Network Flows

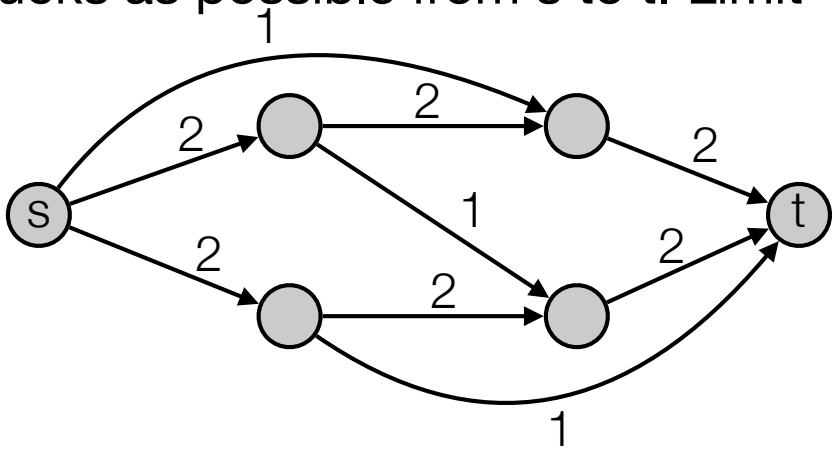
Inge Li Gørtz

Applications

- Matchings
- Job scheduling
- Image segmentation
- Baseball elimination
- Disjoint paths
- Survivable network design

Network Flow

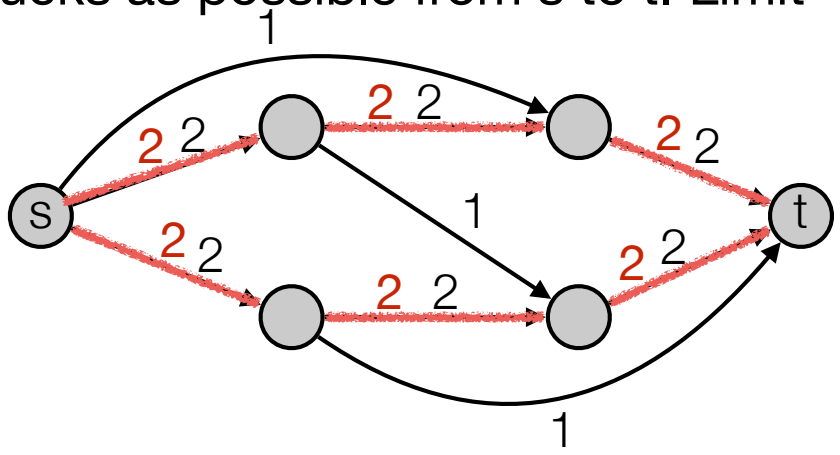
- Truck company: Wants to send as many trucks as possible from s to t. Limit of number of trucks on each road.



Network Flow

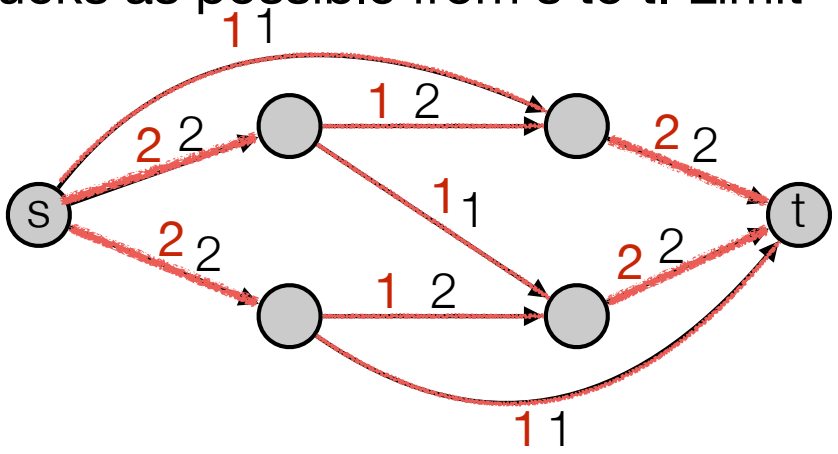
- Truck company: Wants to send as many trucks as possible from s to t. Limit of number of trucks on each road.

- Example 1:
  - Solution 1: 4 trucks



Network Flow

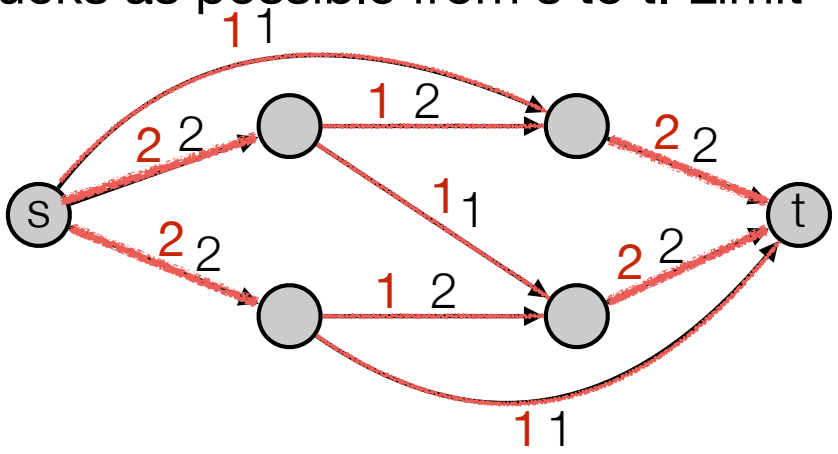
- Truck company: Wants to send as many trucks as possible from s to t. Limit of number of trucks on each road.



- Example 1:
  - Solution 1: 4 trucks
  - Solution 2: 5 trucks

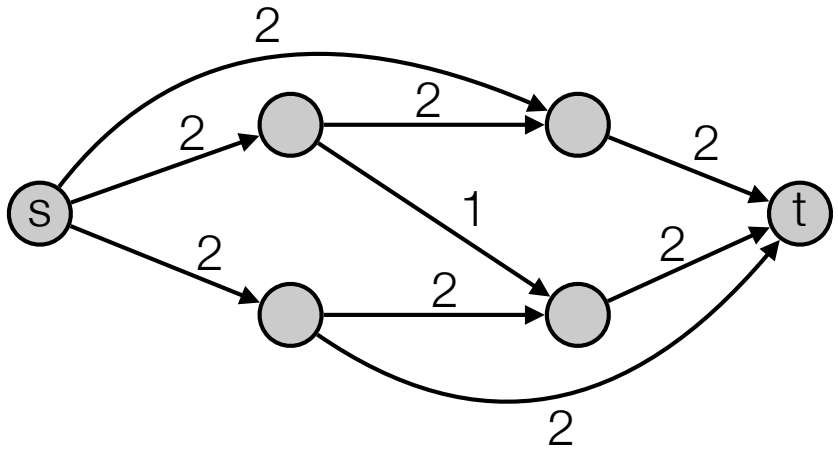
Network Flow

- Truck company: Wants to send as many trucks as possible from s to t. Limit of number of trucks on each road.



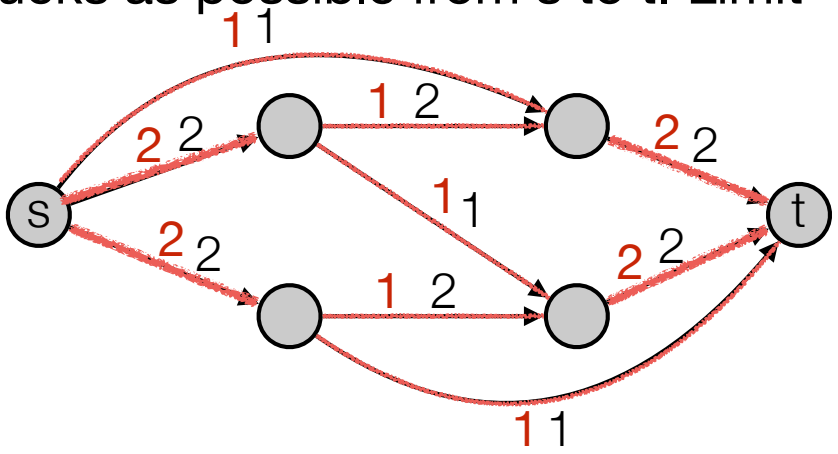
- Example 1:
  - Solution 1: 4 trucks
  - Solution 2: 5 trucks

- Example 2:



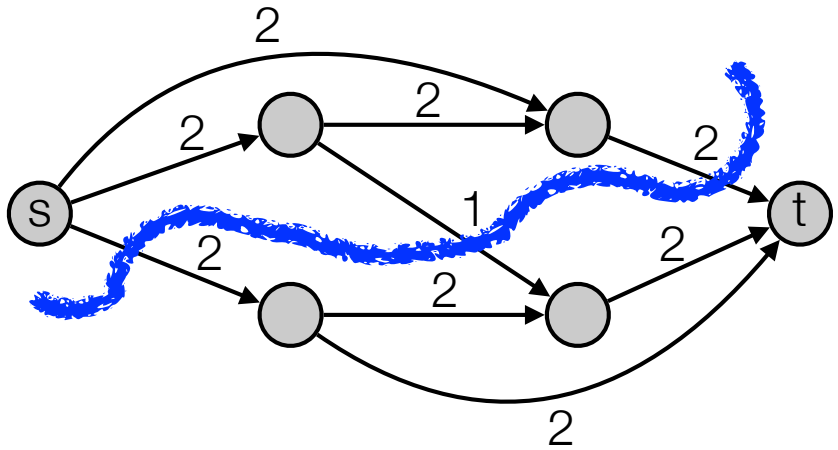
Network Flow

- Truck company: Wants to send as many trucks as possible from s to t. Limit of number of trucks on each road.



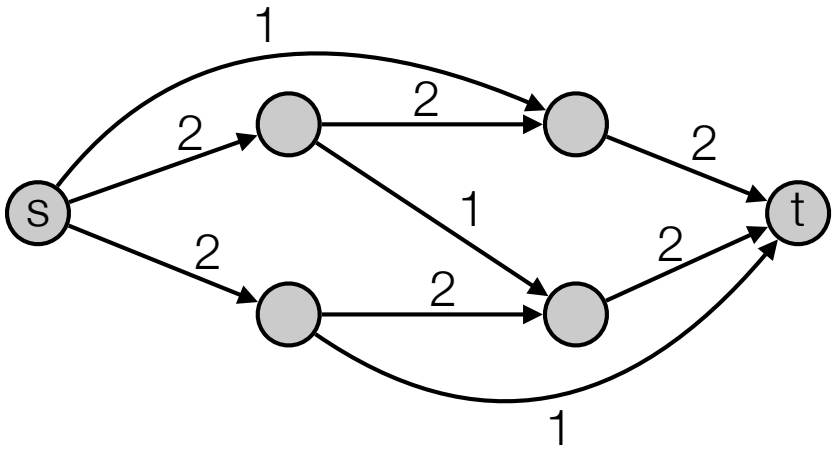
- Example 1:
  - Solution 1: 4 trucks
  - Solution 2: 5 trucks

- Example 2:
  - 5 trucks (need to cross river).

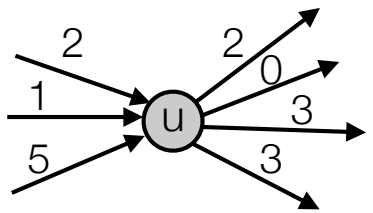


Network Flow

- Network flow:
  - graph G=(V,E).
  - Special vertices s (source) and t (sink).
  - s has no edges in and t has no edges out.
  - Every edge (e) has a (integer) capacity c(e) ≥ 0.
  - Flow:
    - **capacity constraint:** every edge e has a flow 0 ≤ f(e) ≤ c(e).
    - **flow conservation:** for all u ≠ s, t: flow into u equals flow out of u.



$$\sum_{v:(v,u)\in E} f(v,u) = \sum_{v:(u,v)\in E} f(u,v)$$



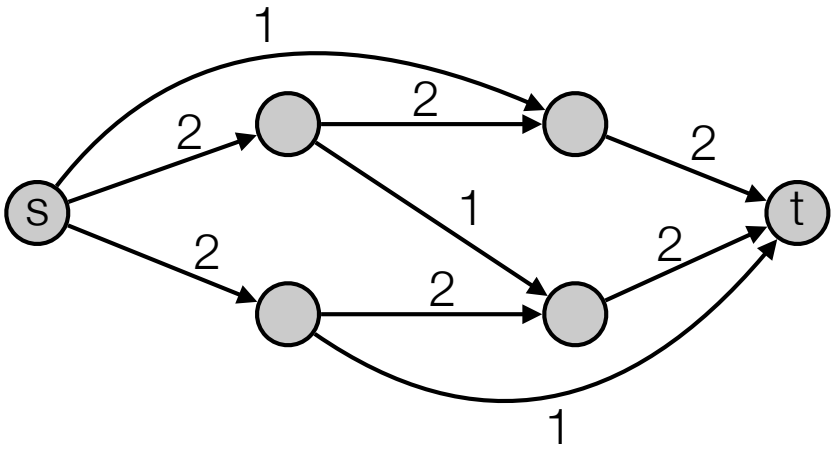
  - Value of flow f is the sum of flows out of s:

$$v(f) = \sum_{v:(s,v)\in E} f(e) = f^{out}(s)$$

- **Maximum flow problem:** find s-t flow of maximum value

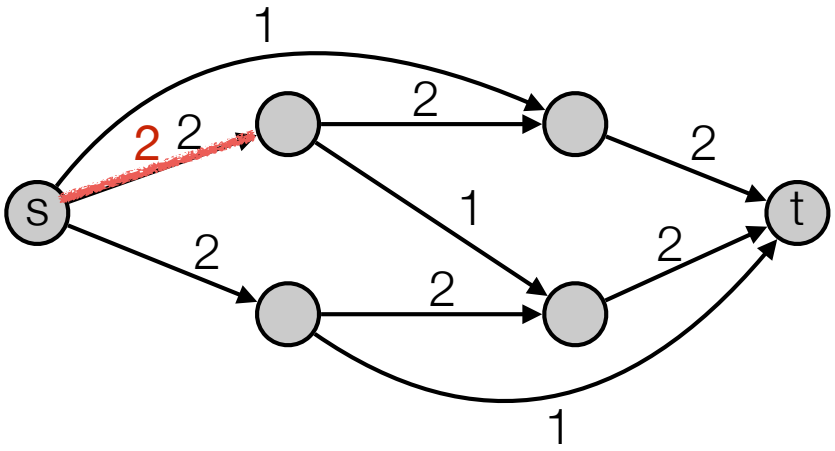
Algorithm

- Find path where we can send more flow.



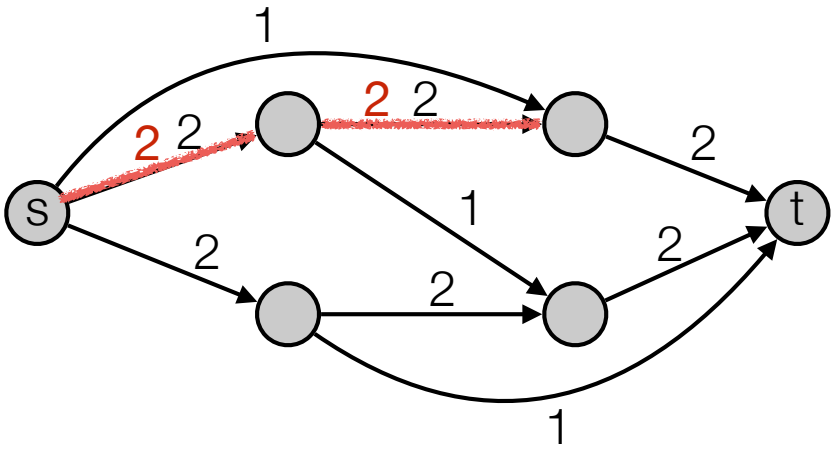
Algorithm

- Find path where we can send more flow.



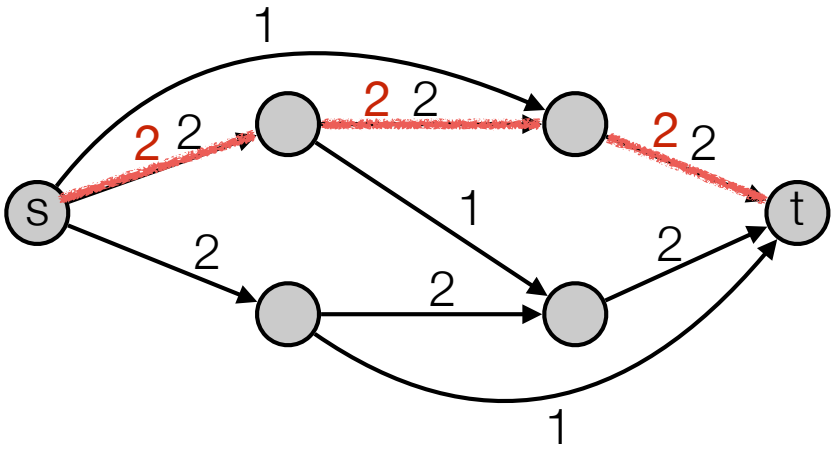
Algorithm

- Find path where we can send more flow.



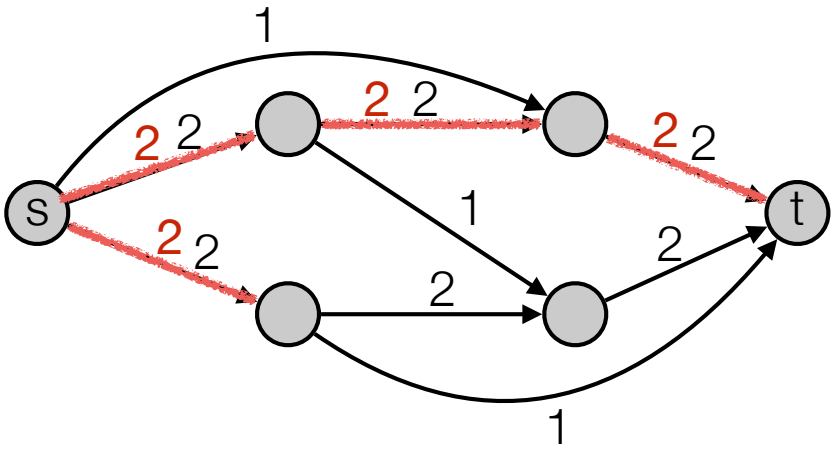
Algorithm

- Find path where we can send more flow.



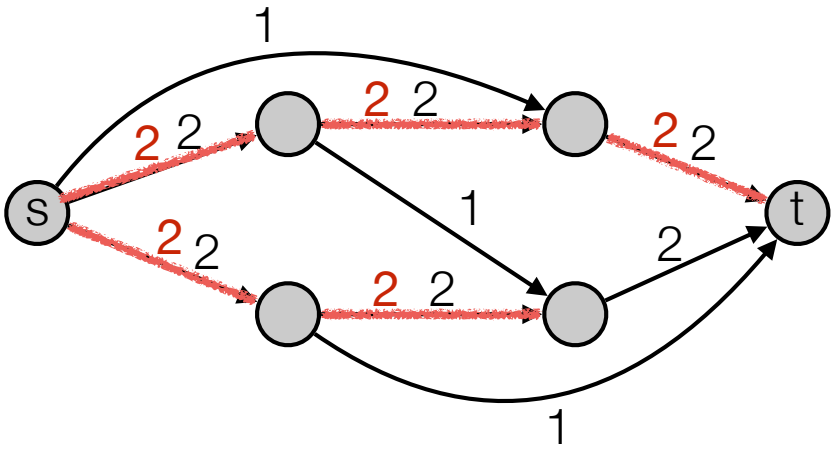
Algorithm

- Find path where we can send more flow.



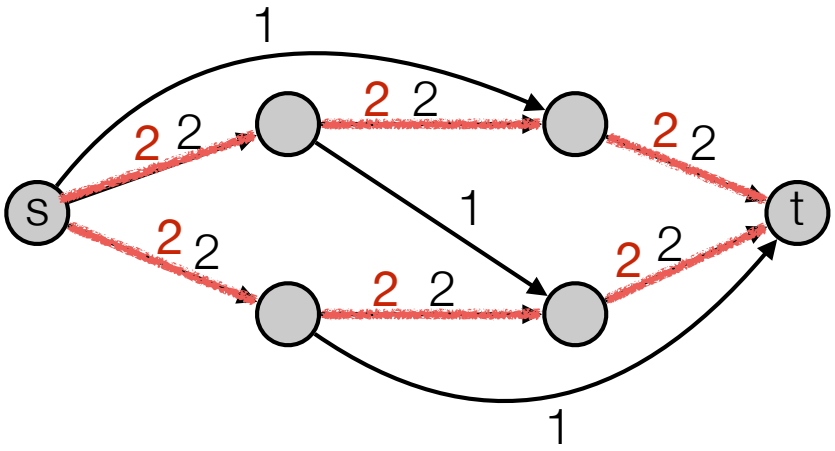
Algorithm

- Find path where we can send more flow.



Algorithm

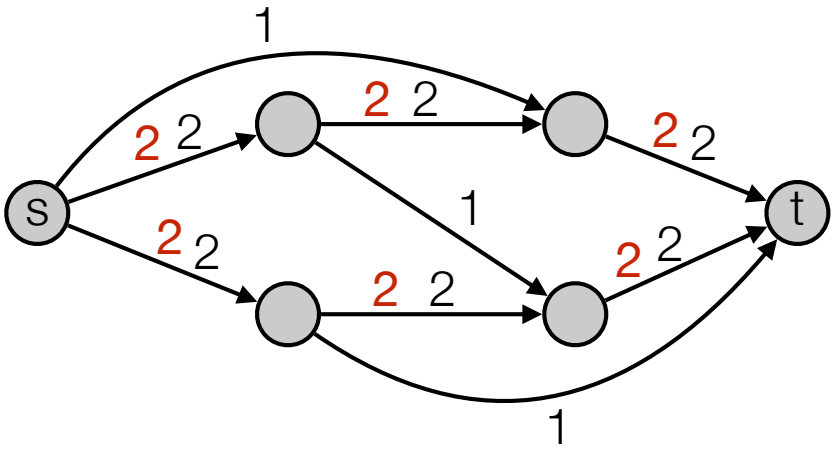
- Find path where we can send more flow.



Algorithm

- Find path where we can send more flow.

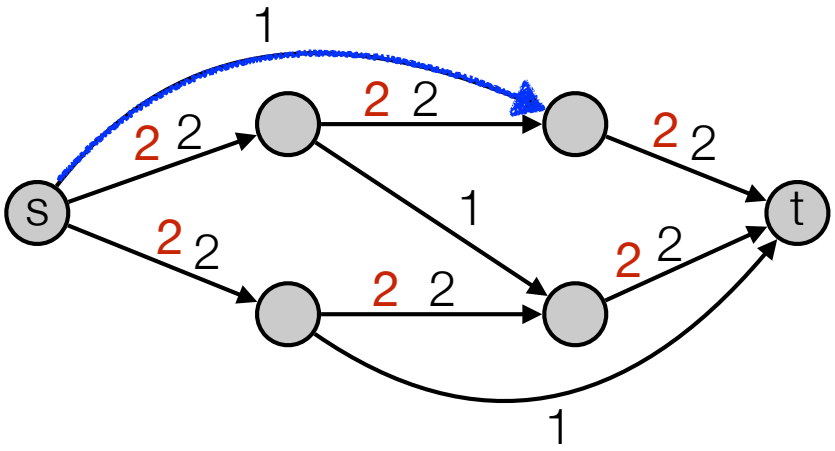
- Send flow back (cancel flow).



Algorithm

- Find path where we can send more flow.

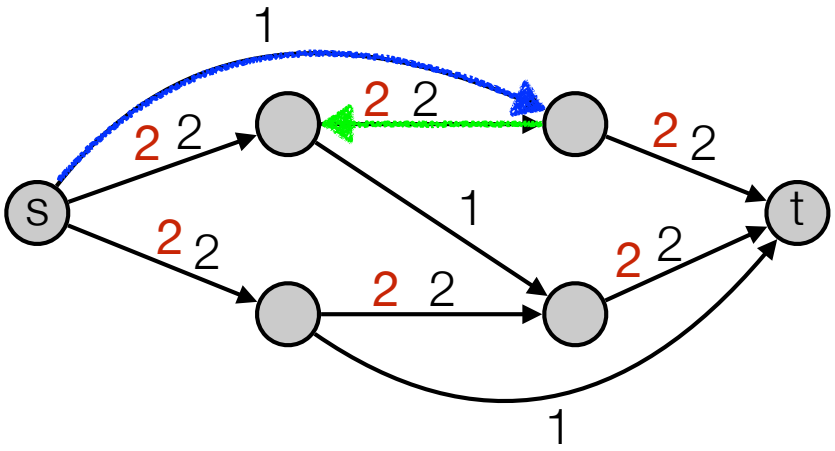
- Send flow back (cancel flow).



Algorithm

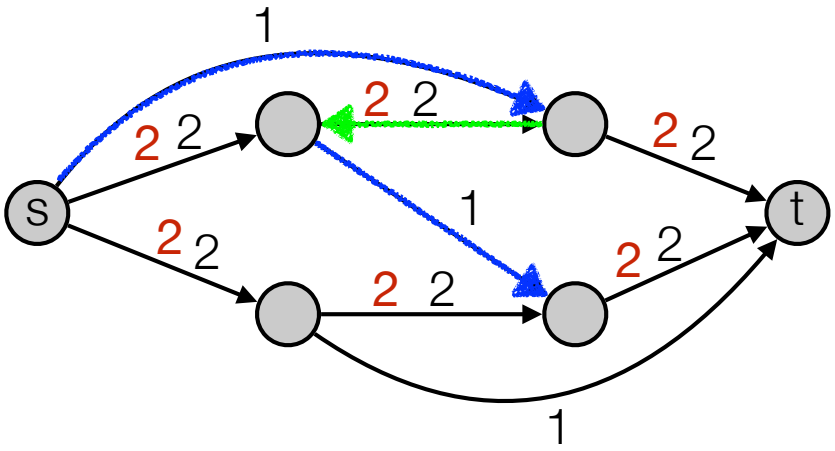
- Find path where we can send more flow.

- Send flow back (cancel flow).



Algorithm

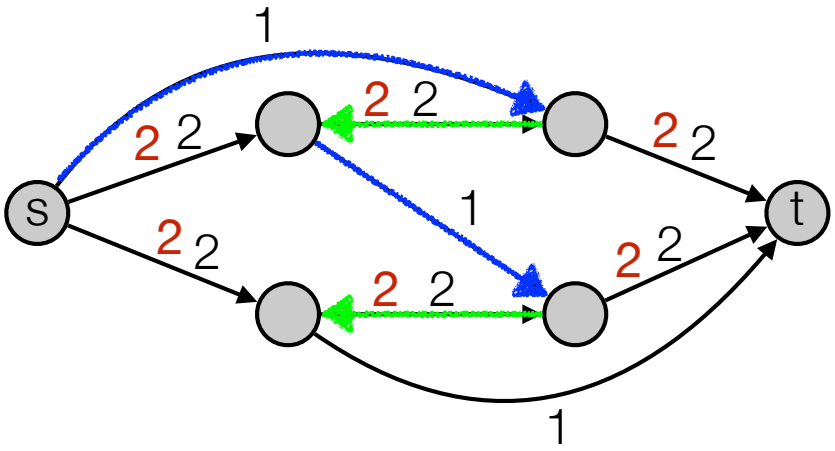
- Find path where we can send more flow.
- Send flow back (cancel flow).



Algorithm

- Find path where we can send more flow.

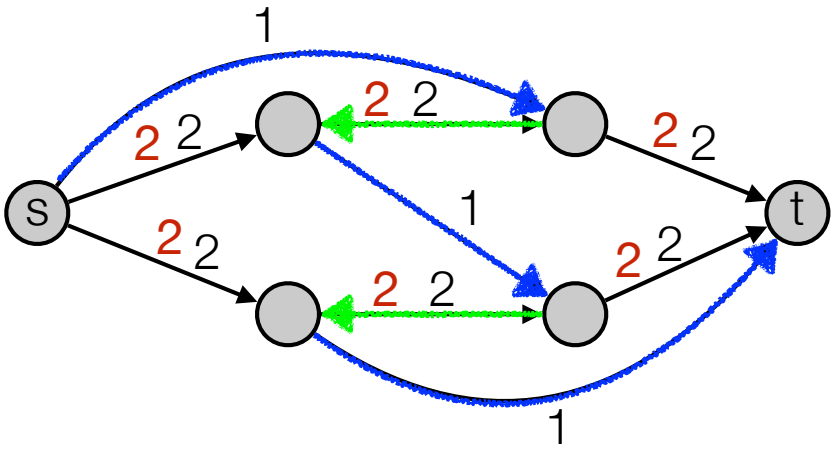
- Send flow back (cancel flow).



Algorithm

- Find path where we can send more flow.

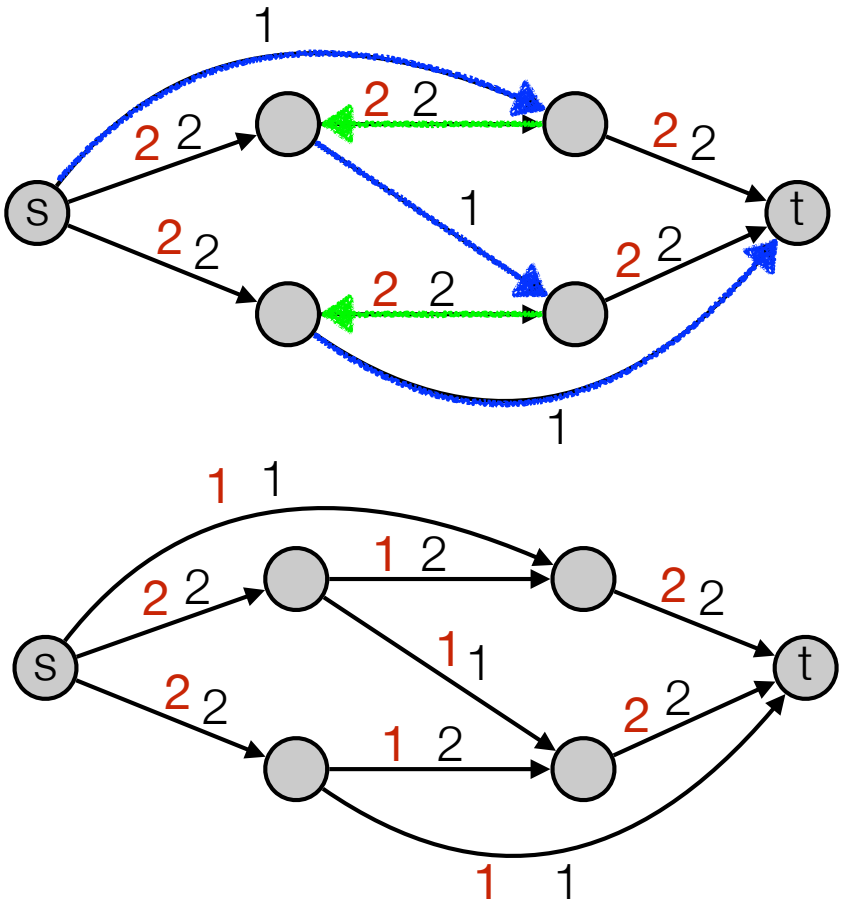
- Send flow back (cancel flow).



Algorithm

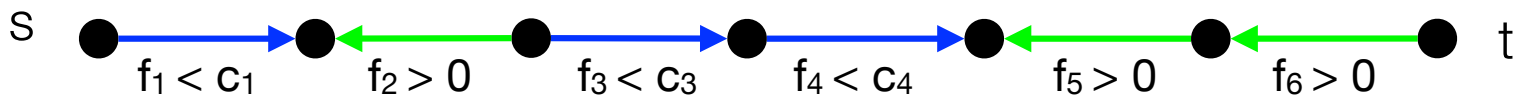
- Find path where we can send more flow.

- Send flow back (cancel flow).

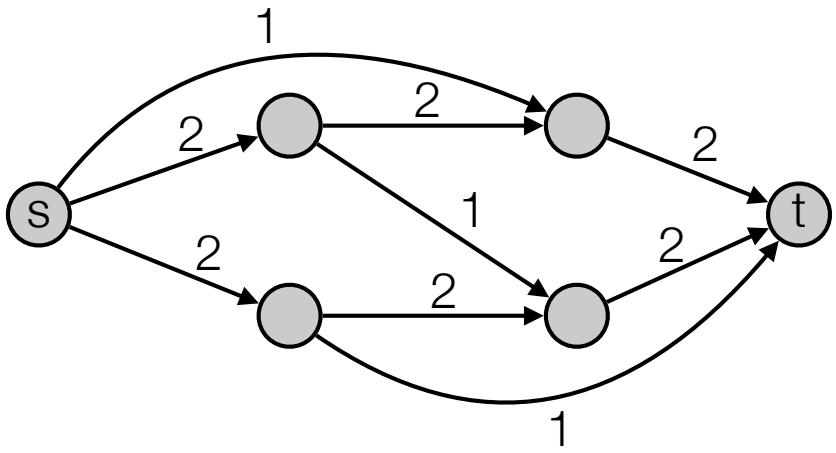


Augmenting Paths

- Augmenting path: s-t path P where
  - forward edges have leftover capacity
  - backwards edges have positive flow

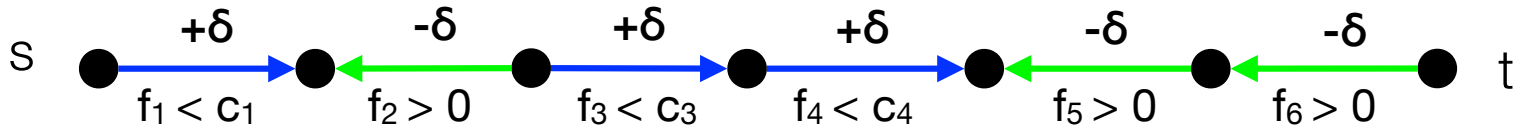


- Can add extra flow: $\min(c_1 - f_1, f_2, c_3 - f_3, c_4 - f_4, f_5, f_6) = \delta$ = bottleneck(P).

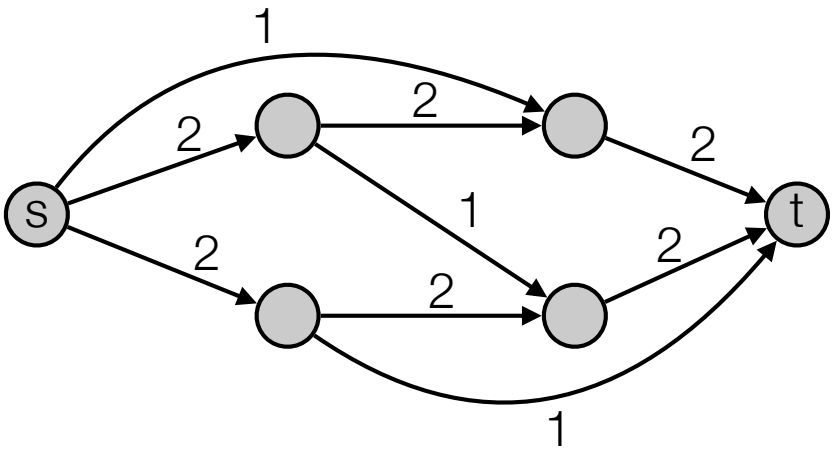


Augmenting Paths

- Augmenting path: s-t path P where
  - forward edges have leftover capacity
  - backwards edges have positive flow

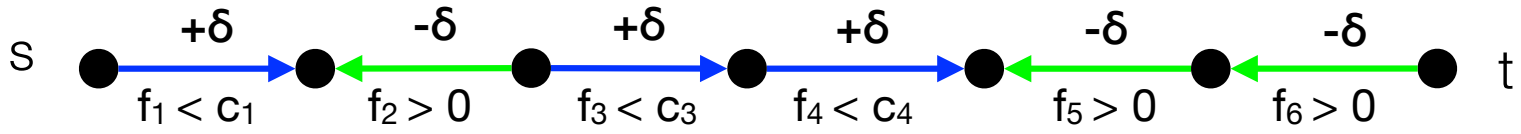


- Can add extra flow: $\min(c_1 - f_1, f_2, c_3 - f_3, c_4 - f_4, f_5, f_6) = \delta$ = bottleneck(P).

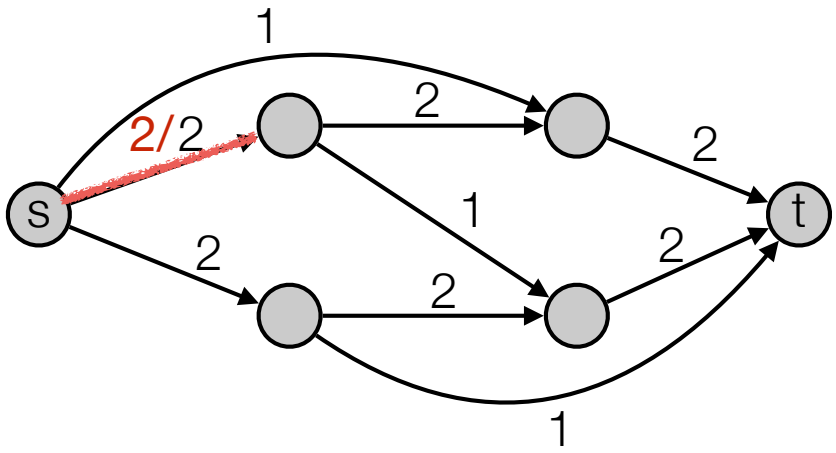


Augmenting Paths

- Augmenting path: s-t path P where
  - forward edges have leftover capacity
  - backwards edges have positive flow

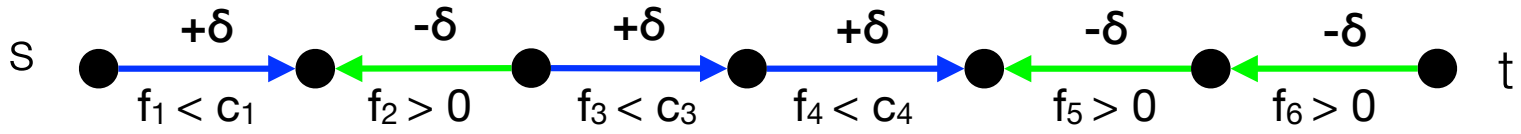


- Can add extra flow: $\min(c_1 - f_1, f_2, c_3 - f_3, c_4 - f_4, f_5, f_6) = \delta$ = bottleneck(P).

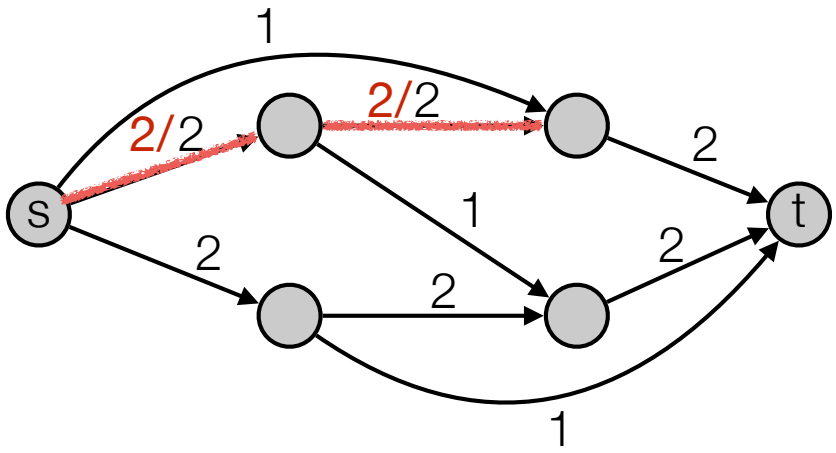


Augmenting Paths

- Augmenting path: s-t path P where
  - forward edges have leftover capacity
  - backwards edges have positive flow

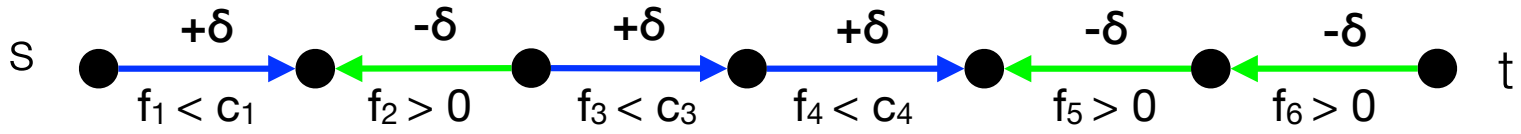


- Can add extra flow: $\min(c_1 - f_1, f_2, c_3 - f_3, c_4 - f_4, f_5, f_6) = \delta$ = bottleneck(P).

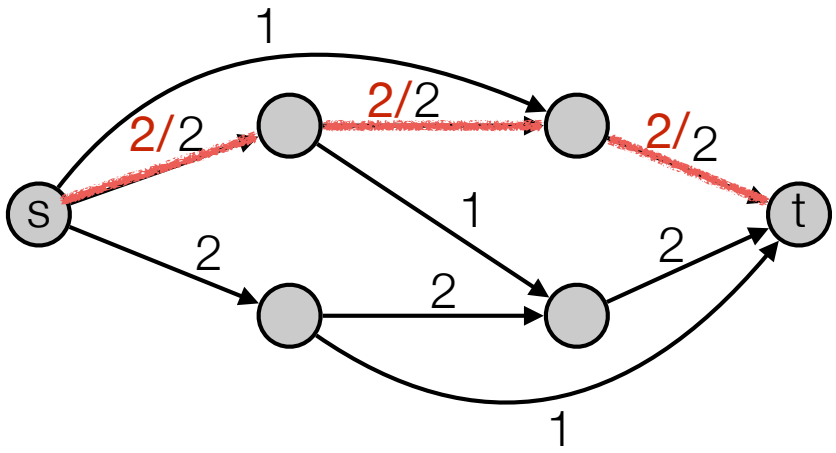


Augmenting Paths

- Augmenting path: s-t path P where
  - forward edges have leftover capacity
  - backwards edges have positive flow

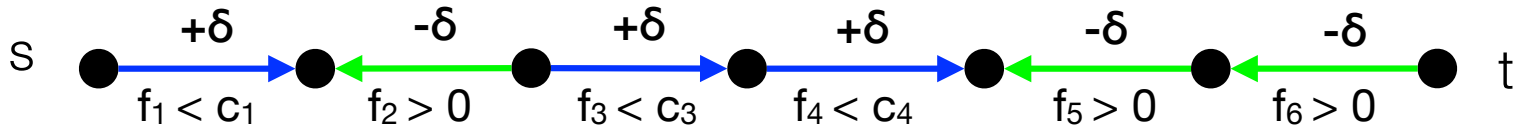


- Can add extra flow: $\min(c_1 - f_1, f_2, c_3 - f_3, c_4 - f_4, f_5, f_6) = \delta$ = bottleneck(P).

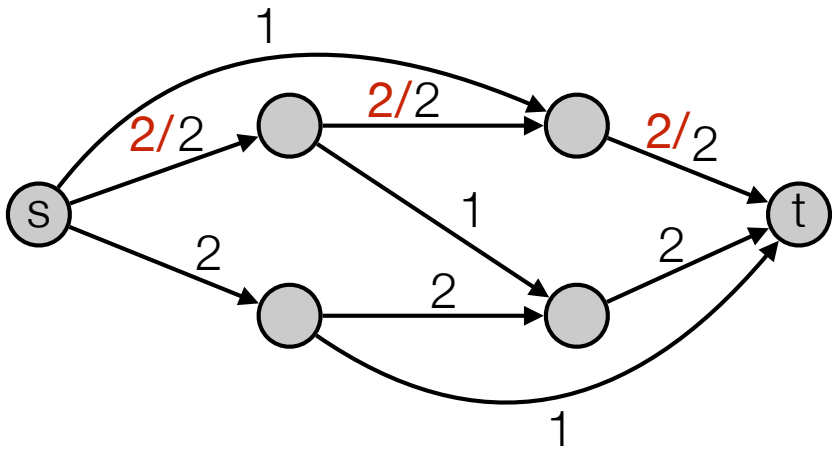


Augmenting Paths

- Augmenting path: s-t path P where
  - forward edges have leftover capacity
  - backwards edges have positive flow

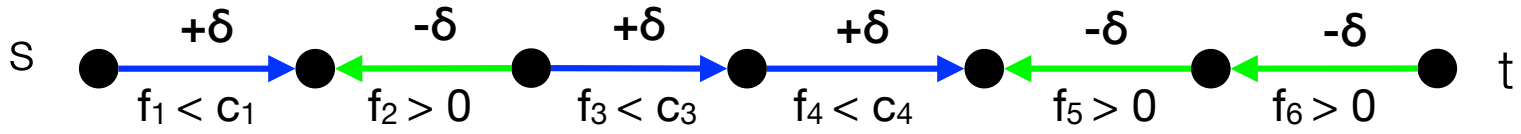


- Can add extra flow: $\min(c_1 - f_1, f_2, c_3 - f_3, c_4 - f_4, f_5, f_6) = \delta$ = bottleneck(P).

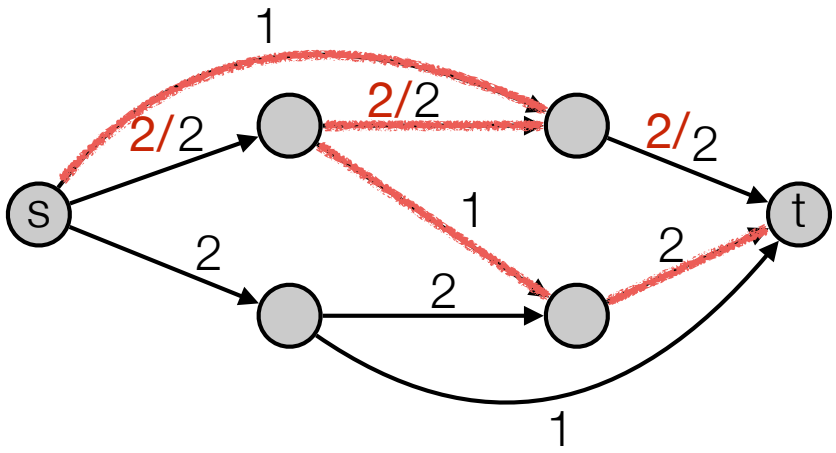


Augmenting Paths

- Augmenting path: s-t path P where
  - forward edges have leftover capacity
  - backwards edges have positive flow

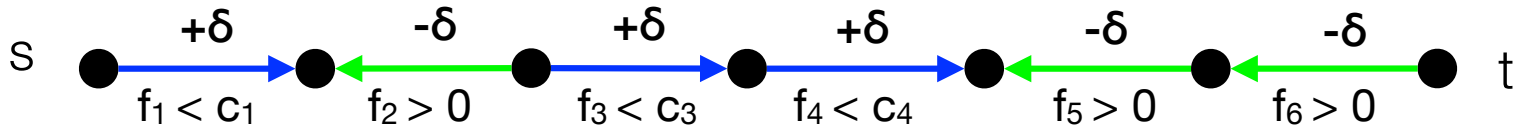


- Can add extra flow: $\min(c_1 - f_1, f_2, c_3 - f_3, c_4 - f_4, f_5, f_6) = \delta$ = bottleneck(P).

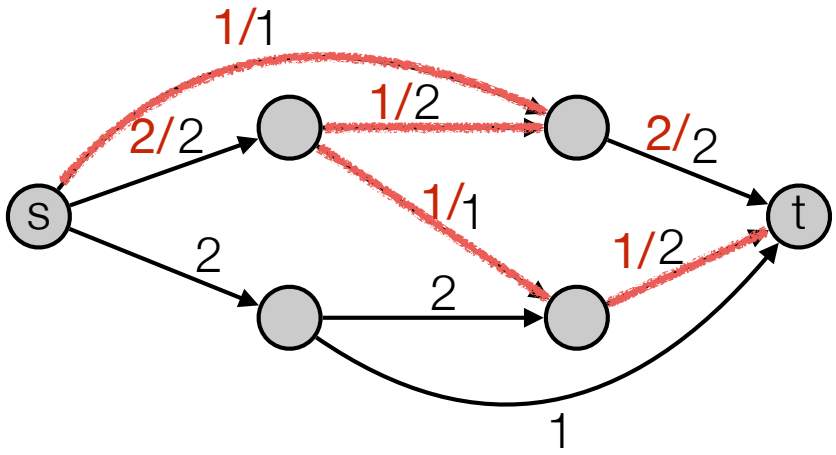


Augmenting Paths

- Augmenting path: s-t path P where
  - forward edges have leftover capacity
  - backwards edges have positive flow

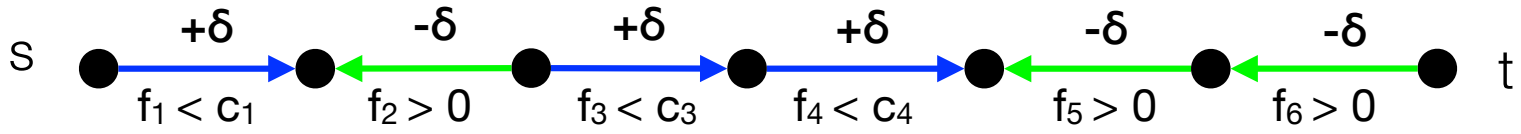


- Can add extra flow: $\min(c_1 - f_1, f_2, c_3 - f_3, c_4 - f_4, f_5, f_6) = \delta$ = bottleneck(P).

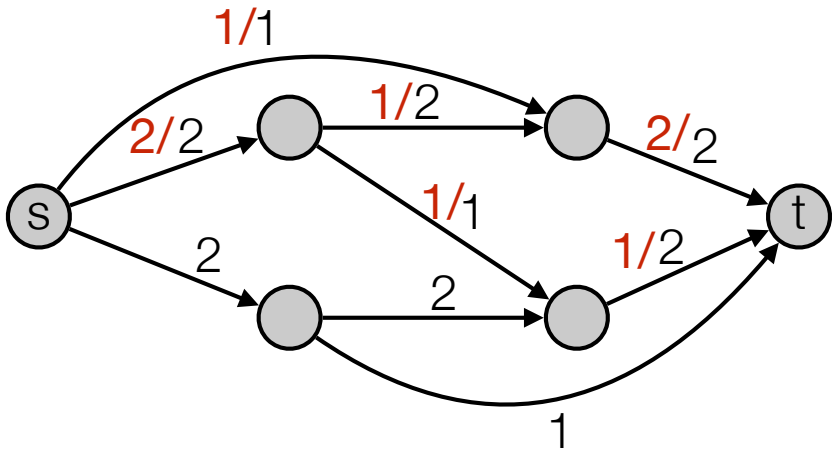


Augmenting Paths

- Augmenting path: s-t path P where
  - forward edges have leftover capacity
  - backwards edges have positive flow

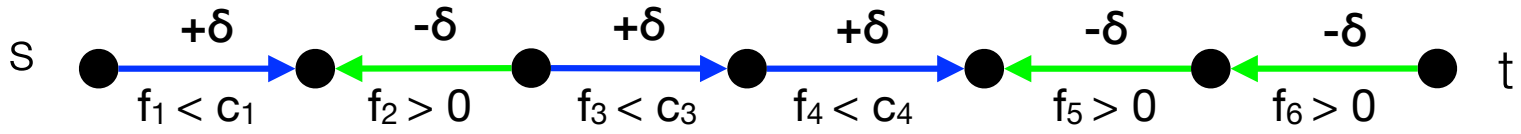


- Can add extra flow: $\min(c_1 - f_1, f_2, c_3 - f_3, c_4 - f_4, f_5, f_6) = \delta$ = bottleneck(P).

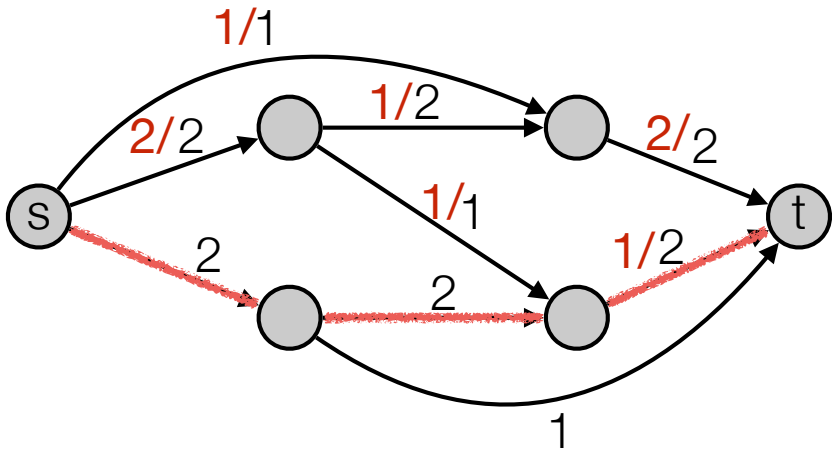


Augmenting Paths

- Augmenting path: s-t path P where
  - forward edges have leftover capacity
  - backwards edges have positive flow

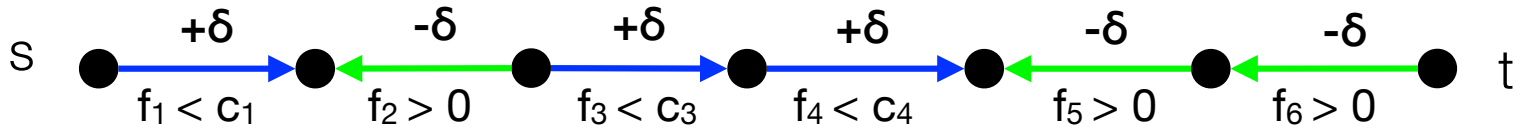


- Can add extra flow: $\min(c_1 - f_1, f_2, c_3 - f_3, c_4 - f_4, f_5, f_6) = \delta$ = bottleneck(P).

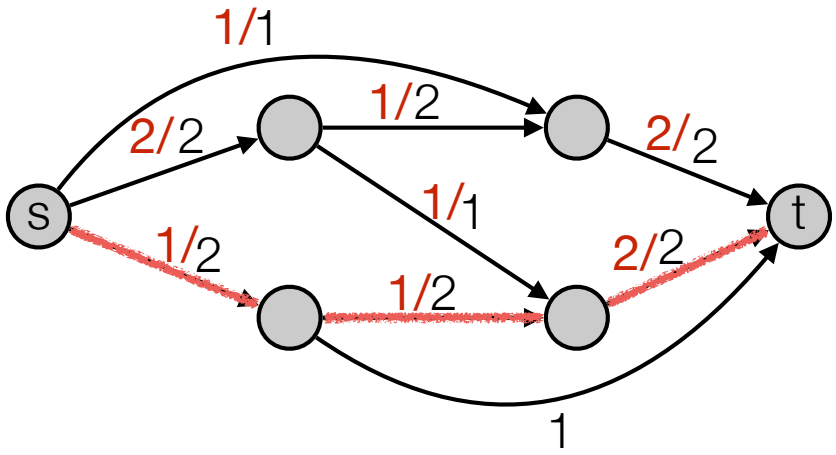


Augmenting Paths

- Augmenting path: s-t path P where
  - forward edges have leftover capacity
  - backwards edges have positive flow

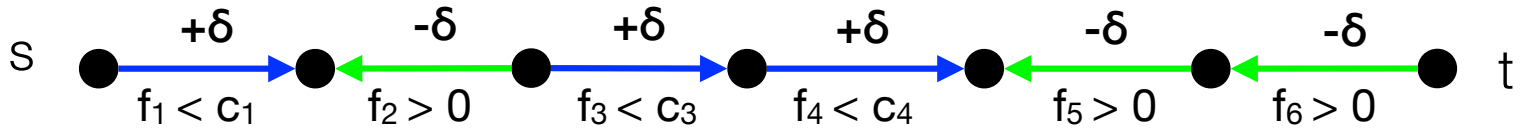


- Can add extra flow: $\min(c_1 - f_1, f_2, c_3 - f_3, c_4 - f_4, f_5, f_6) = \delta$ = bottleneck(P).

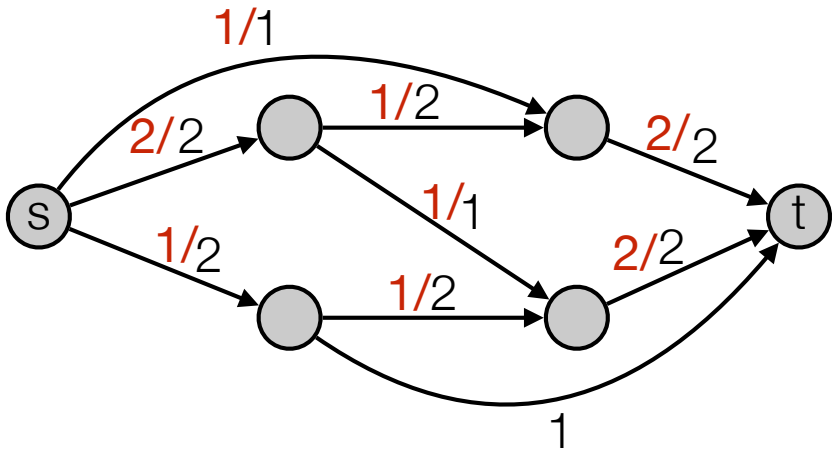


Augmenting Paths

- Augmenting path: s-t path P where
  - forward edges have leftover capacity
  - backwards edges have positive flow

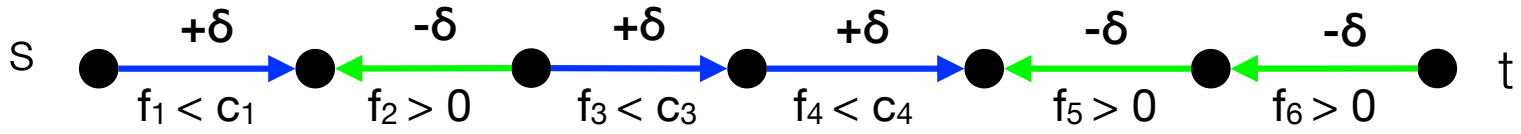


- Can add extra flow: $\min(c_1 - f_1, f_2, c_3 - f_3, c_4 - f_4, f_5, f_6) = \delta$ = bottleneck(P).

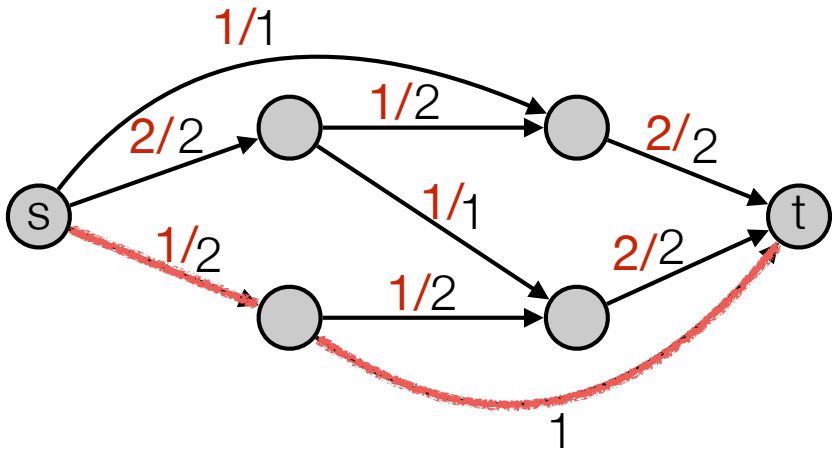


Augmenting Paths

- Augmenting path: s-t path P where
  - forward edges have leftover capacity
  - backwards edges have positive flow

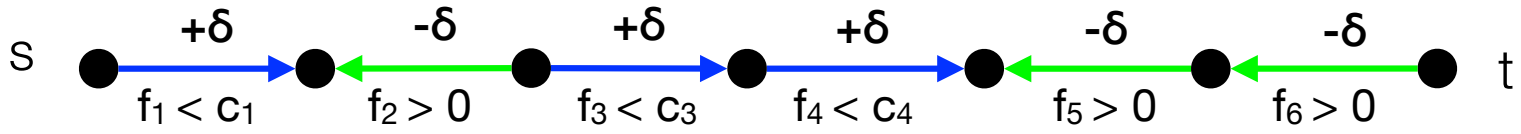


- Can add extra flow: $\min(c_1 - f_1, f_2, c_3 - f_3, c_4 - f_4, f_5, f_6) = \delta$ = bottleneck(P).

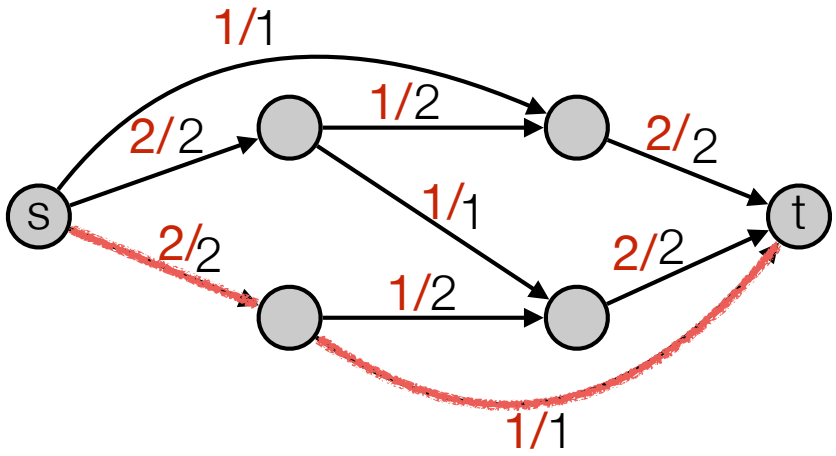


Augmenting Paths

- Augmenting path: s-t path P where
  - forward edges have leftover capacity
  - backwards edges have positive flow

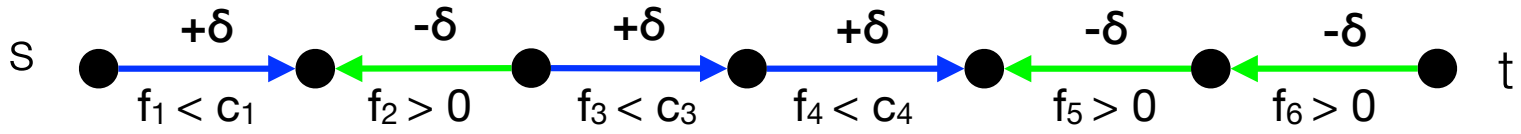


- Can add extra flow: $\min(c_1 - f_1, f_2, c_3 - f_3, c_4 - f_4, f_5, f_6) = \delta$ = bottleneck(P).

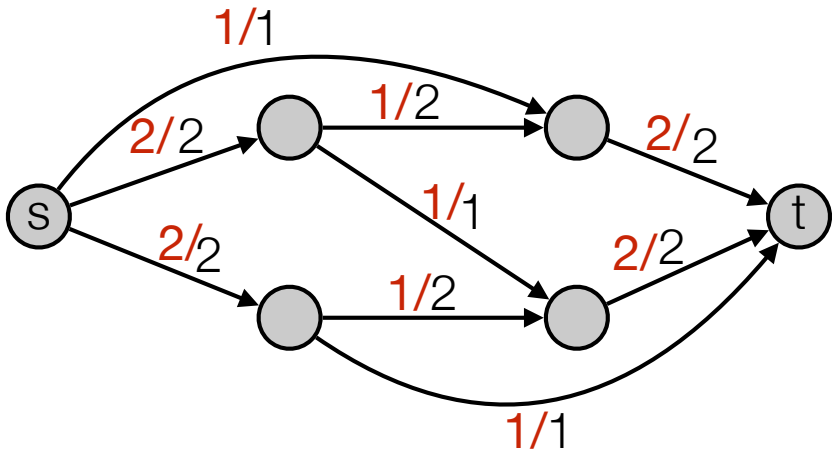


Augmenting Paths

- Augmenting path: s-t path P where
  - forward edges have leftover capacity
  - backwards edges have positive flow

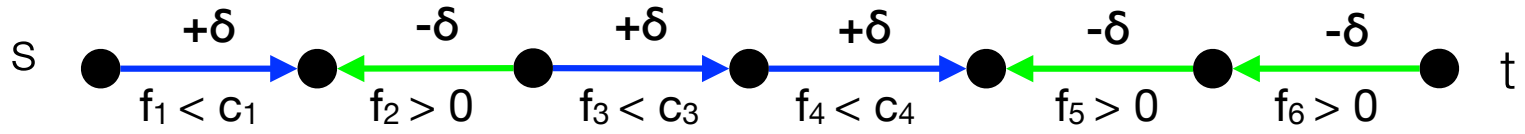


- Can add extra flow: $\min(c_1 - f_1, f_2, c_3 - f_3, c_4 - f_4, f_5, f_6) = \delta$ = bottleneck(P).



Augmenting Paths

- Augmenting path (definition different than in CLRS): s-t path where
  - forward edges have leftover capacity
  - backwards edges have positive flow

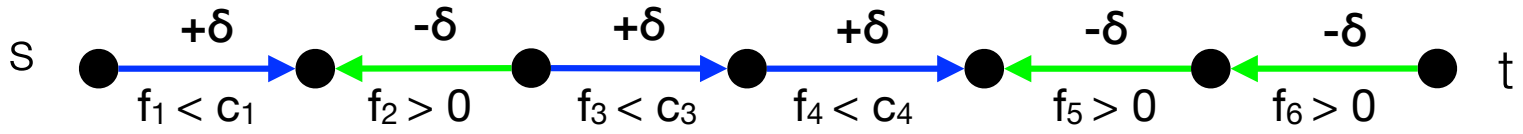


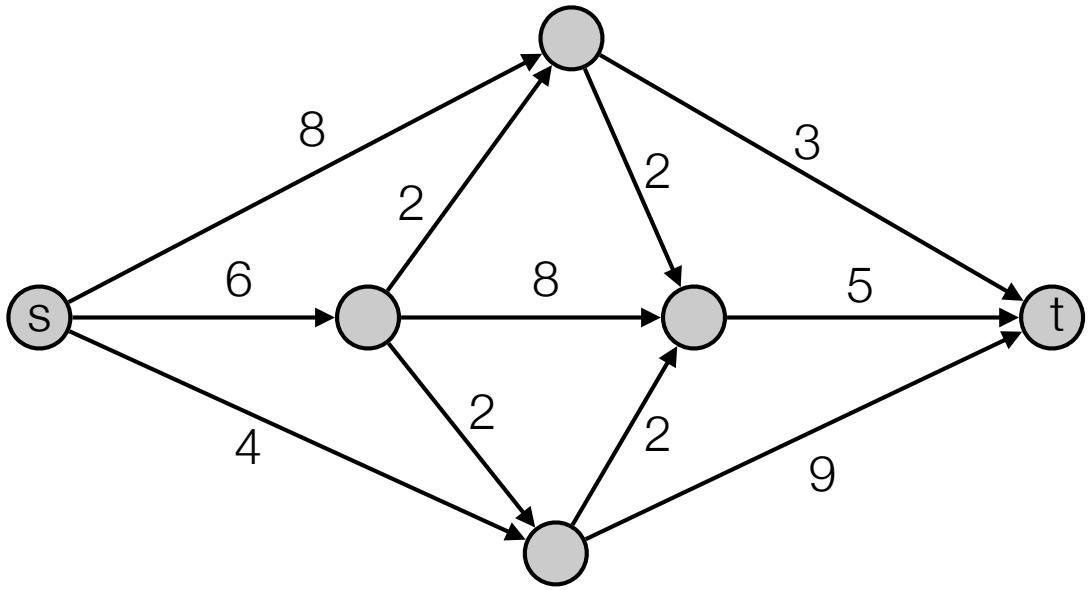
  - Can add extra flow: $\min(c_1 - f_1, f_2, c_3 - f_3, c_4 - f_4, f_5, f_6) = \delta = \text{bottleneck}(P)$.

- Ford-Fulkerson:
  - Find augmenting path, use it
  - Find augmenting path, use it
  - Find augmenting path, use it
  - ……

Ford Fulkerson

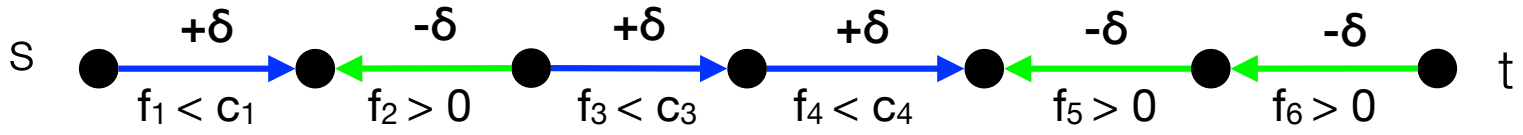
- Augmenting path: s-t path P where
  - forward edges have leftover capacity
  - backwards edges have positive flow

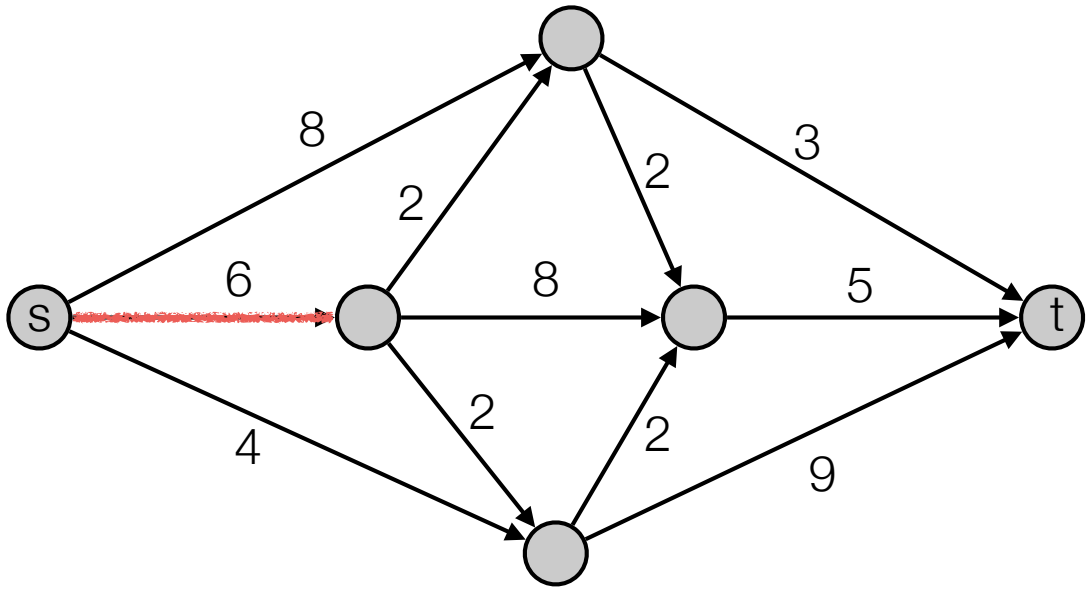




Ford Fulkerson

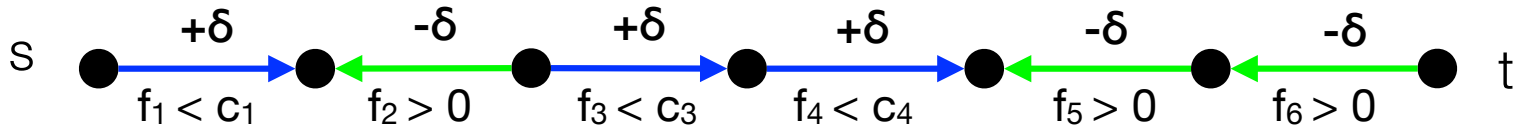
- Augmenting path: s-t path P where
  - forward edges have leftover capacity
  - backwards edges have positive flow

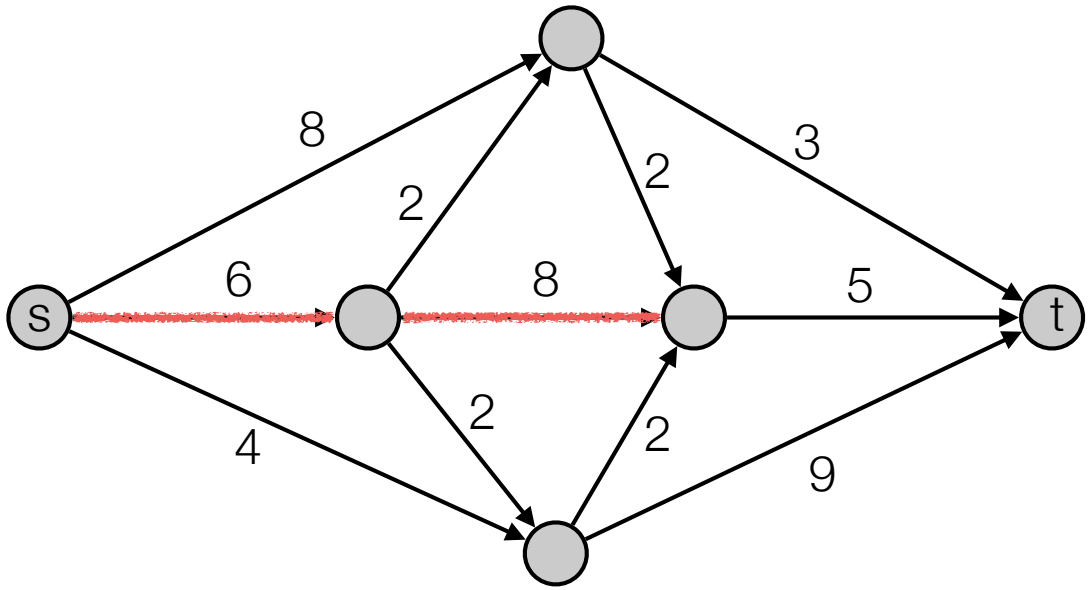




Ford Fulkerson

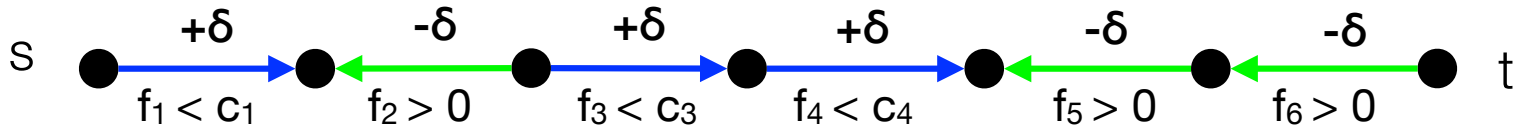
- Augmenting path: s-t path P where
  - forward edges have leftover capacity
  - backwards edges have positive flow

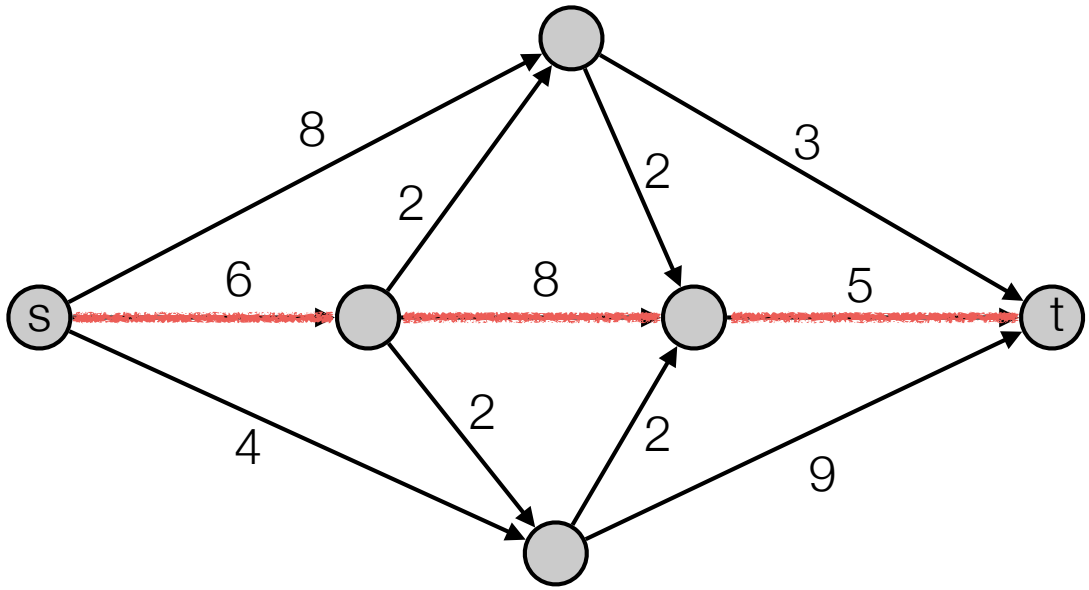




Ford Fulkerson

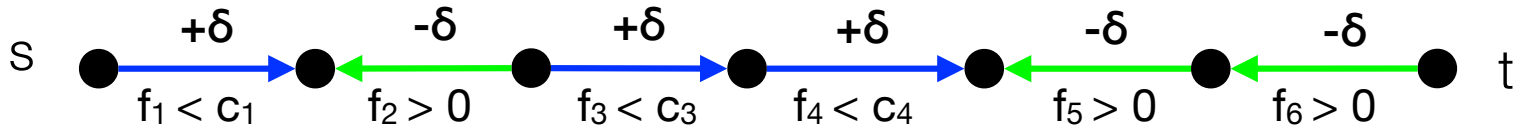
- Augmenting path: s-t path P where
  - forward edges have leftover capacity
  - backwards edges have positive flow

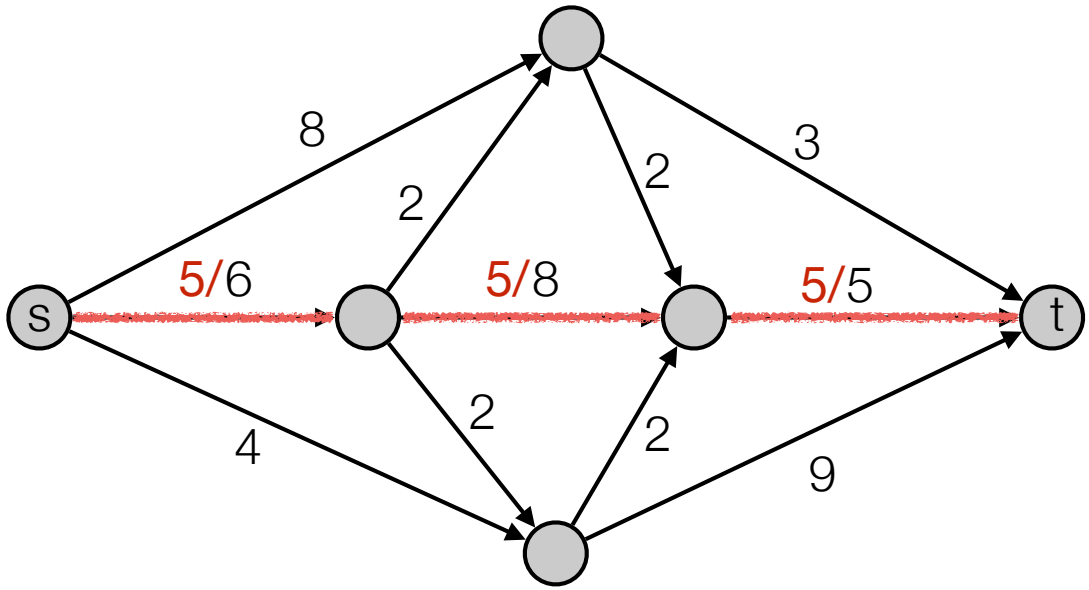




Ford Fulkerson

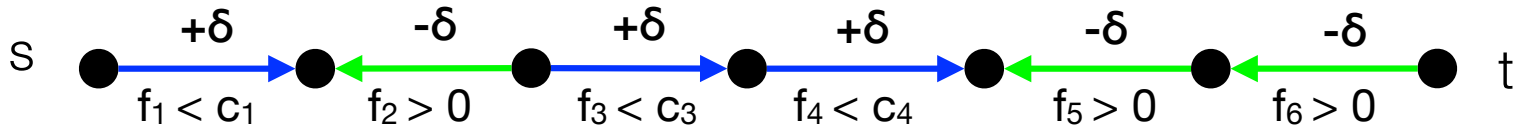
- Augmenting path: s-t path P where
  - forward edges have leftover capacity
  - backwards edges have positive flow

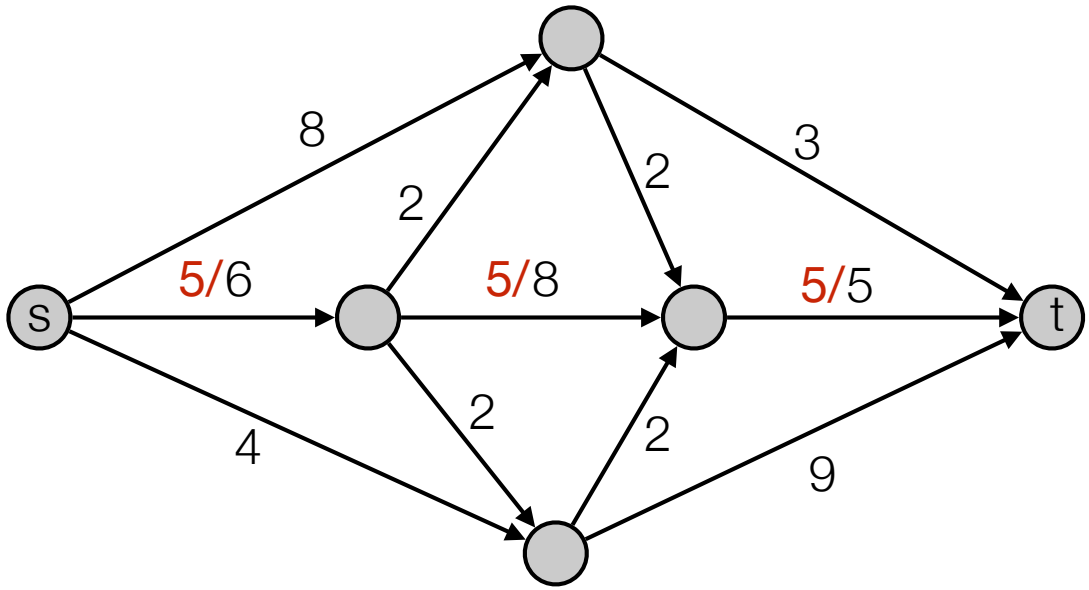




Ford Fulkerson

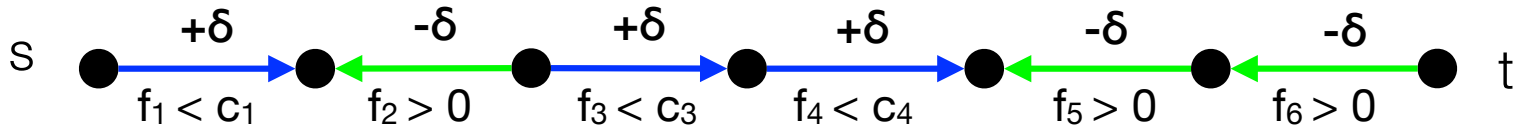
- Augmenting path: s-t path P where
  - forward edges have leftover capacity
  - backwards edges have positive flow

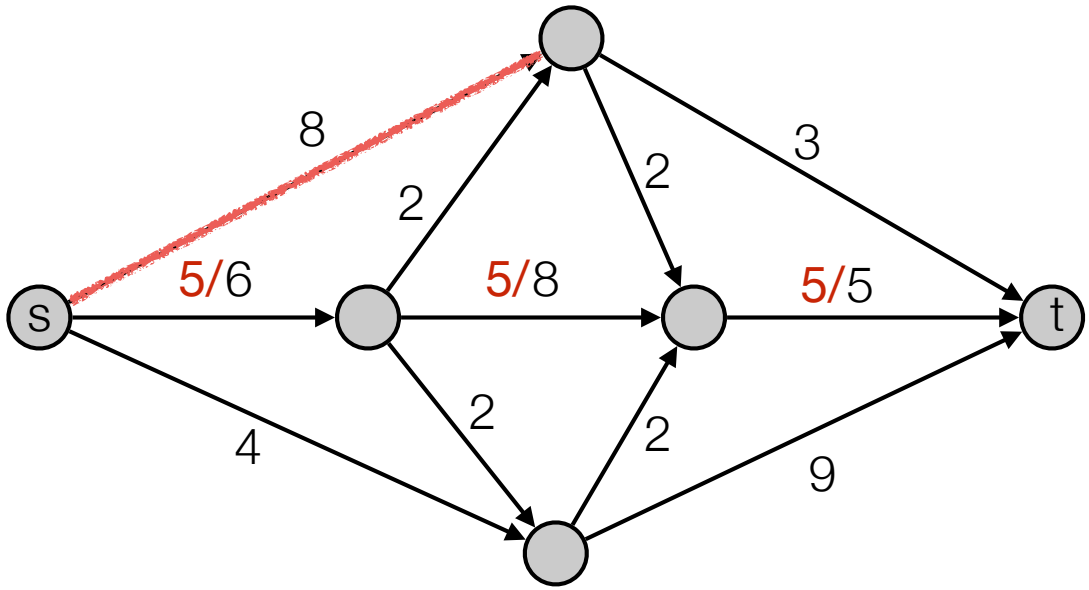




Ford Fulkerson

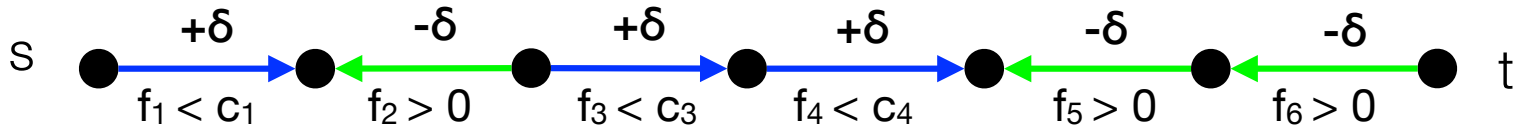
- Augmenting path: s-t path P where
  - forward edges have leftover capacity
  - backwards edges have positive flow

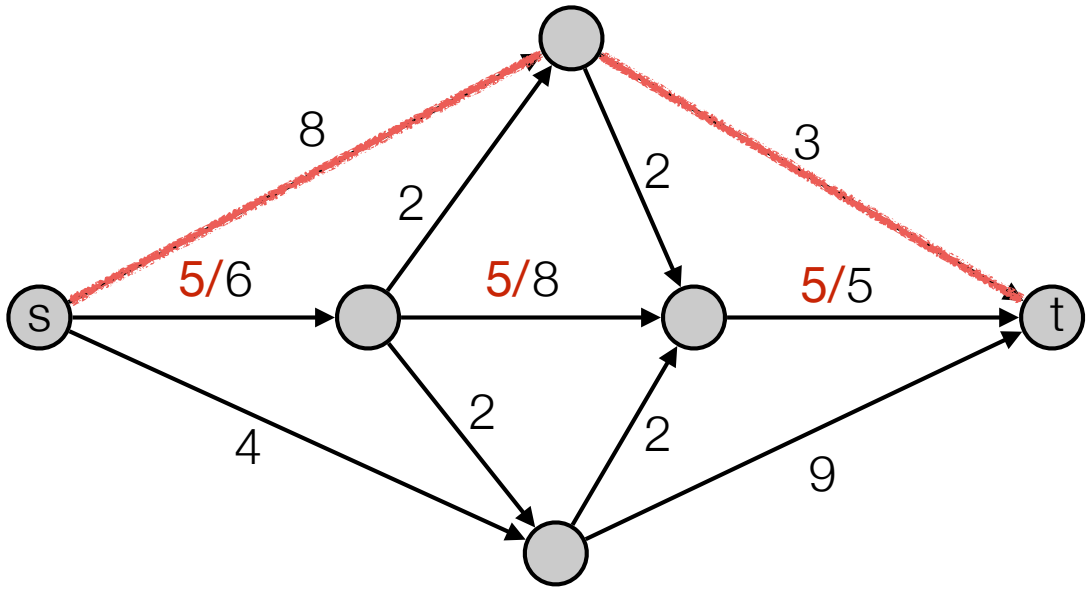




Ford Fulkerson

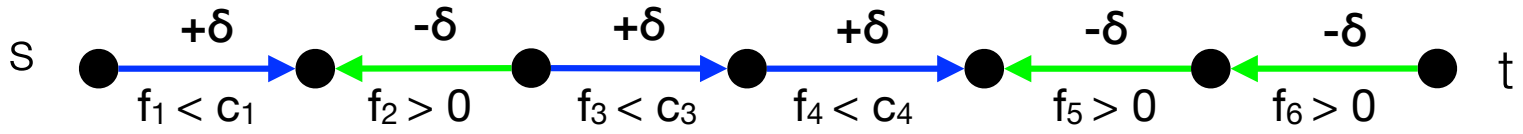
- Augmenting path: s-t path P where
  - forward edges have leftover capacity
  - backwards edges have positive flow

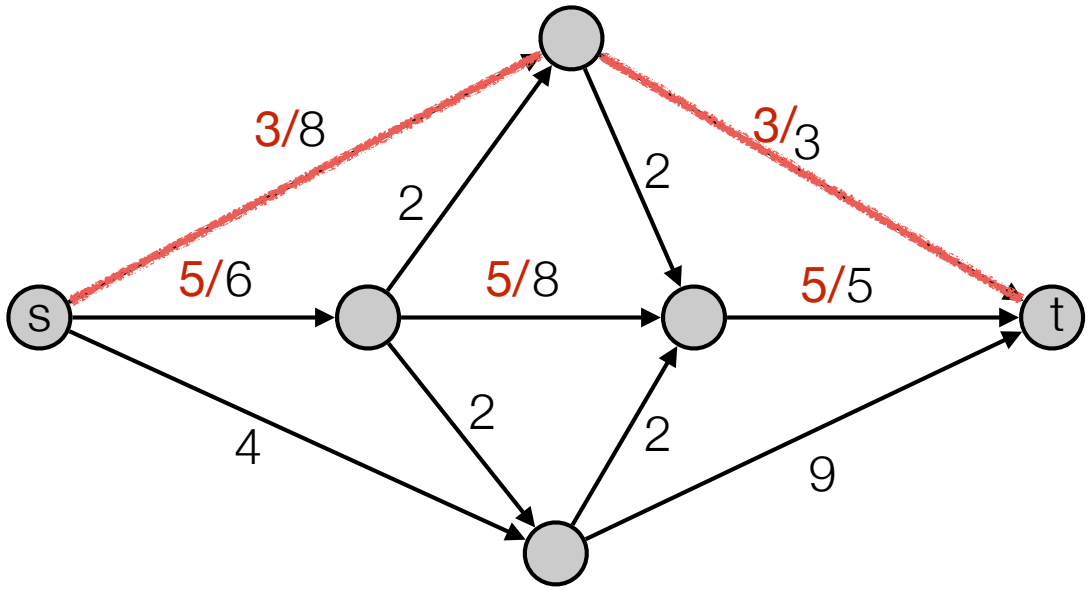




Ford Fulkerson

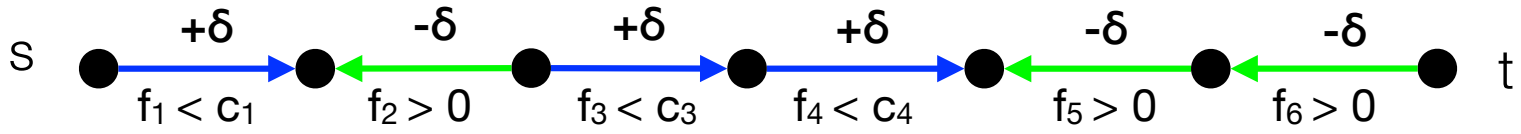
- Augmenting path: s-t path P where
  - forward edges have leftover capacity
  - backwards edges have positive flow

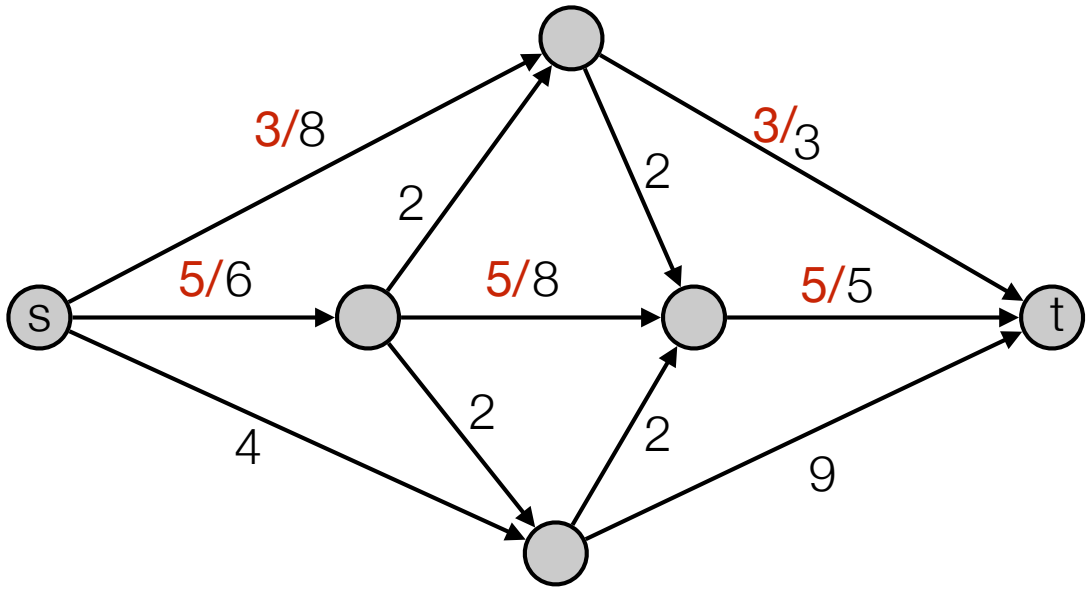




Ford Fulkerson

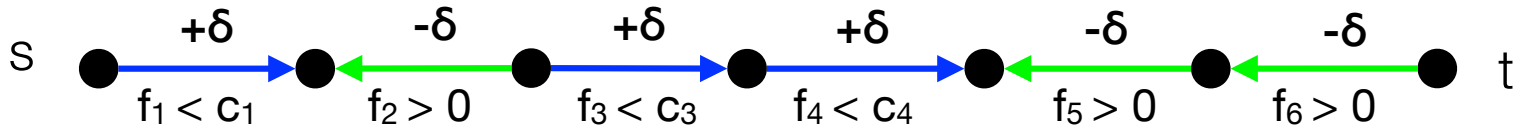
- Augmenting path: s-t path P where
  - forward edges have leftover capacity
  - backwards edges have positive flow

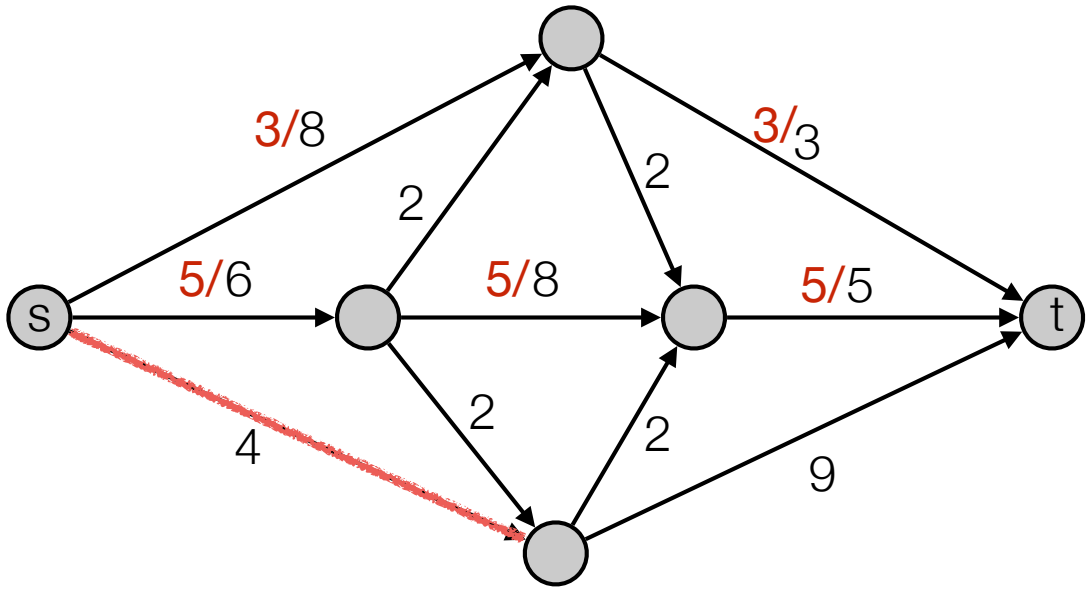




Ford Fulkerson

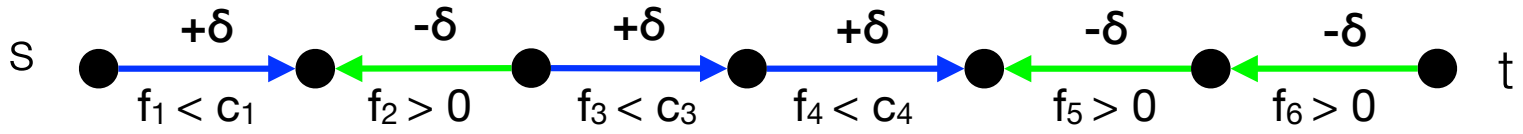
- Augmenting path: s-t path P where
  - forward edges have leftover capacity
  - backwards edges have positive flow

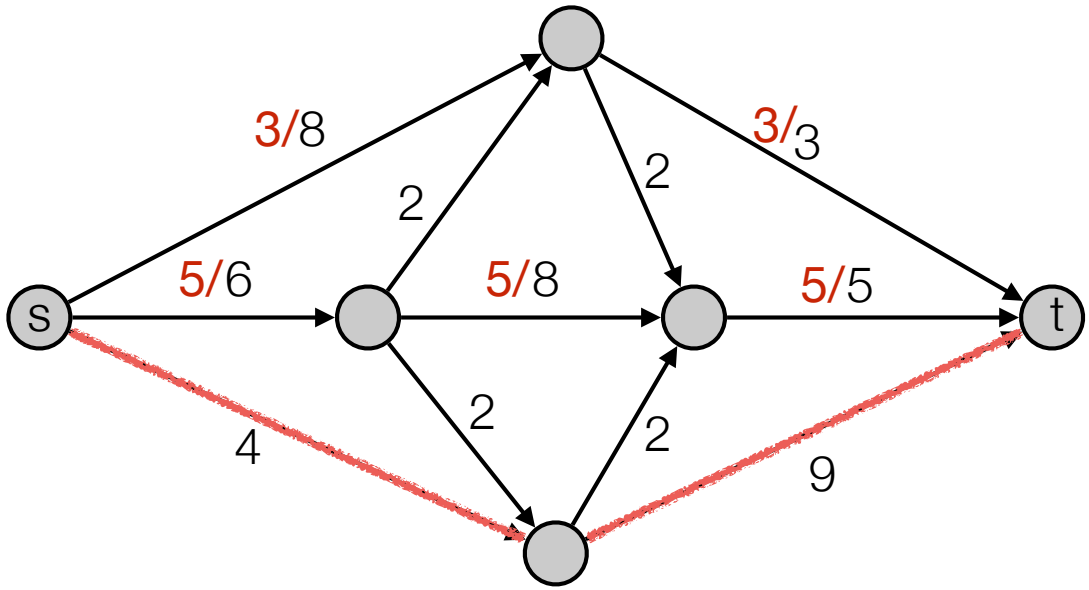




Ford Fulkerson

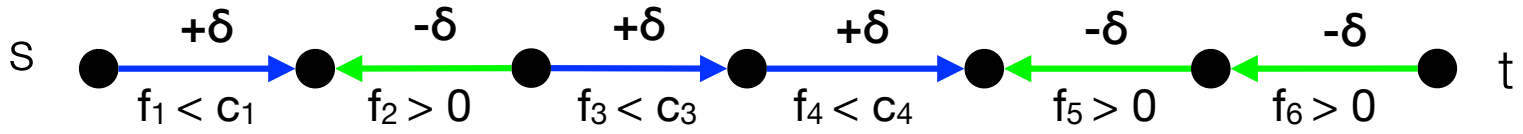
- Augmenting path: s-t path P where
  - forward edges have leftover capacity
  - backwards edges have positive flow

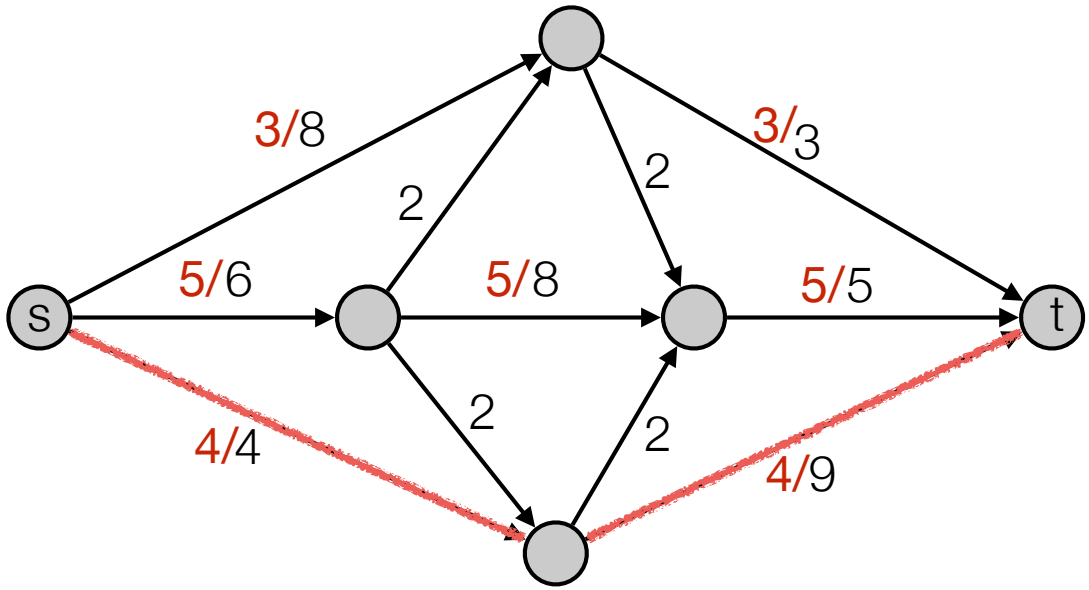




Ford Fulkerson

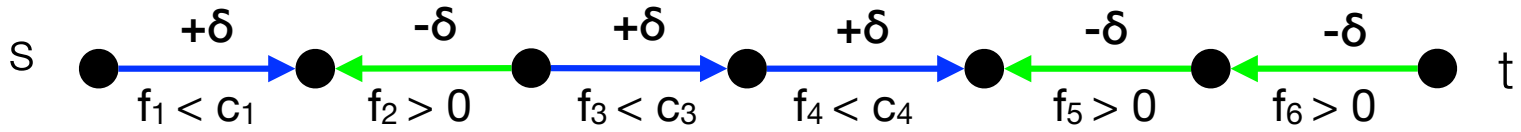
- Augmenting path: s-t path P where
  - forward edges have leftover capacity
  - backwards edges have positive flow

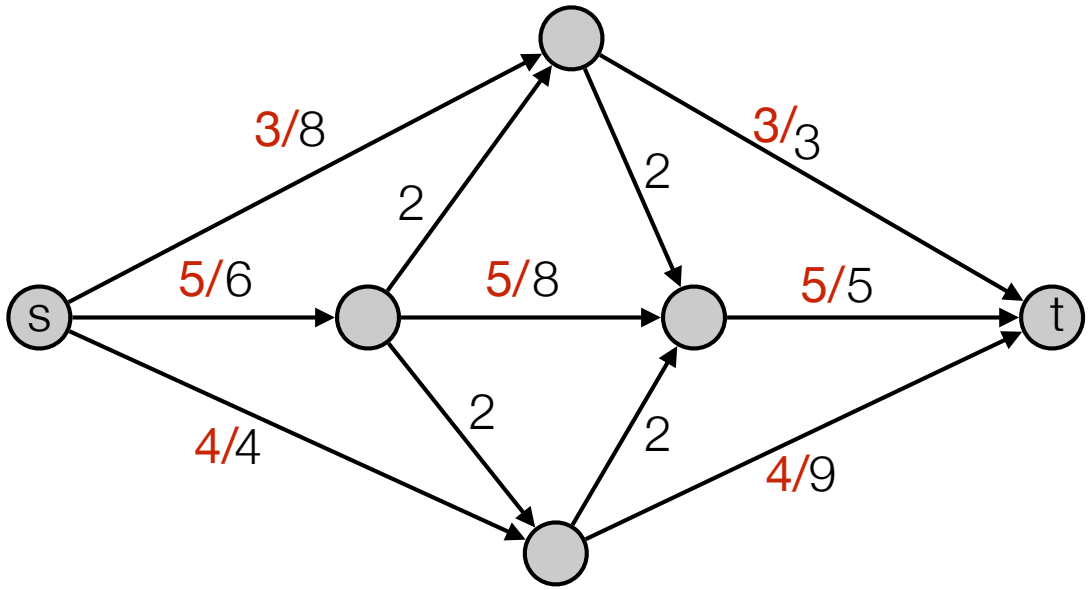




Ford Fulkerson

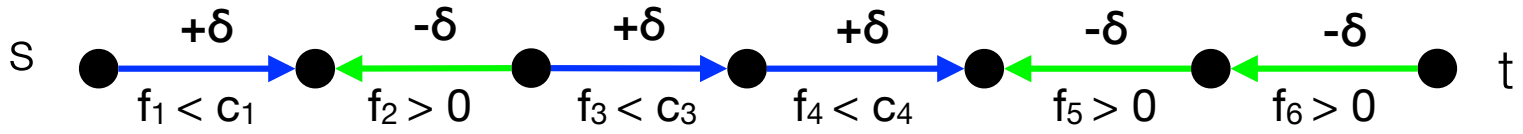
- Augmenting path: s-t path P where
  - forward edges have leftover capacity
  - backwards edges have positive flow

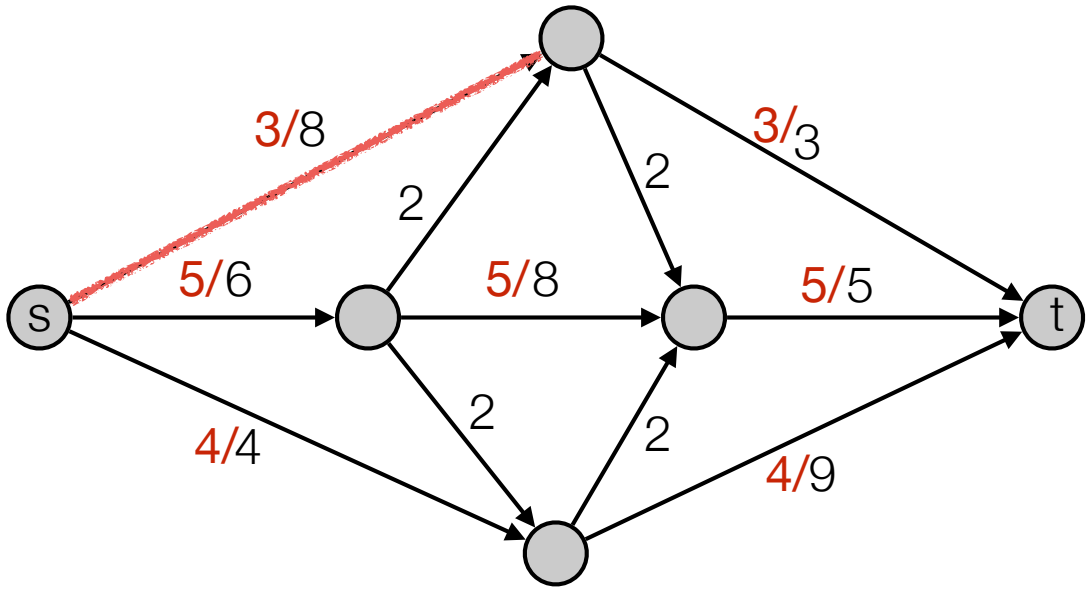




Ford Fulkerson

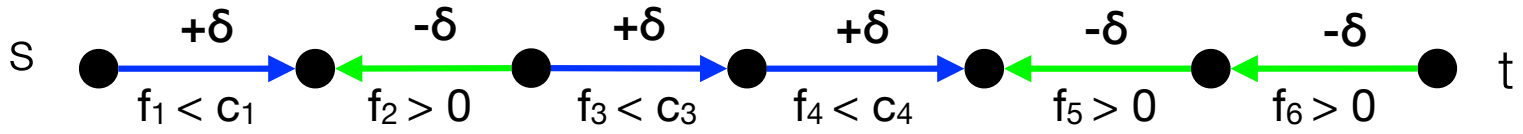
- Augmenting path: s-t path P where
  - forward edges have leftover capacity
  - backwards edges have positive flow

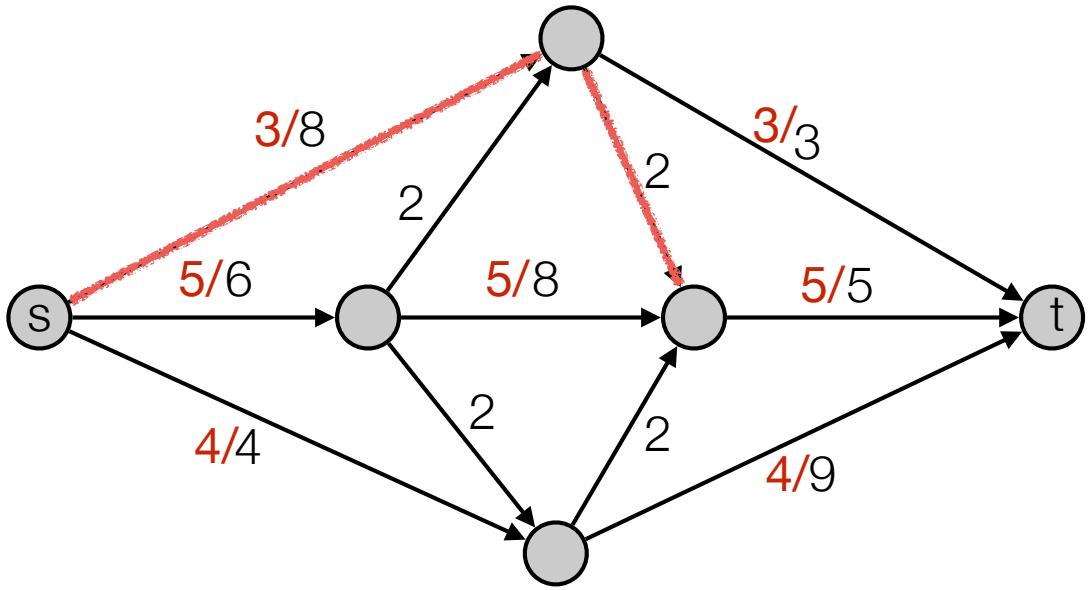




Ford Fulkerson

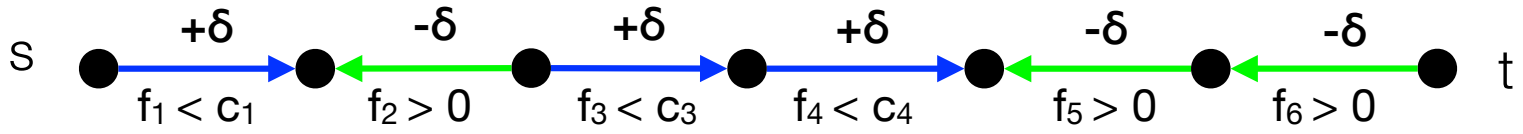
- Augmenting path: s-t path P where
  - forward edges have leftover capacity
  - backwards edges have positive flow

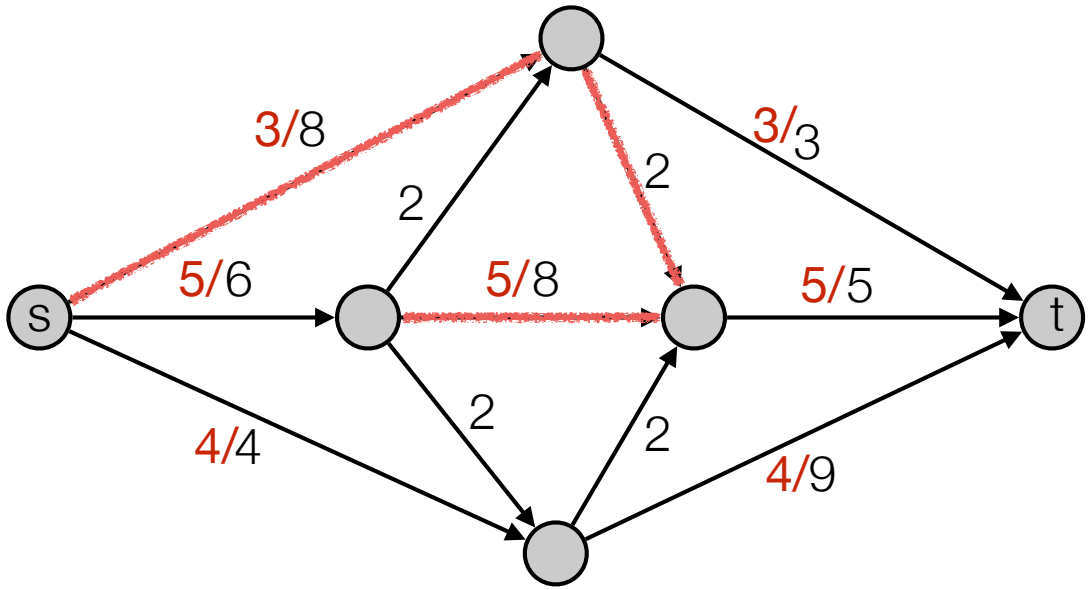




Ford Fulkerson

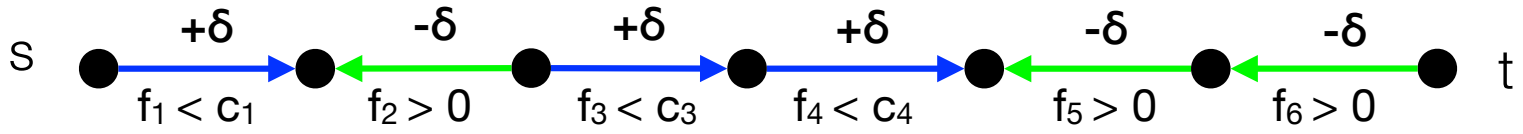
- Augmenting path: s-t path P where
  - forward edges have leftover capacity
  - backwards edges have positive flow

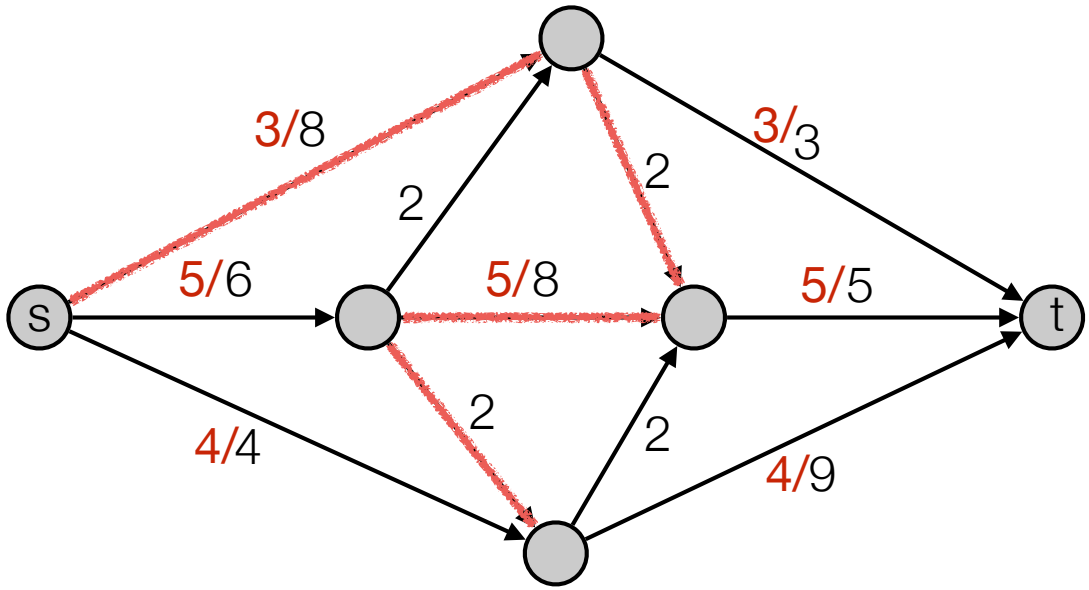




Ford Fulkerson

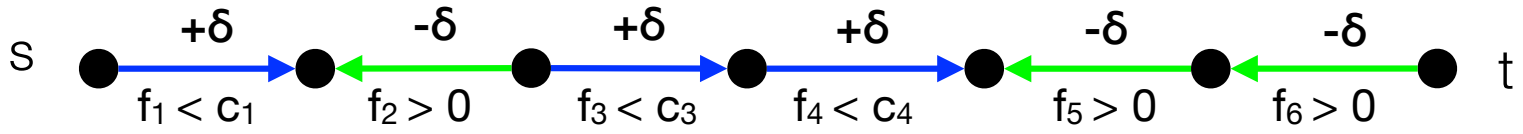
- Augmenting path: s-t path P where
  - forward edges have leftover capacity
  - backwards edges have positive flow

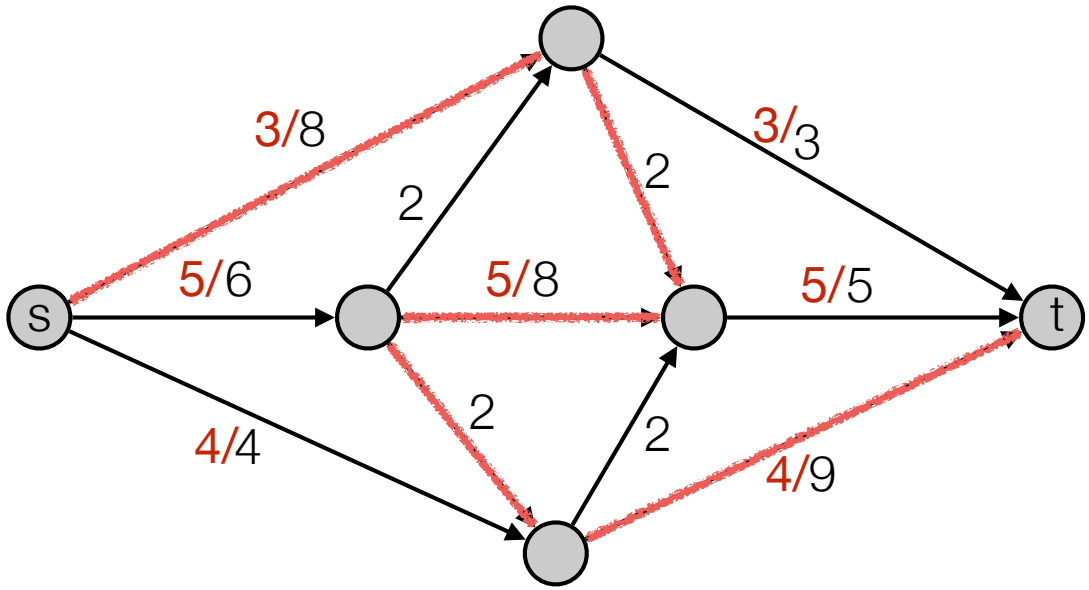




Ford Fulkerson

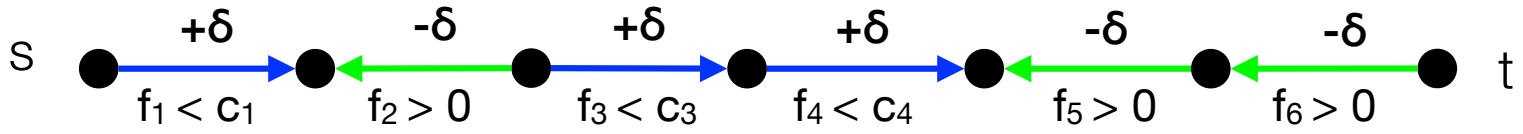
- Augmenting path: s-t path P where
  - forward edges have leftover capacity
  - backwards edges have positive flow

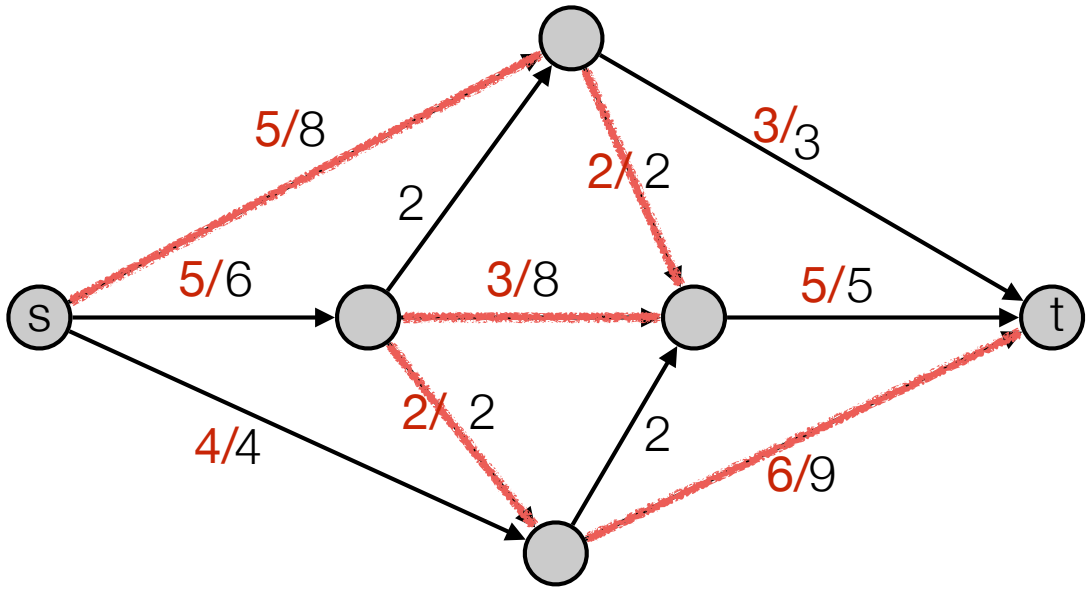




Ford Fulkerson

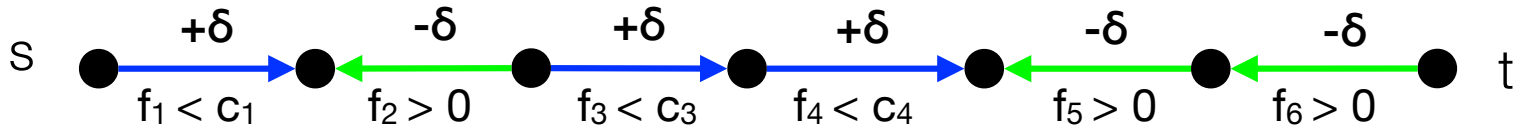
- Augmenting path: s-t path P where
  - forward edges have leftover capacity
  - backwards edges have positive flow

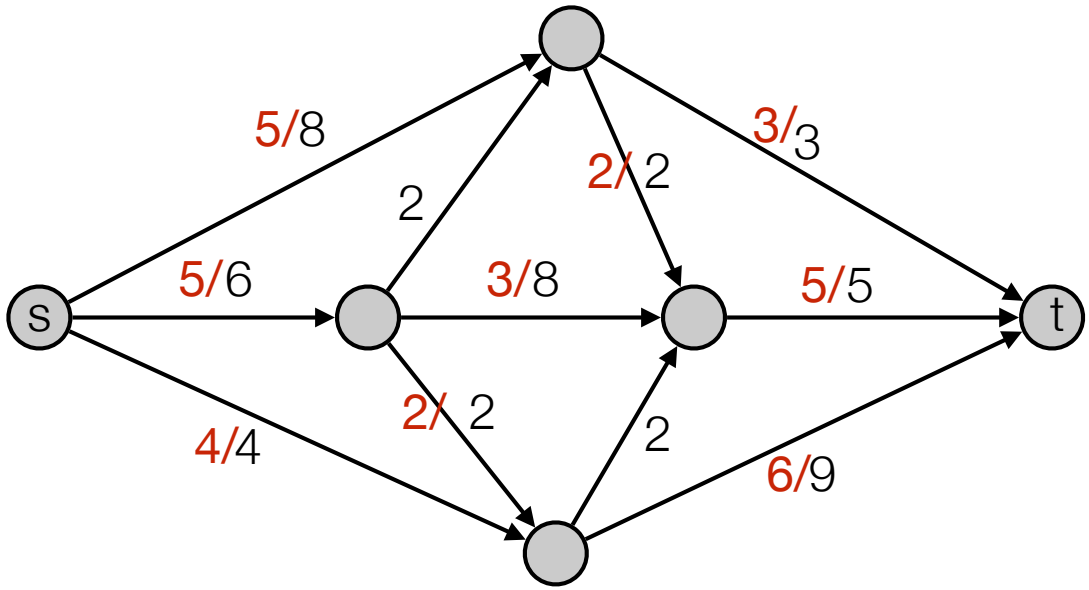




Ford Fulkerson

- Augmenting path: s-t path P where
  - forward edges have leftover capacity
  - backwards edges have positive flow

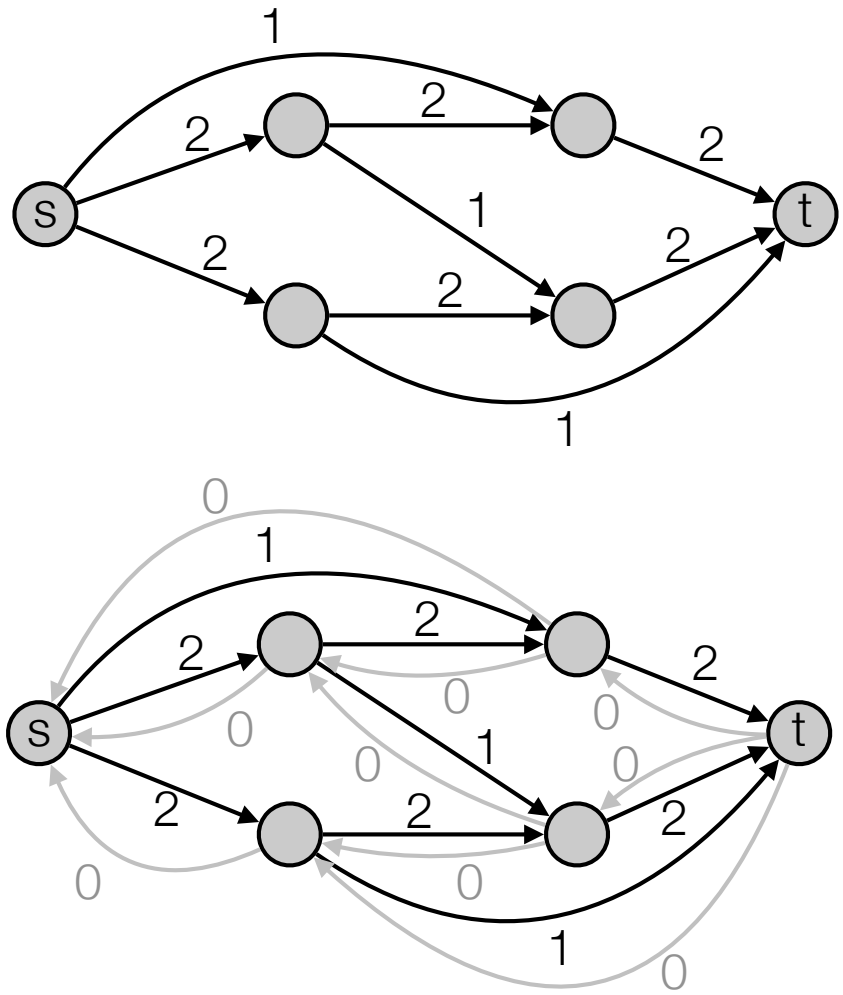




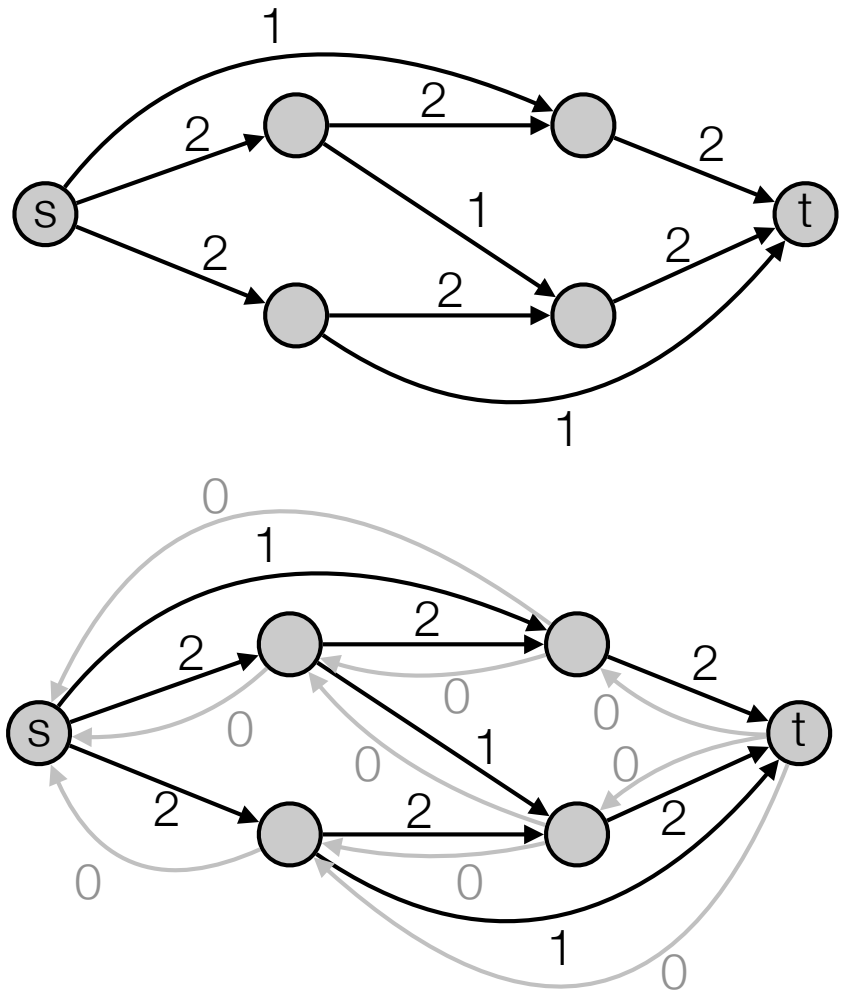
Analysis of Ford-Fulkerson

- Integral capacities implies theres is a maximum flow where all flow values f(e) are integers.
- Number of iterations:
  - Always increment flow by at least 1: #iterations ≤ max flow value f*
- Time for one iteration:
  - Can find augmenting path in linear time: One iteration takes O(m) time.
- Total running time = O(|f*| m).

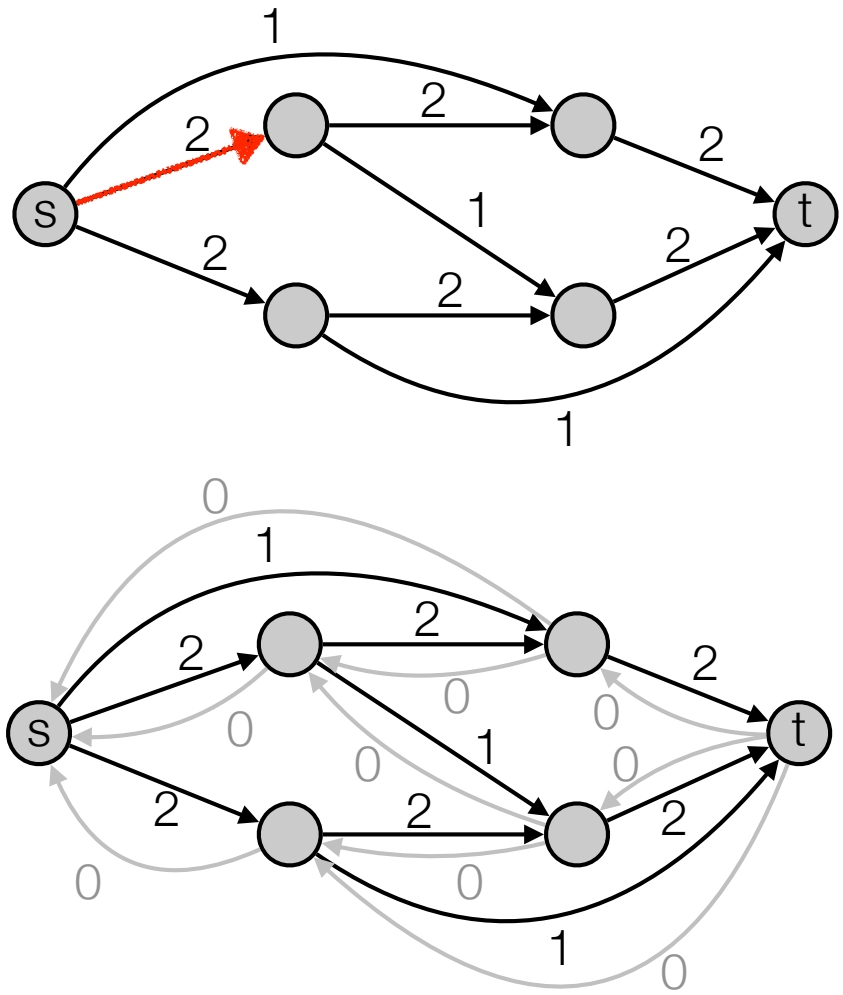
Residual networks



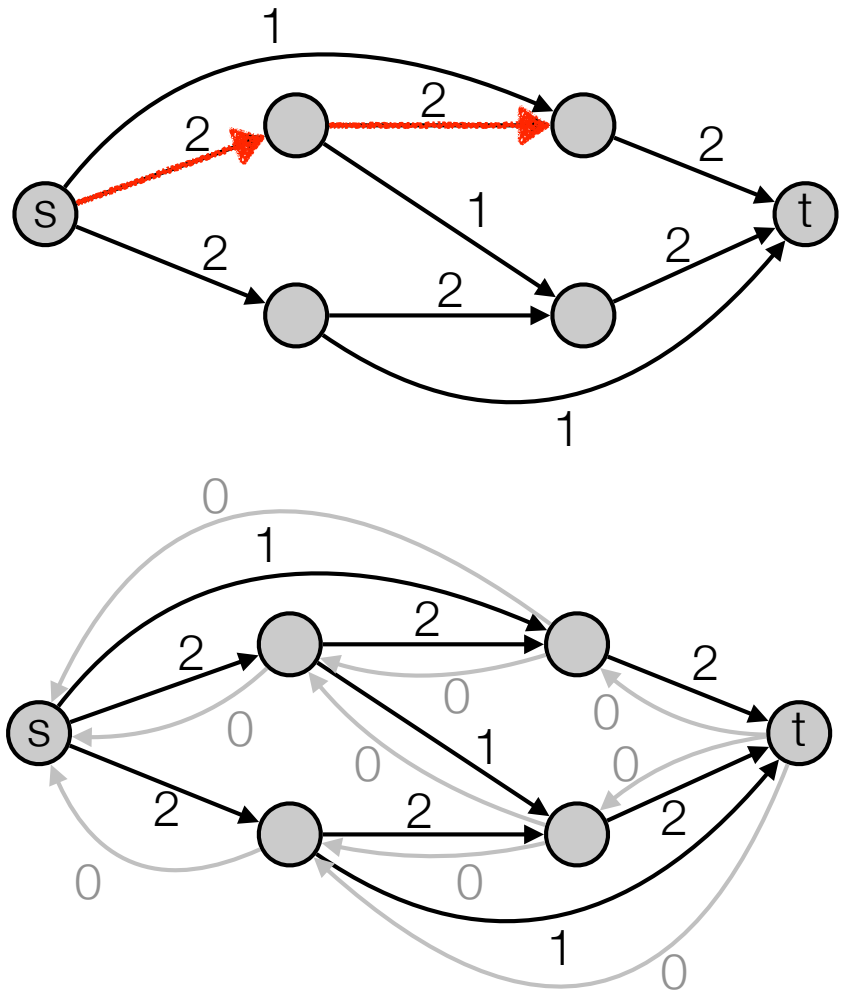
Residual networks



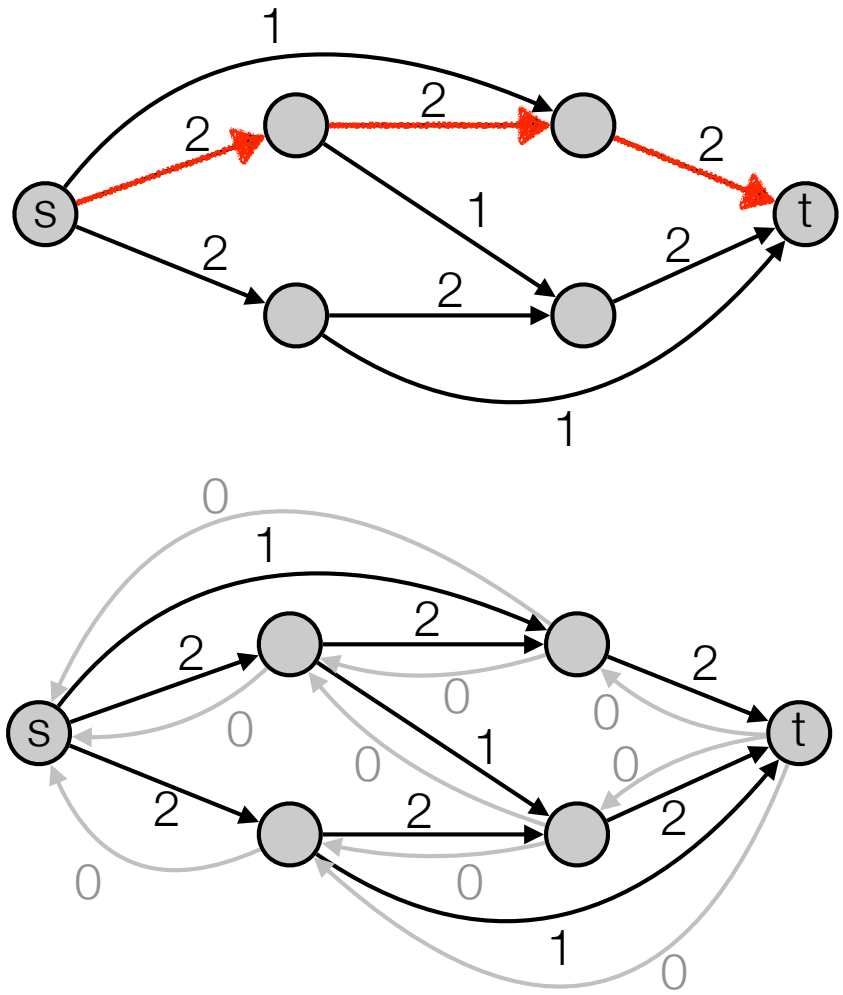
Residual networks



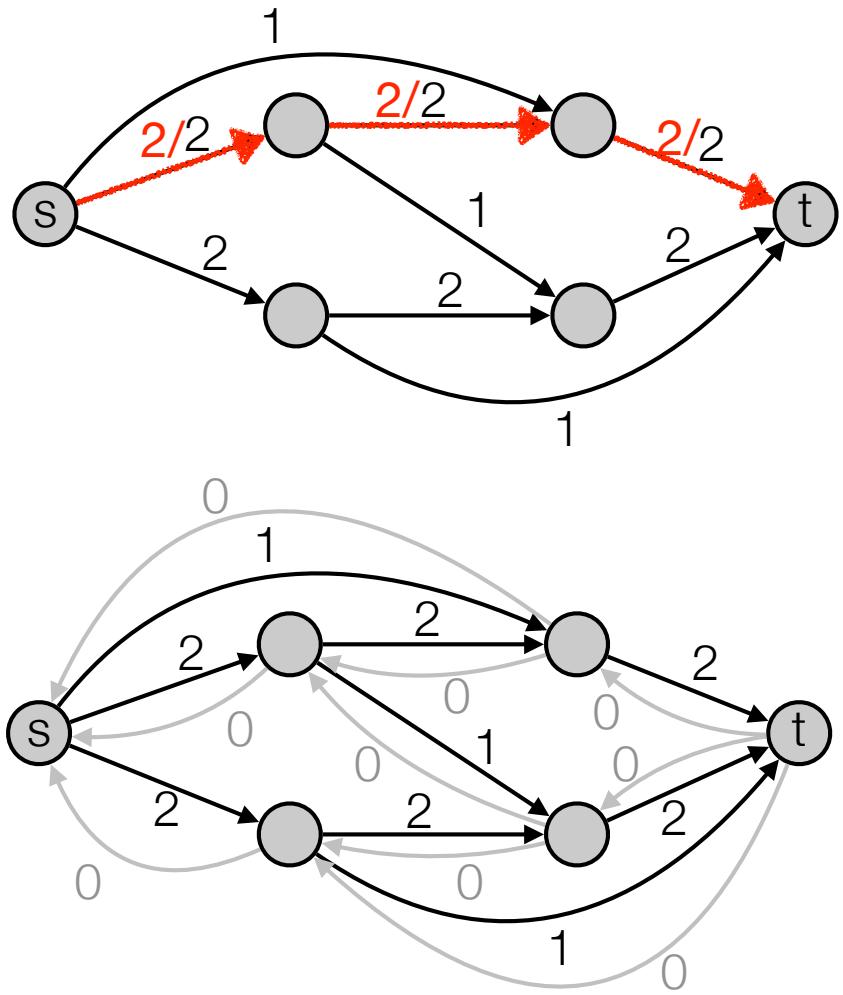
Residual networks



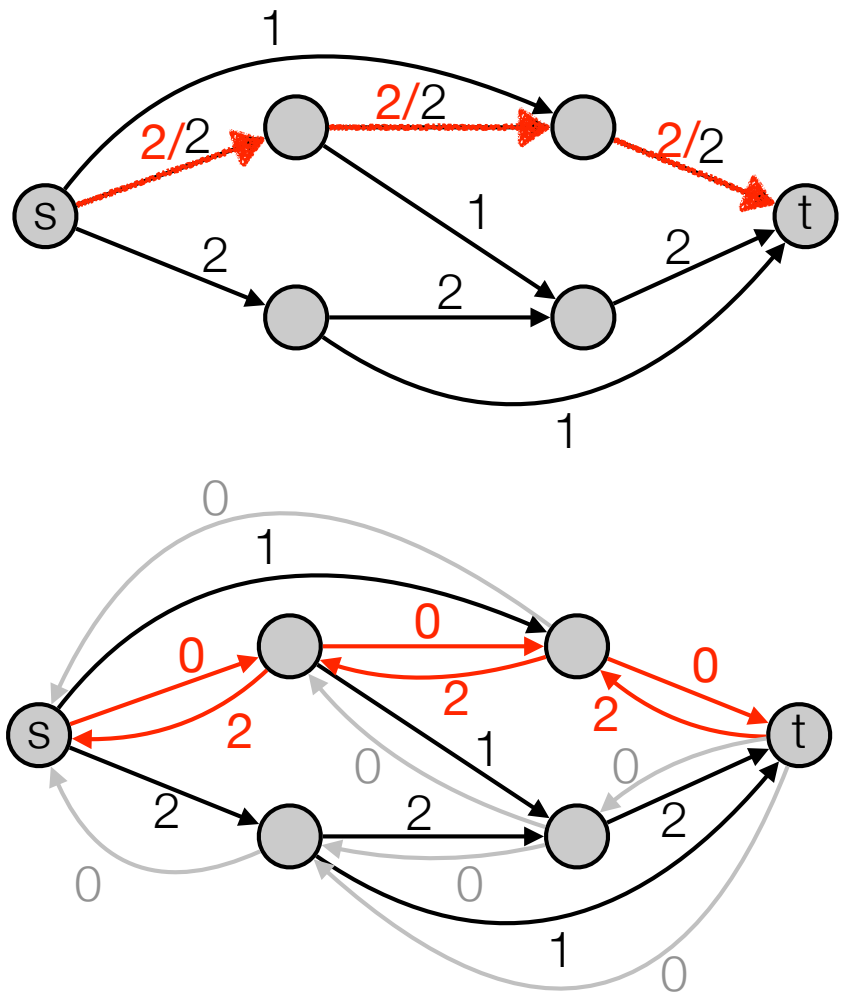
Residual networks



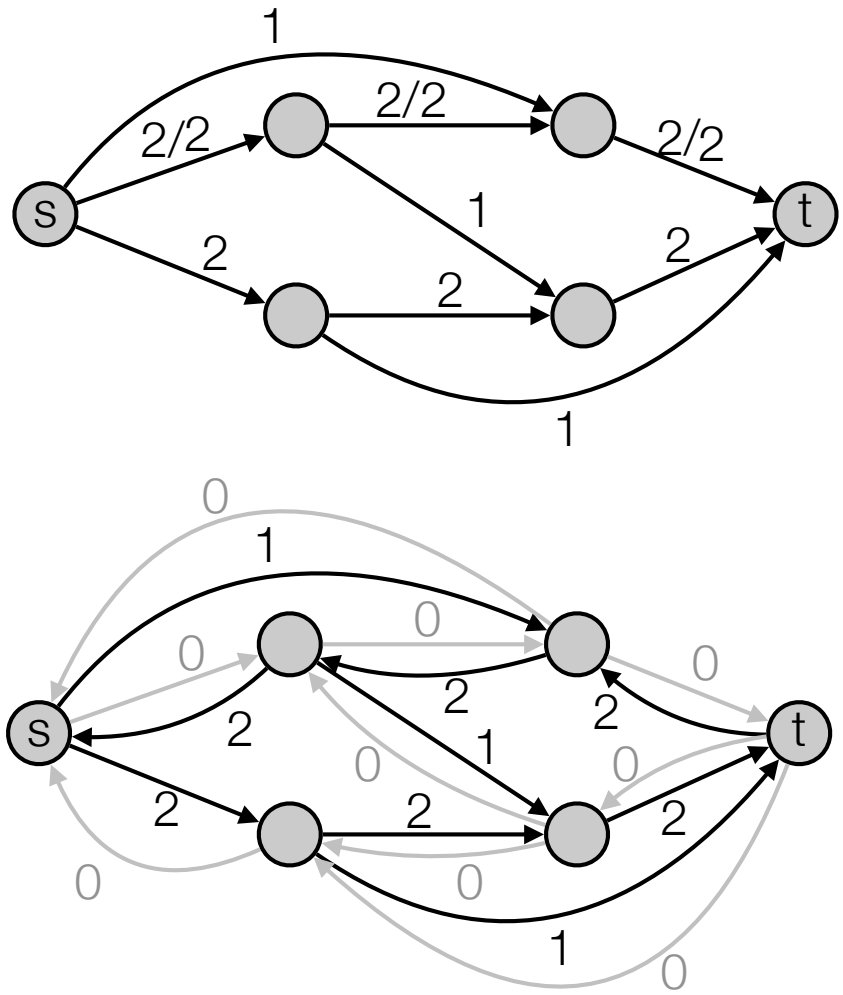
Residual networks



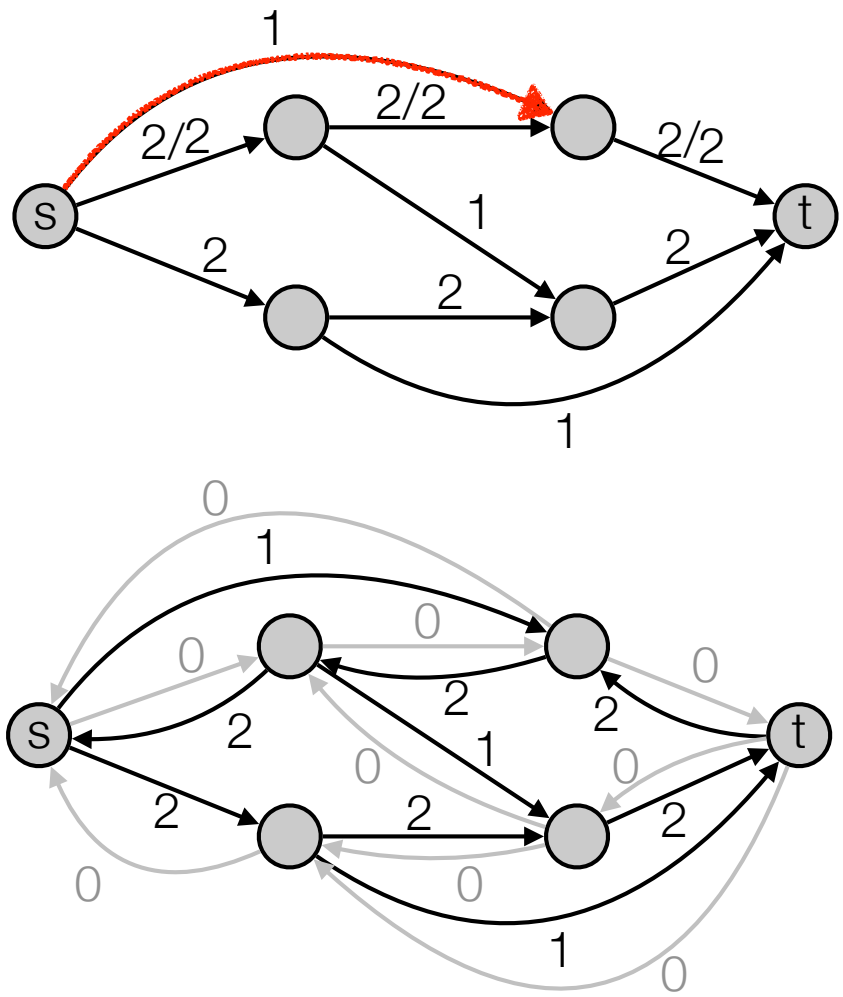
Residual networks



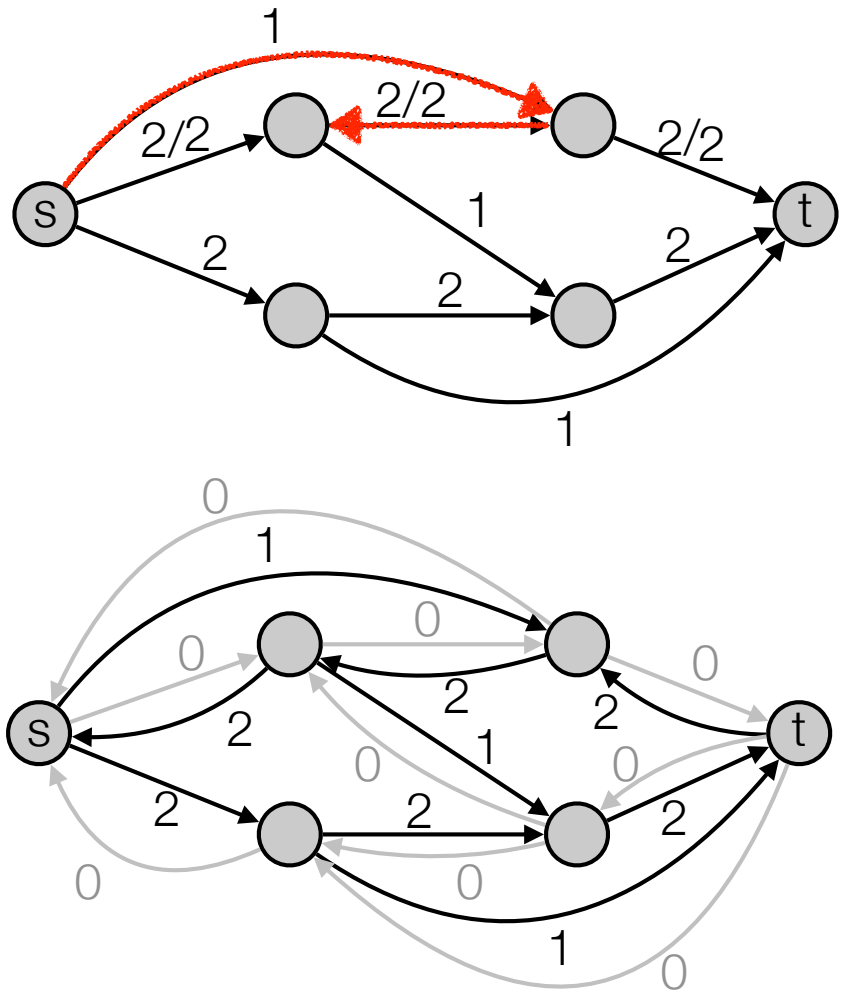
Residual networks



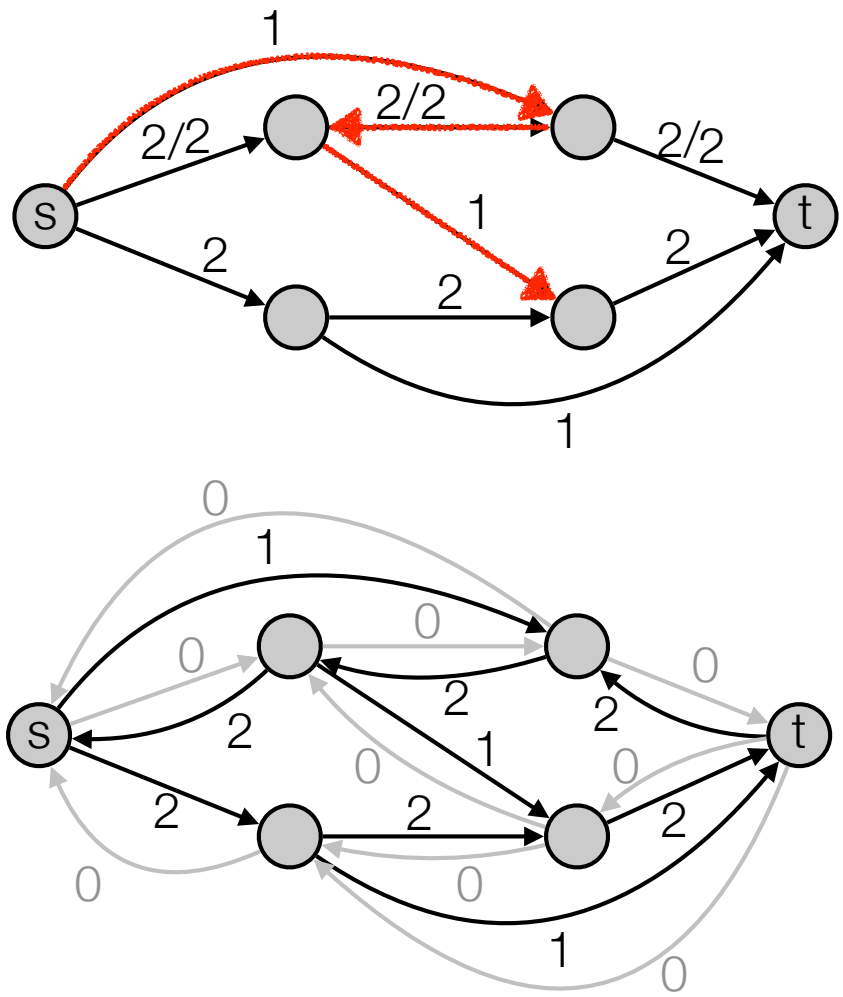
Residual networks



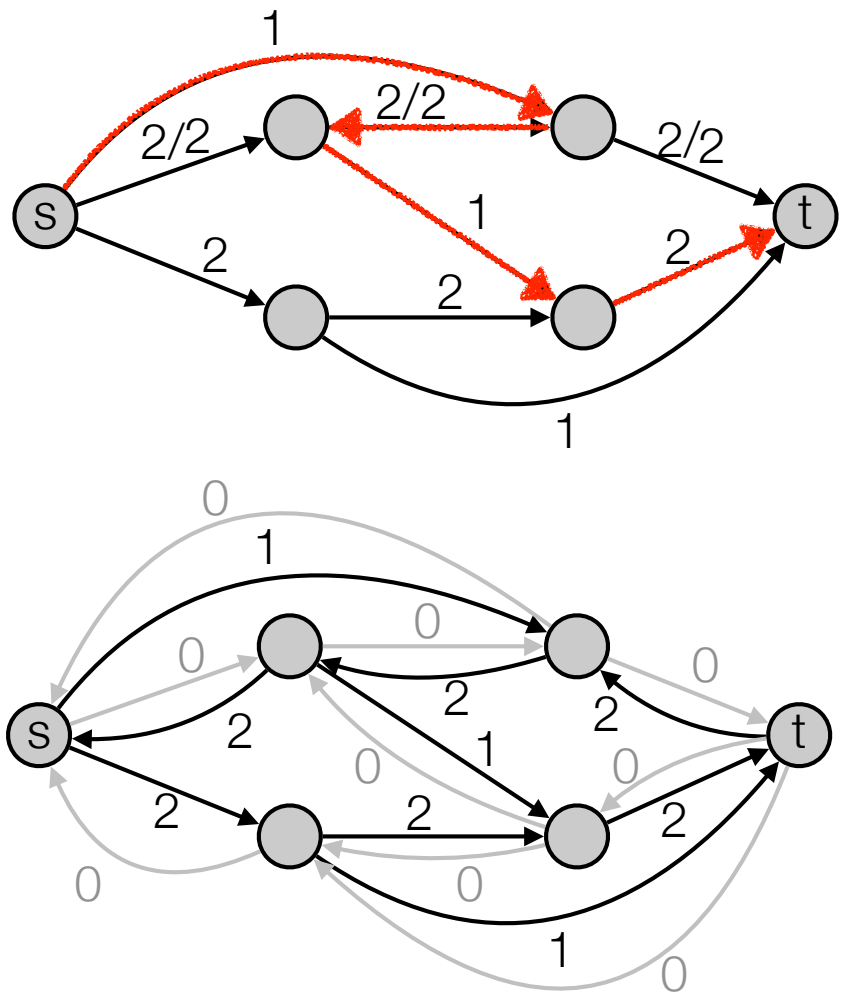
Residual networks



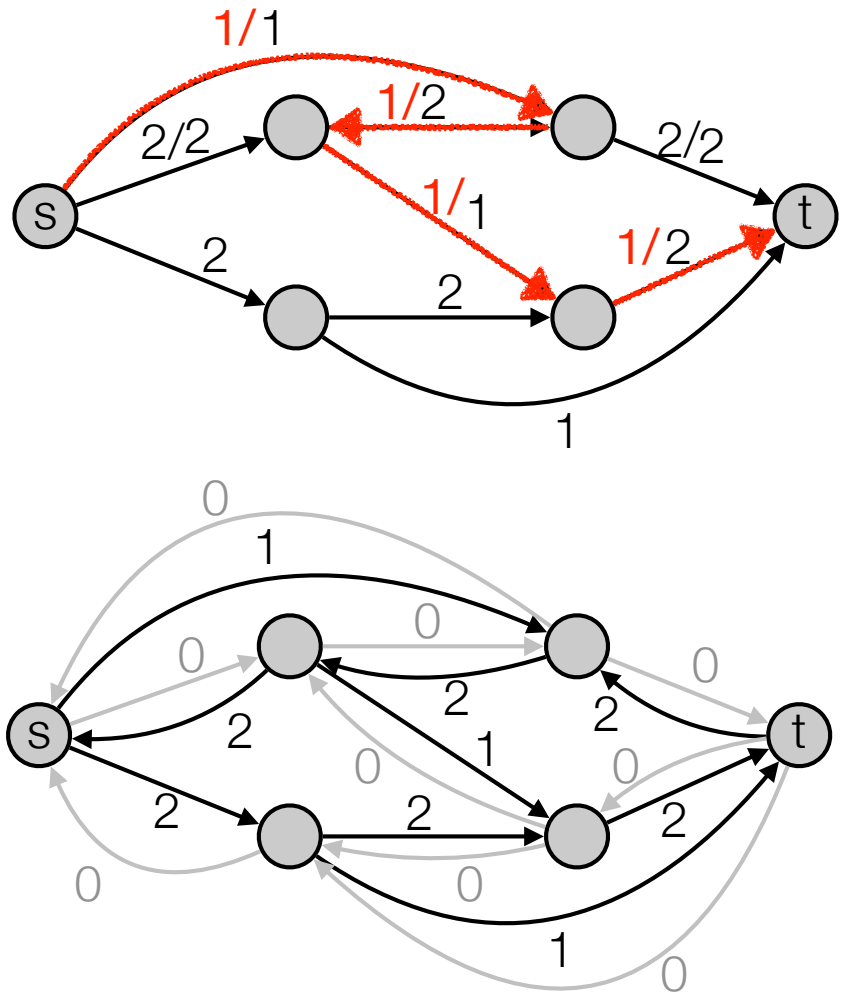
Residual networks



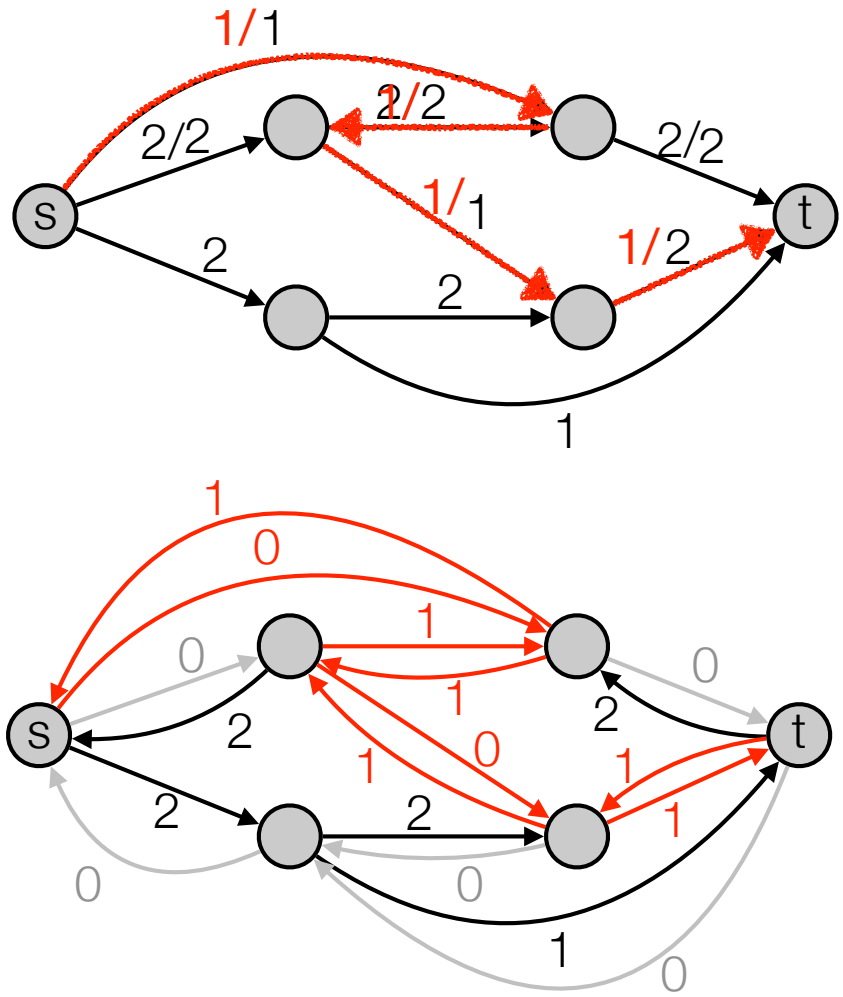
Residual networks



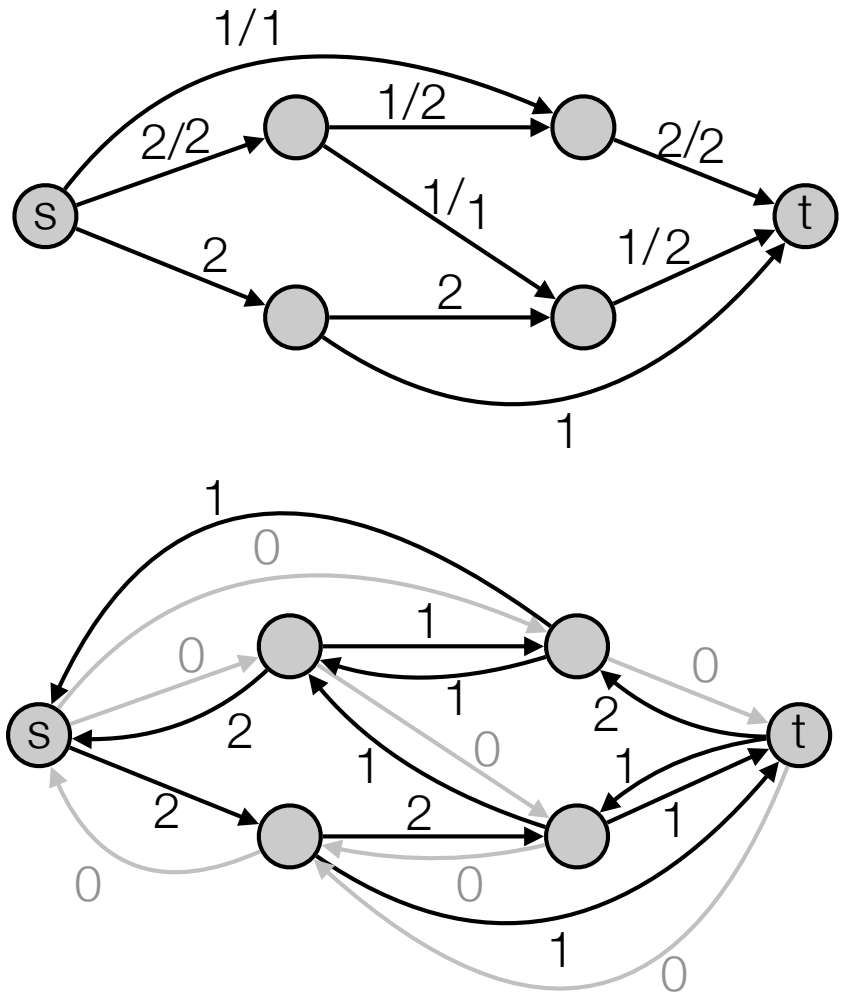
Residual networks



Residual networks



Residual networks

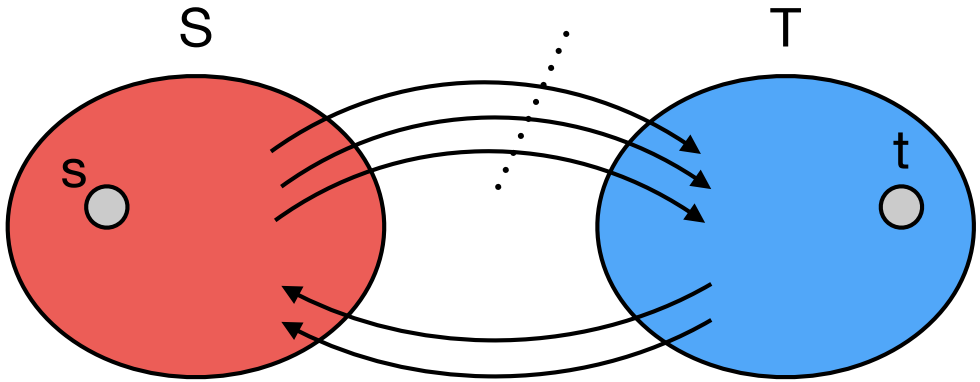


Implementation

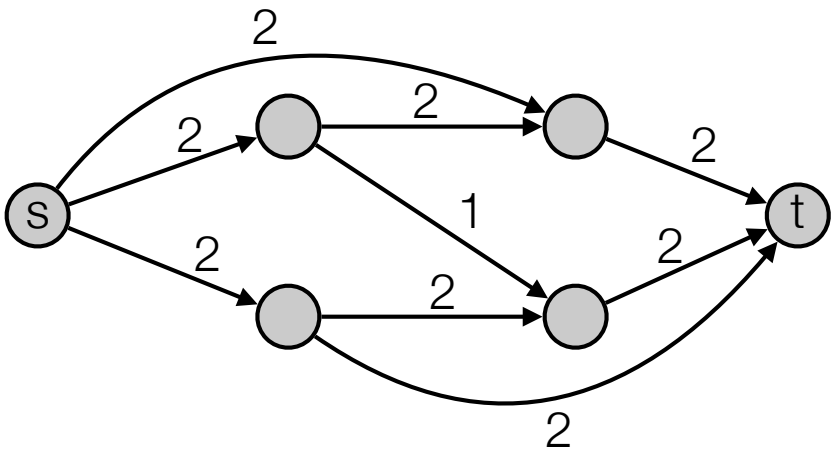
```
adj[0...n-1]                    # adjacency list
cap                             # capacity dictionary

for each edge (u,v,c):
    adj[u].append(v)            # add v to u's adjacency list (adding the edge u -> v)
    adj[v].append(u)            # add u to v's adjacency list (adding the edge v -> u)
    cap[(u,v)] = c              # set capacity on u->v edge to c.
    cap[(v,u)] = 0              # set capacity on u->v edge to 0.

# Graph search algorithm that searches for an augmenting path from u->v      (e.g. BFS or DFS)
AugPath():
    visited[0...n-1]            # visited list initialized to False
    pred[0...n-1]               # predecessor list
    stack S                     # initialize stack S

    push(S,s) and set visited[s] = True
    while S not empty and not visited[t]:
        u = pop(S)
        for v in adj[u]:
            if visited[v] or cap[(u,v)] = 0:
                continue
            visited[v] = True
            pred[v] = u
            push(S,v)
    if visited[t]:              # found augmenting path
        follow pred pointers back from t to s to find delta                 (fill out details yourself)
        follow pred pointers back from t to s to update capacities          (fill out details yourself)
        return delta
    return 0                    # no augmenting path found
```

s-t Cuts

- Cut: Partition of vertices into S and T, such that s ∈ S and t ∈ T.

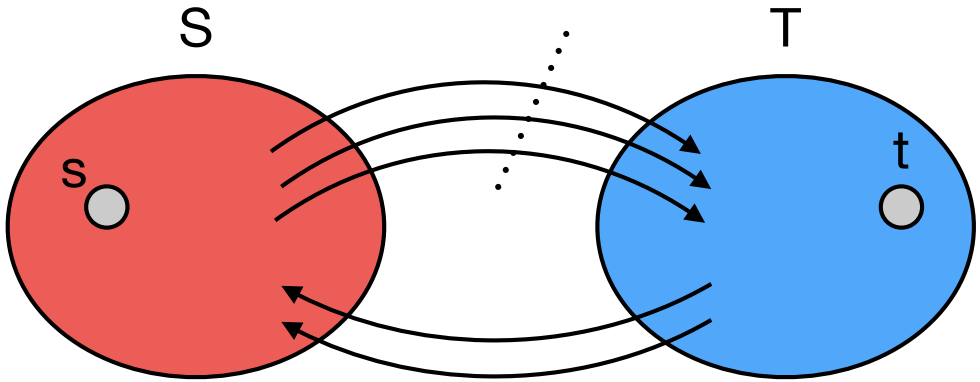


- Capacity of cut: total capacity of edges going *from* S to T.

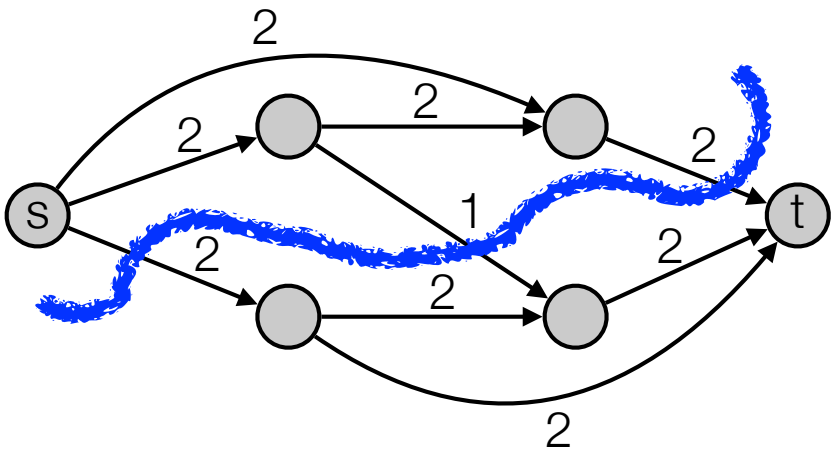


s-t Cuts

- Cut: Partition of vertices into S and T, such that s ∈ S and t ∈ T.



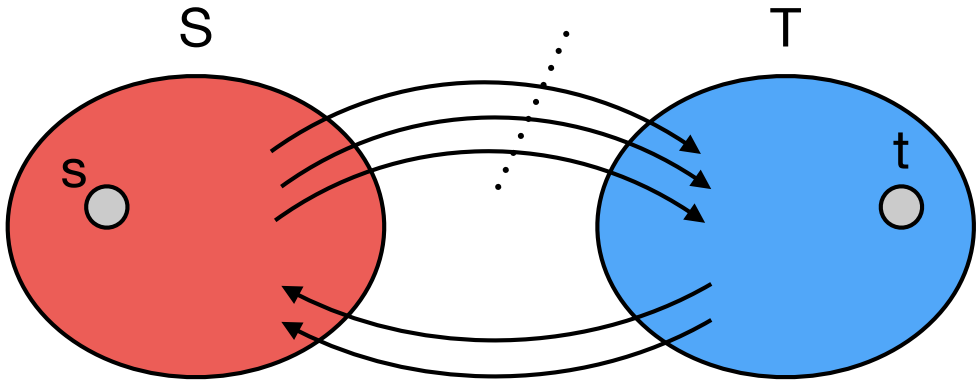
- Capacity of cut: total capacity of edges going *from* S to T.



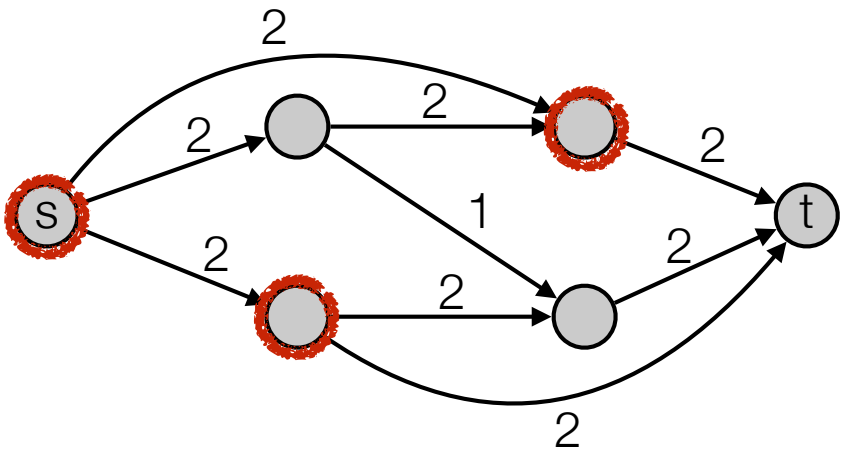
c(S,T) = 5

s-t Cuts

- Cut: Partition of vertices into S and T, such that s ∈ S and t ∈ T.

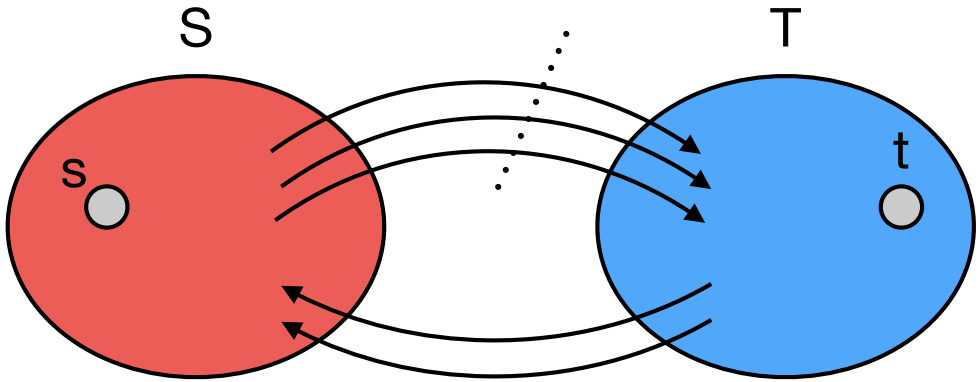


- Capacity of cut: total capacity of edges going *from* S to T.

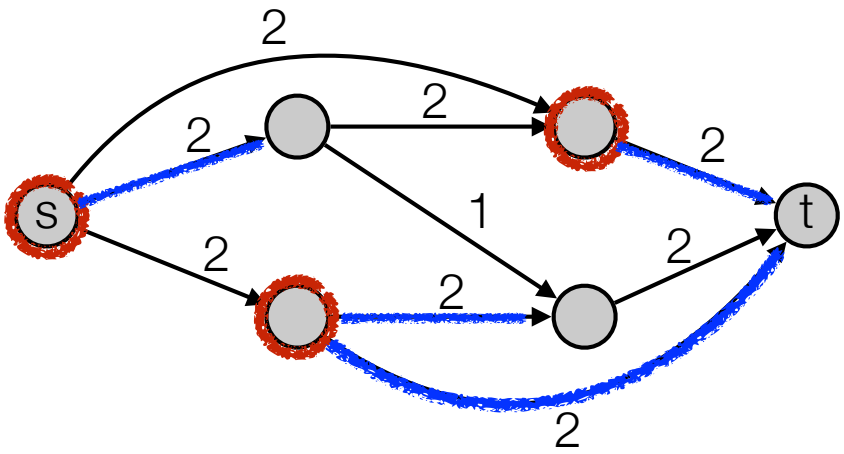


s-t Cuts

- Cut: Partition of vertices into S and T, such that s ∈ S and t ∈ T.

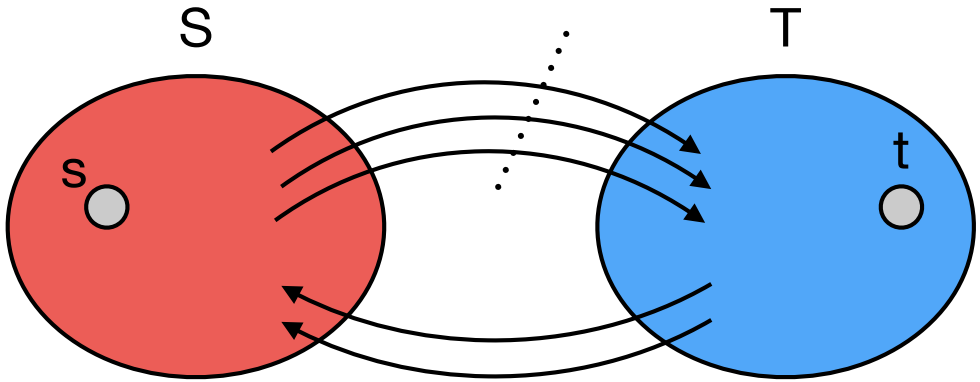


- Capacity of cut: total capacity of edges going *from* S to T.

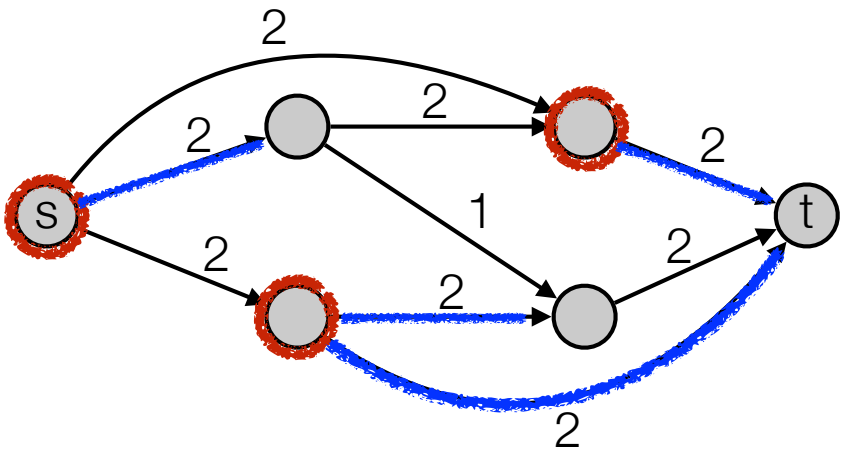


s-t Cuts

- Cut: Partition of vertices into S and T, such that s ∈ S and t ∈ T.



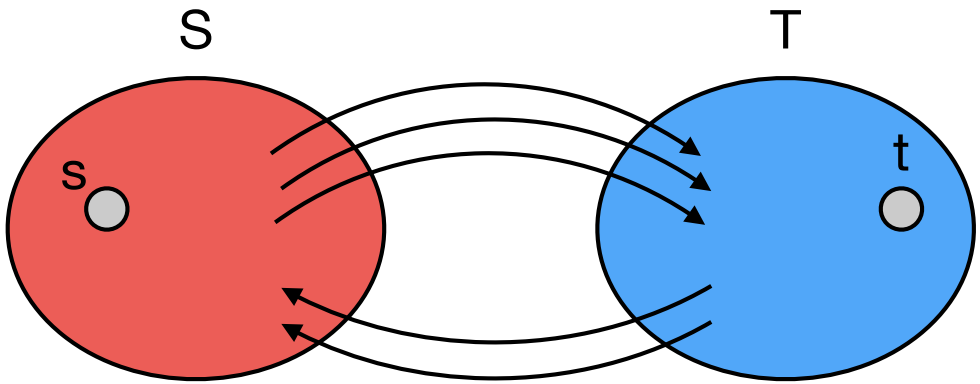
- Capacity of cut: total capacity of edges going *from* S to T.



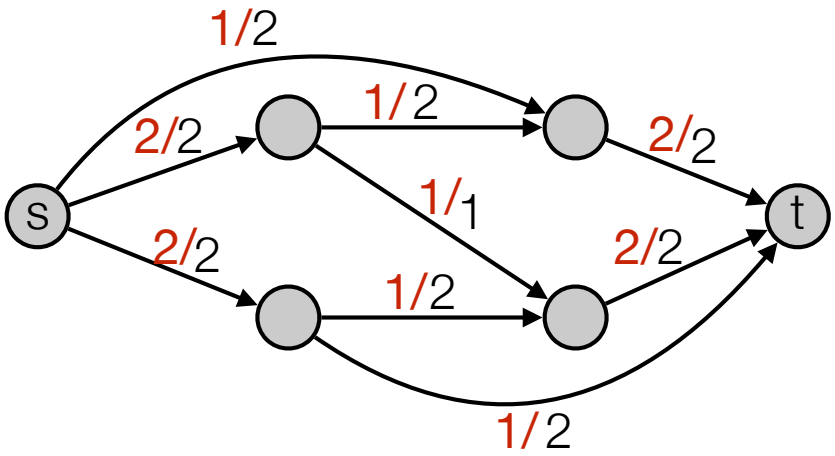
c(S,T) = 8

s-t Cuts

- Cut: Partition of vertices into S and T, such that s ∈ S and t ∈ T.

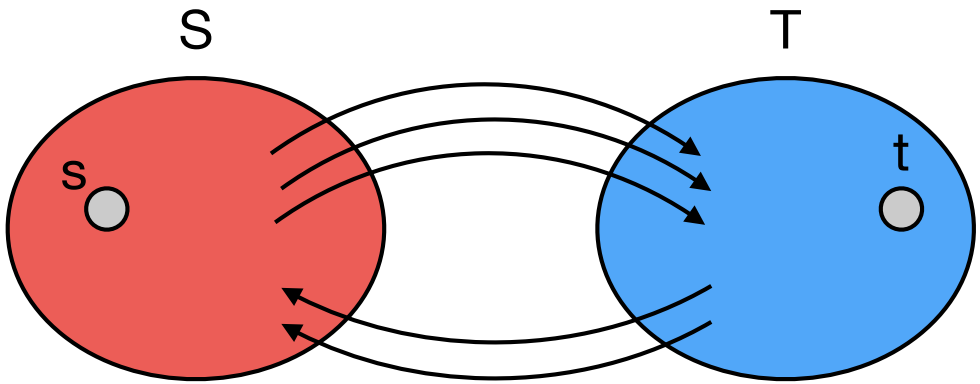


- Flow across cut: = flow *from* S to T *minus flow from* T to S.

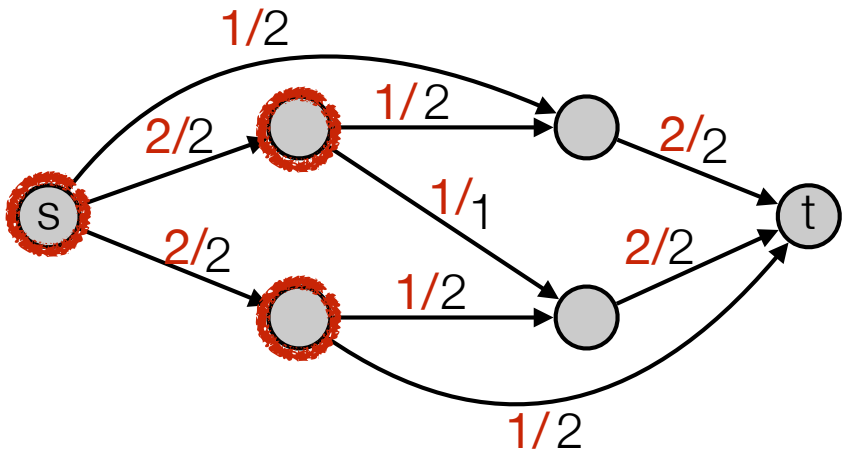


s-t Cuts

- Cut: Partition of vertices into S and T, such that s ∈ S and t ∈ T.

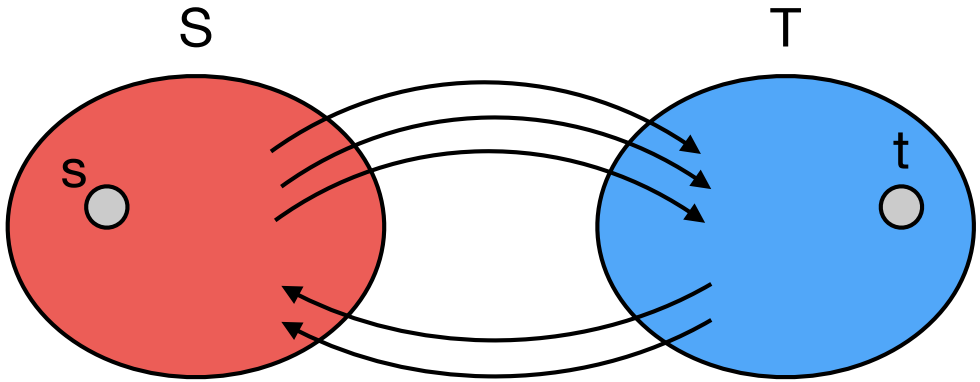


- Flow across cut: = flow *from* S to T *minus flow from* T to S.

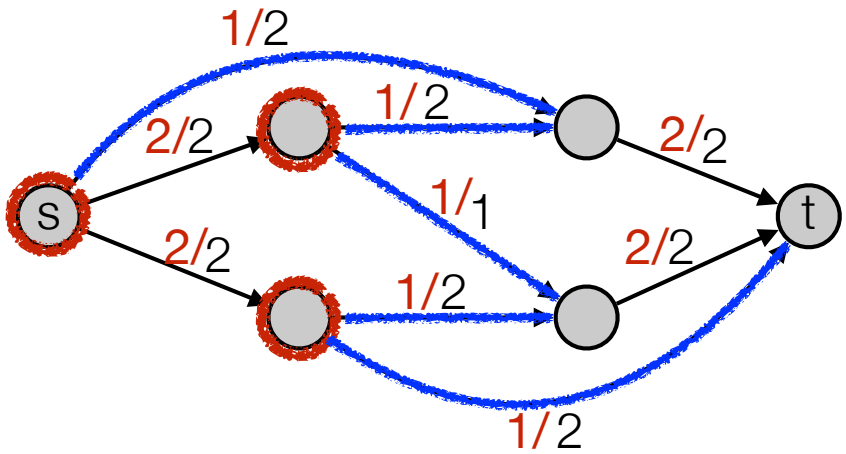


s-t Cuts

- Cut: Partition of vertices into S and T, such that s ∈ S and t ∈ T.

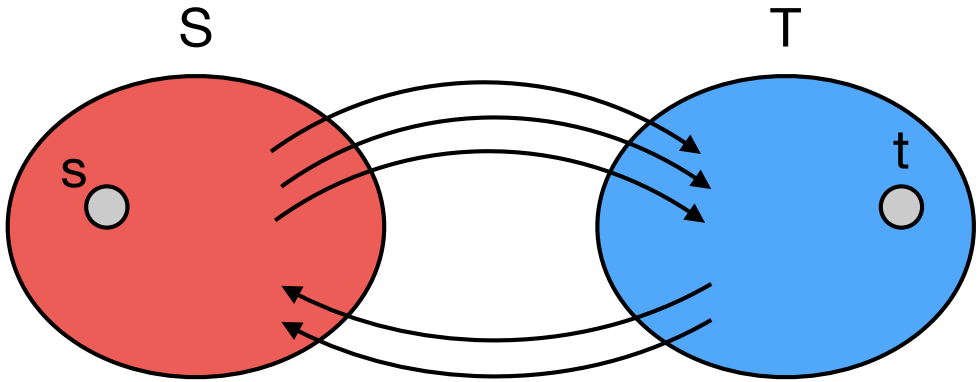


- Flow across cut: = flow *from* S to T *minus flow from* T to S.

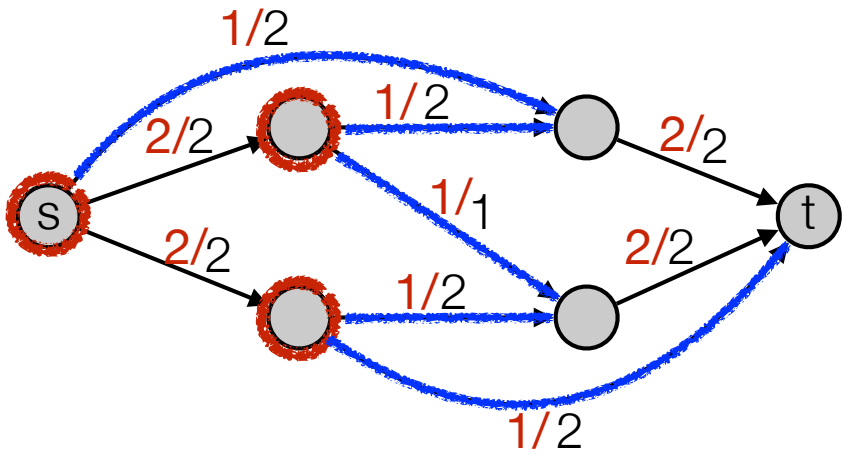


s-t Cuts

- Cut: Partition of vertices into S and T, such that s ∈ S and t ∈ T.



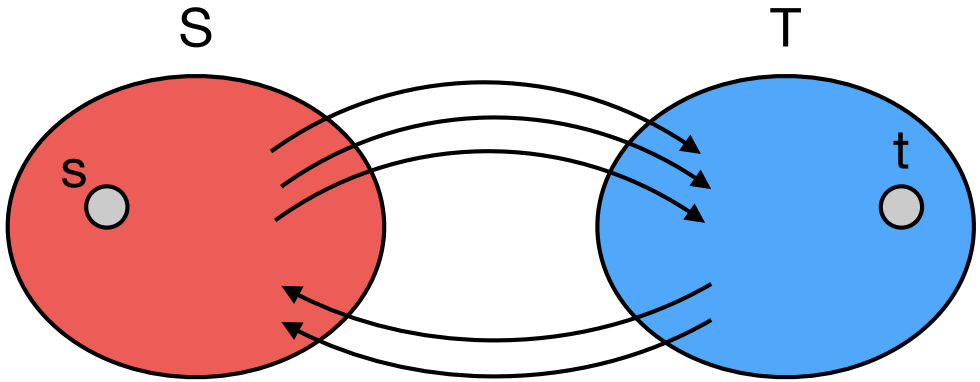
- Flow across cut: = flow *from* S to T *minus flow from* T to S.



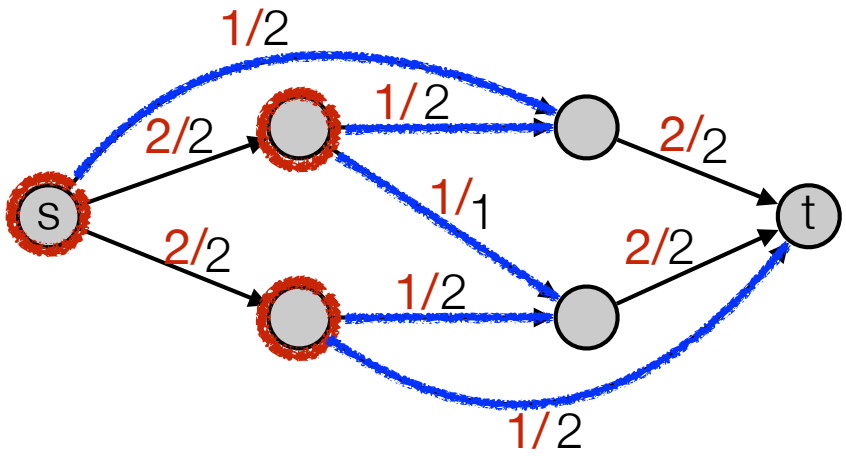
c(S,T) = 9

s-t Cuts

- Cut: Partition of vertices into S and T, such that s ∈ S and t ∈ T.



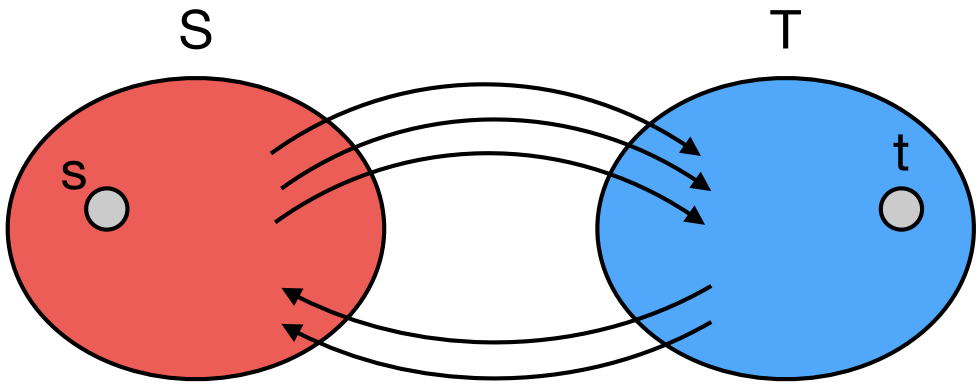
- Flow across cut: = flow *from* S to T *minus flow from* T to S.



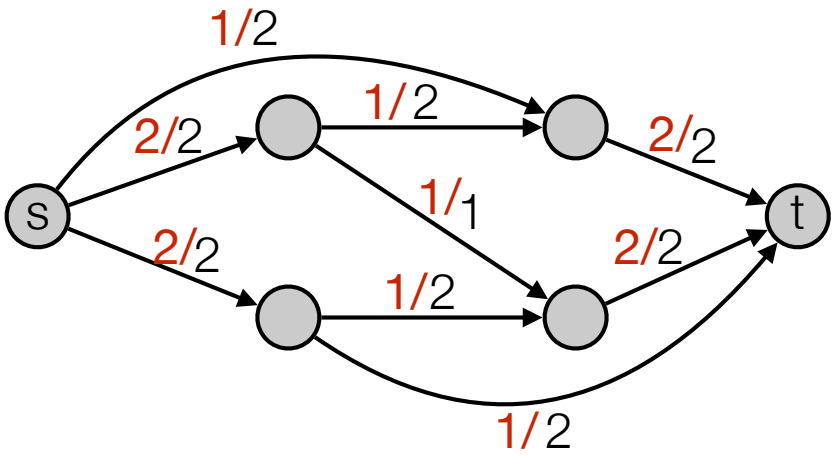
c(S,T) = 9 f(S,T) = 5

s-t Cuts

- Cut: Partition of vertices into S and T, such that s ∈ S and t ∈ T.

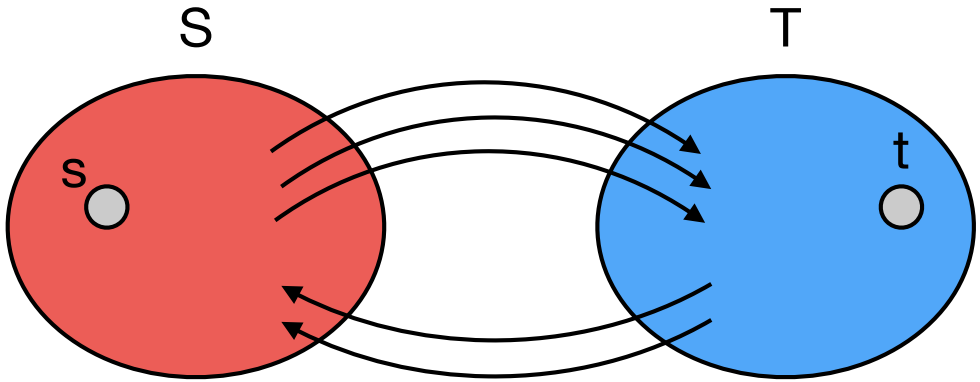


- Flow across cut: = flow *from* S to T *minus flow from* T to S.

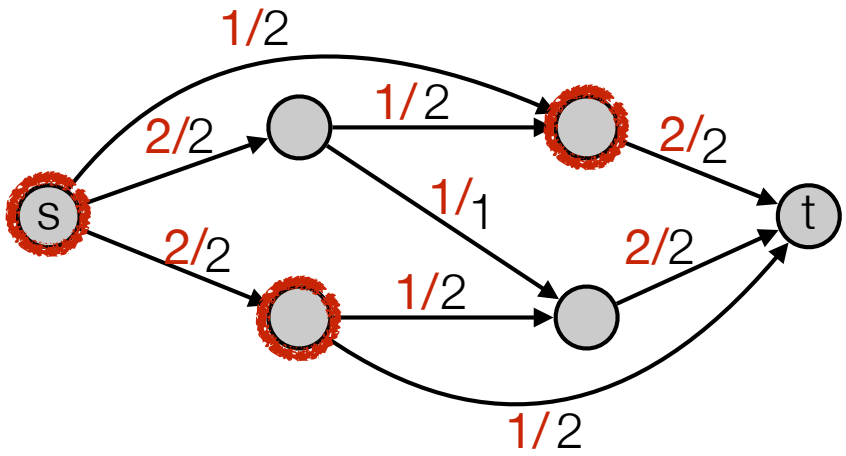


s-t Cuts

- Cut: Partition of vertices into S and T, such that s ∈ S and t ∈ T.

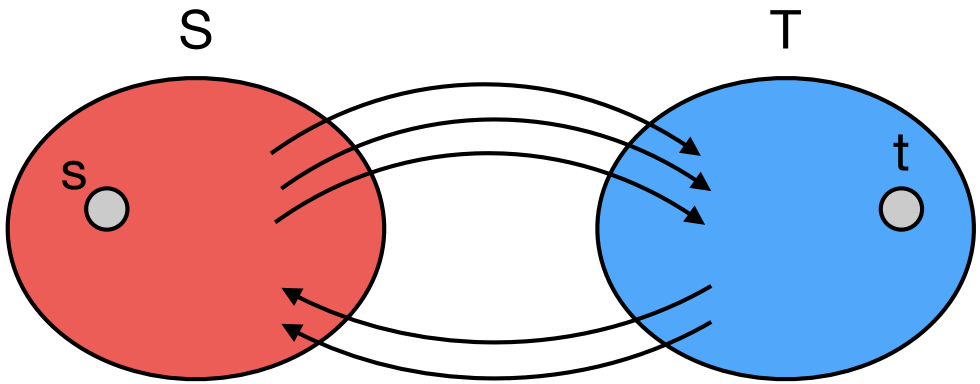


- Flow across cut: = flow *from* S to T *minus flow from* T to S.

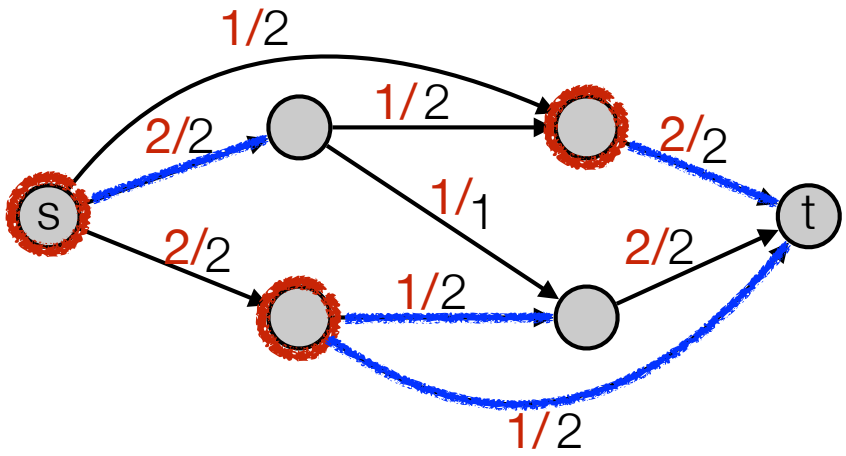


s-t Cuts

- Cut: Partition of vertices into S and T, such that s ∈ S and t ∈ T.

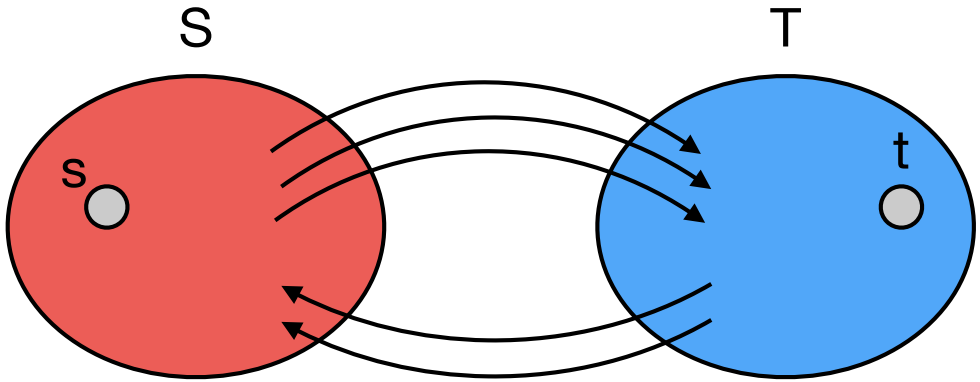


- Flow across cut: = flow *from* S to T *minus flow from* T to S.

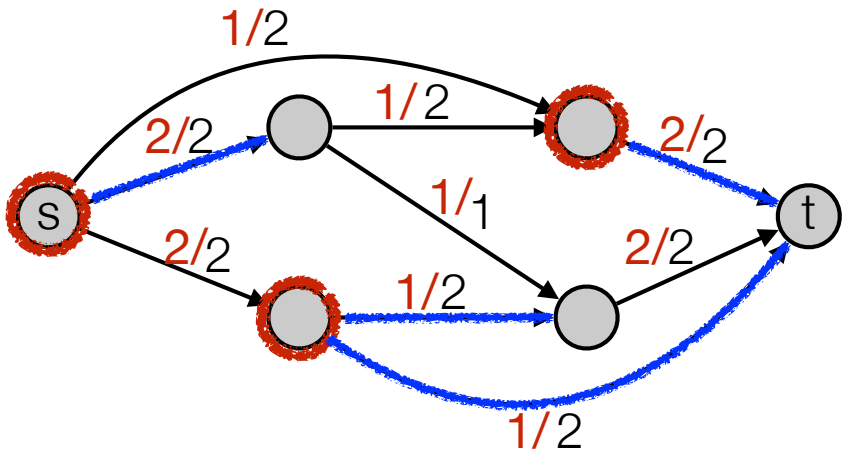


s-t Cuts

- Cut: Partition of vertices into S and T, such that s ∈ S and t ∈ T.



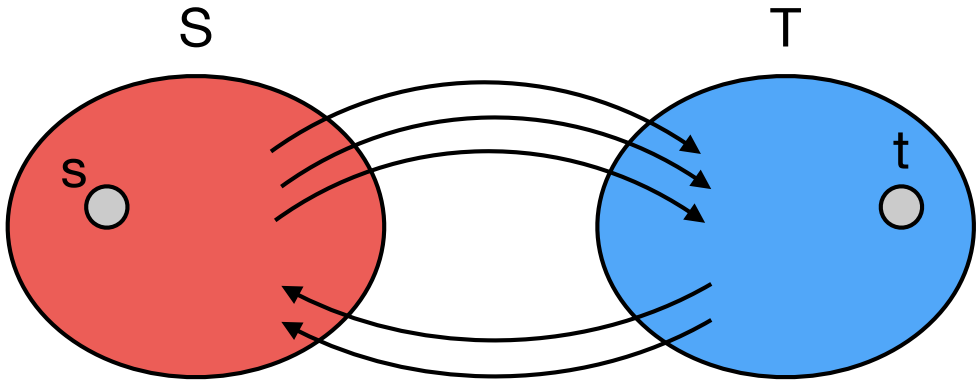
- Flow across cut: = flow *from* S to T *minus flow from* T to S.



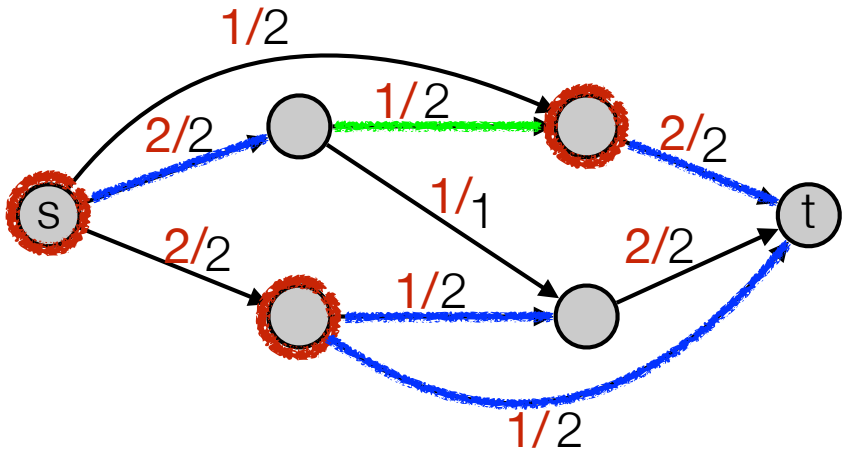
c(S,T) = 8

s-t Cuts

- Cut: Partition of vertices into S and T, such that s ∈ S and t ∈ T.



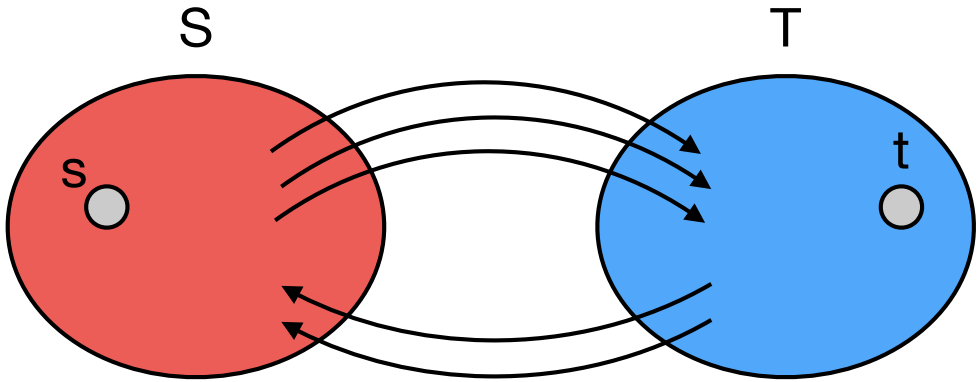
- Flow across cut: = flow *from* S to T *minus flow from* T to S.



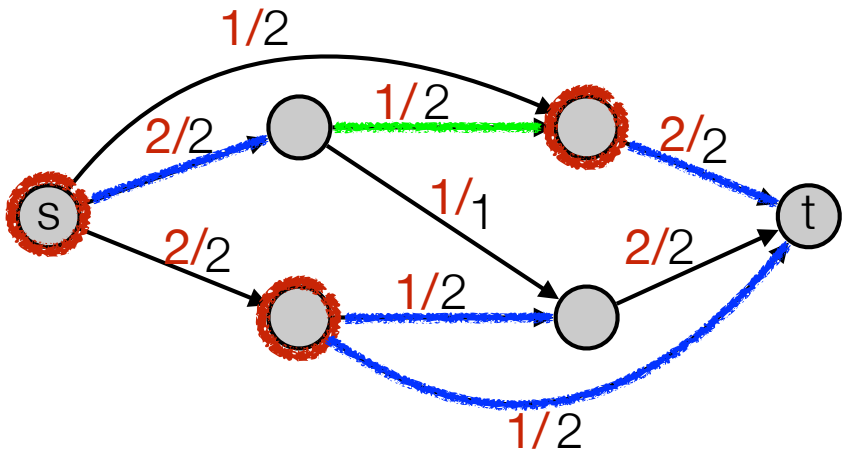
c(S,T) = 8

s-t Cuts

- Cut: Partition of vertices into S and T, such that s ∈ S and t ∈ T.



- Flow across cut: = flow *from* S to T *minus flow from* T to S.

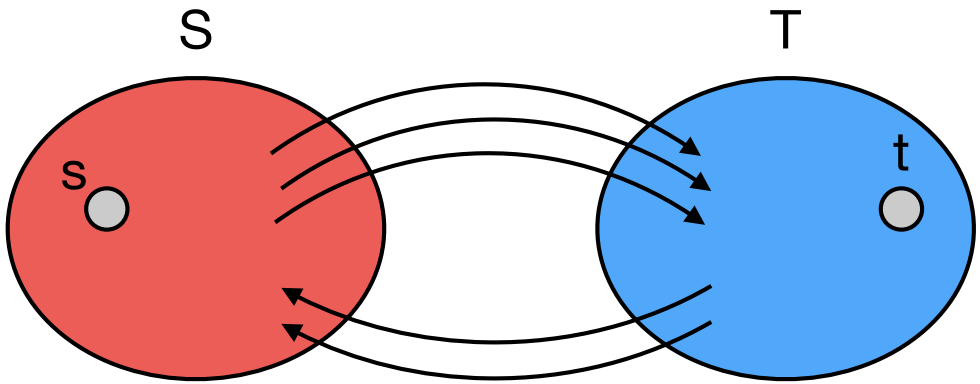


c(S,T) = 8

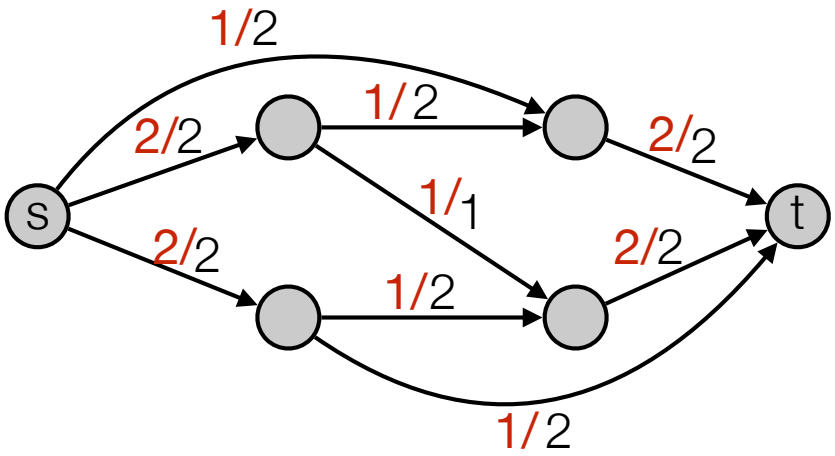
f(S,T) = 6 - 1 = 5

s-t Cuts

- Cut: Partition of vertices into S and T, such that s ∈ S and t ∈ T.

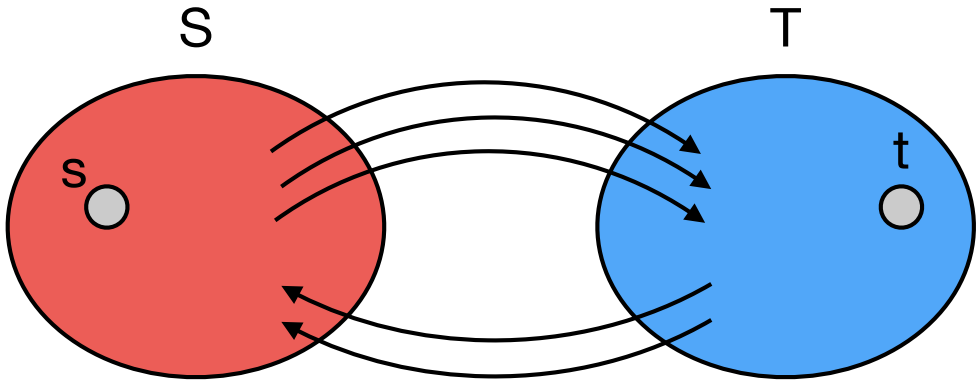


- Flow across cut: = flow *from* S to T *minus flow from* T to S.

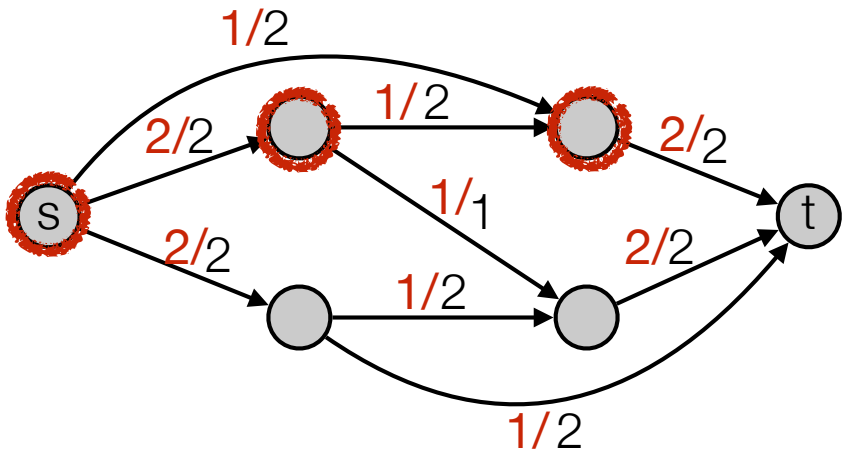


s-t Cuts

- Cut: Partition of vertices into S and T, such that s ∈ S and t ∈ T.

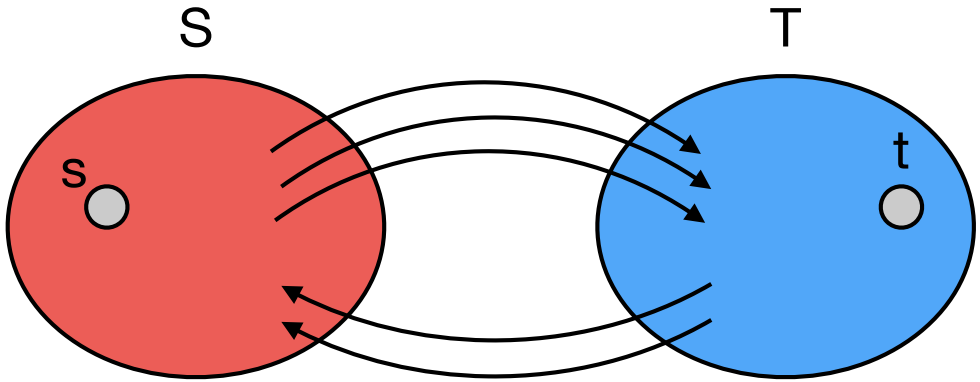


- Flow across cut: = flow *from* S to T *minus flow from* T to S.

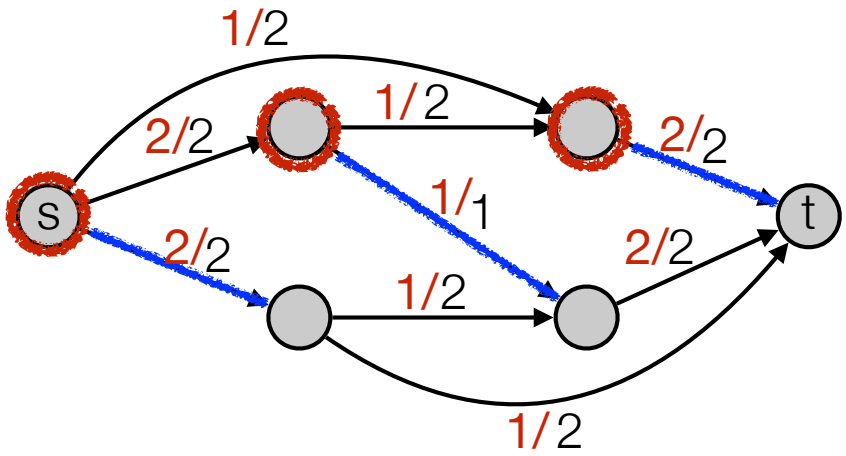


s-t Cuts

- Cut: Partition of vertices into S and T, such that s ∈ S and t ∈ T.

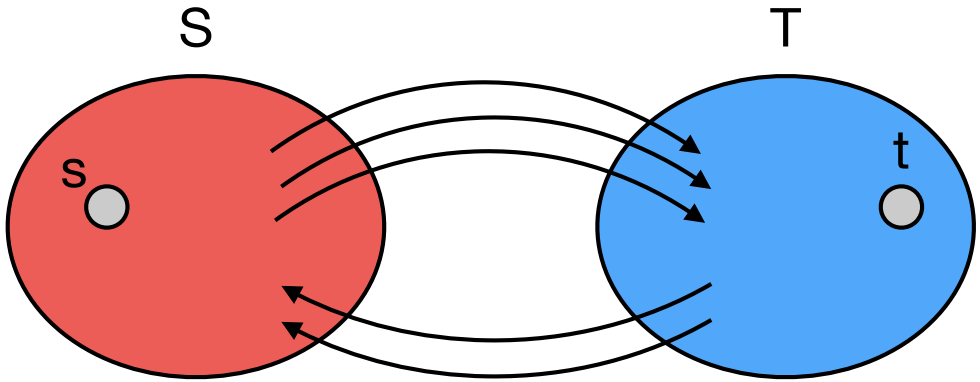


- Flow across cut: = flow *from* S to T *minus flow from* T to S.

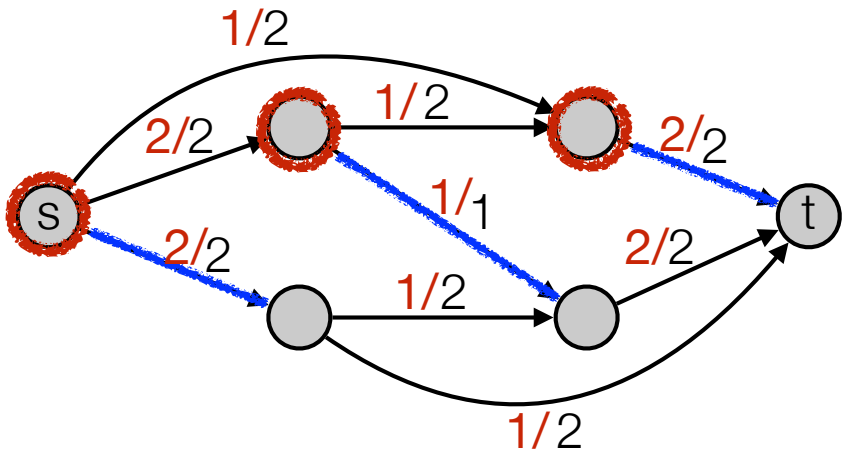


s-t Cuts

- Cut: Partition of vertices into S and T, such that s ∈ S and t ∈ T.



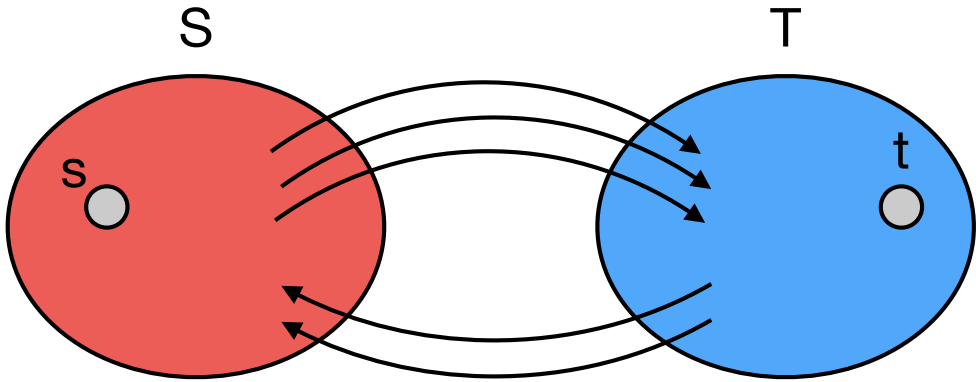
- Flow across cut: = flow *from* S to T *minus flow from* T to S.



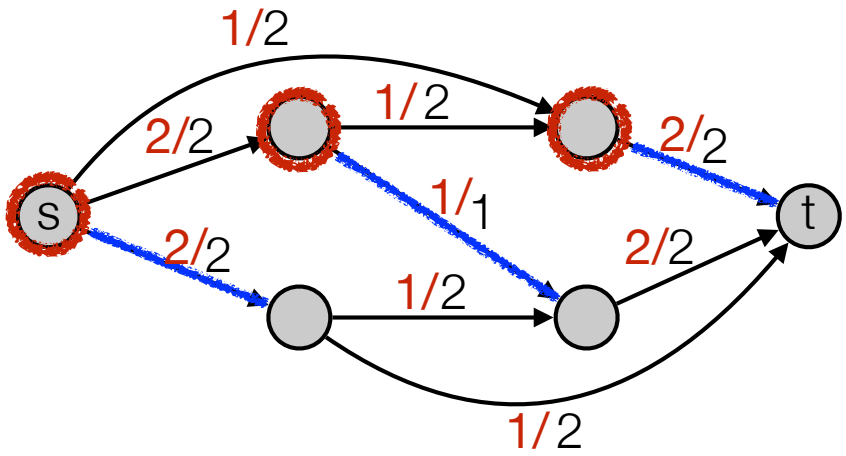
c(S,T) = 5

s-t Cuts

- Cut: Partition of vertices into S and T, such that s ∈ S and t ∈ T.



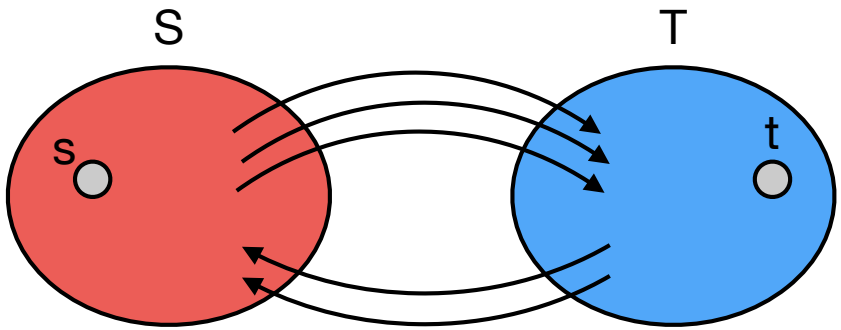
- Flow across cut: = flow *from* S to T *minus flow from* T to S.



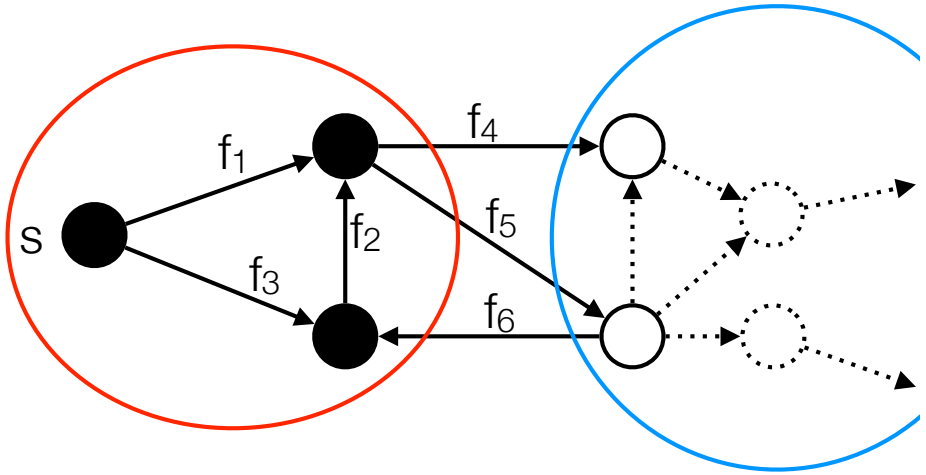
c(S,T) = 5 f(S,T) = 5

s-t Cuts

- Cut: Partition of vertices into S and T, such that s ∈ S and t ∈ T.

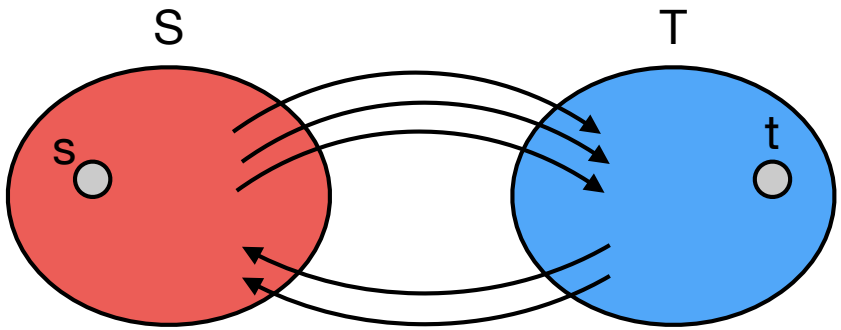


- Flow across cut = flow *from* S to T *minus flow from* T to S.

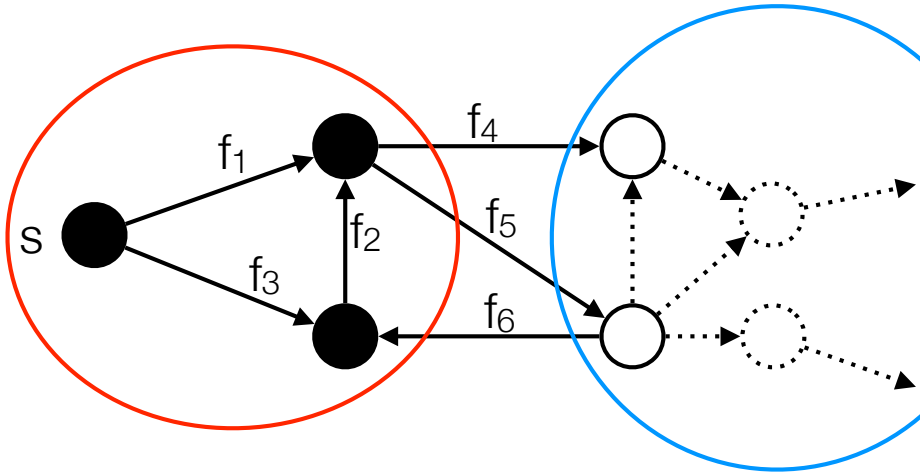


s-t Cuts

- Cut: Partition of vertices into S and T, such that s ∈ S and t ∈ T.

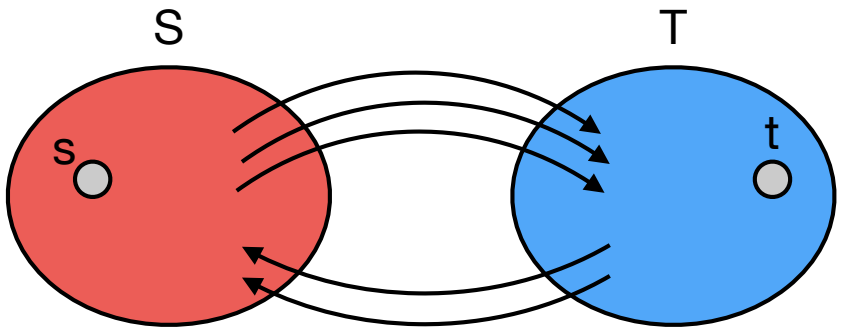


- Flow across cut = flow *from* S to T *minus flow from* T to S.
- Flow across cut: $f_4 + f_5 - f_6 = ?$

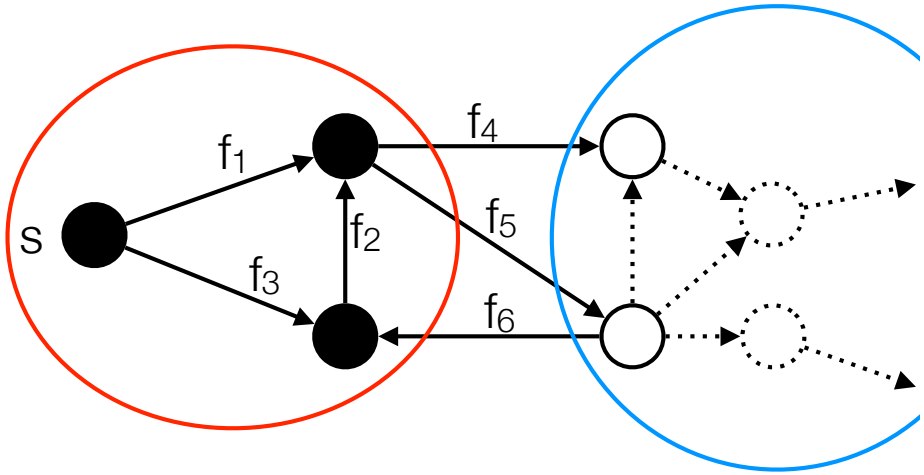


s-t Cuts

- Cut: Partition of vertices into S and T, such that s ∈ S and t ∈ T.

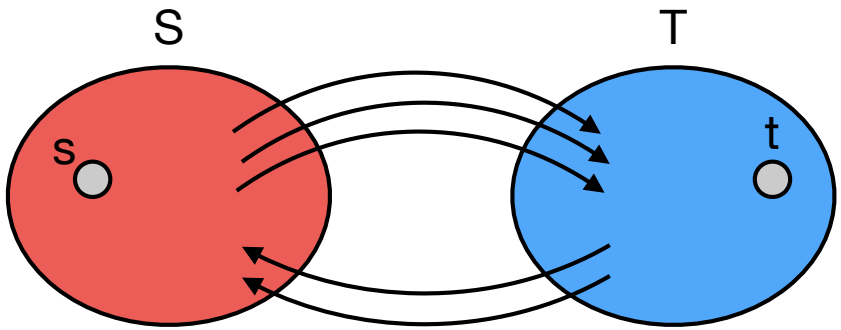


- Flow across cut = flow *from* S to T *minus flow from* T to S.
- Flow across cut: $f_4 + f_5 - f_6 = ?$
  - $f_4 + f_5 - f_1 - f_2 = 0$

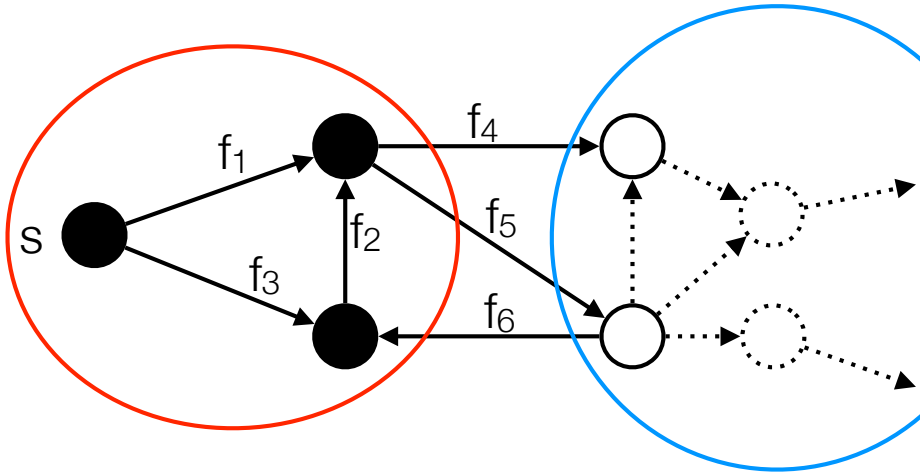


s-t Cuts

- Cut: Partition of vertices into S and T, such that s ∈ S and t ∈ T.

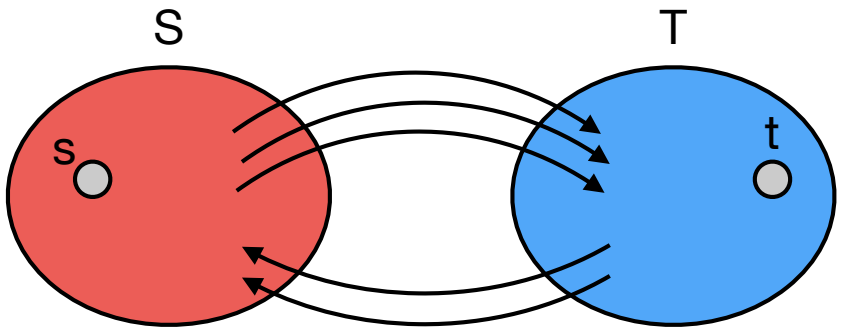


- Flow across cut = flow *from* S to T *minus flow from* T to S.
- Flow across cut: $f_4 + f_5 - f_6 = ?$
  - $f_4 + f_5 - f_1 - f_2 = 0$
  - $f_2 - f_6 - f_3 = 0$

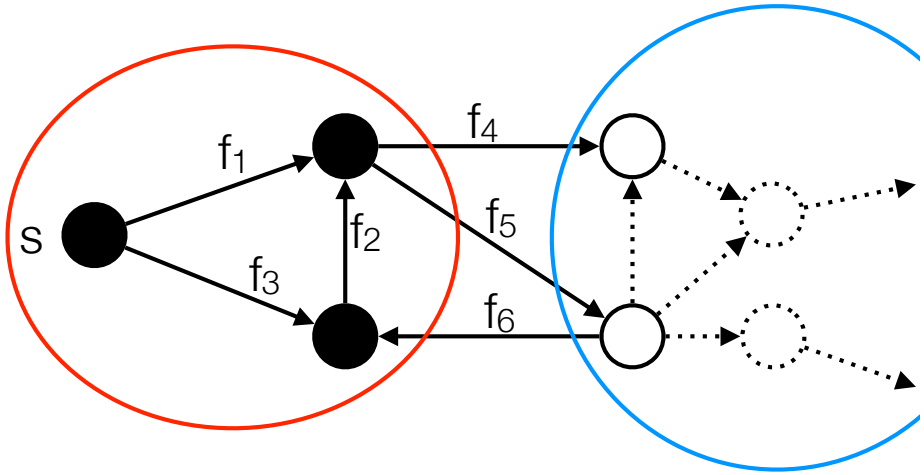


s-t Cuts

- Cut: Partition of vertices into S and T, such that s ∈ S and t ∈ T.

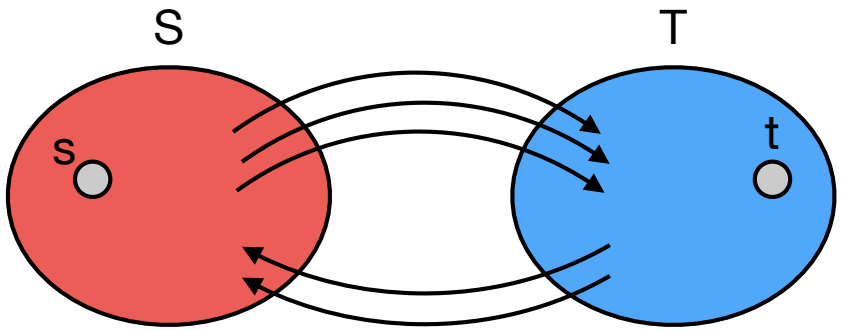


- Flow across cut = flow *from* S to T *minus flow from* T to S.
- Flow across cut: $f_4 + f_5 - f_6 = ?$
  - $f_4 + f_5 - f_1 - f_2 = 0$
  - $f_2 - f_6 - f_3 = 0$
  - $f_1 + f_3 = |f|$

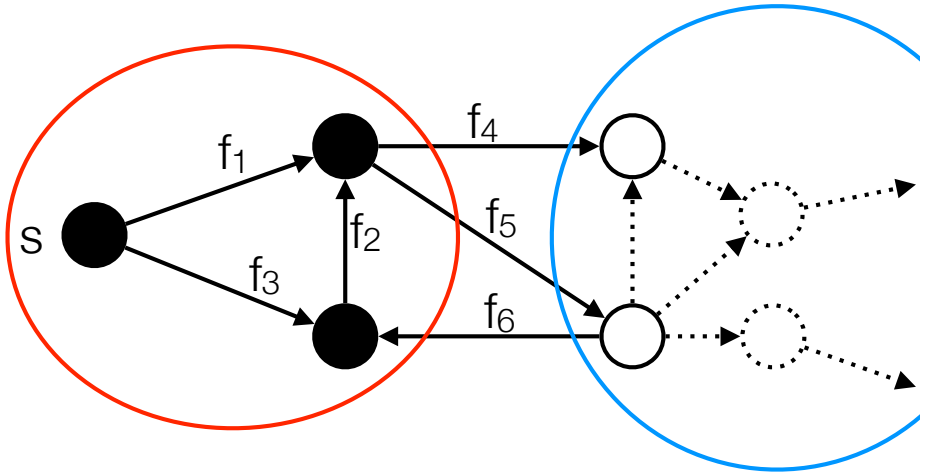


s-t Cuts

- Cut: Partition of vertices into S and T, such that s ∈ S and t ∈ T.

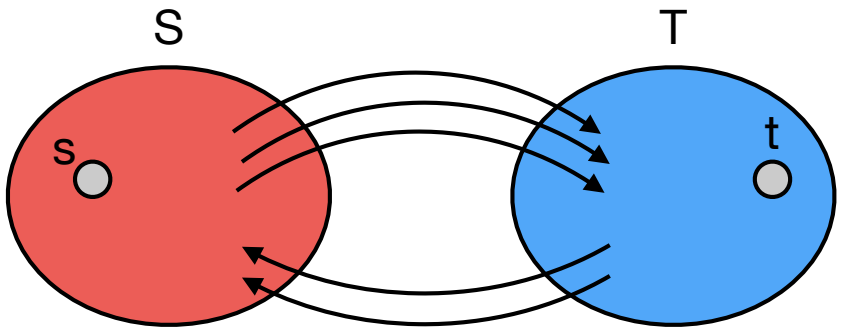


- Flow across cut = flow *from* S to T *minus flow from* T to S.
- Flow across cut: $f_4 + f_5 - f_6 = ?$
  - $f_4 + f_5 - f_1 - f_2 = 0$
  - $f_2 - f_6 - f_3 = 0$
  - $f_1 + f_3 = |f|$
  - $(f_4 + f_5 - f_1 - f_2) + (f_2 - f_6 - f_3) + (f_1 + f_3) = |f|$

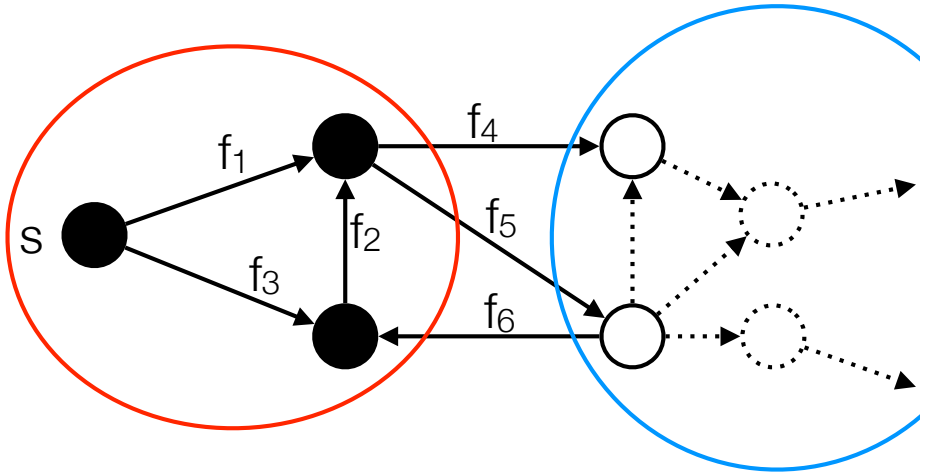


s-t Cuts

- Cut: Partition of vertices into S and T, such that s ∈ S and t ∈ T.

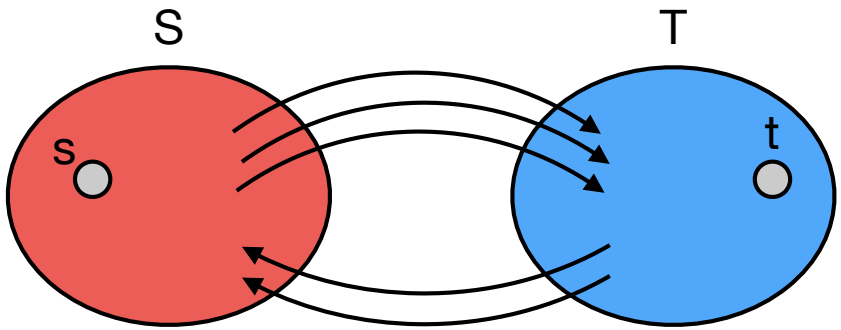


- Flow across cut = flow *from* S to T *minus flow from* T to S.
- Flow across cut: $f_4 + f_5 - f_6 = ?$
  - $f_4 + f_5 - f_1 - f_2 = 0$
  - $f_2 - f_6 - f_3 = 0$
  - $f_1 + f_3 = |f|$
  - $(f_4 + f_5 - f_1 - f_2) + (f_2 - f_6 - f_3) + (f_1 + f_3) = |f|$

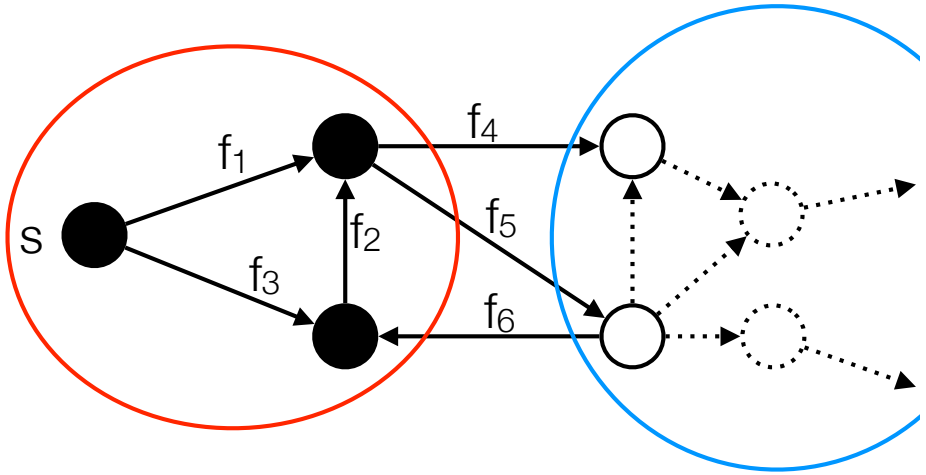


s-t Cuts

- Cut: Partition of vertices into S and T, such that s ∈ S and t ∈ T.

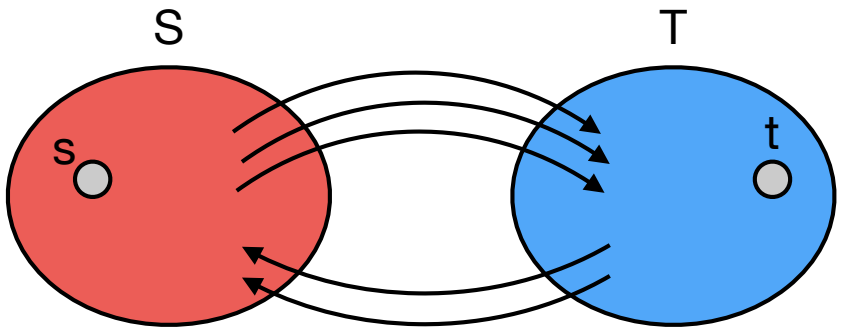


- Flow across cut = flow *from* S to T *minus flow from* T to S.
- Flow across cut: $f_4 + f_5 - f_6 = ?$
  - $f_4 + f_5 - f_1 - f_2 = 0$
  - $f_2 - f_6 - f_3 = 0$
  - $f_1 + f_3 = |f|$
  - $(f_4 + f_5 - f_1 - f_2) + (f_2 - f_6 - f_3) + (f_1 + f_3) = |f|$

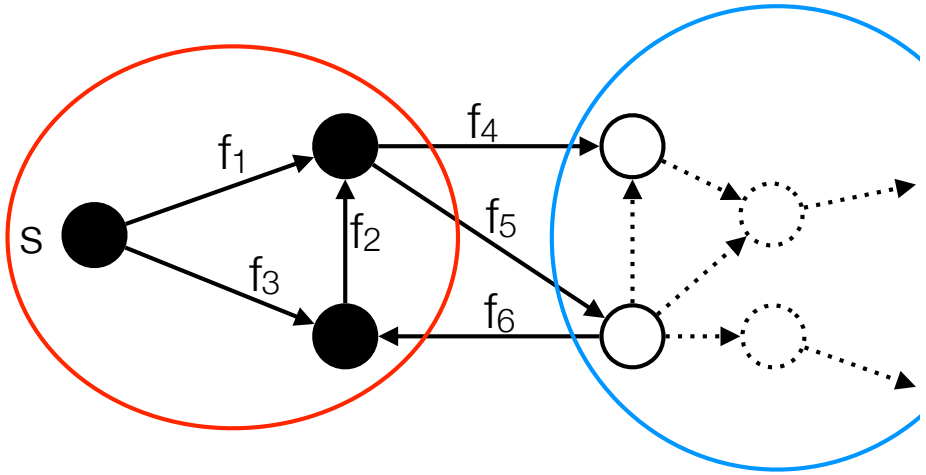


s-t Cuts

- Cut: Partition of vertices into S and T, such that s ∈ S and t ∈ T.

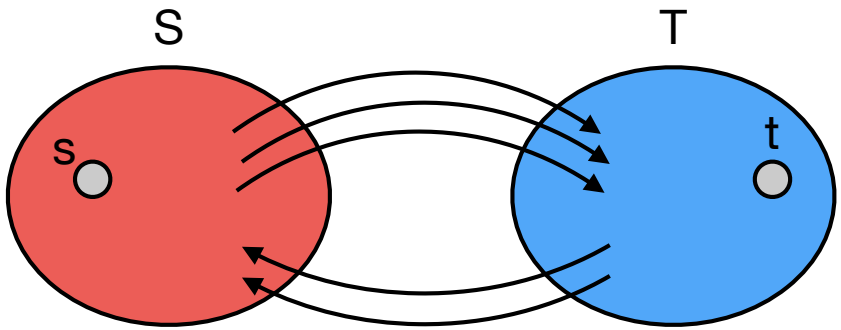


- Flow across cut = flow *from* S to T *minus flow from* T to S.
- Flow across cut: $f_4 + f_5 - f_6 = ?$
  - $f_4 + f_5 - f_1 - f_2 = 0$
  - $f_2 - f_6 - f_3 = 0$
  - $f_1 + f_3 = |f|$
  - $(f_4 + f_5 - f_1 - f_2) + (f_2 - f_6 - f_3) + (f_1 + f_3) = |f|$

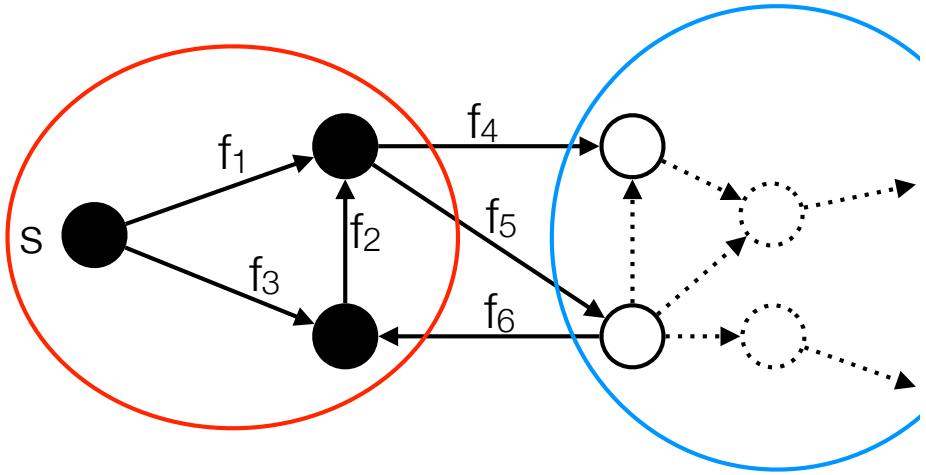


s-t Cuts

- Cut: Partition of vertices into S and T, such that s ∈ S and t ∈ T.

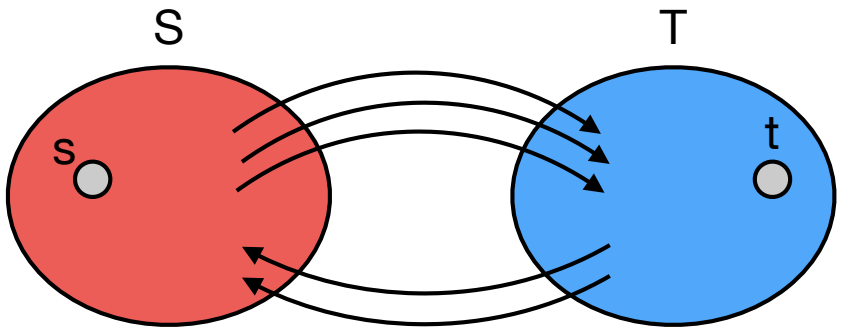


- Flow across cut = flow *from* S to T *minus flow from* T to S.
- Flow across cut: $f_4 + f_5 - f_6 = ?$
  - $f_4 + f_5 - f_1 - f_2 = 0$
  - $f_2 - f_6 - f_3 = 0$
  - $f_1 + f_3 = |f|$
  - $(f_4 + f_5 - f_1 - f_2) + (f_2 - f_6 - f_3) + (f_1 + f_3) = |f|$
  - $f_4 + f_5 - f_6 = |f|$

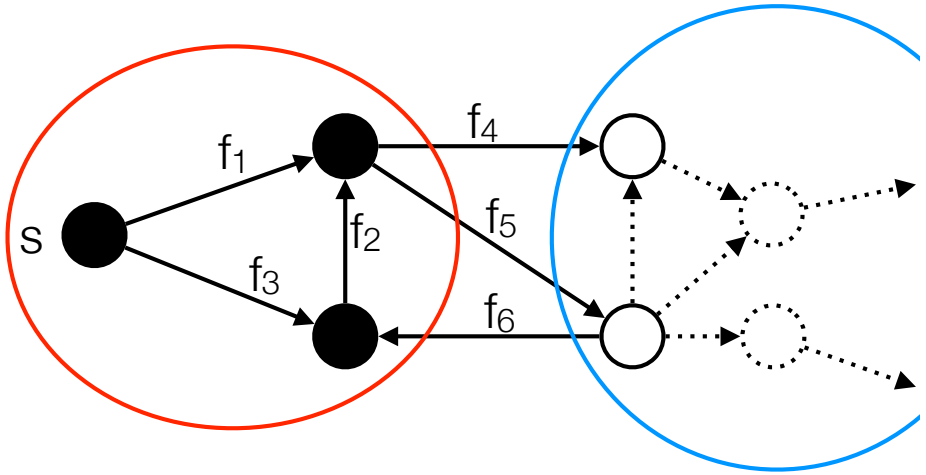


s-t Cuts

- Cut: Partition of vertices into S and T, such that s ∈ S and t ∈ T.



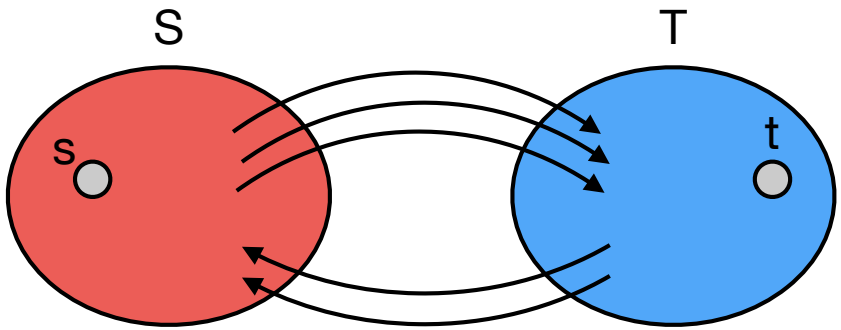
- Flow across cut = flow *from* S to T *minus flow from* T to S.
- Flow across cut: $f_4 + f_5 - f_6 = ?$
  - $f_4 + f_5 - f_1 - f_2 = 0$
  - $f_2 - f_6 - f_3 = 0$
  - $f_1 + f_3 = |f|$
  - $(f_4 + f_5 - f_1 - f_2) + (f_2 - f_6 - f_3) + (f_1 + f_3) = |f|$
  - $f_4 + f_5 - f_6 = |f|$



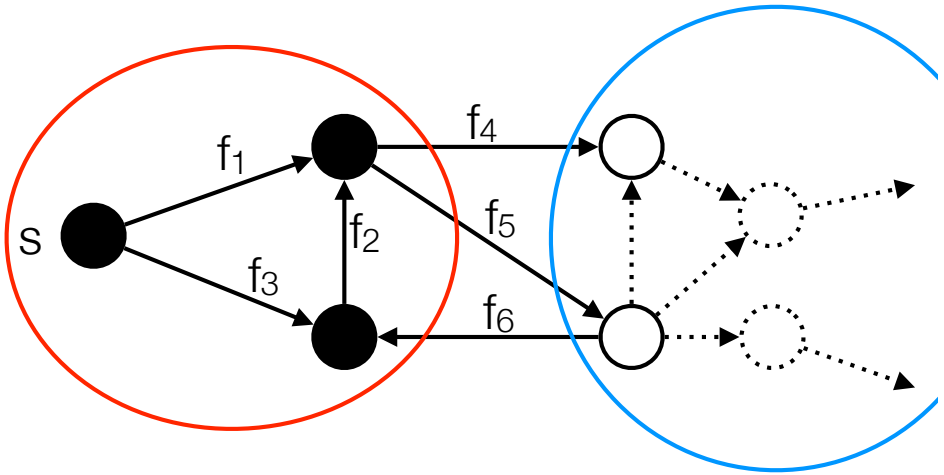
- Flow across cut is |f| for all cuts => flow out of s = flow into t.

s-t Cuts

- Cut: Partition of vertices into S and T, such that $s \in S$ and $t \in T$.

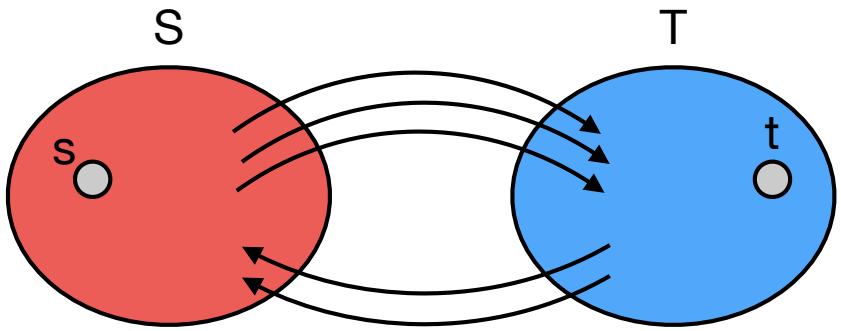


- Flow across cut is |f| for all cuts => flow out of s = flow into t.
- $|f| \leq c(S,T)$:
  - $|f| = f_4 + f_5 - f_6 \leq f_4 + f_5 \leq c_4 + c_5 = c(S,T)$



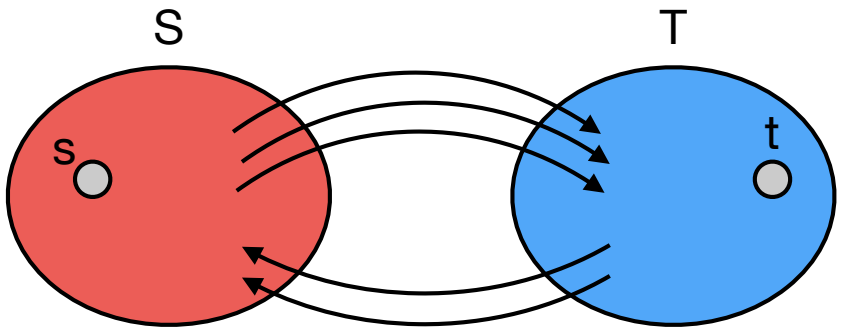
s-t Cuts

- Cut: Partition of vertices into S and T, such that $s \in S$ and $t \in T$.



s-t Cuts

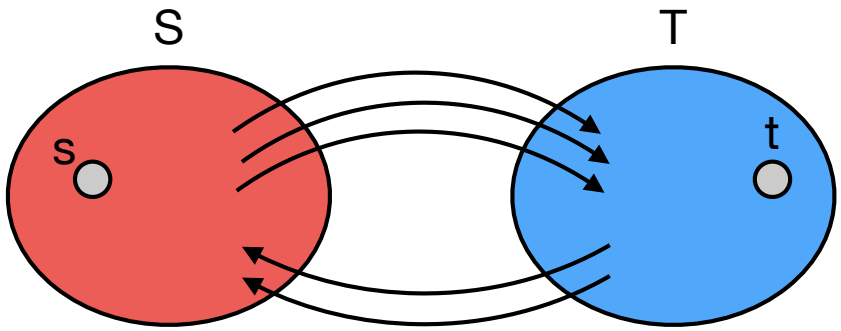
- Cut: Partition of vertices into S and T, such that $s \in S$ and $t \in T$.



- Suppose we have found flow f and cut (S,T) such that $|f| = c(S,T)$. Then f is a maximum flow and (S,T) is a minimum cut.

s-t Cuts

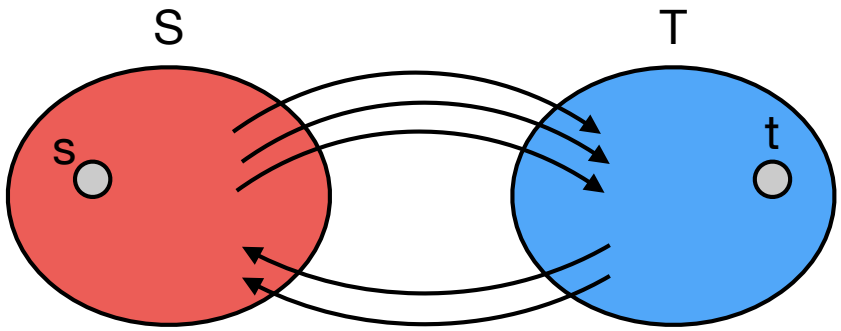
- Cut: Partition of vertices into S and T, such that $s \in S$ and $t \in T$.



- Suppose we have found flow f and cut (S,T) such that $|f| = c(S,T)$. Then f is a maximum flow and (S,T) is a minimum cut.
  - Let f^* be the maximum flow and the (S^*,T^*) minimum cut:

s-t Cuts

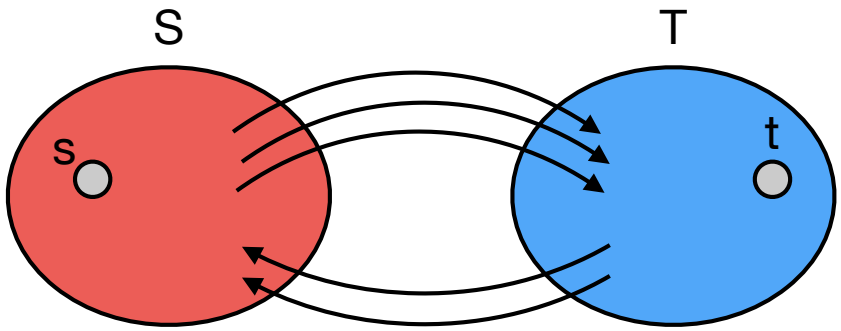
- Cut: Partition of vertices into S and T, such that $s \in S$ and $t \in T$.



- Suppose we have found flow f and cut (S,T) such that $|f| = c(S,T)$. Then f is a maximum flow and (S,T) is a minimum cut.
  - Let f^* be the maximum flow and the (S^*,T^*) minimum cut:
  - $|f| \leq |f^*| \leq c(S^*,T^*) \leq c(S,T)$.

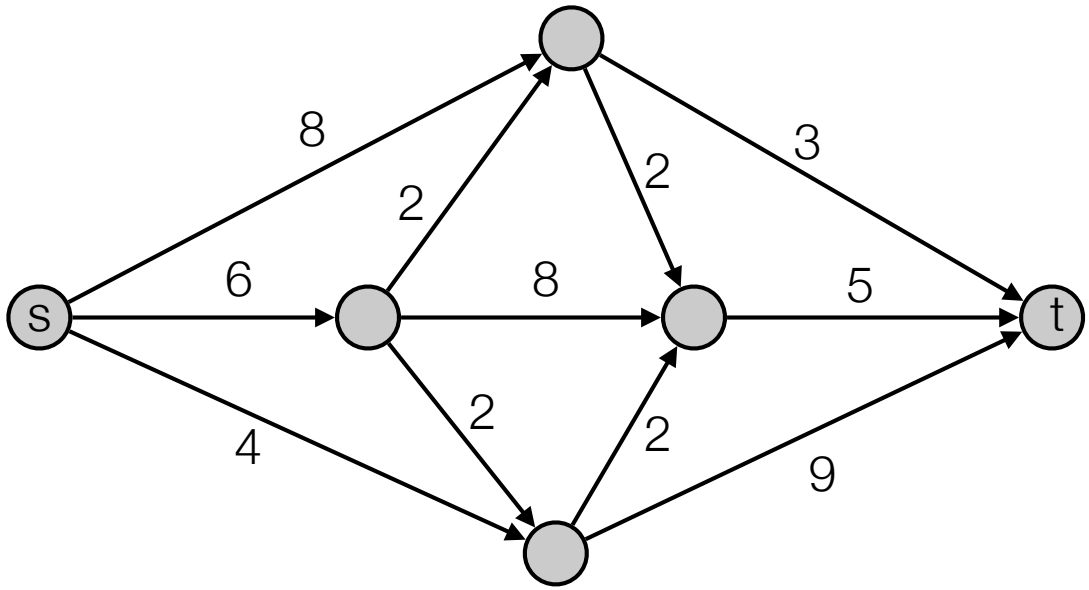
s-t Cuts

- Cut: Partition of vertices into S and T, such that $s \in S$ and $t \in T$.



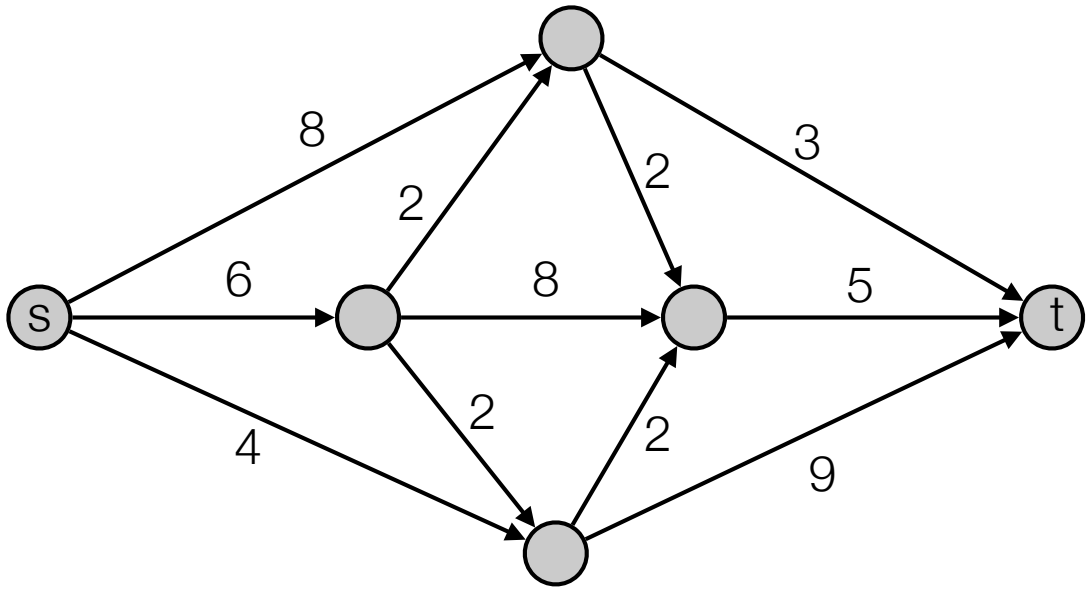
- Suppose we have found flow f and cut (S,T) such that $|f| = c(S,T)$. Then f is a maximum flow and (S,T) is a minimum cut.
  - Let f^* be the maximum flow and the (S^*,T^*) minimum cut:
  - $|f| \leq |f^*| \leq c(S^*,T^*) \leq c(S,T)$.
  - Since $|f| = c(S,T)$ this implies $|f| = |f^*|$ and $c(S,T) = c(S^*,T^*)$.

Finding minimum cuts



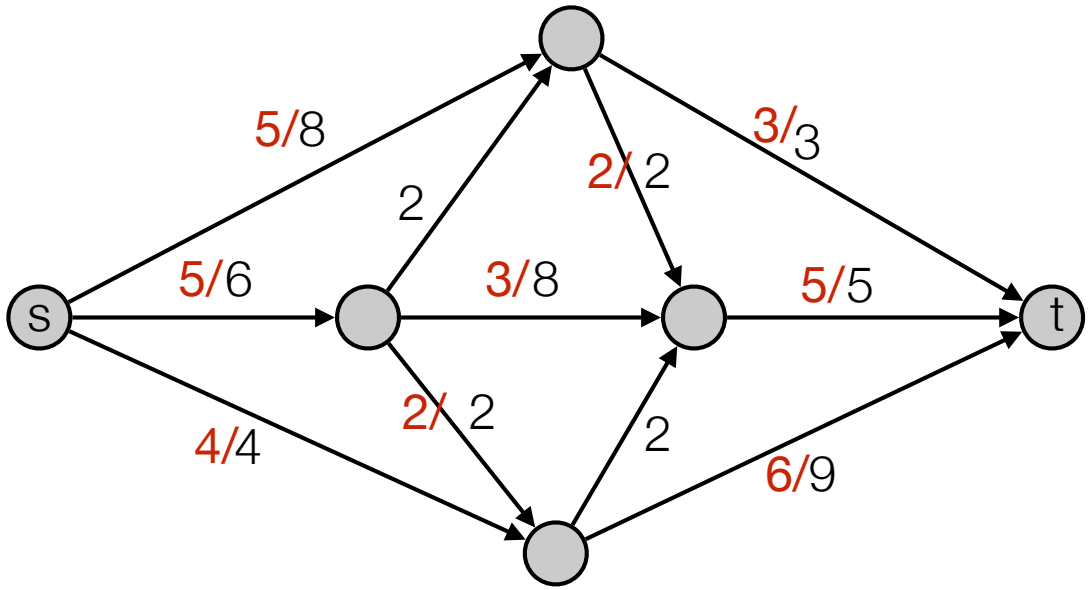
Finding minimum cuts

- Use Ford-Fulkerson to find a max-flow (finding augmenting paths).



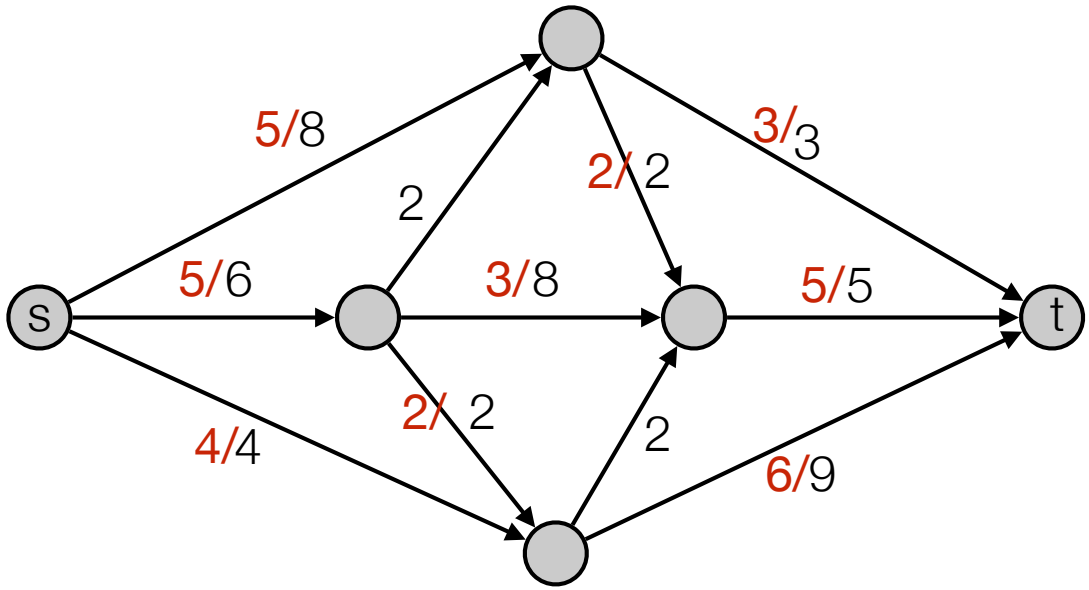
Finding minimum cuts

- Use Ford-Fulkerson to find a max-flow (finding augmenting paths).



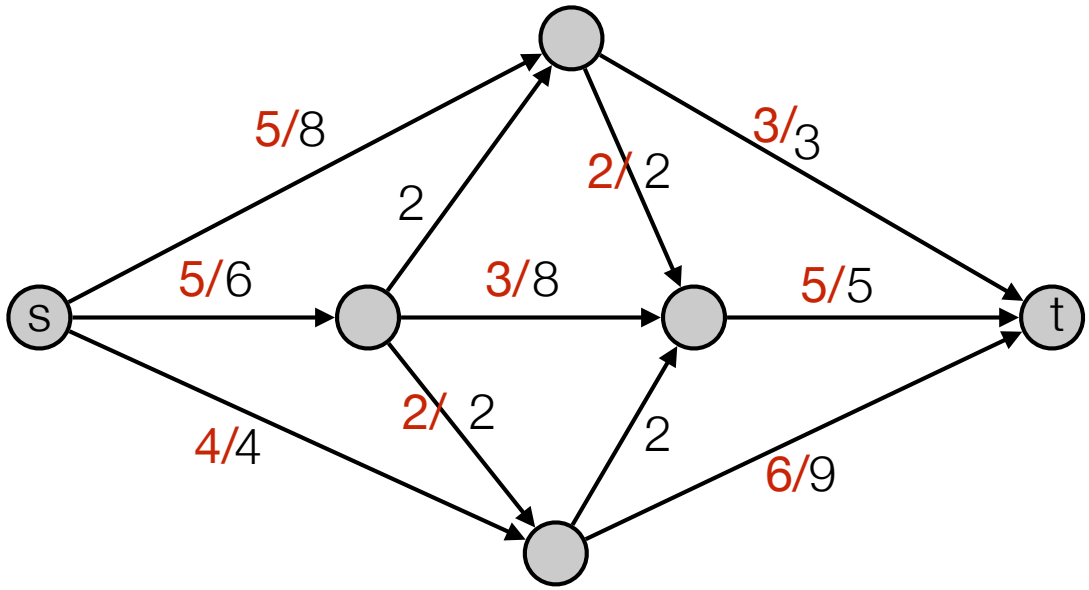
Finding minimum cuts

- Use Ford-Fulkerson to find a max-flow (finding augmenting paths).
- When no augmenting s-t path:



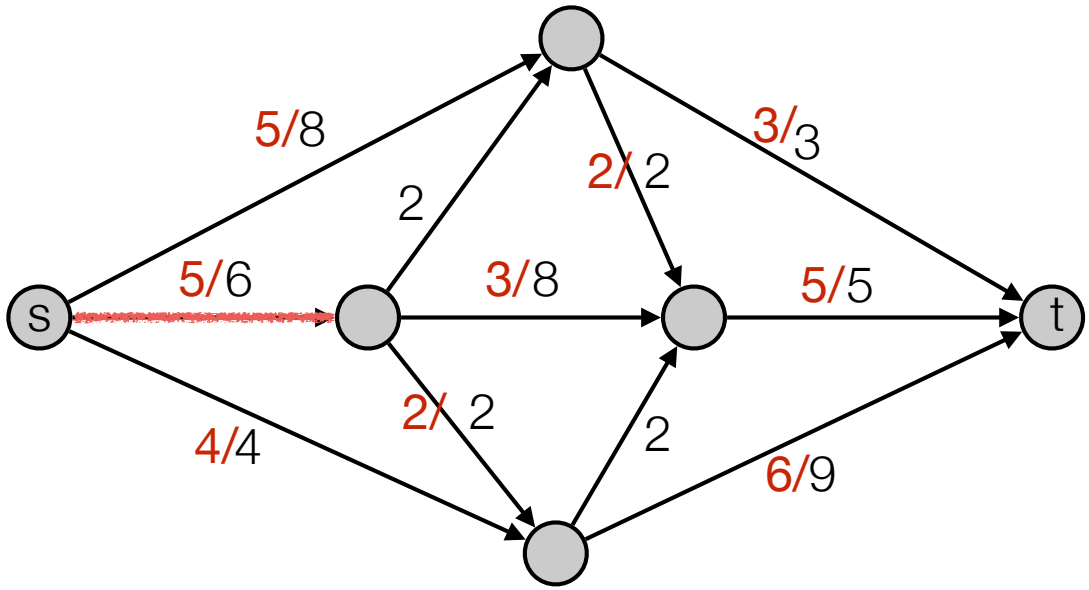
Finding minimum cuts

- Use Ford-Fulkerson to find a max-flow (finding augmenting paths).
- When no augmenting s-t path:
  - Let S be all vertices to which there exists an augmenting path from s.



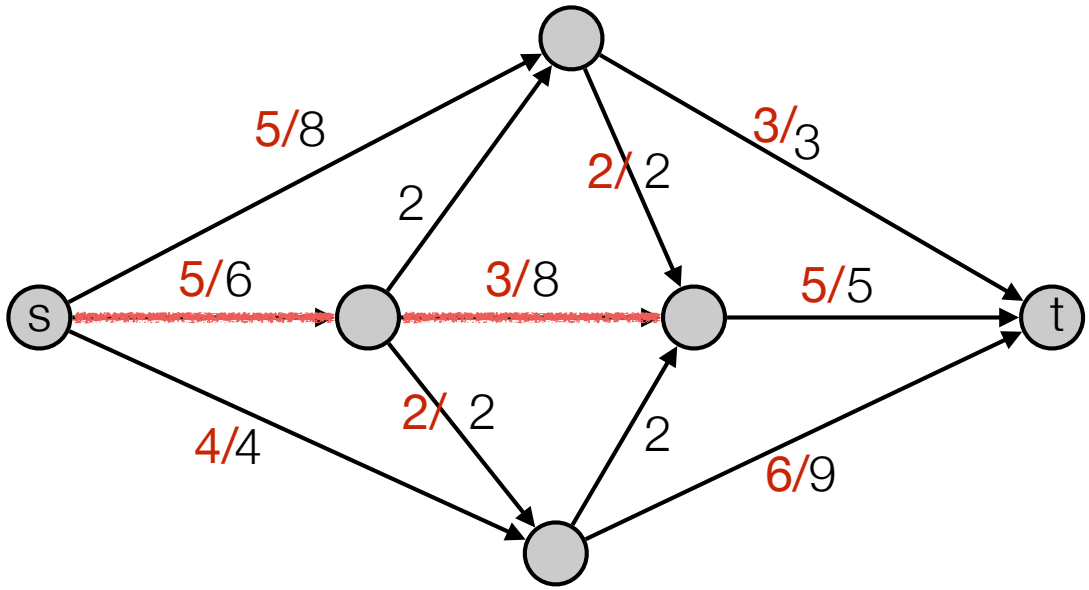
Finding minimum cuts

- Use Ford-Fulkerson to find a max-flow (finding augmenting paths).
- When no augmenting s-t path:
  - Let S be all vertices to which there exists an augmenting path from s.



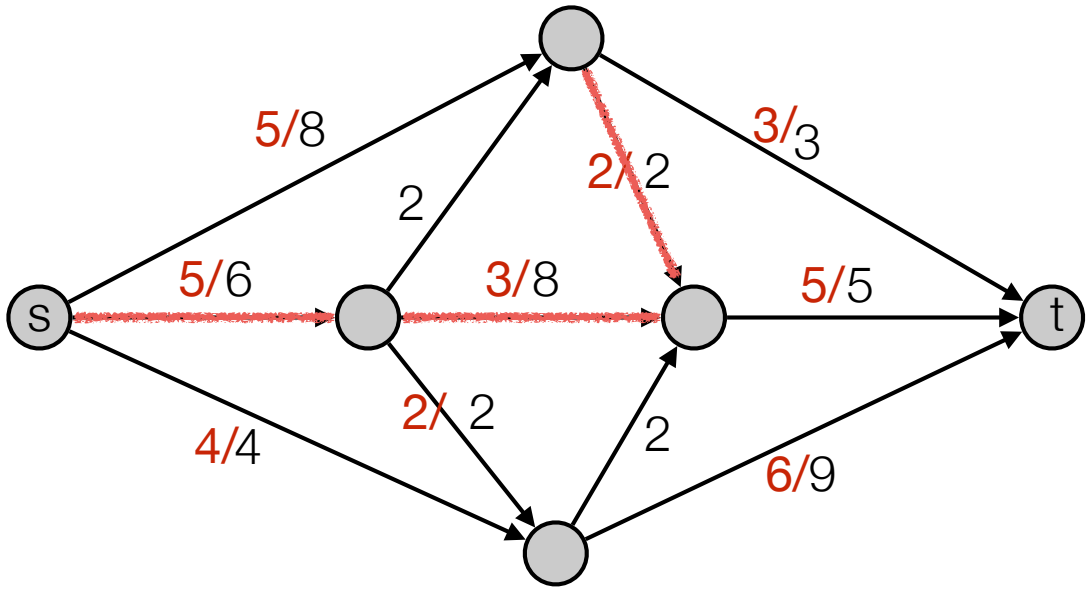
Finding minimum cuts

- Use Ford-Fulkerson to find a max-flow (finding augmenting paths).
- When no augmenting s-t path:
  - Let S be all vertices to which there exists an augmenting path from s.



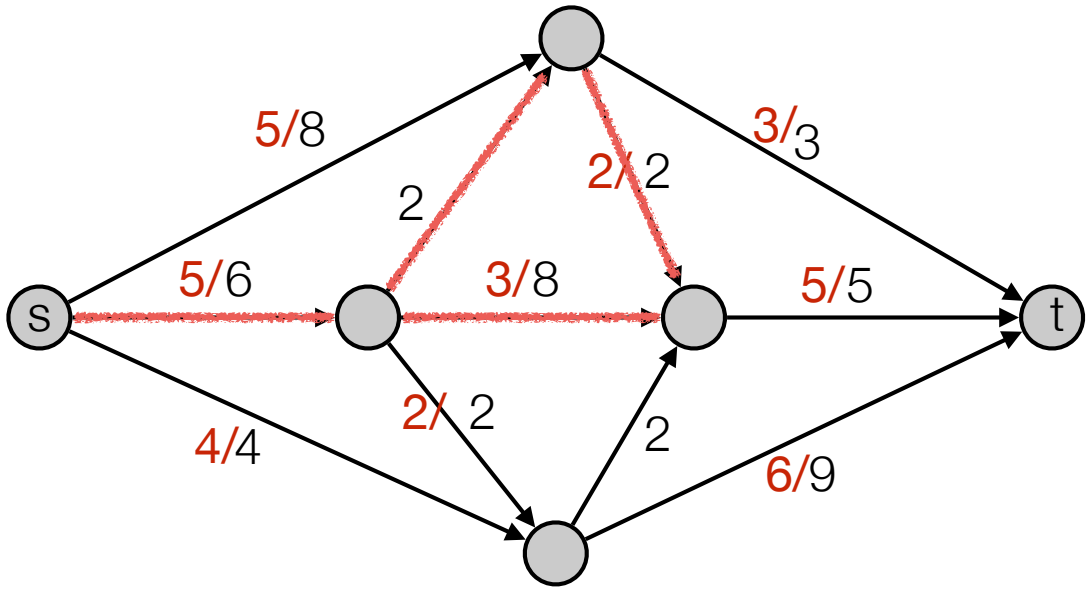
Finding minimum cuts

- Use Ford-Fulkerson to find a max-flow (finding augmenting paths).
- When no augmenting s-t path:
  - Let S be all vertices to which there exists an augmenting path from s.



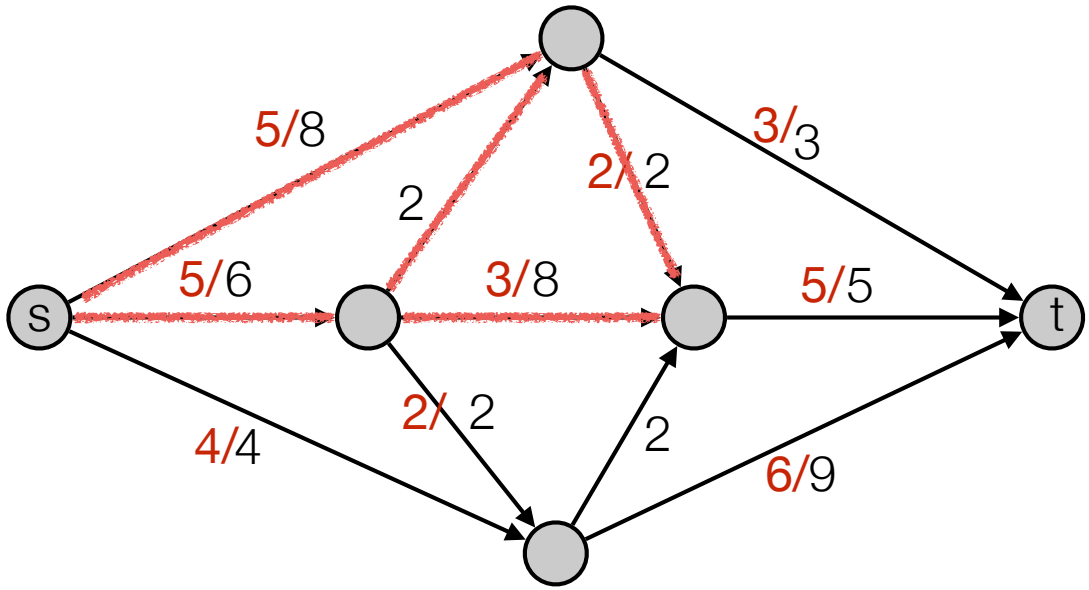
Finding minimum cuts

- Use Ford-Fulkerson to find a max-flow (finding augmenting paths).
- When no augmenting s-t path:
  - Let S be all vertices to which there exists an augmenting path from s.



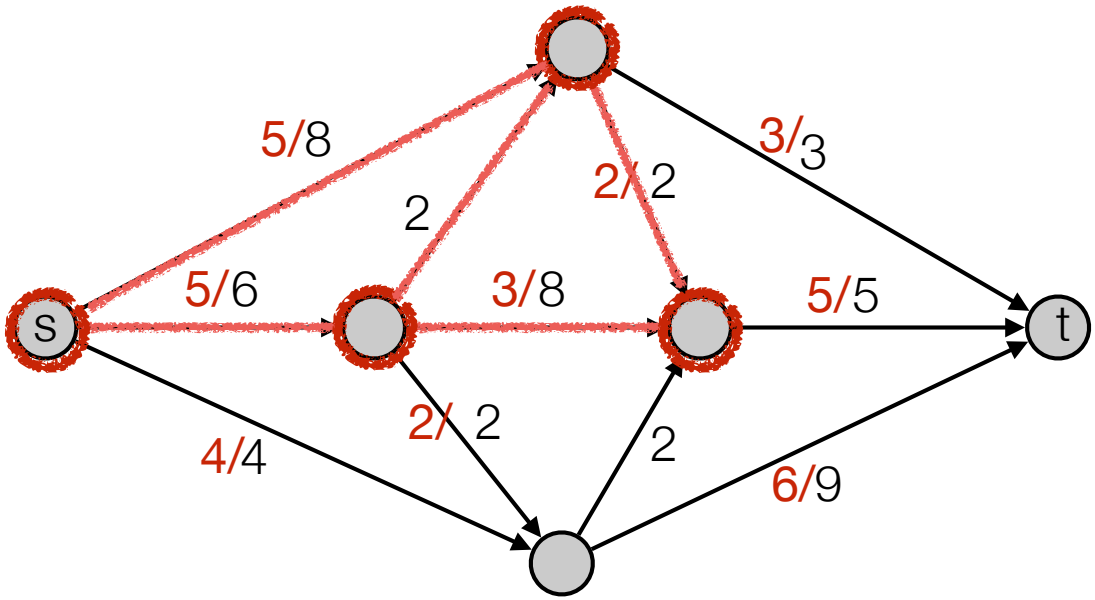
Finding minimum cuts

- Use Ford-Fulkerson to find a max-flow (finding augmenting paths).
- When no augmenting s-t path:
  - Let S be all vertices to which there exists an augmenting path from s.



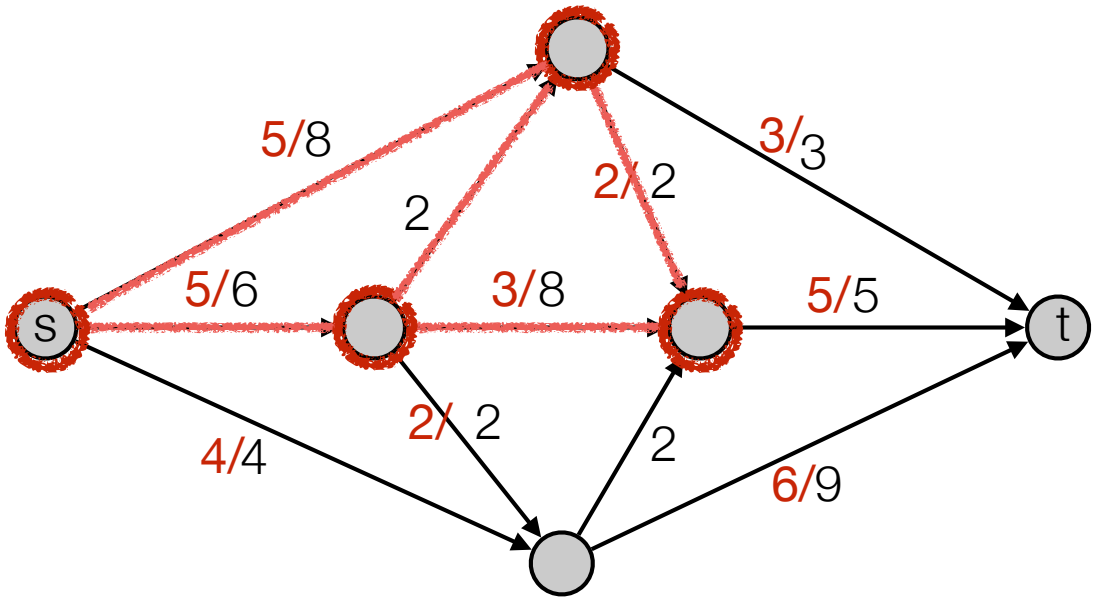
Finding minimum cuts

- Use Ford-Fulkerson to find a max-flow (finding augmenting paths).
- When no augmenting s-t path:
  - Let S be all vertices to which there exists an augmenting path from s.



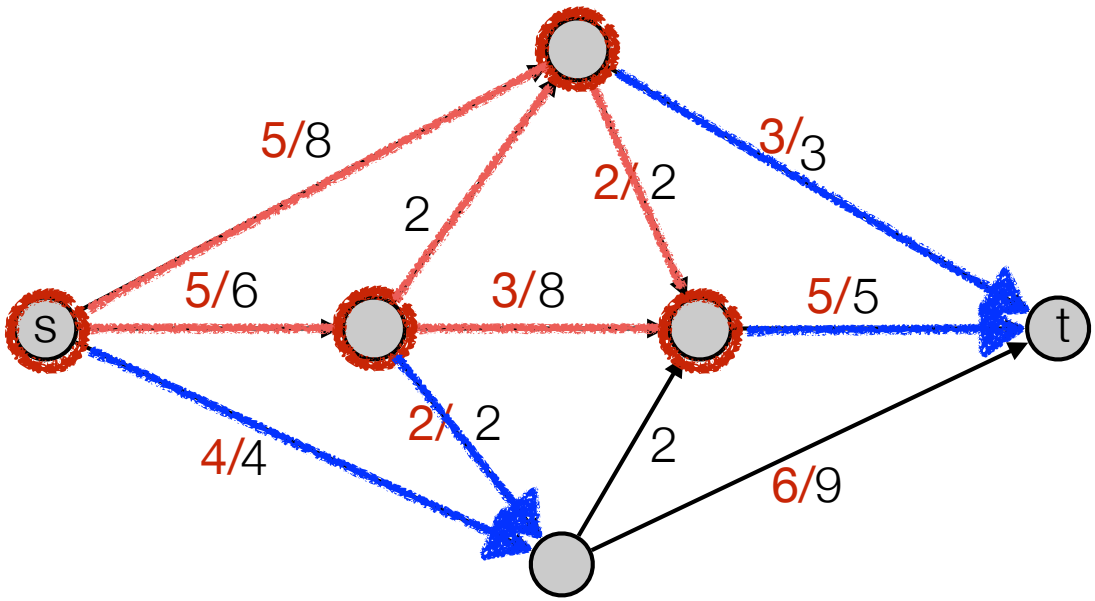
Finding minimum cuts

- Use Ford-Fulkerson to find a max-flow (finding augmenting paths).
- When no augmenting s-t path:
  - Let S be all vertices to which there exists an augmenting path from s.
  - value of flow (S,T) = capacity of the cut:



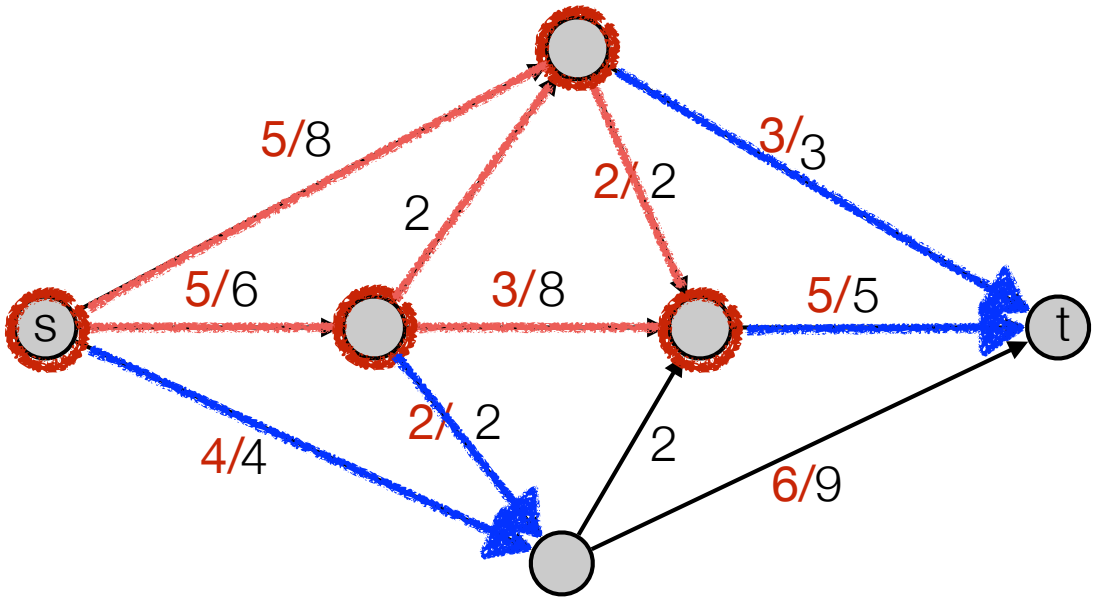
Finding minimum cuts

- Use Ford-Fulkerson to find a max-flow (finding augmenting paths).
- When no augmenting s-t path:
  - Let S be all vertices to which there exists an augmenting path from s.
  - value of flow (S,T) = capacity of the cut:



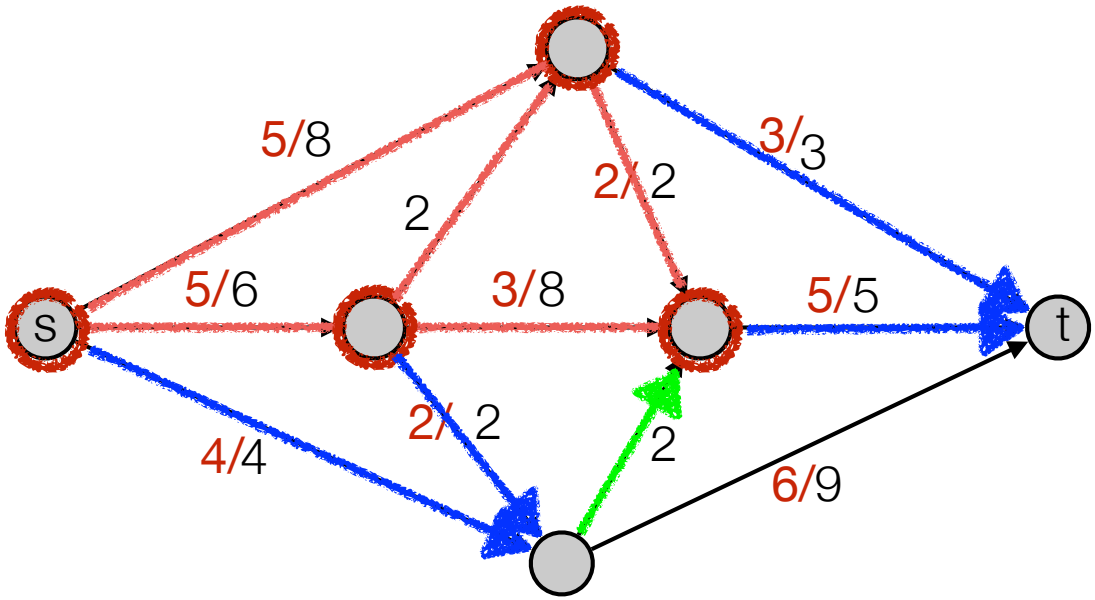
Finding minimum cuts

- Use Ford-Fulkerson to find a max-flow (finding augmenting paths).
- When no augmenting s-t path:
  - Let S be all vertices to which there exists an augmenting path from s.
  - value of flow (S,T) = capacity of the cut:
    - All forward edges in the minimum cut are “full” (flow = capacity).



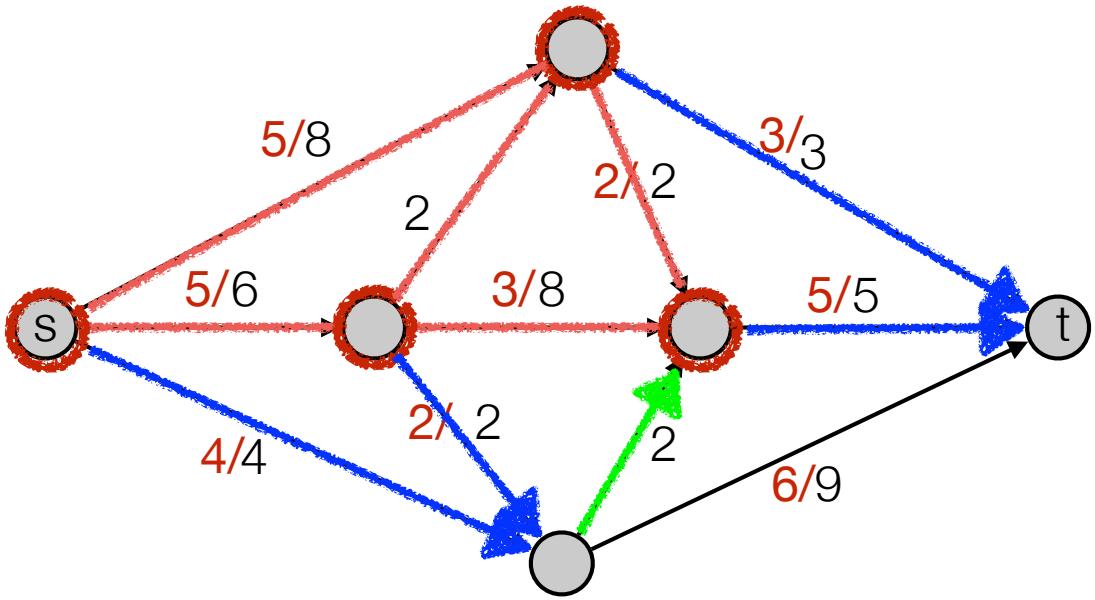
Finding minimum cuts

- Use Ford-Fulkerson to find a max-flow (finding augmenting paths).
- When no augmenting s-t path:
  - Let S be all vertices to which there exists an augmenting path from s.
  - value of flow (S,T) = capacity of the cut:
    - All forward edges in the minimum cut are “full” (flow = capacity).



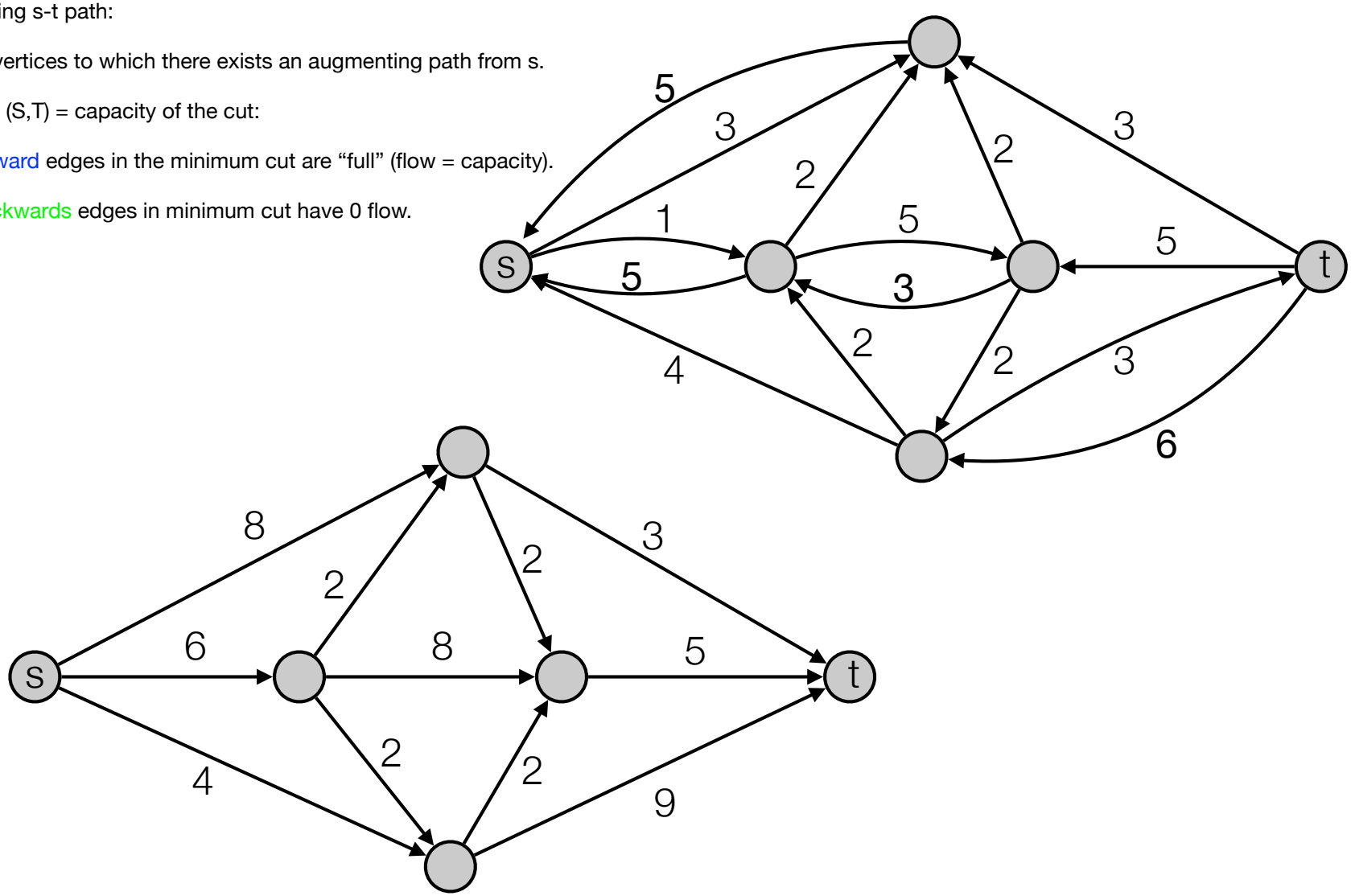
Finding minimum cuts

- Use Ford-Fulkerson to find a max-flow (finding augmenting paths).
- When no augmenting s-t path:
  - Let S be all vertices to which there exists an augmenting path from s.
  - value of flow (S,T) = capacity of the cut:
    - All forward edges in the minimum cut are “full” (flow = capacity).
    - All backwards edges in minimum cut have 0 flow.



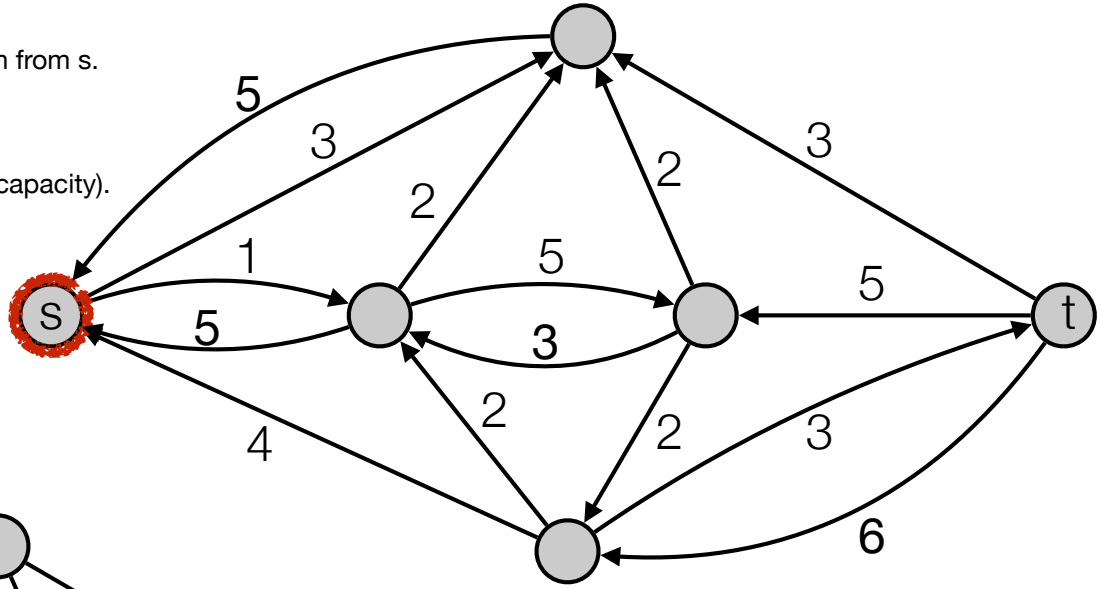
Finding minimum cuts (with residual network).

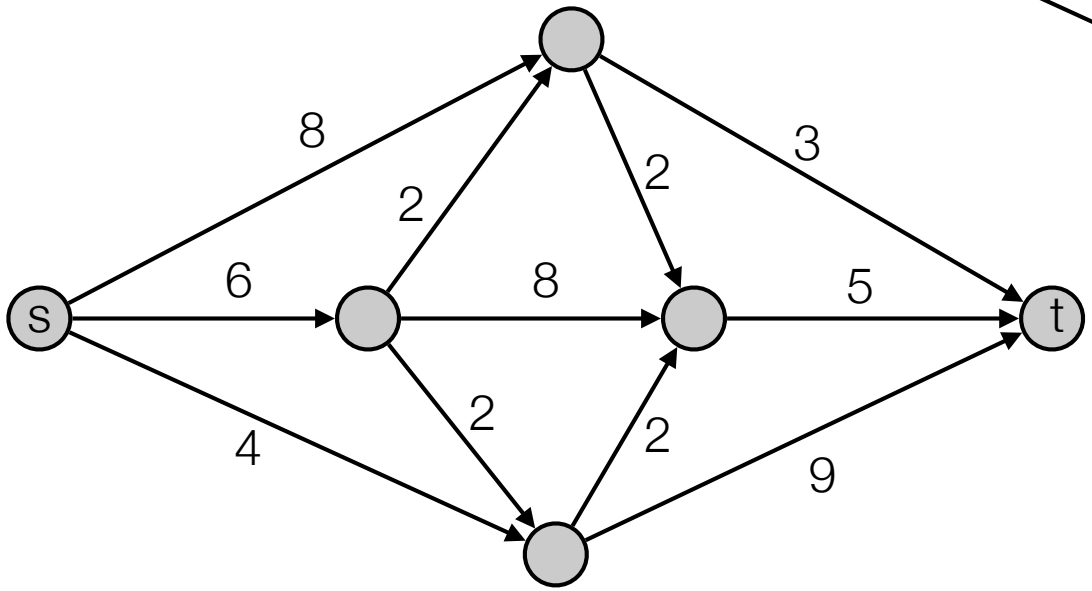
- Use Ford-Fulkerson to find a max-flow (finding augmenting paths).
- When no augmenting s-t path:
  - Let S be all vertices to which there exists an augmenting path from s.
  - value of flow (S,T) = capacity of the cut:
    - All forward edges in the minimum cut are “full” (flow = capacity).
    - All backwards edges in minimum cut have 0 flow.



Finding minimum cuts (with residual network).

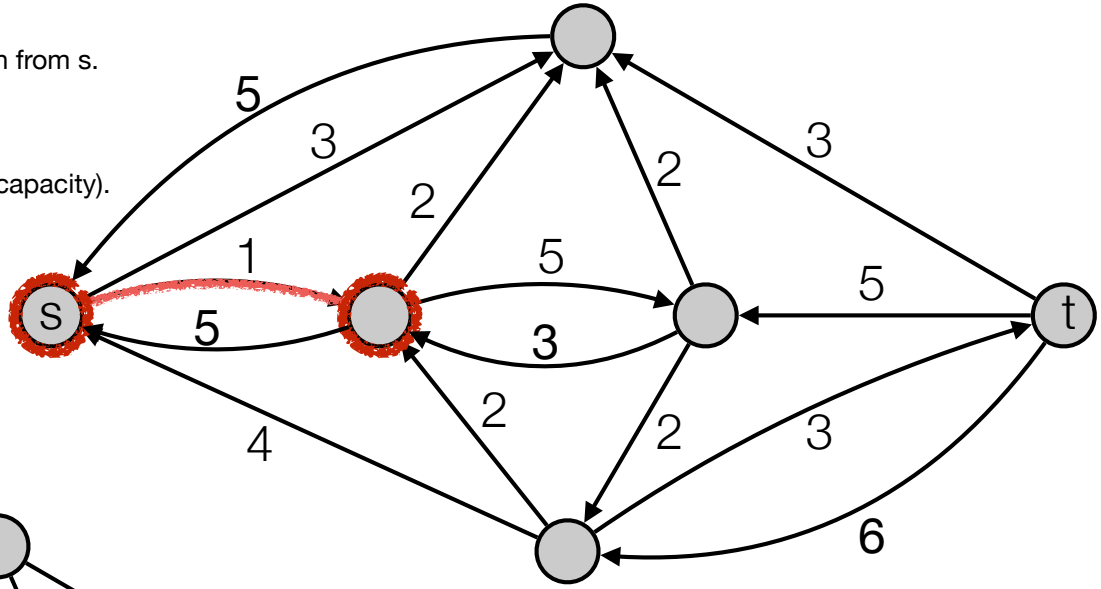
- Use Ford-Fulkerson to find a max-flow (finding augmenting paths).
- When no augmenting s-t path:
  - Let S be all vertices to which there exists an augmenting path from s.
  - value of flow (S,T) = capacity of the cut:
    - All forward edges in the minimum cut are “full” (flow = capacity).
    - All backwards edges in minimum cut have 0 flow.

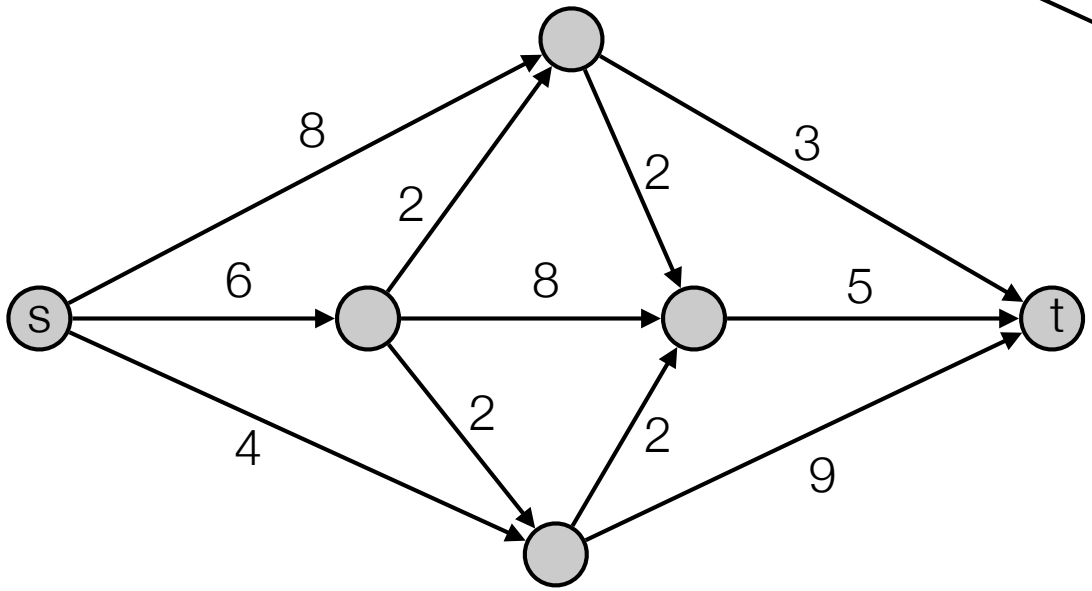




Finding minimum cuts (with residual network).

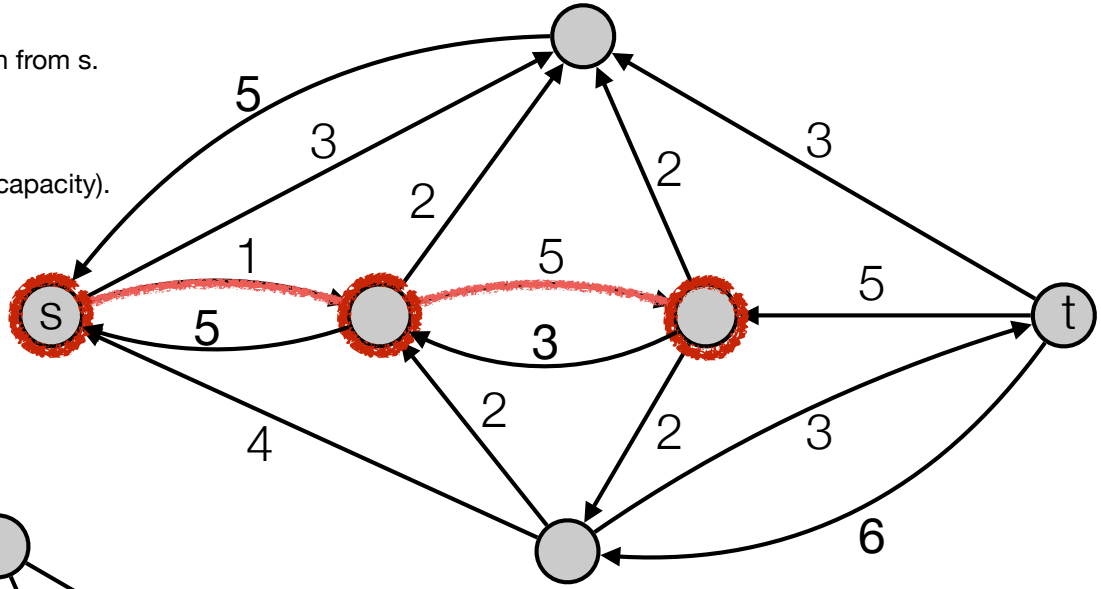
- Use Ford-Fulkerson to find a max-flow (finding augmenting paths).
- When no augmenting s-t path:
  - Let S be all vertices to which there exists an augmenting path from s.
  - value of flow (S,T) = capacity of the cut:
    - All forward edges in the minimum cut are “full” (flow = capacity).
    - All backwards edges in minimum cut have 0 flow.

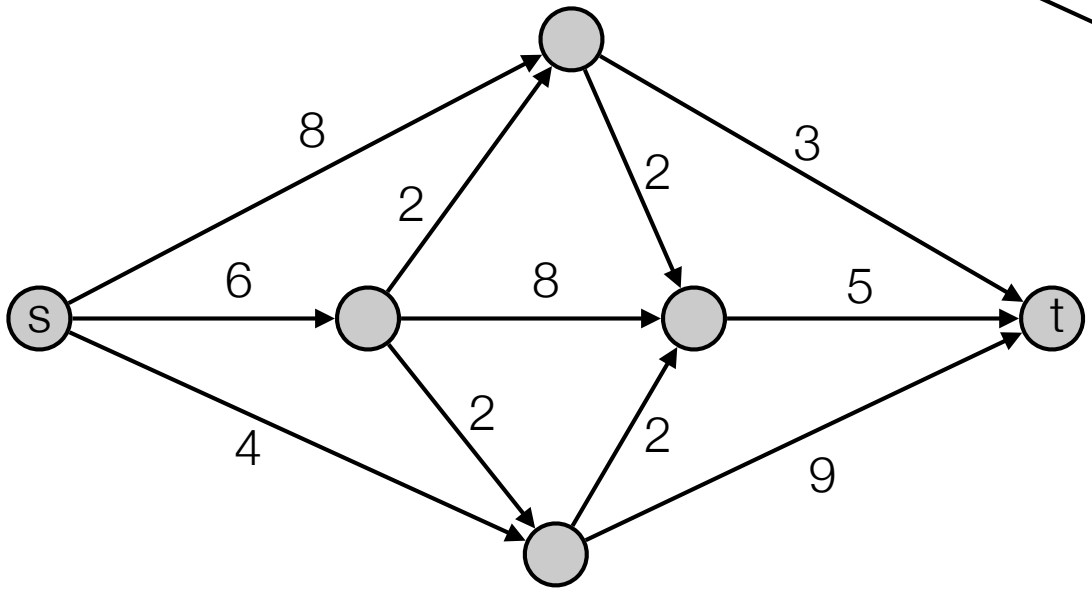




Finding minimum cuts (with residual network).

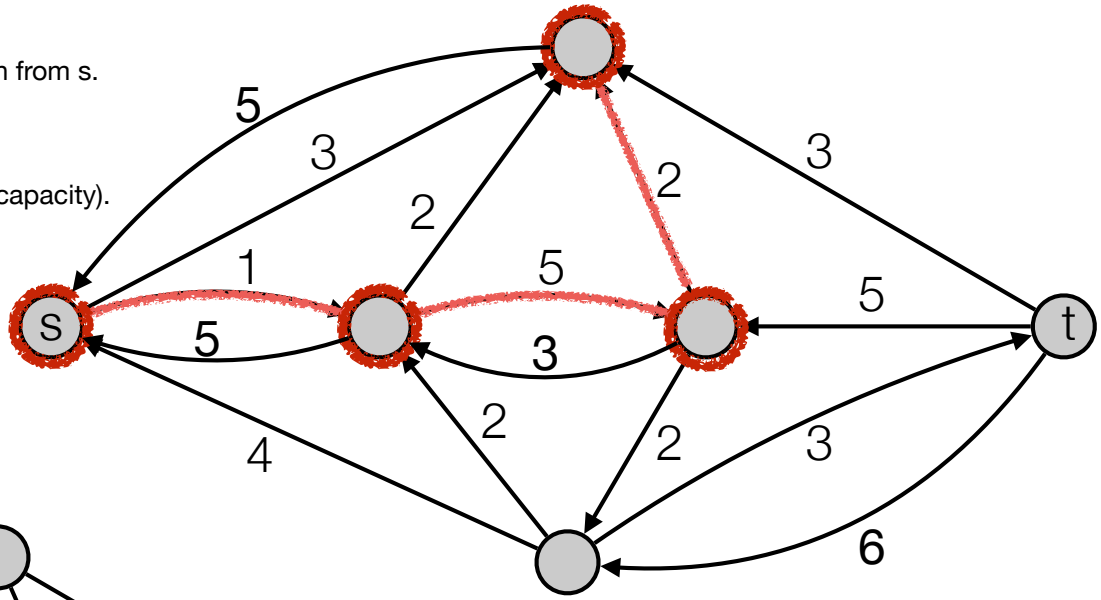
- Use Ford-Fulkerson to find a max-flow (finding augmenting paths).
- When no augmenting s-t path:
  - Let S be all vertices to which there exists an augmenting path from s.
  - value of flow (S,T) = capacity of the cut:
    - All forward edges in the minimum cut are “full” (flow = capacity).
    - All backwards edges in minimum cut have 0 flow.

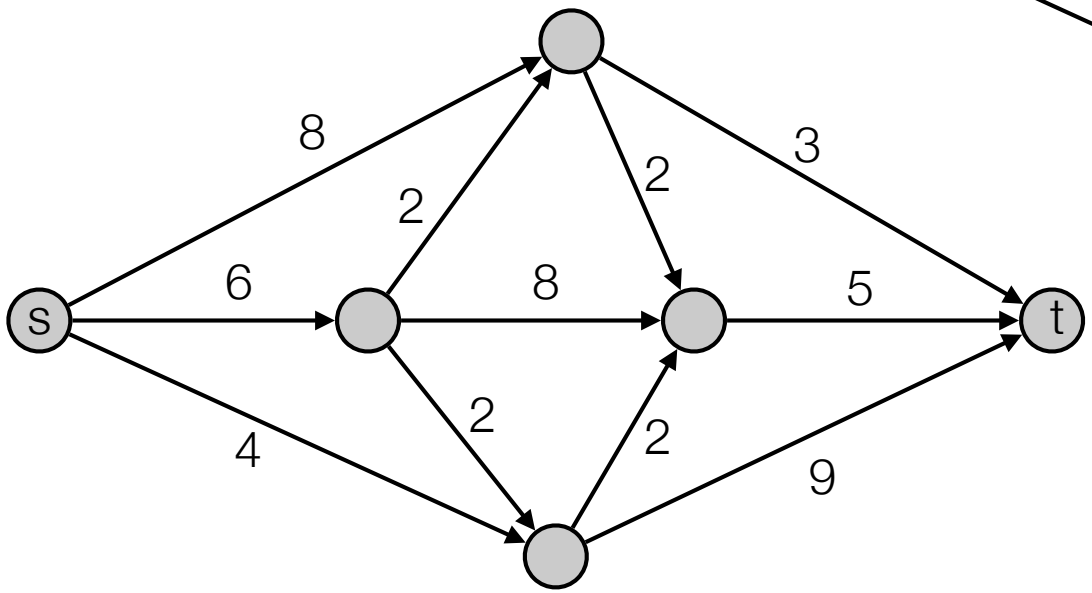




Finding minimum cuts (with residual network).

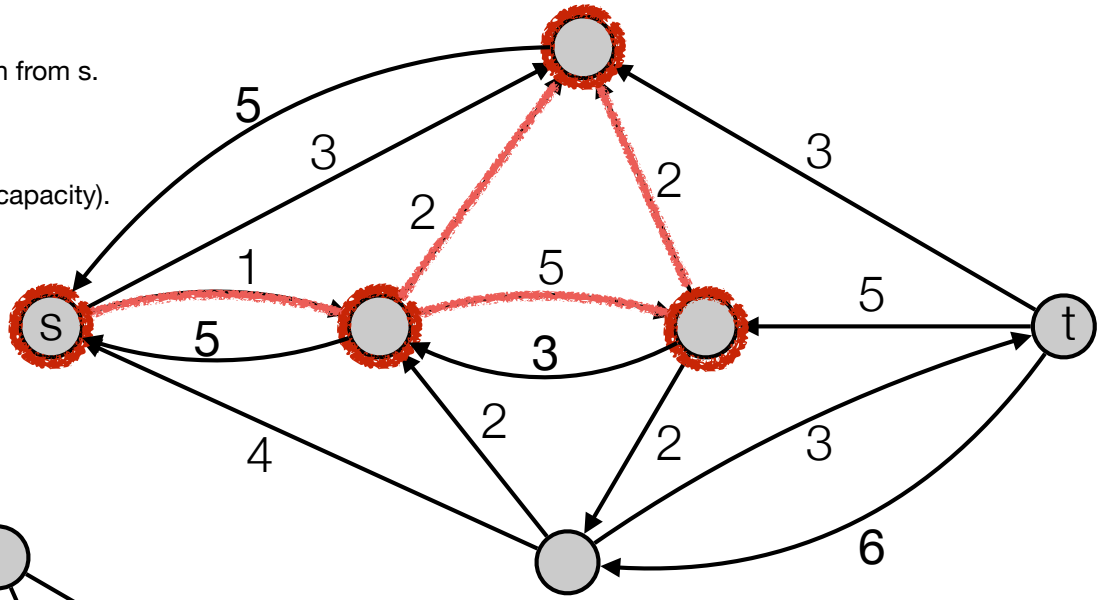
- Use Ford-Fulkerson to find a max-flow (finding augmenting paths).
- When no augmenting s-t path:
  - Let S be all vertices to which there exists an augmenting path from s.
  - value of flow (S,T) = capacity of the cut:
    - All forward edges in the minimum cut are “full” (flow = capacity).
    - All backwards edges in minimum cut have 0 flow.

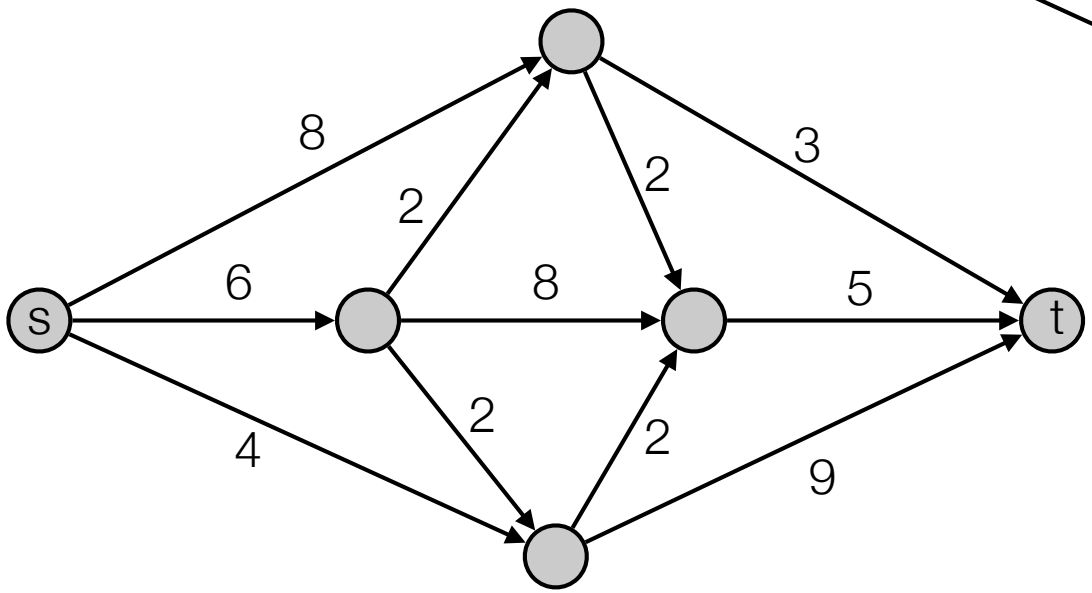




Finding minimum cuts (with residual network).

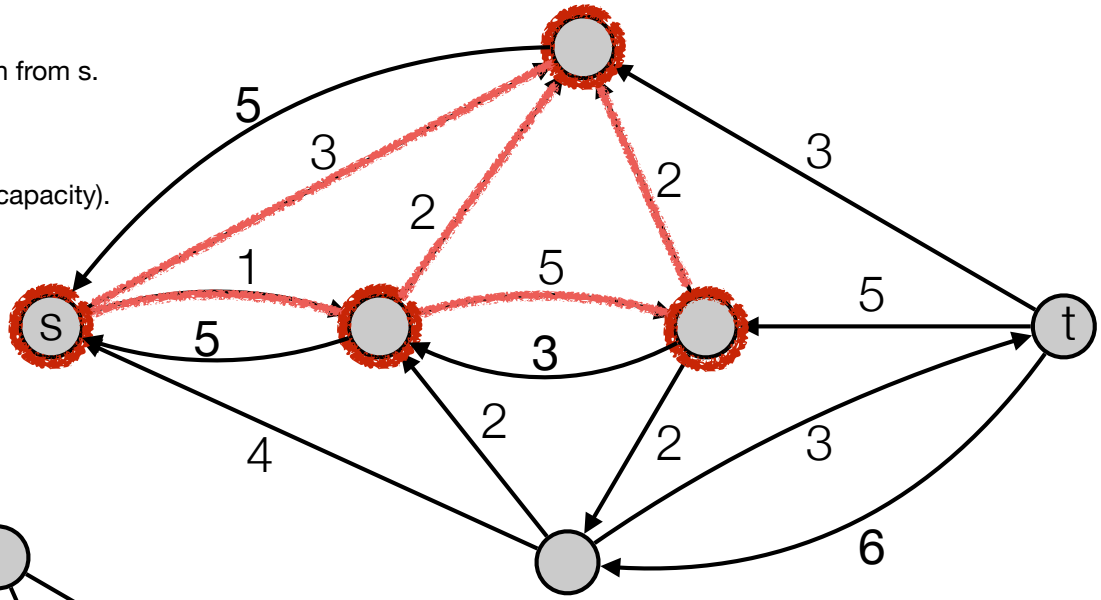
- Use Ford-Fulkerson to find a max-flow (finding augmenting paths).
- When no augmenting s-t path:
  - Let S be all vertices to which there exists an augmenting path from s.
  - value of flow (S,T) = capacity of the cut:
    - All forward edges in the minimum cut are “full” (flow = capacity).
    - All backwards edges in minimum cut have 0 flow.

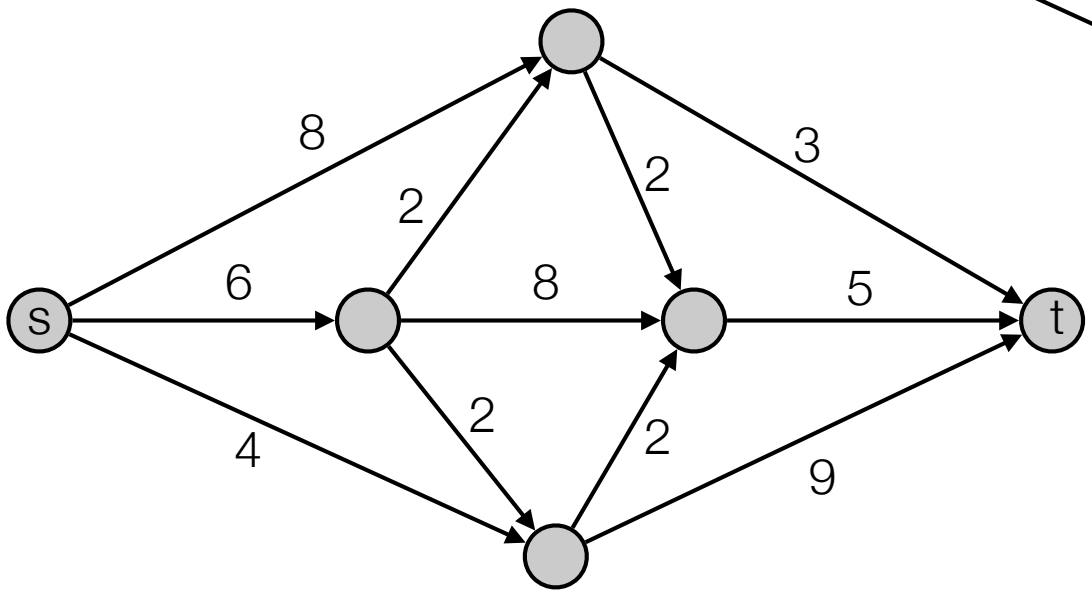




Finding minimum cuts (with residual network).

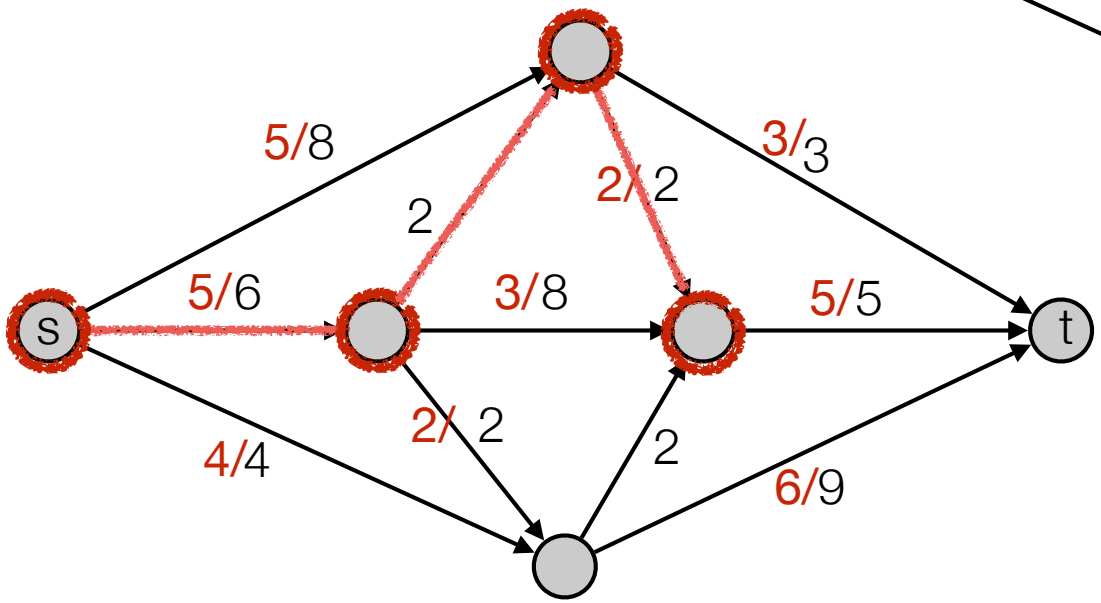
- Use Ford-Fulkerson to find a max-flow (finding augmenting paths).
- When no augmenting s-t path:
  - Let S be all vertices to which there exists an augmenting path from s.
  - value of flow (S,T) = capacity of the cut:
    - All forward edges in the minimum cut are “full” (flow = capacity).
    - All backwards edges in minimum cut have 0 flow.

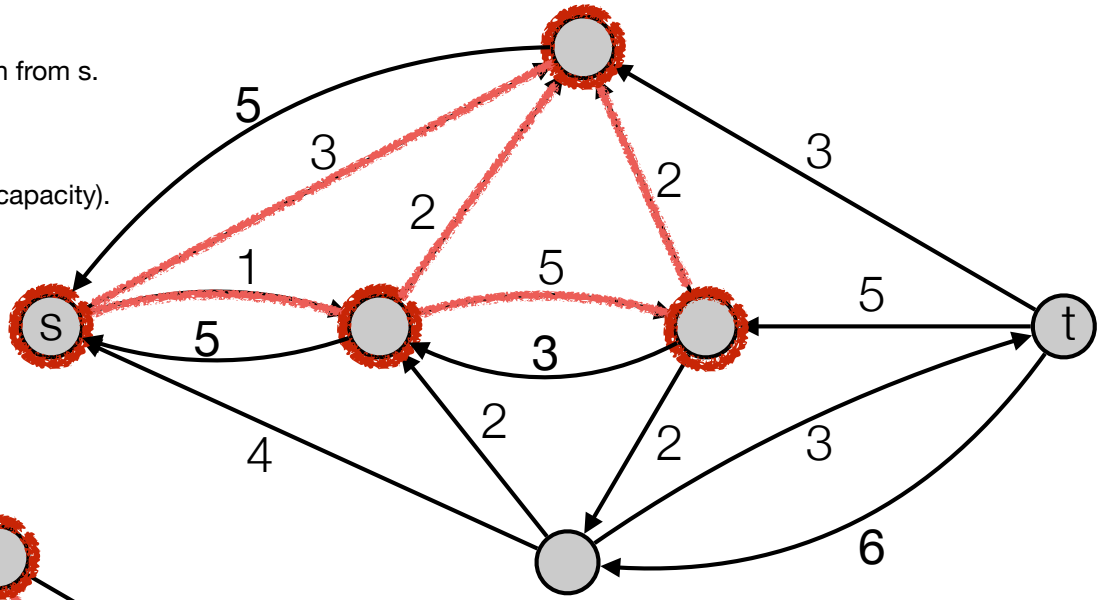




Finding minimum cuts (with residual network).

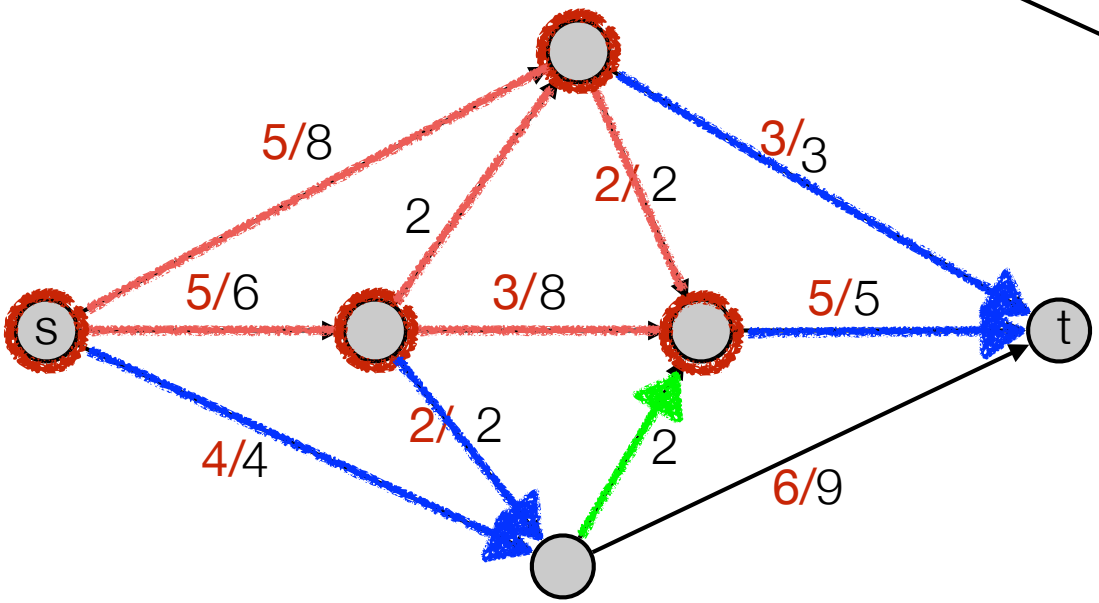
- Use Ford-Fulkerson to find a max-flow (finding augmenting paths).
- When no augmenting s-t path:
  - Let S be all vertices to which there exists an augmenting path from s.
  - value of flow (S,T) = capacity of the cut:
    - All forward edges in the minimum cut are “full” (flow = capacity).
    - All backwards edges in minimum cut have 0 flow.

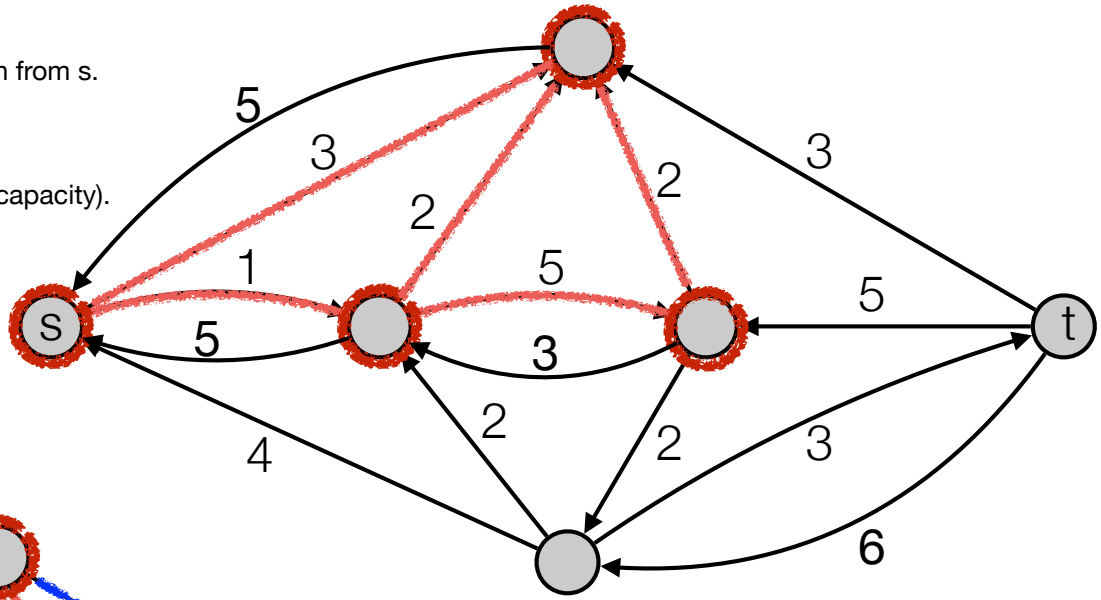




Finding minimum cuts (with residual network).

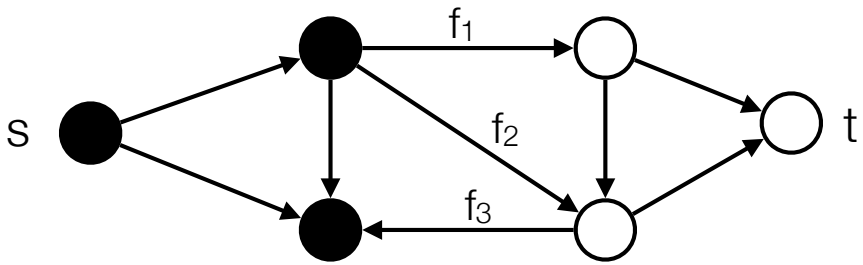
- Use Ford-Fulkerson to find a max-flow (finding augmenting paths).
- When no augmenting s-t path:
  - Let S be all vertices to which there exists an augmenting path from s.
  - value of flow (S,T) = capacity of the cut:
    - All forward edges in the minimum cut are “full” (flow = capacity).
    - All backwards edges in minimum cut have 0 flow.





Use of Max-flow min-cut theorem

- There is no augmenting path <=> f is a maximum flow.
  - *f maximum flow => no augmenting path:*
    - Show that exists augmenting path => f not maximum flow.
  - *no augmenting path => f maximum flow*
    - no augmenting path => exists cut (S,T) where $|f| = c(S,T)$:
      - Let S be all vertices to which there exists an augmenting path from s.
      - t not in S (since there is no augmenting s-t path).
      - Edges from S to T: $f_1 = c_1$ and $f_2 = c_2$.
      - Edges from T to S: $f_3 = 0$.
      - => $|f| = f_1 + f_2 - f_3 = f_1 + f_2 = c_1 + c_2 = c(S,T)$.
      - => f a maximum flow and (S,T) a minimum cut.



Removing assumptions

- Edges into s and out of t:

$$v(f) = f^{out}(s) - f^{in}(s)$$

- Capacities not integers.

Network Flow

- Multiple sources and sinks: