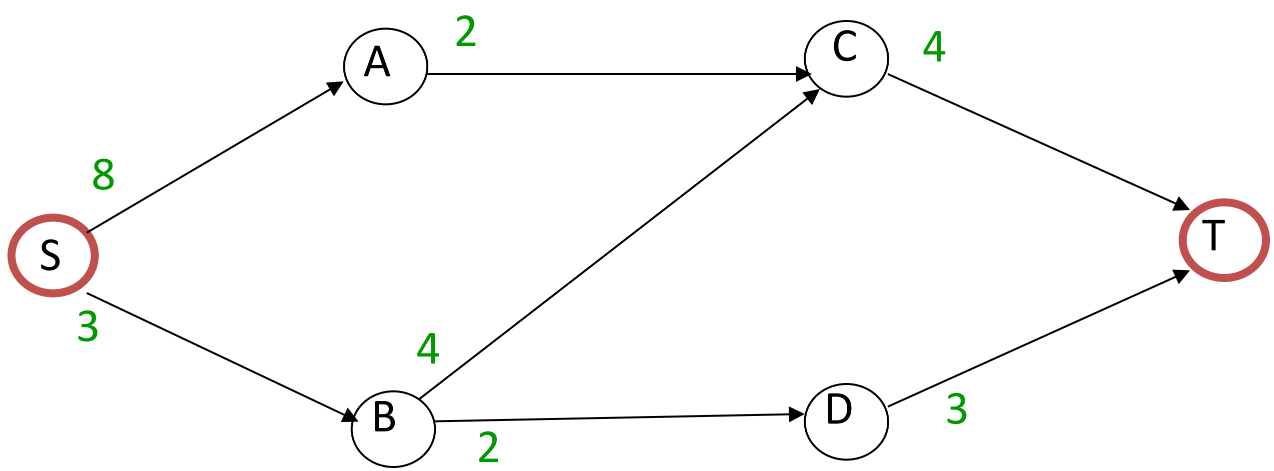
Observation

- Can we give upper bounds on the maximum flow value before finding any augmenting paths?
  - One possible upper bound is the total capacity of the arcs leaving the source:
  - Another upper bound is the total capacity of the arcs entering the sink:

Ideally, this upper bound is equal to the maximum flow value.

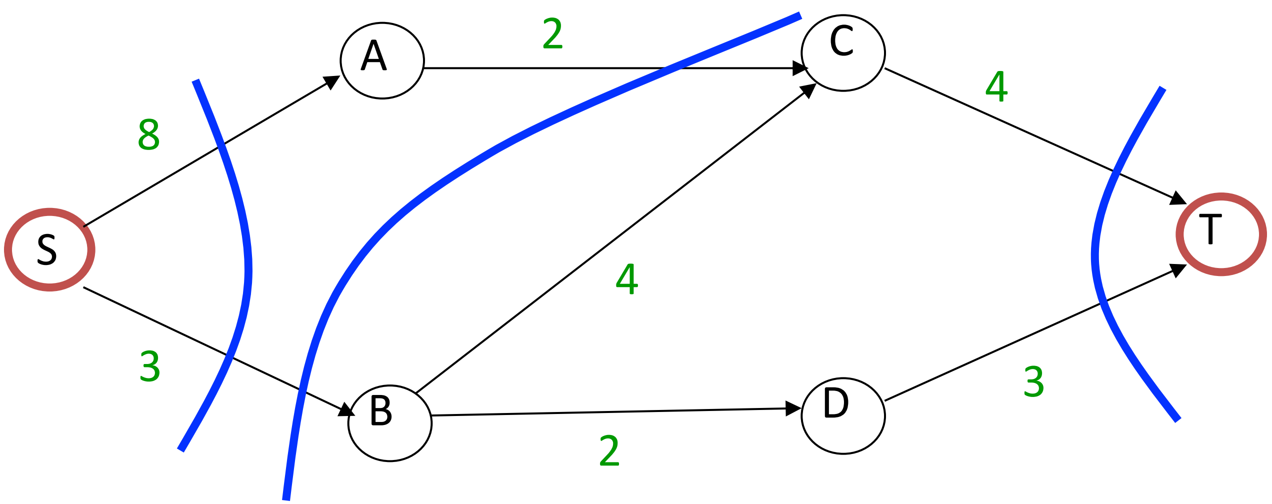
Thus, we could recognize that the algorithm output is optimal simply by **comparing the flow value with the upper bound.**

Example



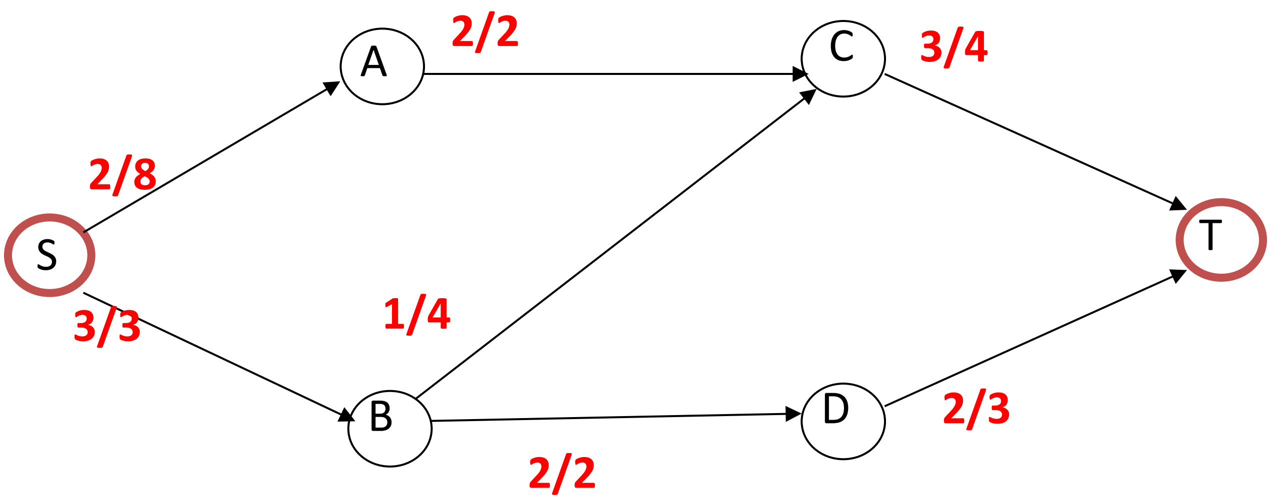
Example

- **Find the upper bound of the flow value**
  - Choice 1: S→A, S→B (cut weight 8+3 = 11)
  - Choice 2: A→C, S→B (cut weight 3+2 = 5)
  - Choice 3: C→T, D→T (cut weight 4+3 = 7)



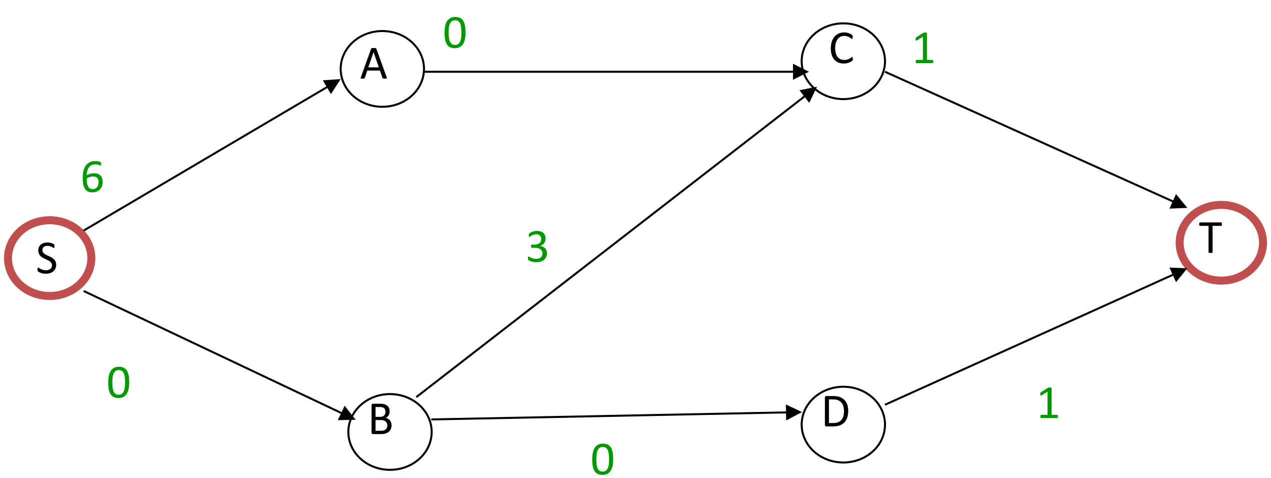
Example

- ❑ **Maximum flow: 5**



Example

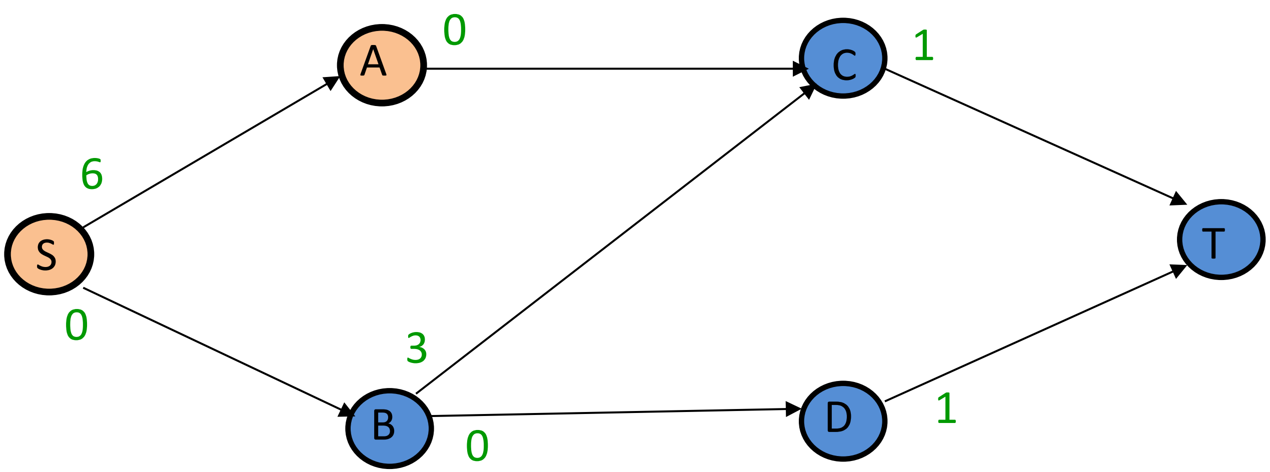
- **Residual network tells us the separation!**
  - For brevity, we only show normal directions



  - All nodes reachable from s belong to the same side!

Example

- **We can do another traversal to find the partition**
  - Cut on s side: {S, A}
  - Cut on t side: {B, C, D, T}



Summary

- Maximum flow
- Bipartite matching
- Minimum s-t cut