Network Flows

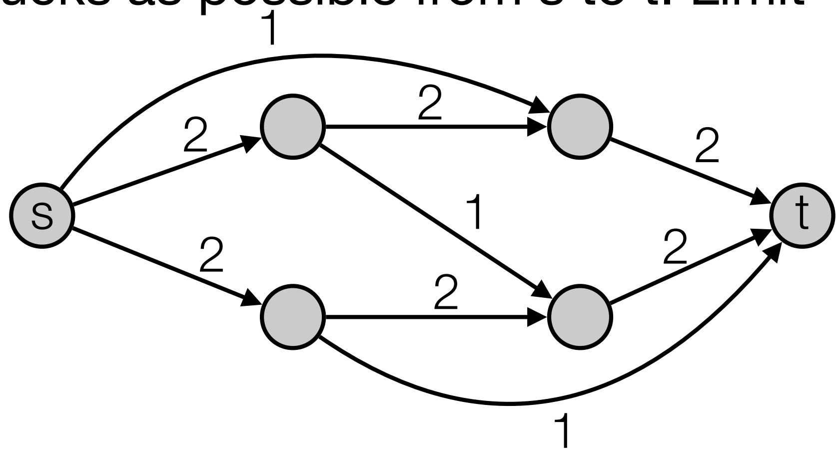
Inge Li Gørtz

CLRS Chapter 26.0-26.2

Network Flow

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

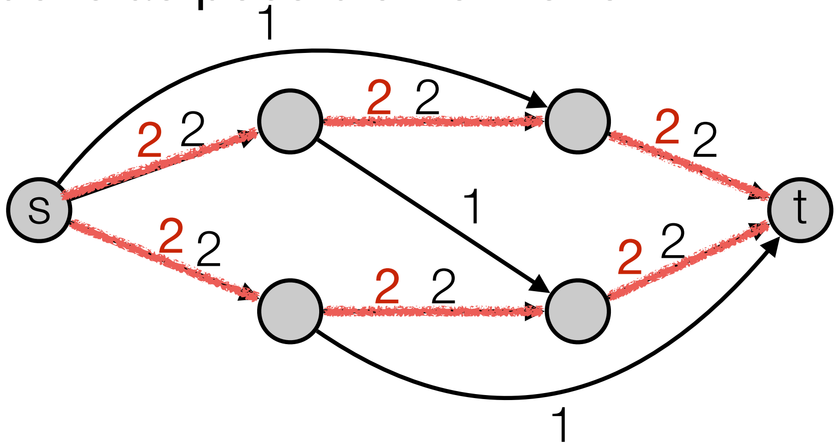
- Example 1:



Network Flow

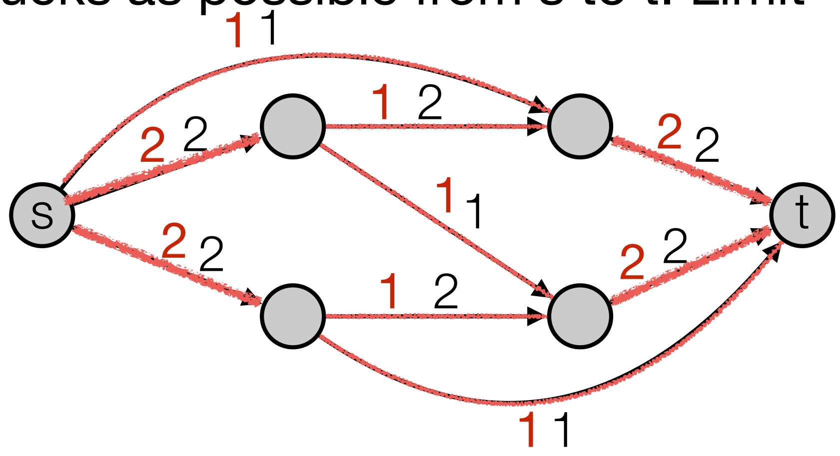
- Truck company: Wants to send as many trucks as possible from s to t. Limit of number of trucks on each road.

- 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.

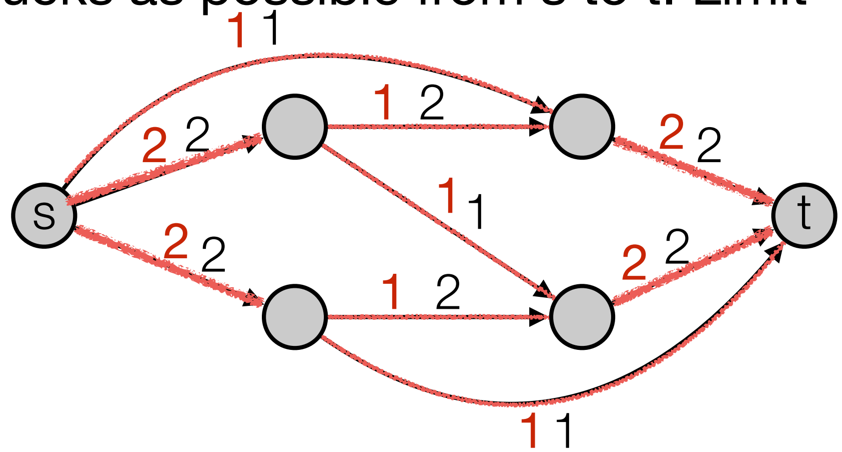


- Example 1:
  - Solution 1: 4 trucks
  - Solution 2: 5 trucks

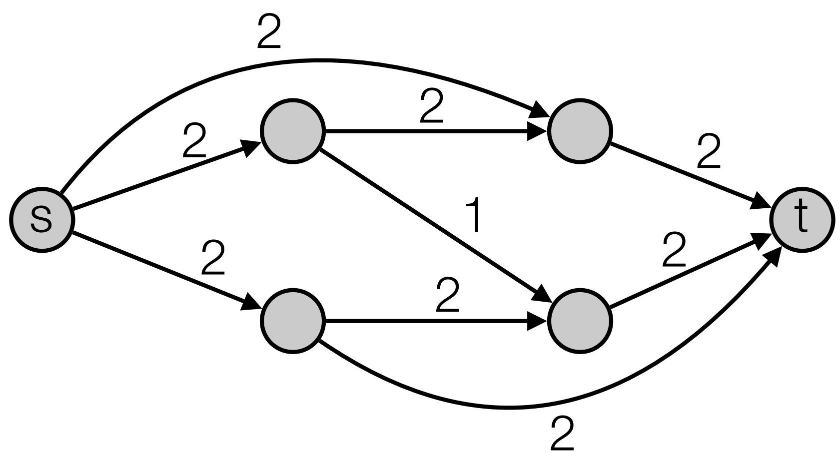
Network Flow

- 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

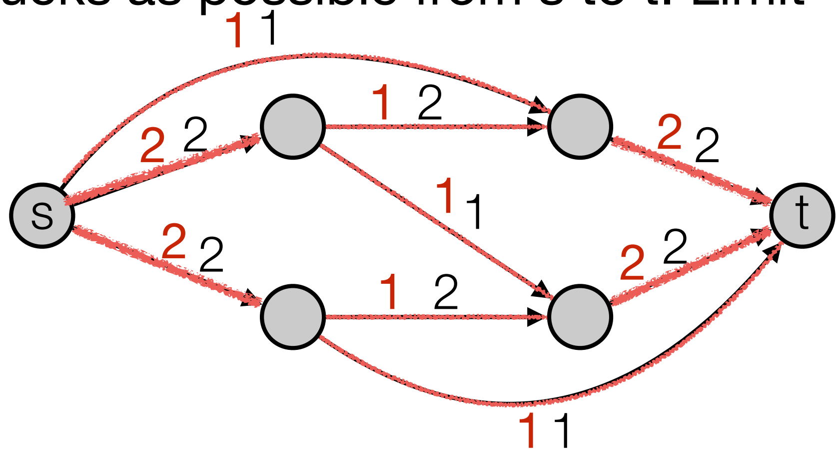


- Example 2:



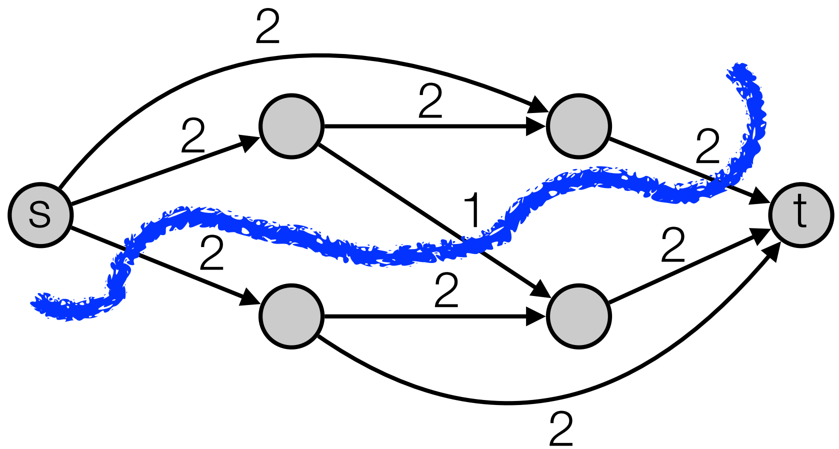
Network Flow

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



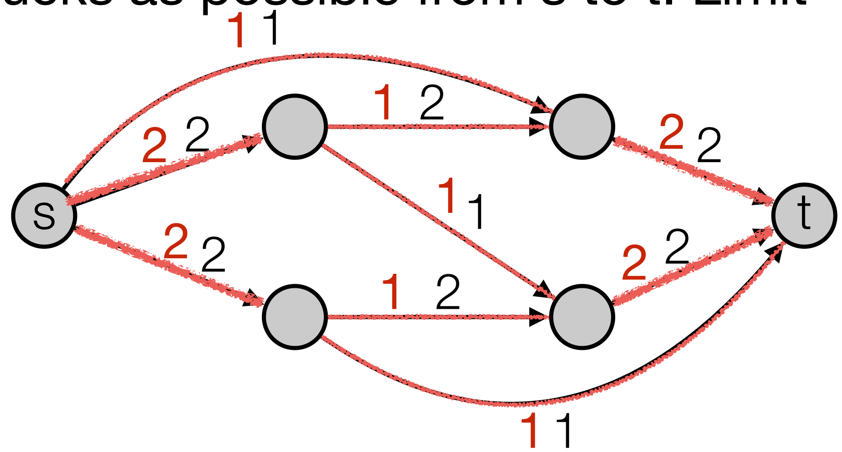
- Example 1:
  - Solution 1: 4 trucks
  - Solution 2: 5 trucks

- Example 2:



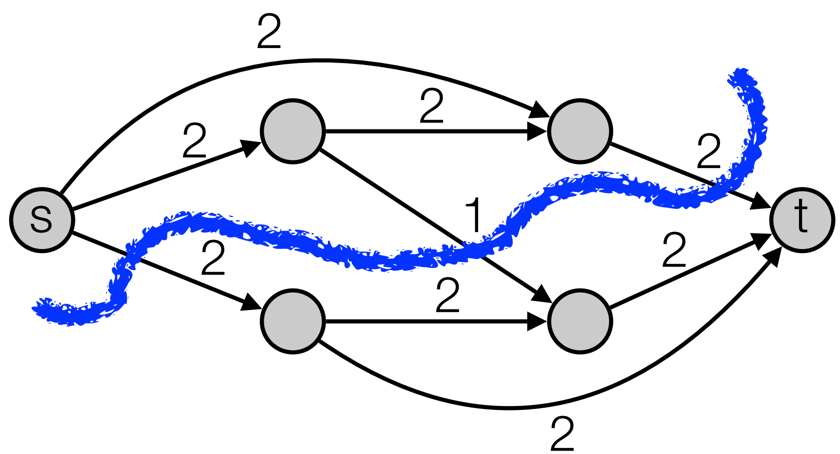
Network Flow

- 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

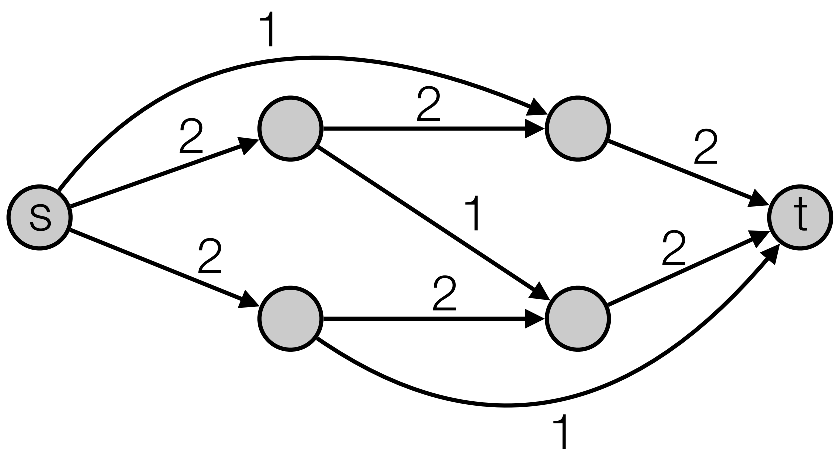
- Example 2:
  - 5 trucks (need to cross river).

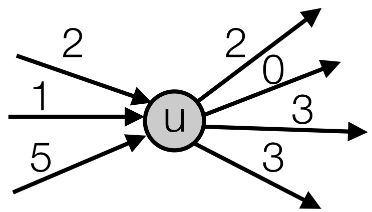


Network Flow

- Network flow:
  - graph G=(V,E).
  - Special vertices s (source) and t (sink).
  - Every edge (u,v) has a capacity c(u,v) ≥ 0.
  - Flow:
    - **capacity constraint:** every edge e has a flow 0 ≤ f(u,v) ≤ c(u,v).
    - **flow conservation:** for all u ≠ s, t: flow into u equals flow out of u.

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





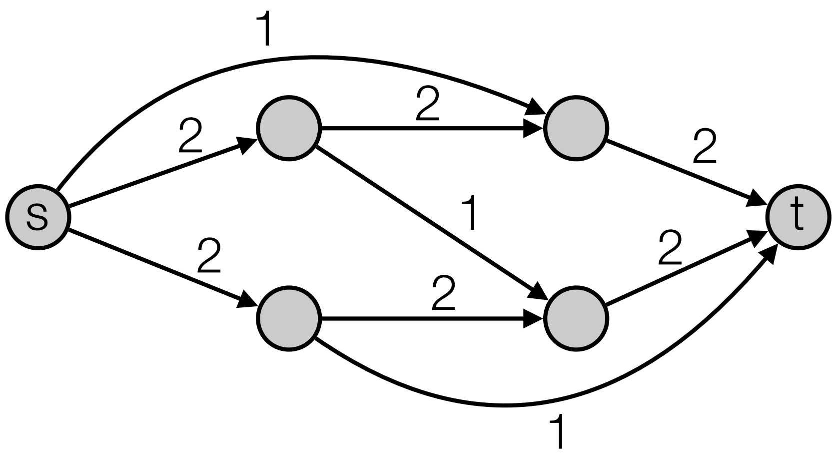
    - Value of flow f is the sum of flows out of s minus sum of flows into s:

$$|f| = \sum_{v:(s,v)\in E} f(s,v) - \sum_{v:(v,s)\in E} f(v,s)$$

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

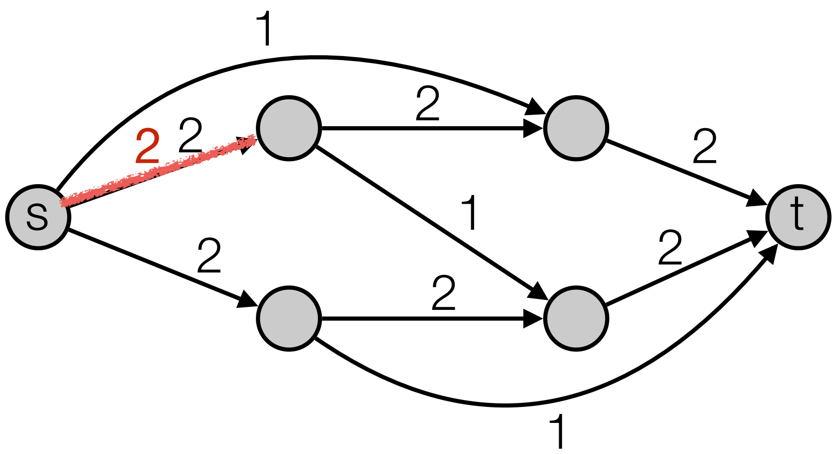
Algorithm

- Find path where we can send more flow.



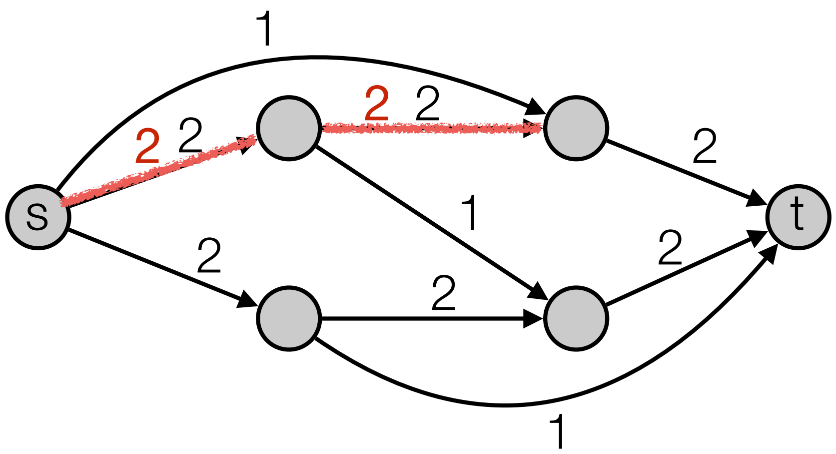
Algorithm

- Find path where we can send more flow.



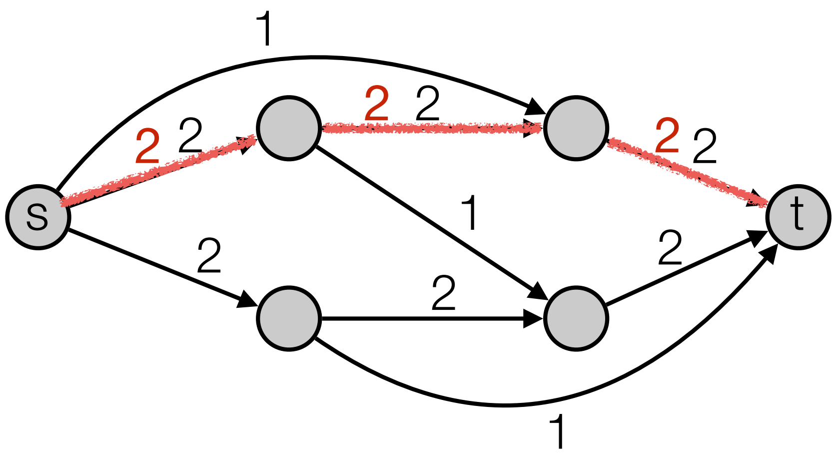
Algorithm

- Find path where we can send more flow.



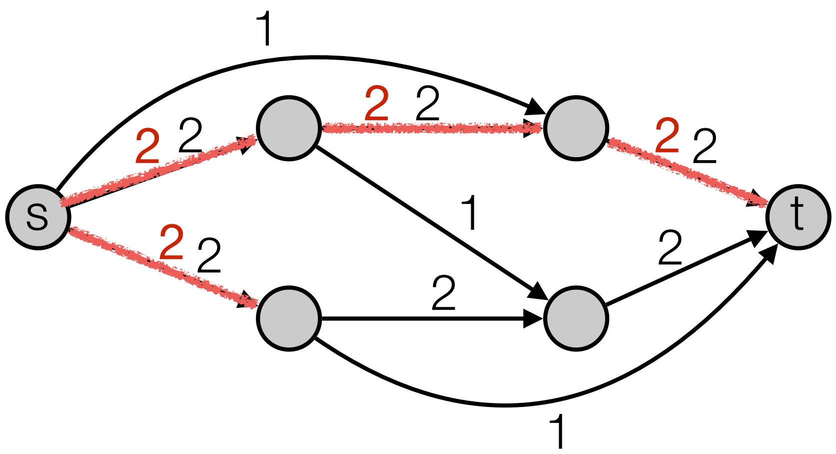
Algorithm

- Find path where we can send more flow.



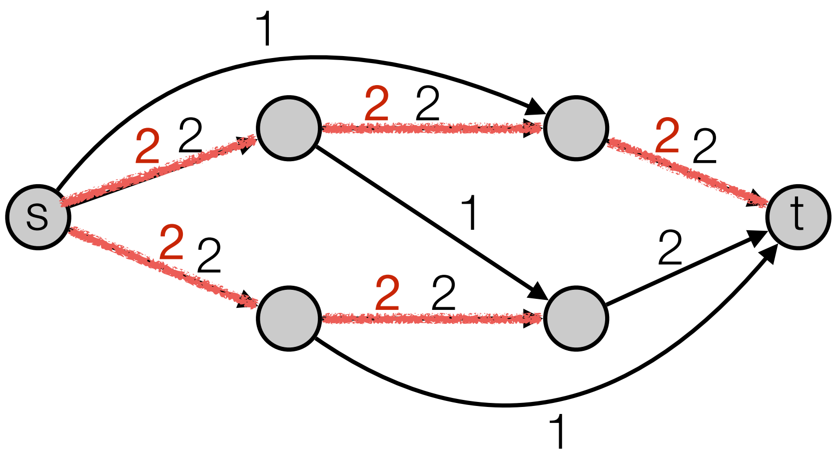
Algorithm

- Find path where we can send more flow.



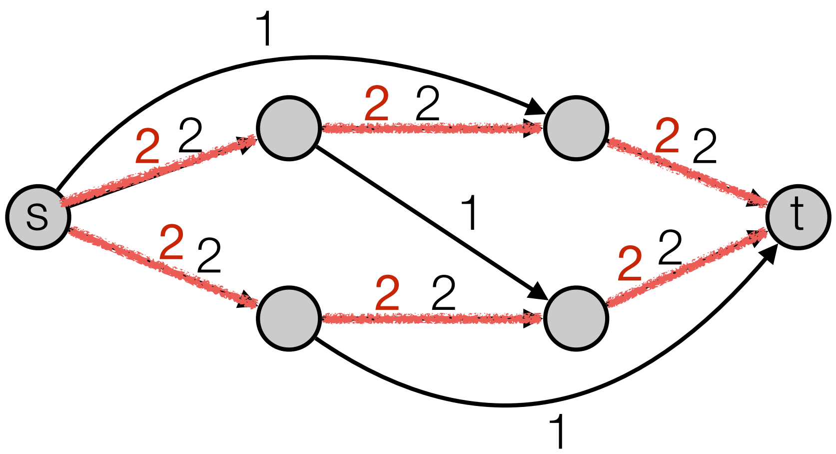
Algorithm

- Find path where we can send more flow.



Algorithm

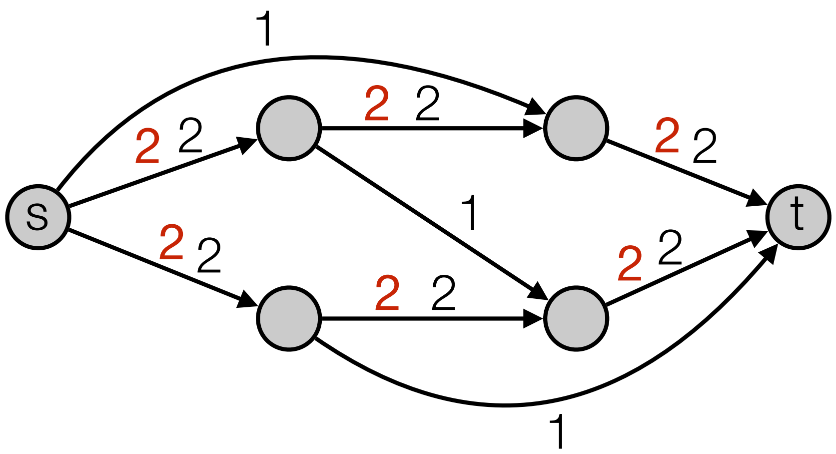
- Find path where we can send more flow.



Algorithm

- Find path where we can send more flow.

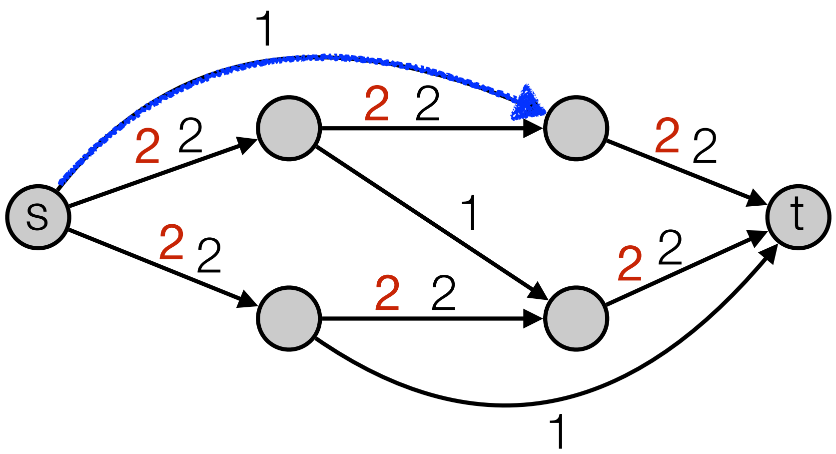
- Send flow back (cancel flow).



Algorithm

- Find path where we can send more flow.

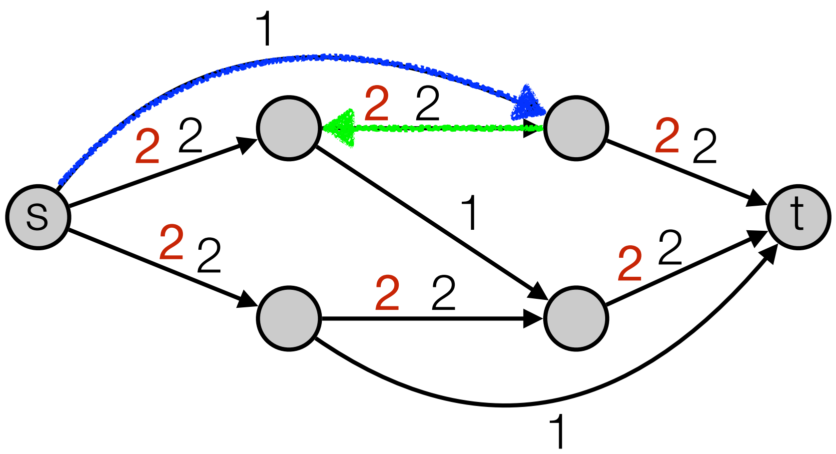
- Send flow back (cancel flow).



Algorithm

- Find path where we can send more flow.

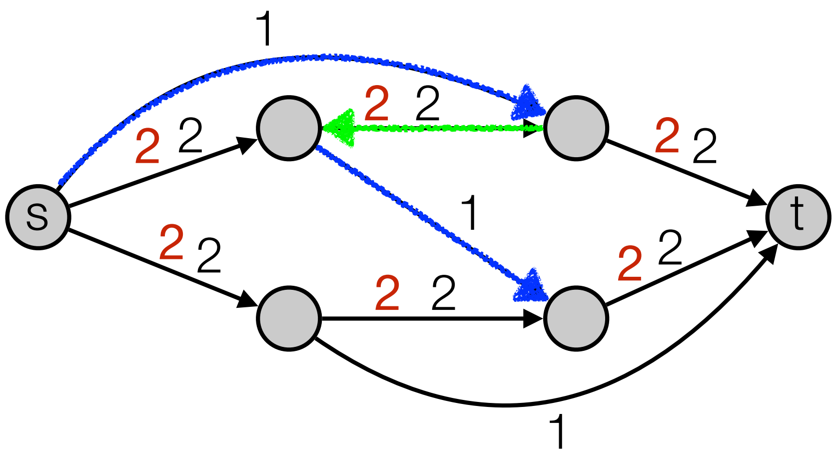
- Send flow back (cancel flow).



Algorithm

- Find path where we can send more flow.

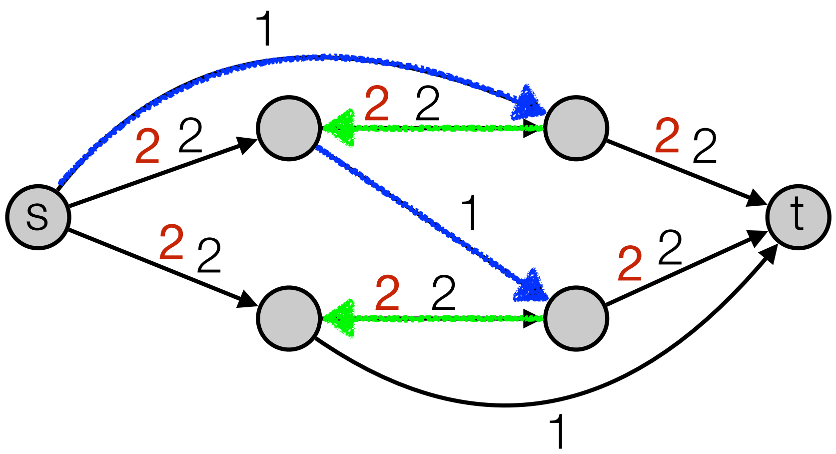
- Send flow back (cancel flow).



Algorithm

- Find path where we can send more flow.

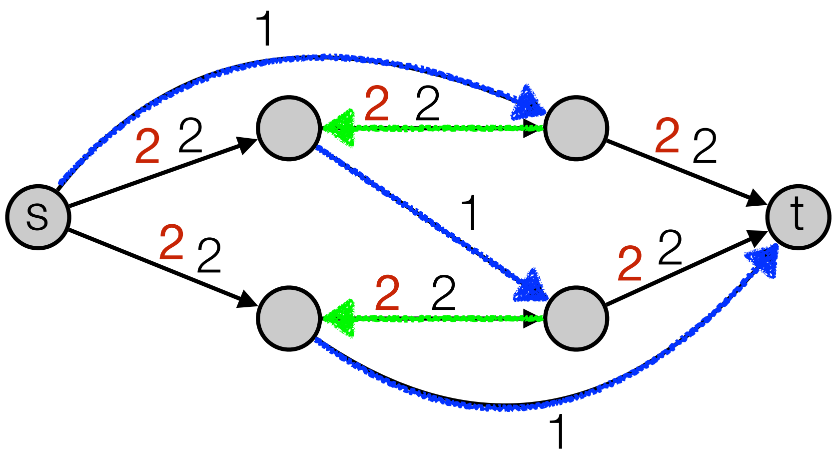
- Send flow back (cancel flow).



Algorithm

- Find path where we can send more flow.

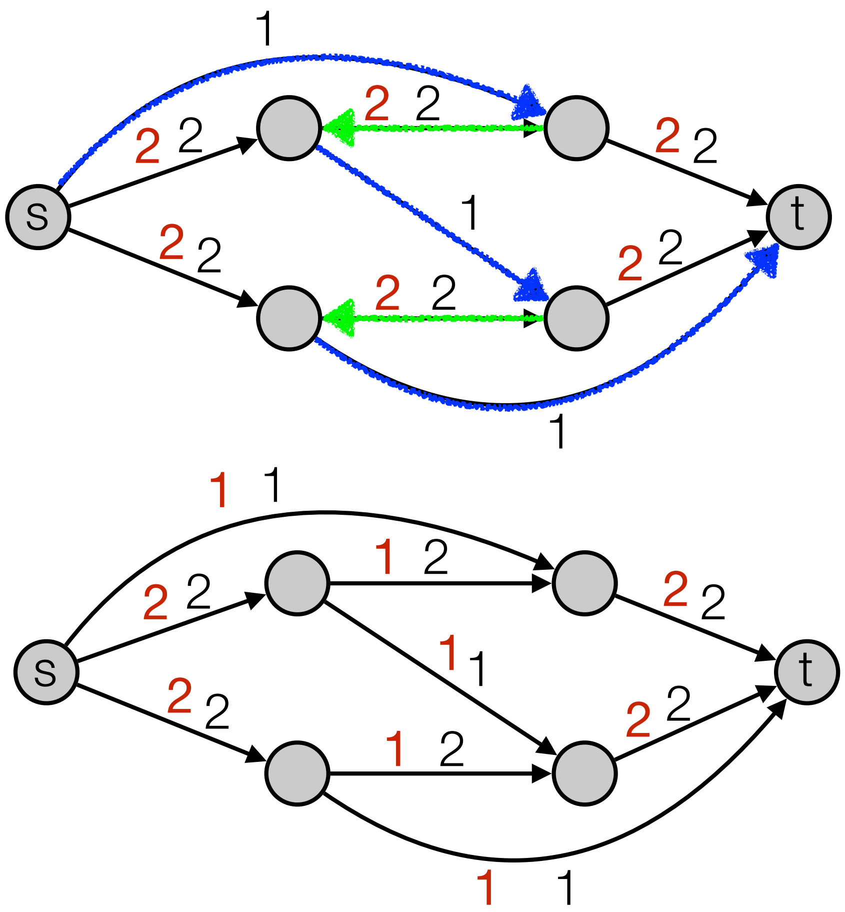
- Send flow back (cancel flow).



Algorithm

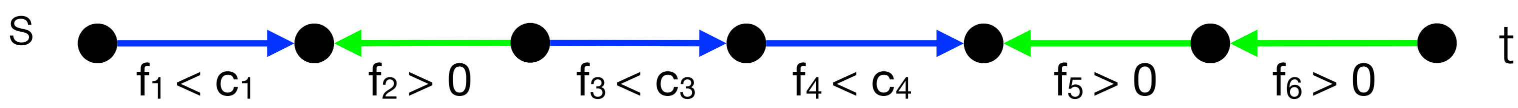
- Find path where we can send more flow.

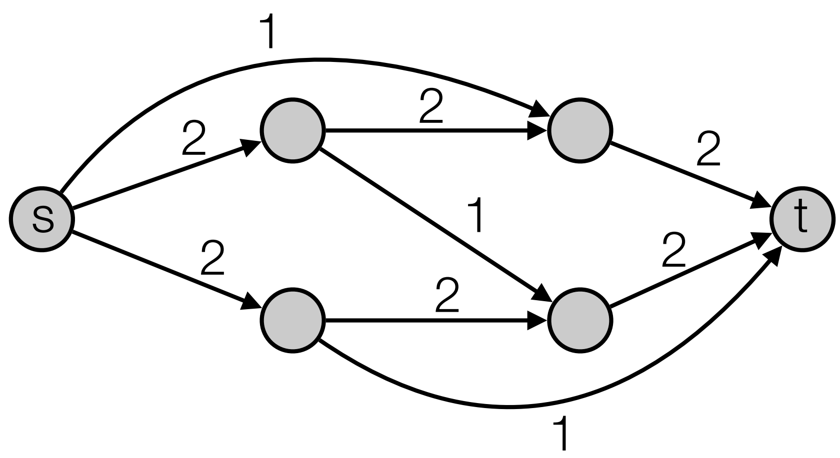
- Send flow back (cancel flow).



Augmenting Paths

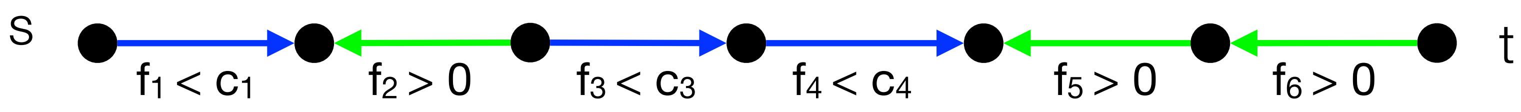
- Augmenting path (definition different than in CLRS): s-t path where
  - forward edges have leftover capacity
  - backwards edges have positive flow



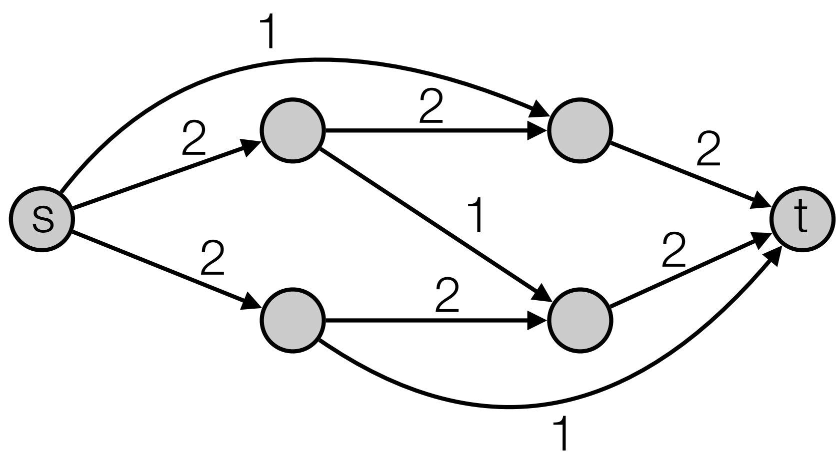


Augmenting Paths

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

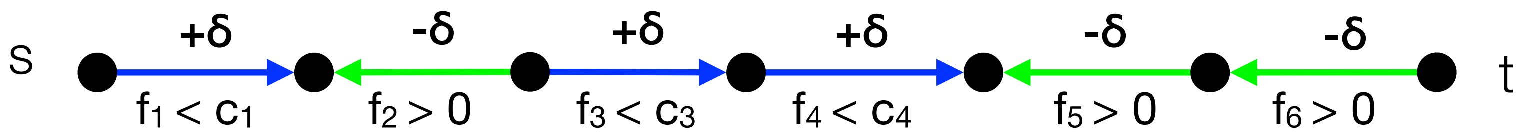


- Can add extra flow: $\min(c_1 - f_1, f_2, c_3 - f_3, c_4 - f_4, f_5, f_6) = \delta$.

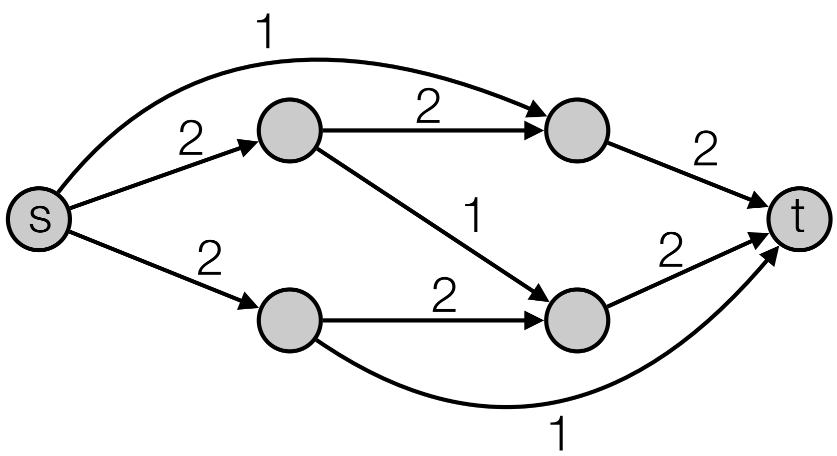


Augmenting Paths

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

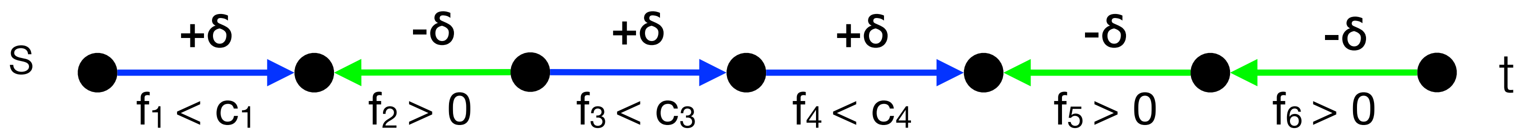


- Can add extra flow: $\min(c_1 - f_1, f_2, c_3 - f_3, c_4 - f_4, f_5, f_6) = \delta$.

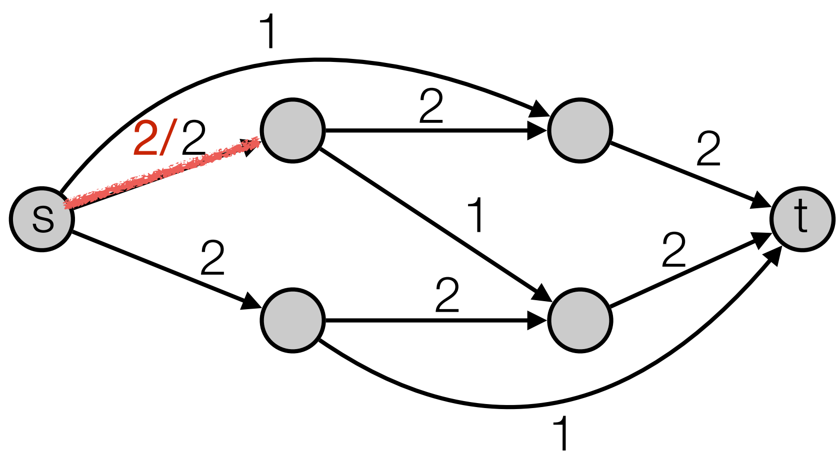


Augmenting Paths

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

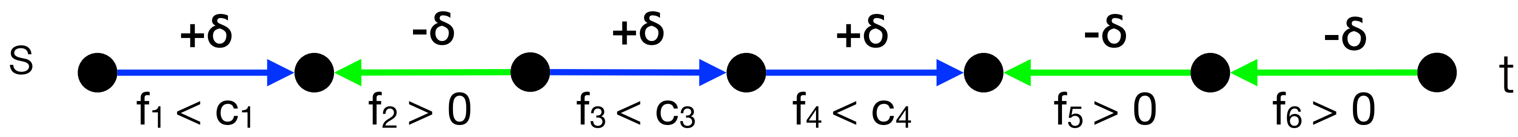


  - Can add extra flow: $\min(c_1 - f_1, f_2, c_3 - f_3, c_4 - f_4, f_5, f_6) = \delta$.

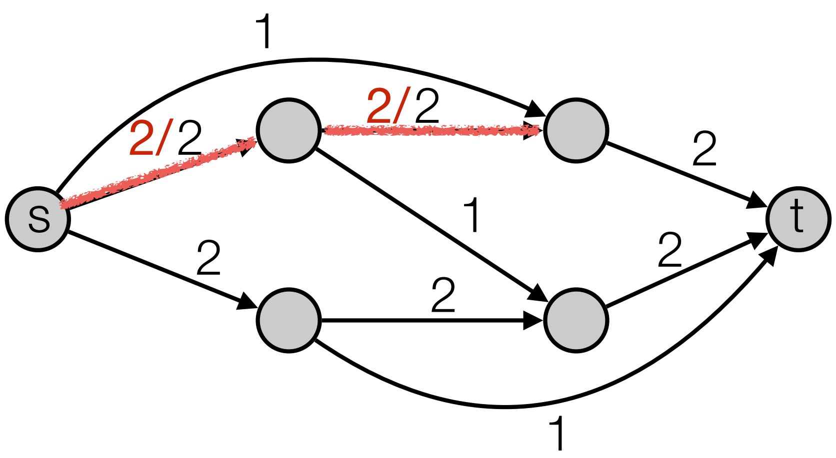


Augmenting Paths

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

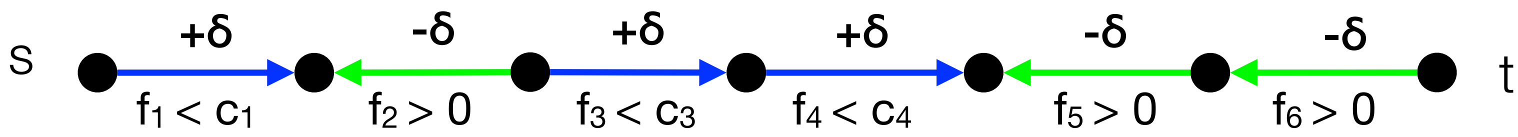


  - Can add extra flow: $\min(c_1 - f_1, f_2, c_3 - f_3, c_4 - f_4, f_5, f_6) = \delta$.

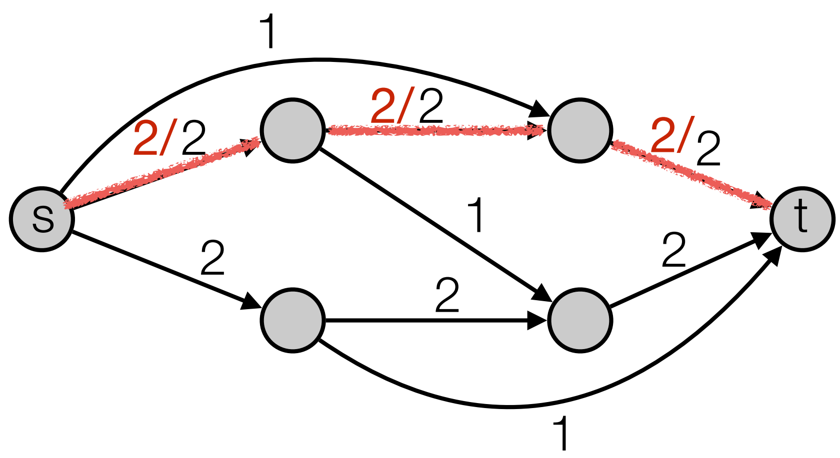


Augmenting Paths

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

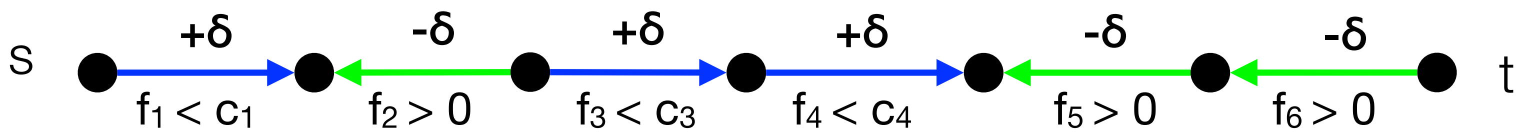


- Can add extra flow: $\min(c_1 - f_1, f_2, c_3 - f_3, c_4 - f_4, f_5, f_6) = \delta$.

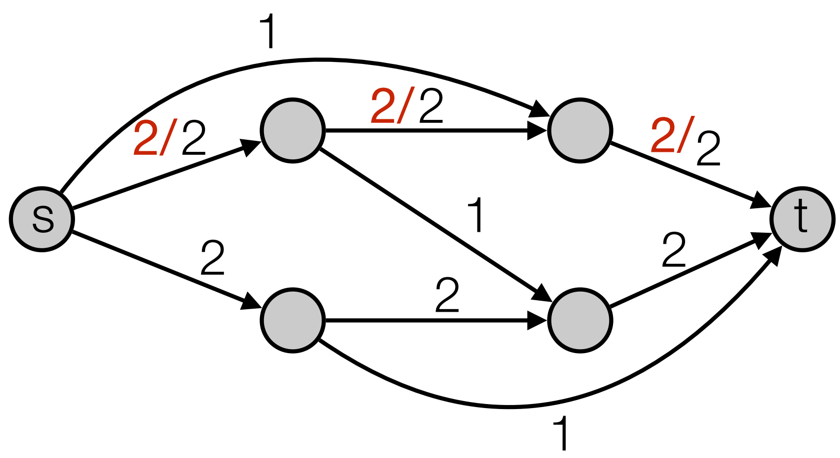


Augmenting Paths

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

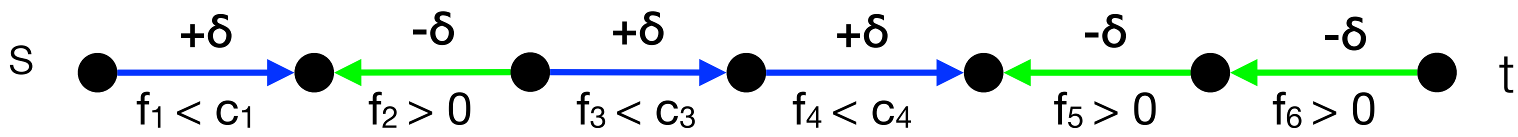


  - Can add extra flow: $\min(c_1 - f_1, f_2, c_3 - f_3, c_4 - f_4, f_5, f_6) = \delta$.

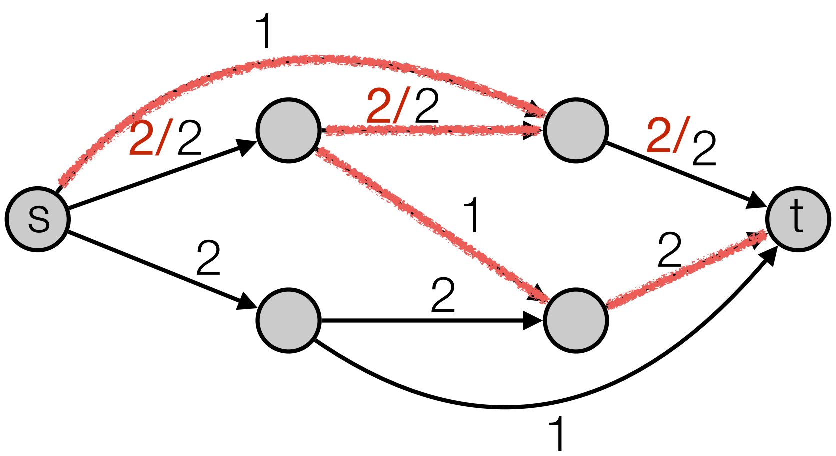


Augmenting Paths

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

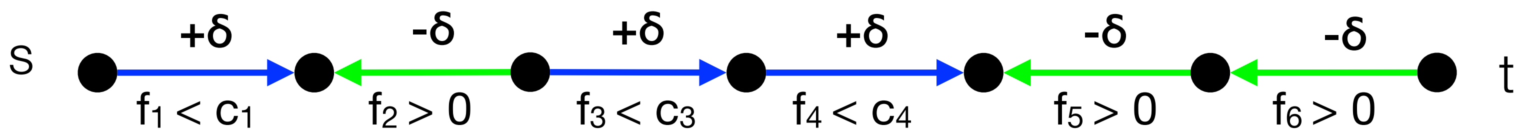


  - Can add extra flow: $\min(c_1 - f_1, f_2, c_3 - f_3, c_4 - f_4, f_5, f_6) = \delta$.

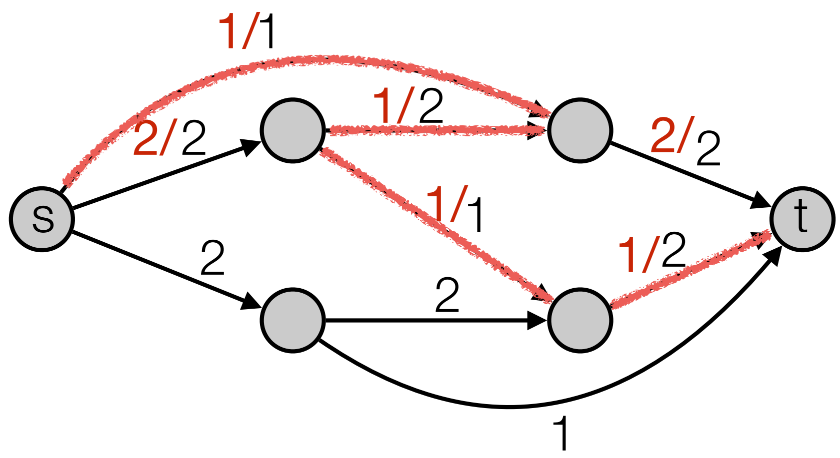


Augmenting Paths

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

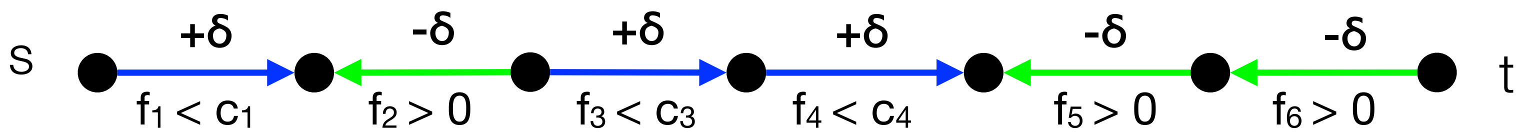


- Can add extra flow: $\min(c_1 - f_1, f_2, c_3 - f_3, c_4 - f_4, f_5, f_6) = \delta$.

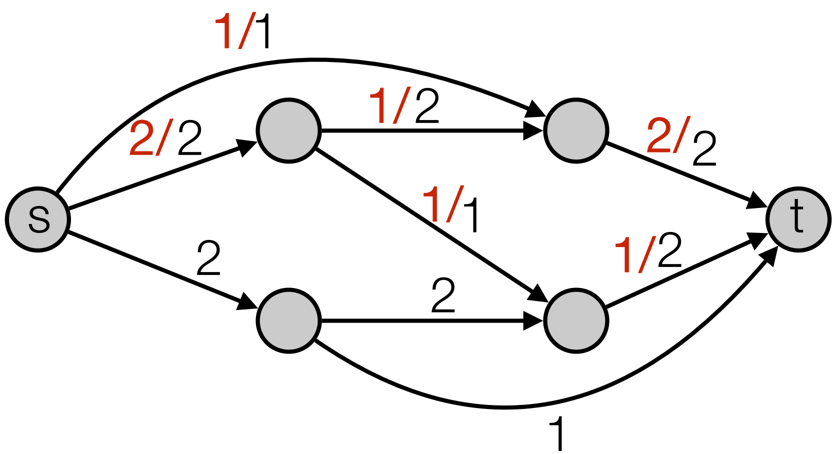


Augmenting Paths

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

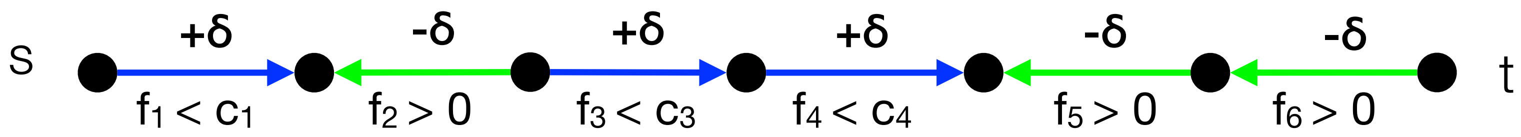


  - Can add extra flow: $\min(c_1 - f_1, f_2, c_3 - f_3, c_4 - f_4, f_5, f_6) = \delta$.

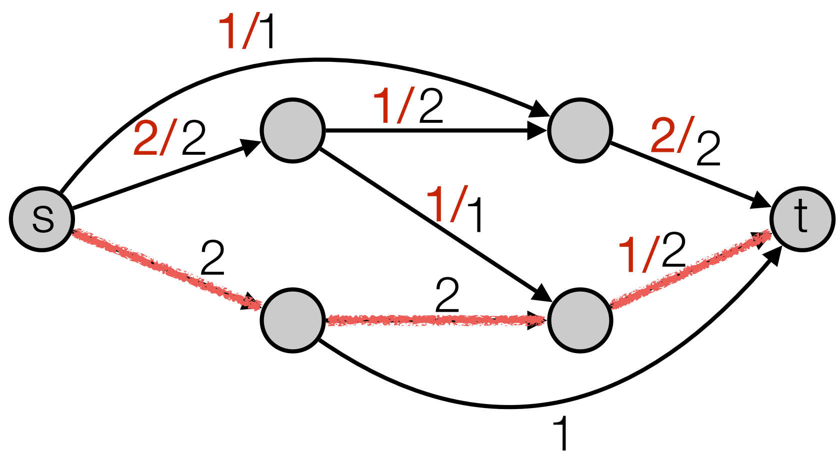


Augmenting Paths

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

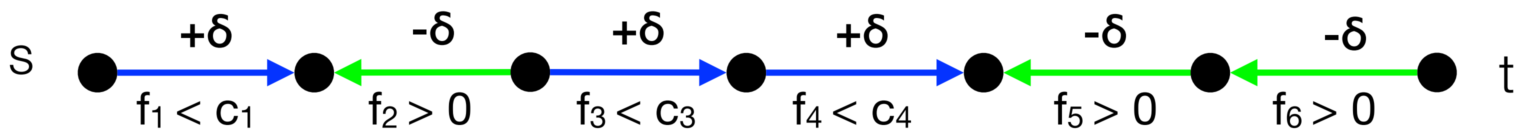


  - Can add extra flow: $\min(c_1 - f_1, f_2, c_3 - f_3, c_4 - f_4, f_5, f_6) = \delta$.

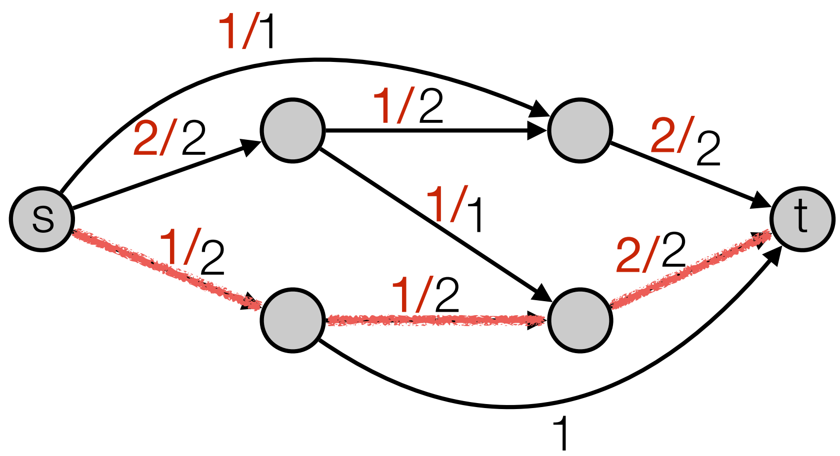


Augmenting Paths

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

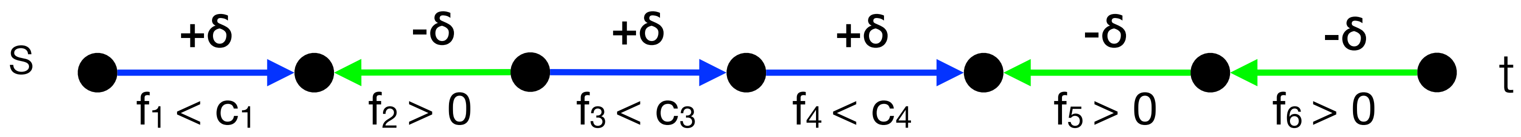


  - Can add extra flow: $\min(c_1 - f_1, f_2, c_3 - f_3, c_4 - f_4, f_5, f_6) = \delta$.

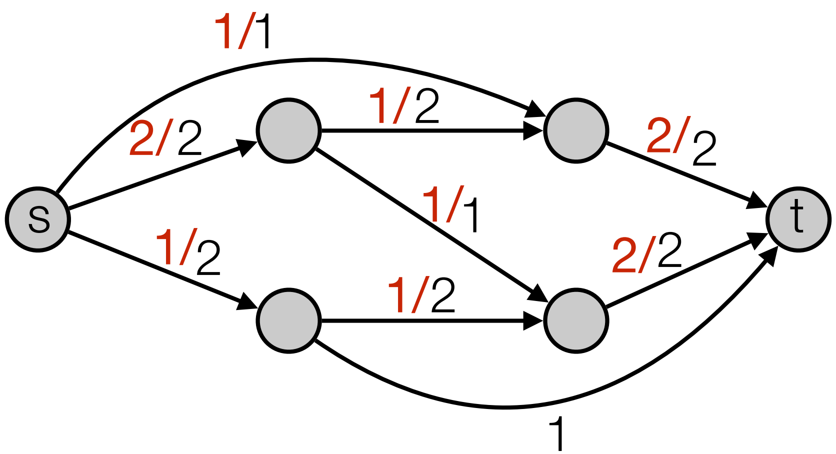


Augmenting Paths

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

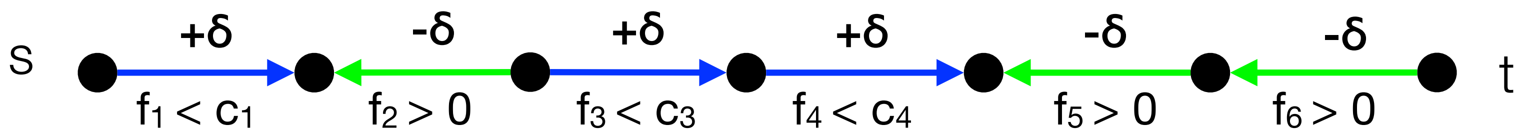


- Can add extra flow: $\min(c_1 - f_1, f_2, c_3 - f_3, c_4 - f_4, f_5, f_6) = \delta$.

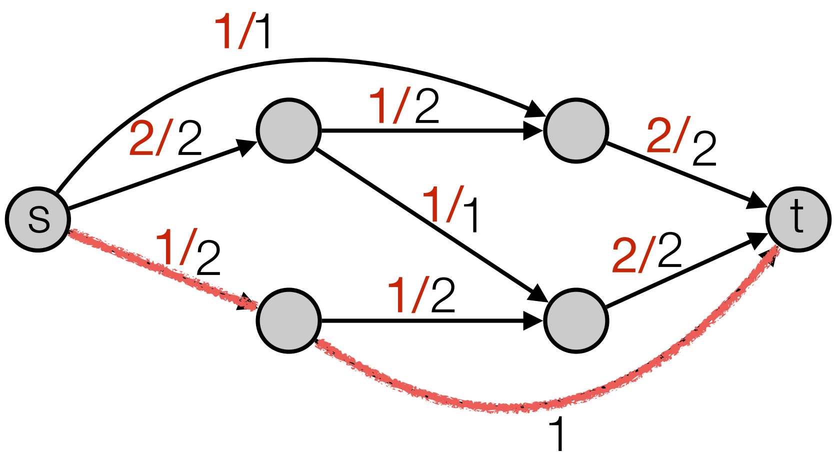


Augmenting Paths

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

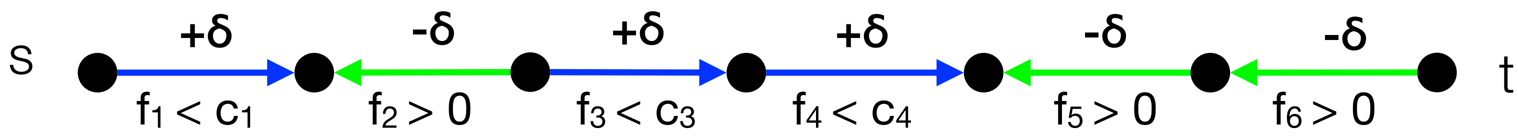


  - Can add extra flow: $\min(c_1 - f_1, f_2, c_3 - f_3, c_4 - f_4, f_5, f_6) = \delta$.

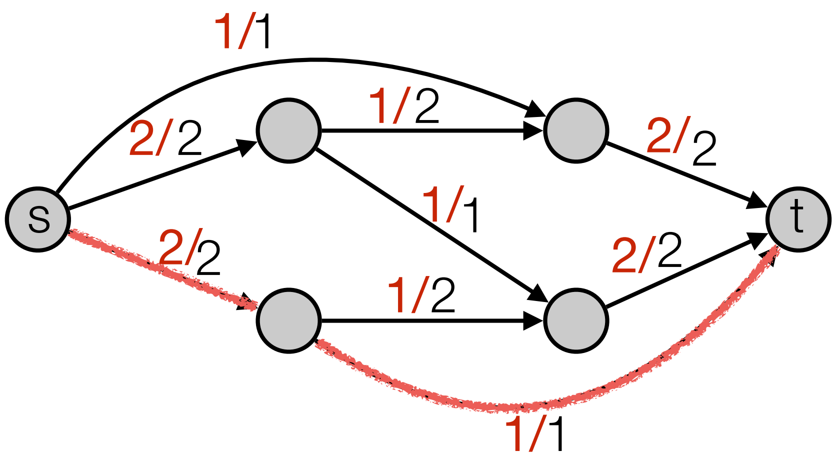


Augmenting Paths

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

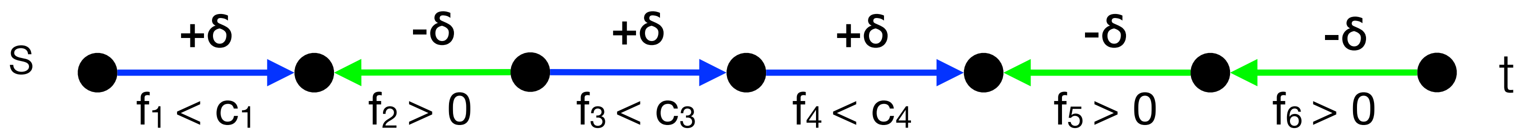


- Can add extra flow: $\min(c_1 - f_1, f_2, c_3 - f_3, c_4 - f_4, f_5, f_6) = \delta$.

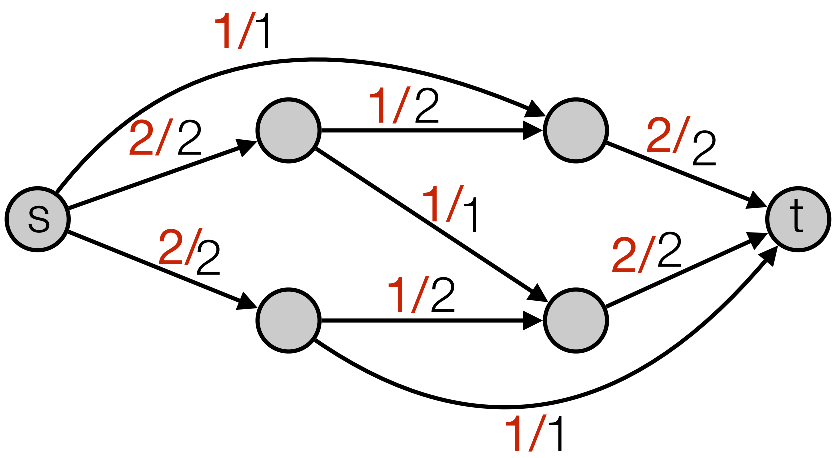


Augmenting Paths

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

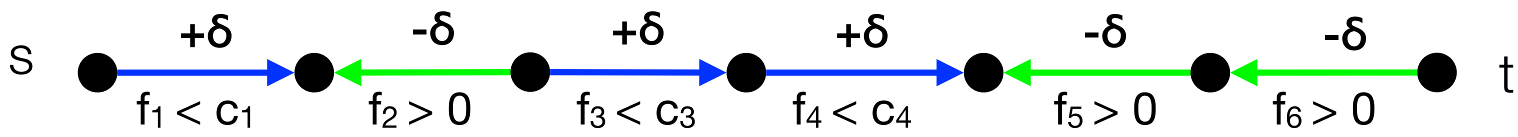


  - Can add extra flow: $\min(c_1 - f_1, f_2, c_3 - f_3, c_4 - f_4, f_5, f_6) = \delta$.



Augmenting Paths

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

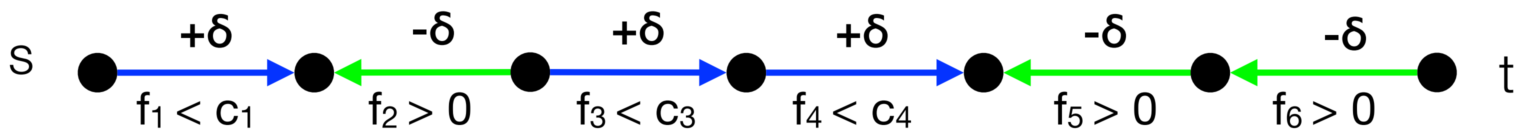


  - Can add extra flow: $\min(c_1 - f_1, f_2, c_3 - f_3, c_4 - f_4, f_5, f_6) = \delta$

- Ford-Fulkerson:

Augmenting Paths

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

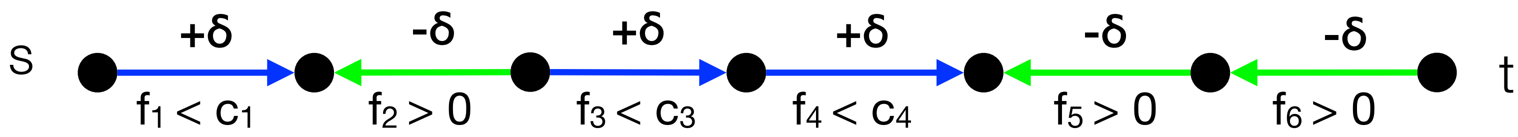


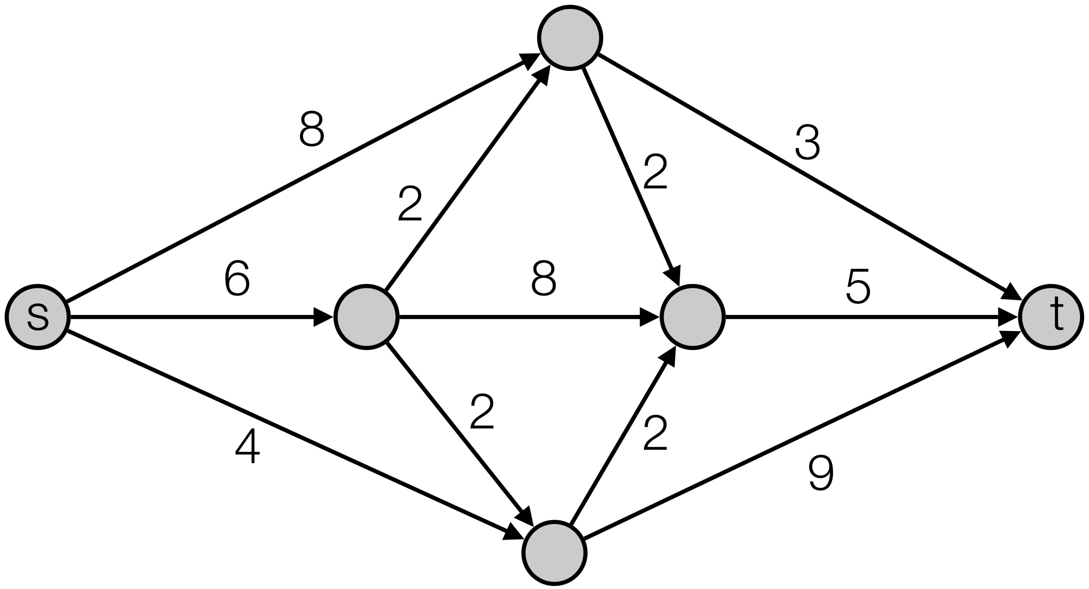
  - Can add extra flow: $\min(c_1 - f_1, f_2, c_3 - f_3, c_4 - f_4, f_5, f_6) = \delta$

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

Ford Fulkerson

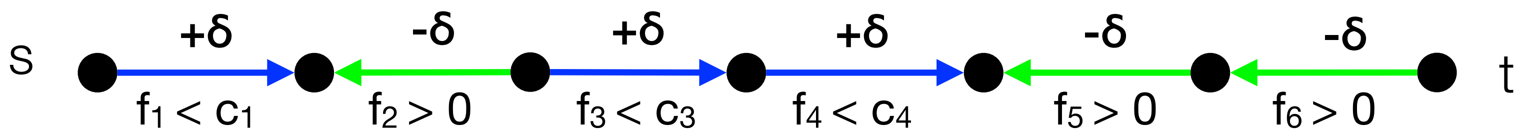
- Augmenting path (definition different than in CLRS): s-t path where
  - forward edges have leftover capacity
  - backwards edges have positive flow

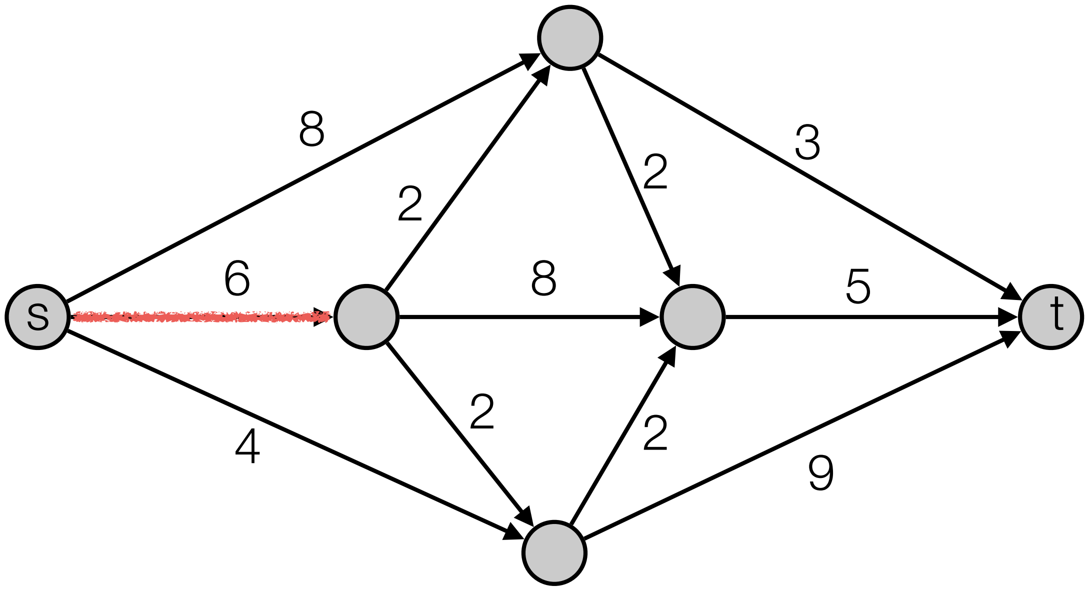




Ford Fulkerson

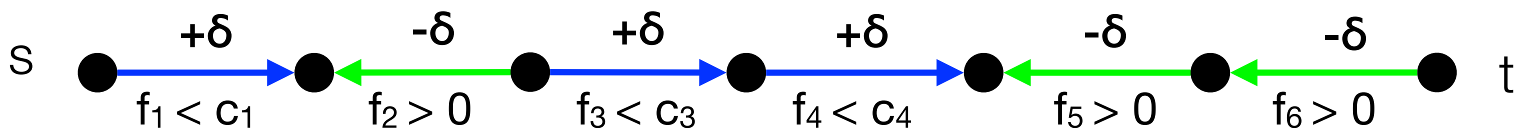
- Augmenting path (definition different than in CLRS): s-t path where
  - forward edges have leftover capacity
  - backwards edges have positive flow

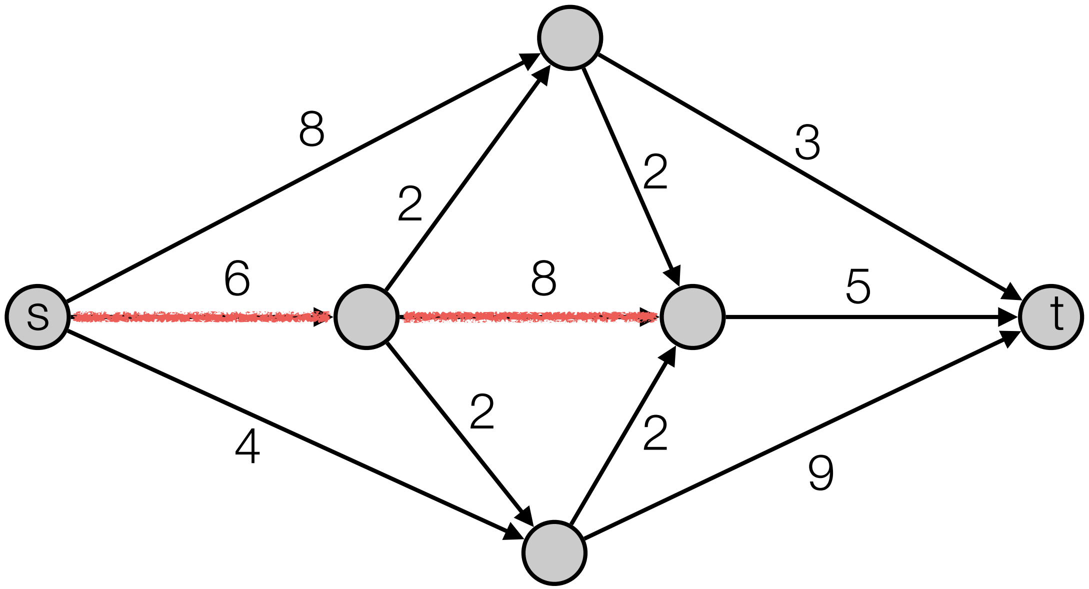




Ford Fulkerson

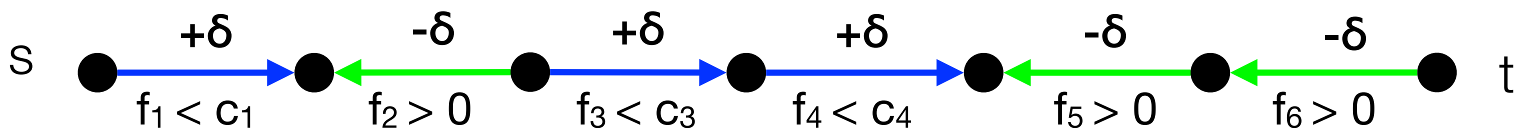
- Augmenting path (definition different than in CLRS): s-t path where
  - forward edges have leftover capacity
  - backwards edges have positive flow

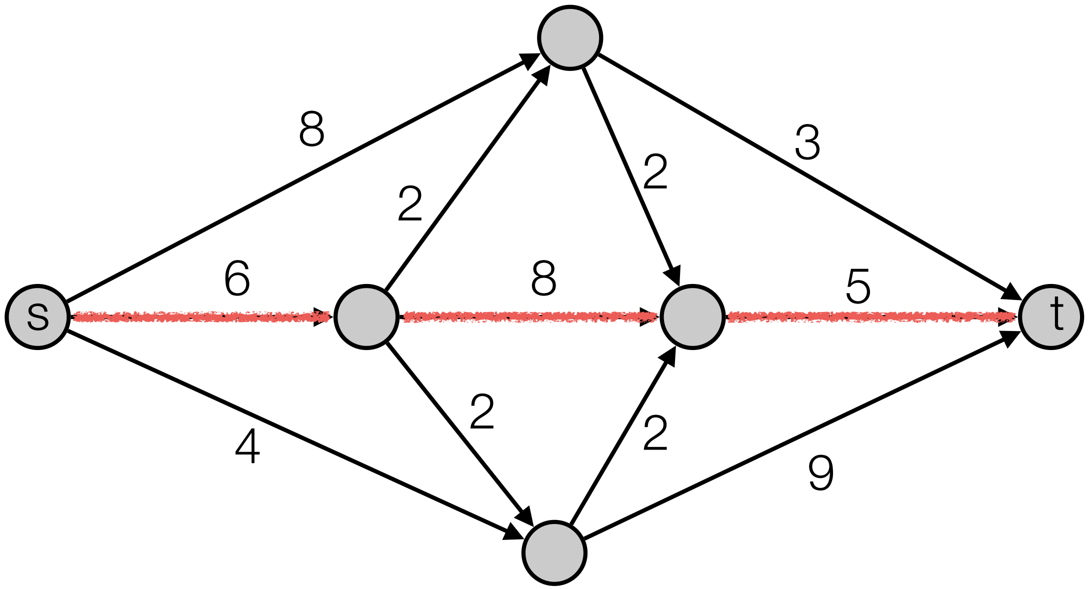




Ford Fulkerson

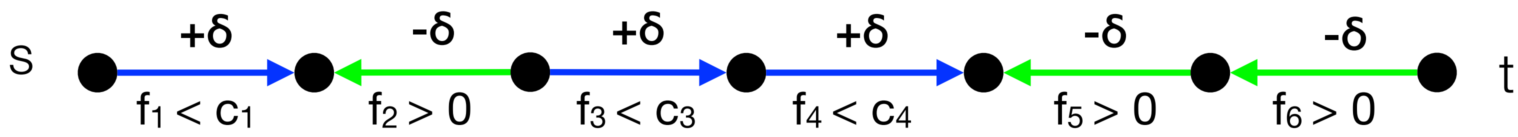
- Augmenting path (definition different than in CLRS): s-t path where
  - forward edges have leftover capacity
  - backwards edges have positive flow

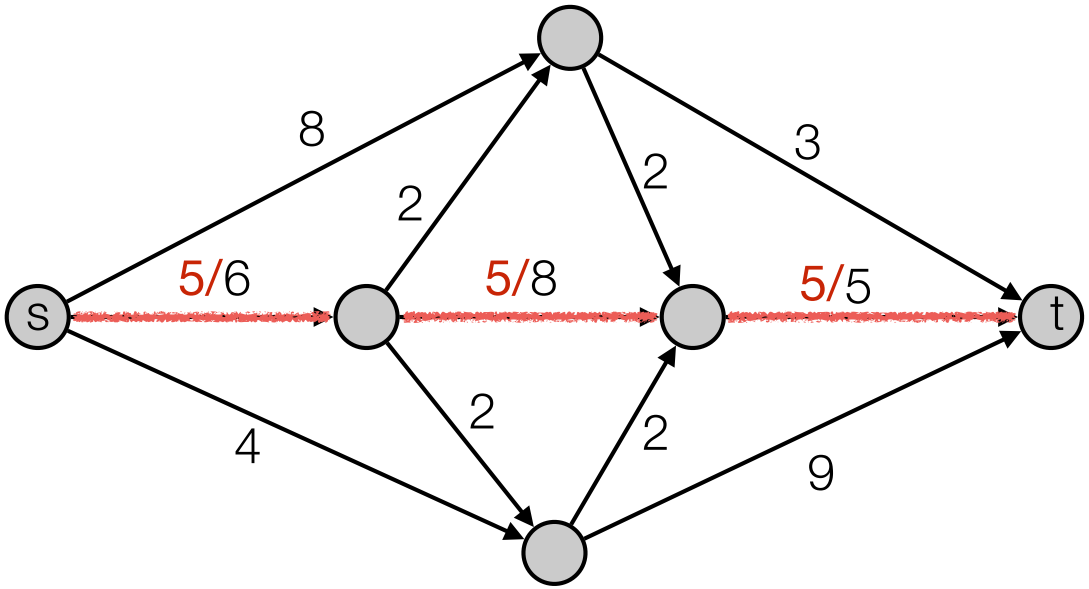




Ford Fulkerson

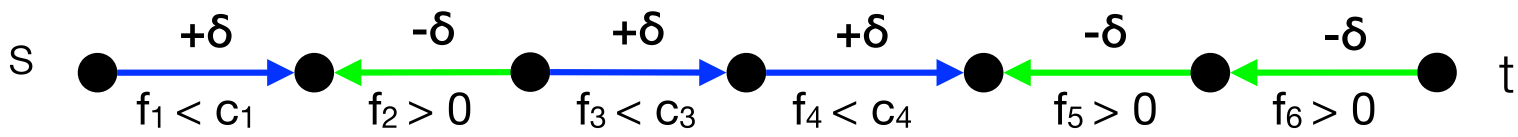
- Augmenting path (definition different than in CLRS): s-t path where
  - forward edges have leftover capacity
  - backwards edges have positive flow

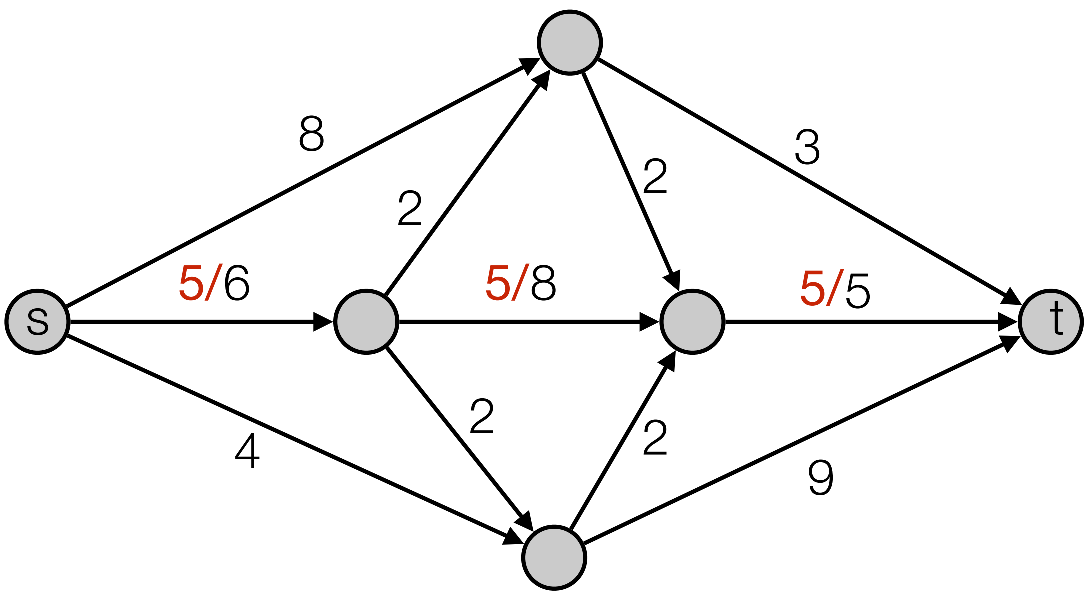




Ford Fulkerson

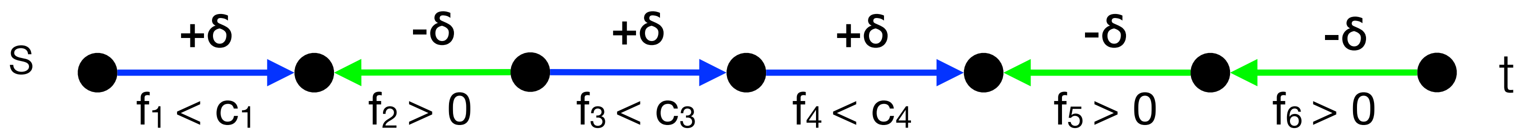
- Augmenting path (definition different than in CLRS): s-t path where
  - forward edges have leftover capacity
  - backwards edges have positive flow

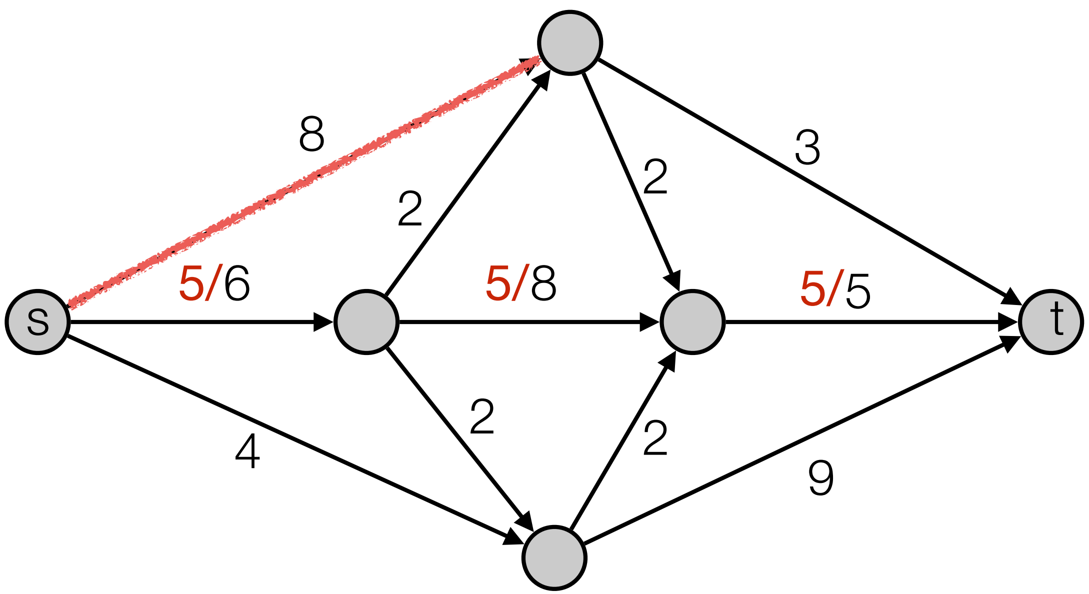




Ford Fulkerson

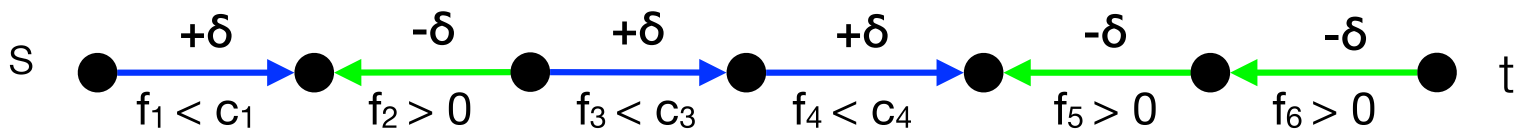
- Augmenting path (definition different than in CLRS): s-t path where
  - forward edges have leftover capacity
  - backwards edges have positive flow

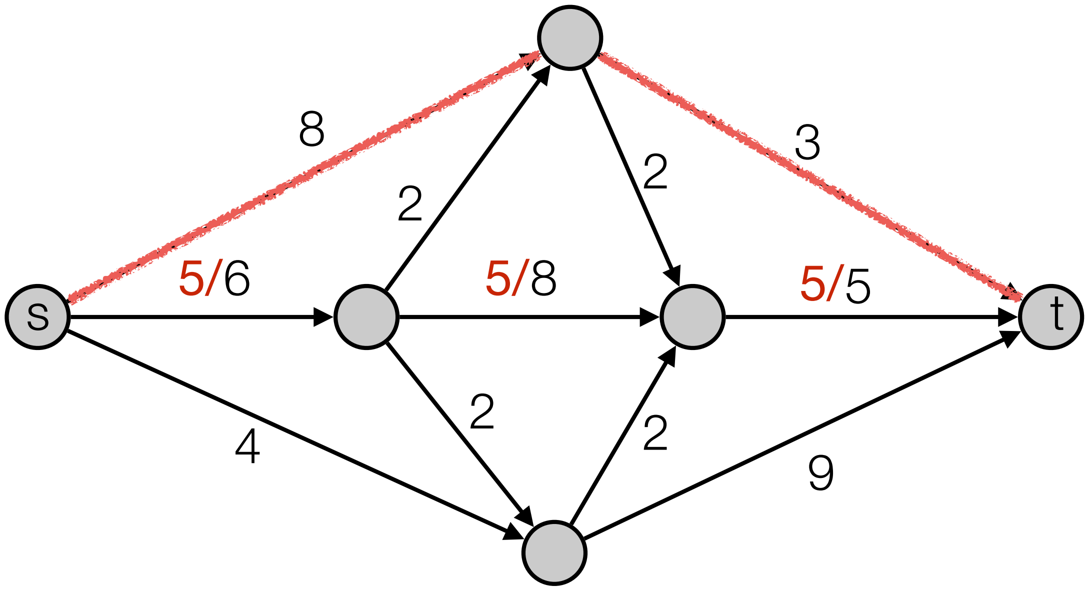




Ford Fulkerson

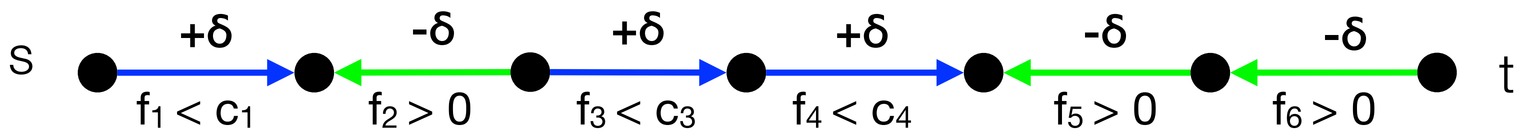
- Augmenting path (definition different than in CLRS): s-t path where
  - forward edges have leftover capacity
  - backwards edges have positive flow

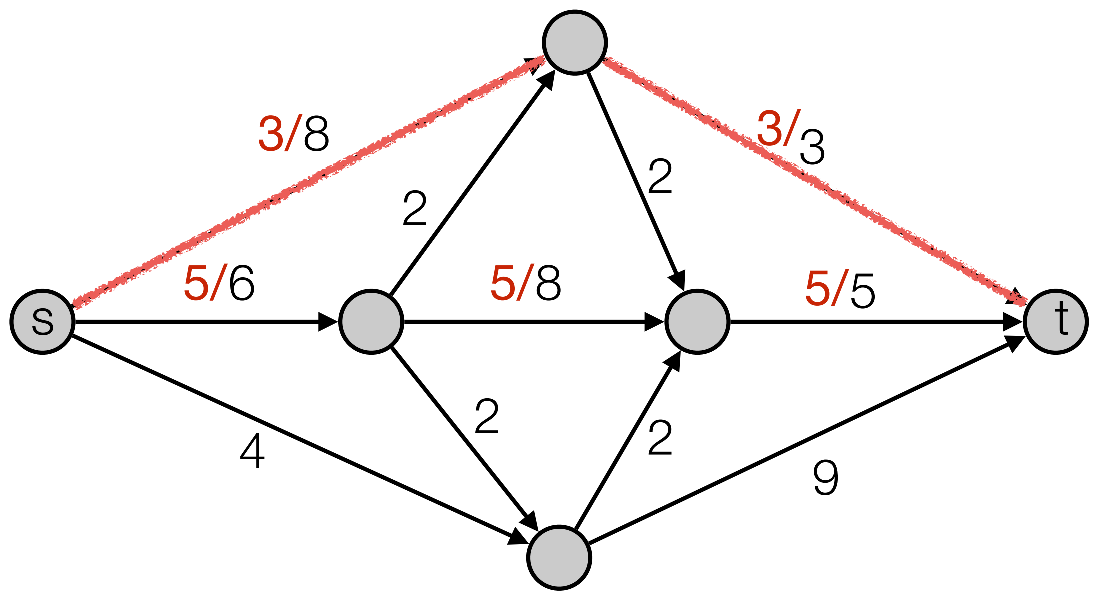




Ford Fulkerson

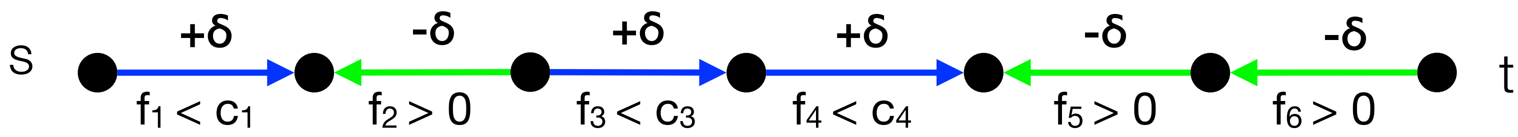
- Augmenting path (definition different than in CLRS): s-t path where
  - forward edges have leftover capacity
  - backwards edges have positive flow

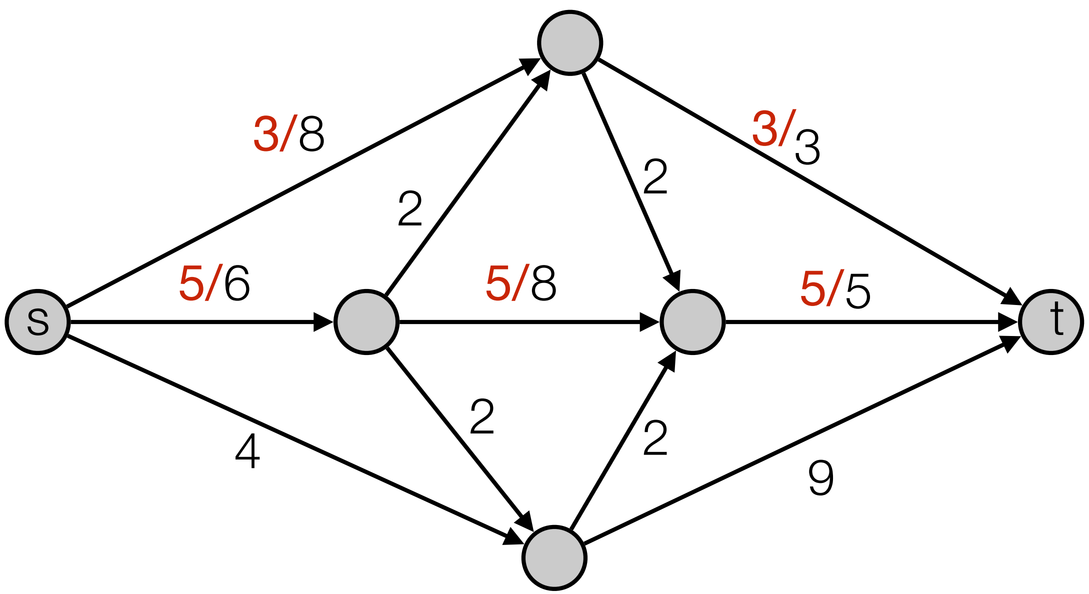




Ford Fulkerson

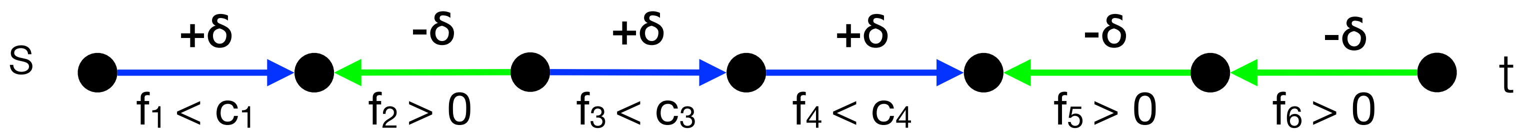
- Augmenting path (definition different than in CLRS): s-t path where
  - forward edges have leftover capacity
  - backwards edges have positive flow

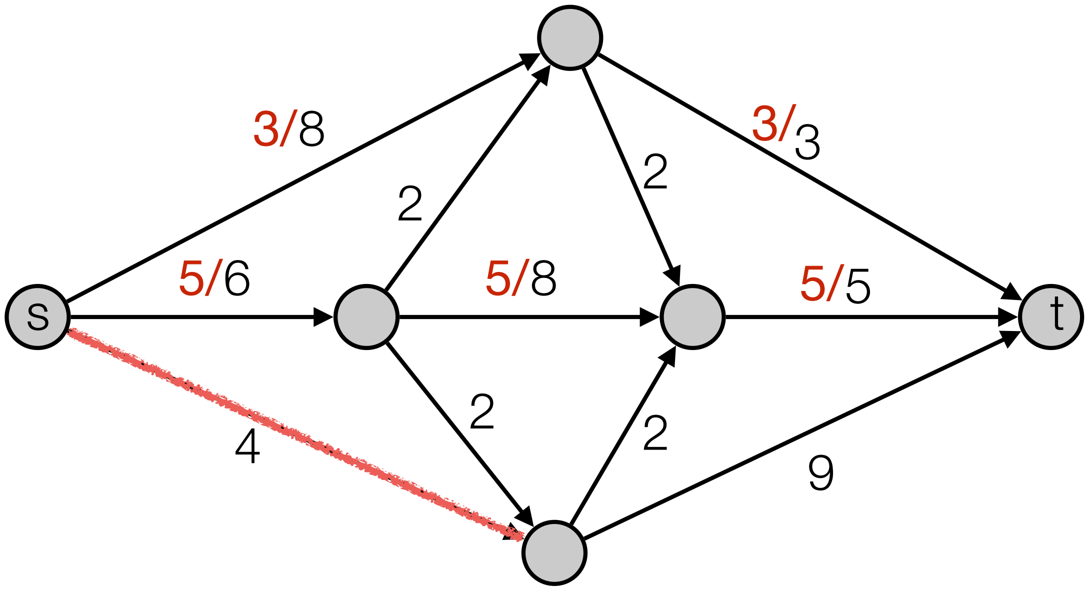




Ford Fulkerson

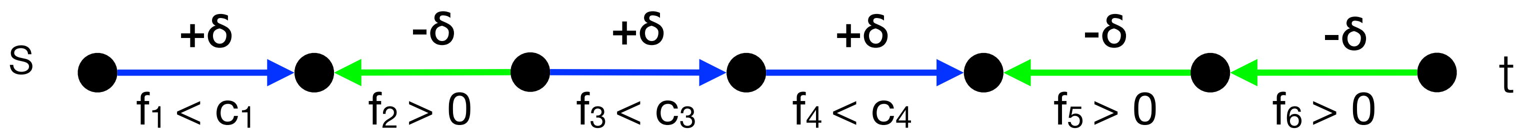
- Augmenting path (definition different than in CLRS): s-t path where
  - forward edges have leftover capacity
  - backwards edges have positive flow

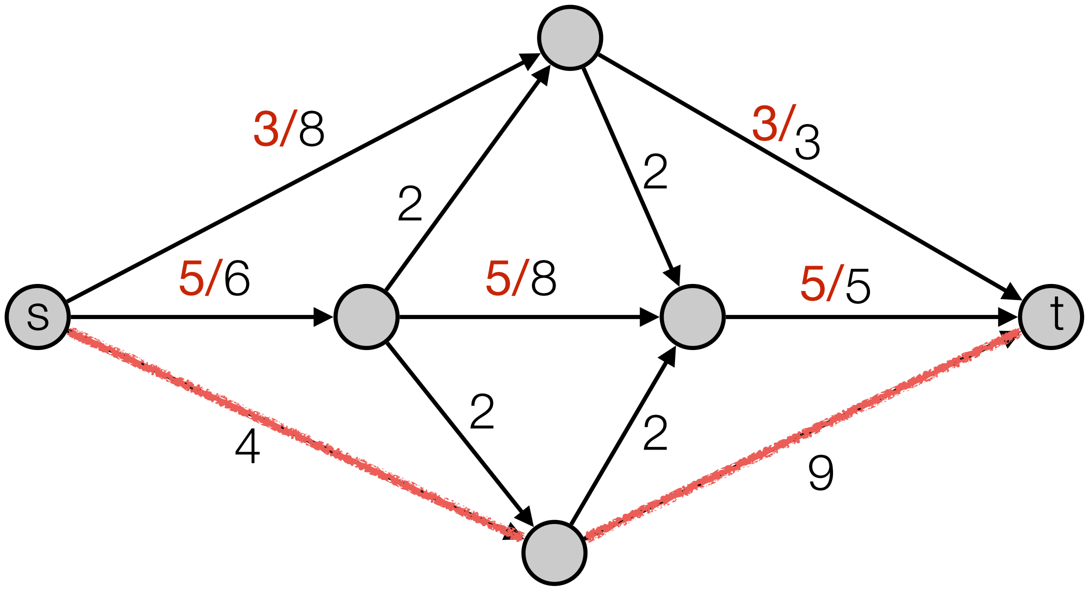




Ford Fulkerson

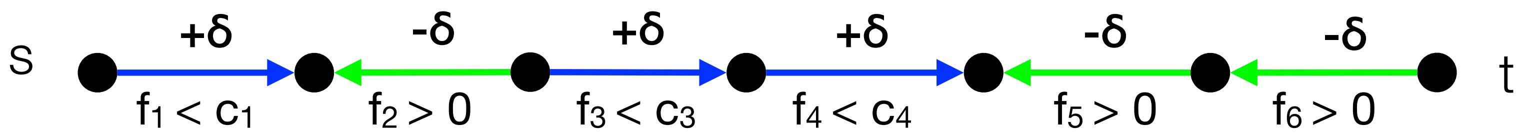
- Augmenting path (definition different than in CLRS): s-t path where
  - forward edges have leftover capacity
  - backwards edges have positive flow

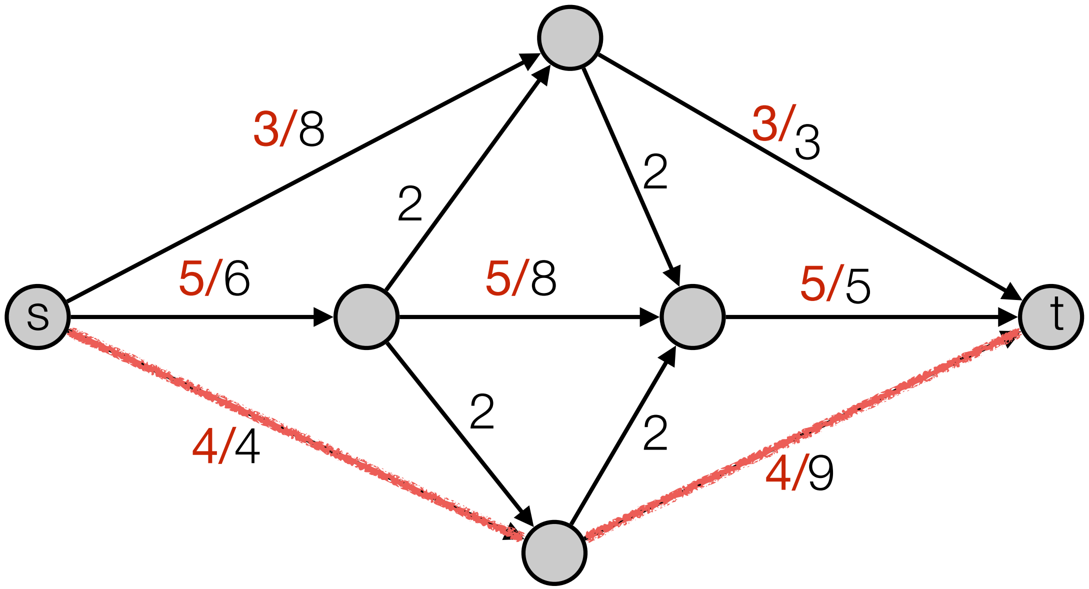




Ford Fulkerson

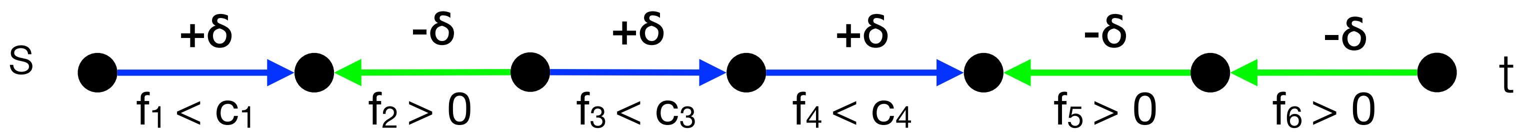
- Augmenting path (definition different than in CLRS): s-t path where
  - forward edges have leftover capacity
  - backwards edges have positive flow

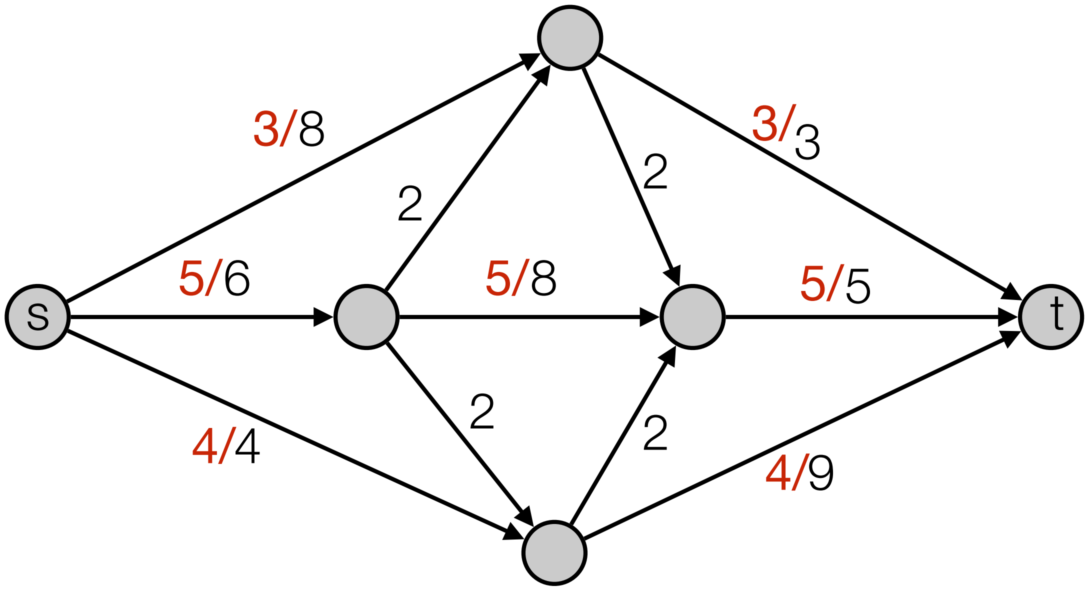




Ford Fulkerson

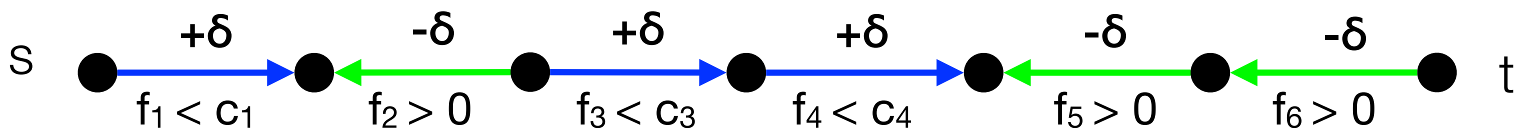
- Augmenting path (definition different than in CLRS): s-t path where
  - forward edges have leftover capacity
  - backwards edges have positive flow

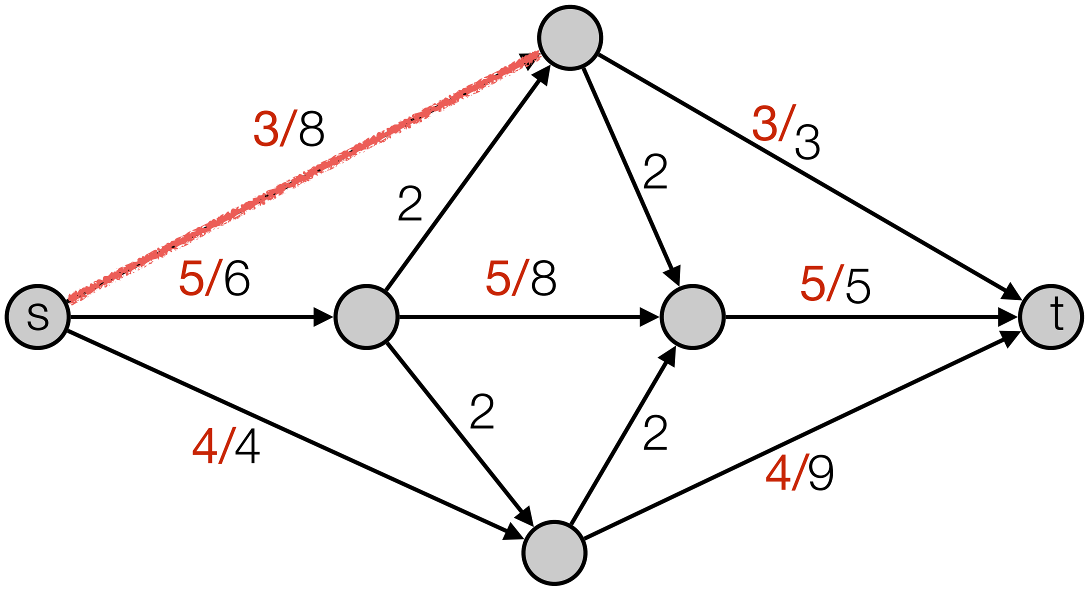




Ford Fulkerson

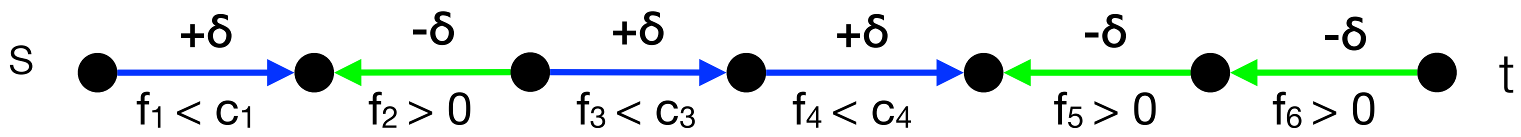
- Augmenting path (definition different than in CLRS): s-t path where
    - forward edges have leftover capacity
    - backwards edges have positive flow

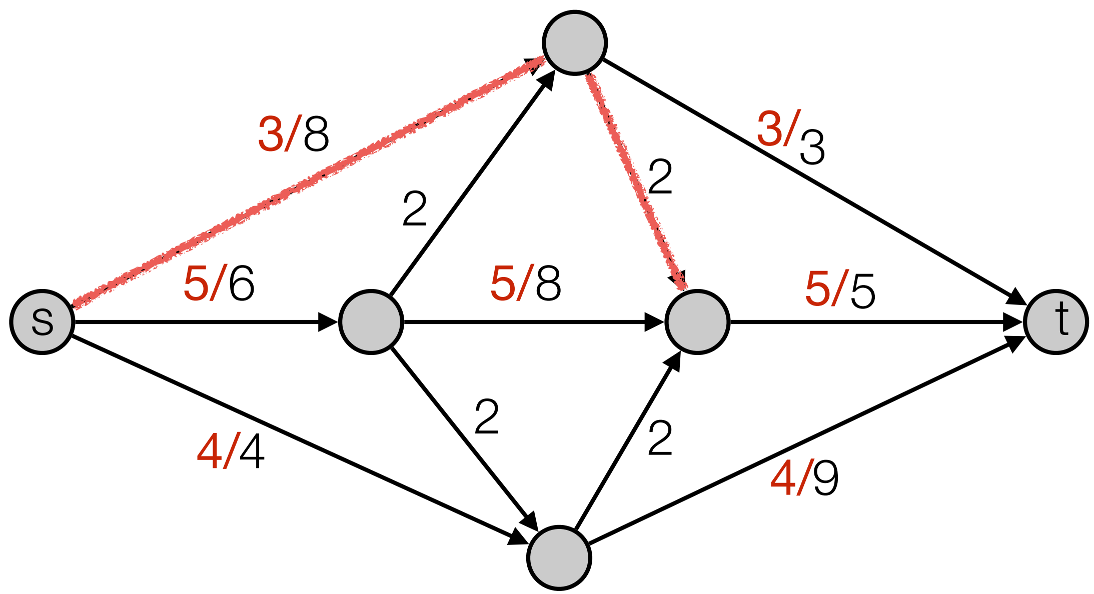




Ford Fulkerson

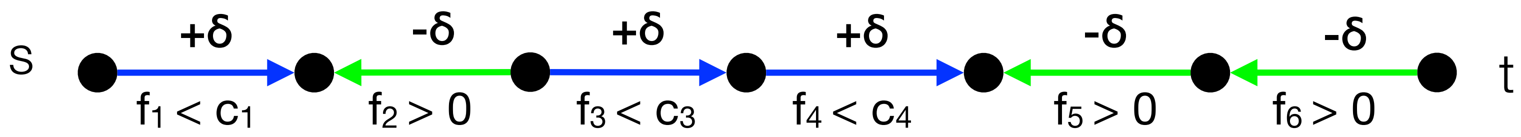
- Augmenting path (definition different than in CLRS): s-t path where
  - forward edges have leftover capacity
  - backwards edges have positive flow

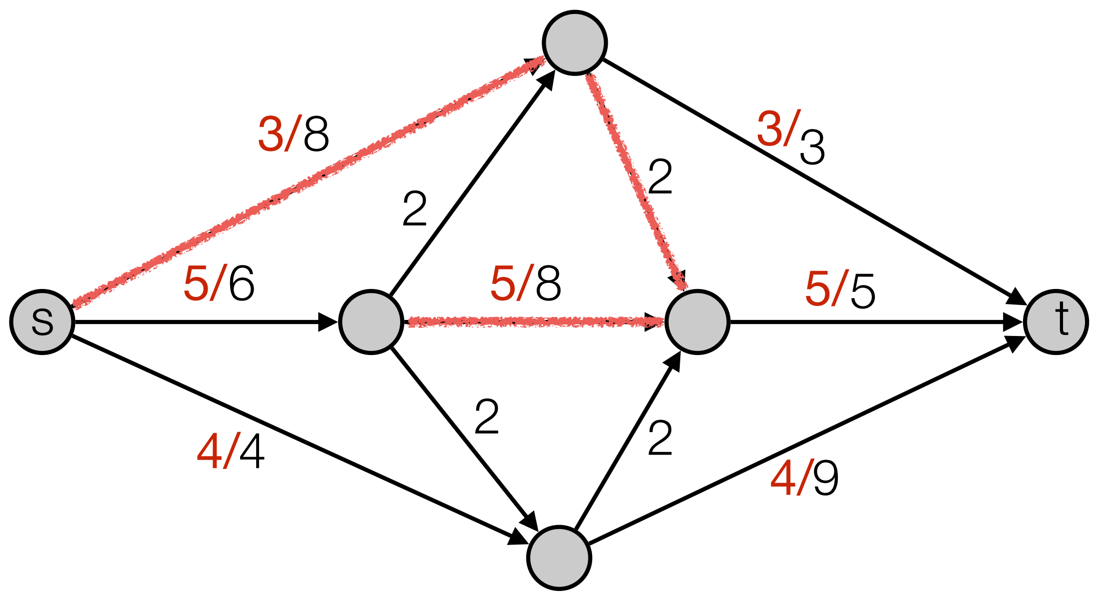




Ford Fulkerson

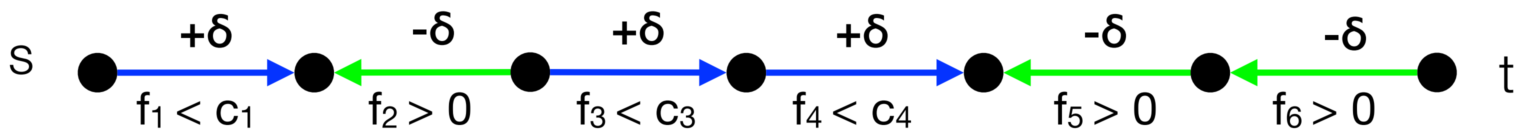
- Augmenting path (definition different than in CLRS): s-t path where
  - forward edges have leftover capacity
  - backwards edges have positive flow

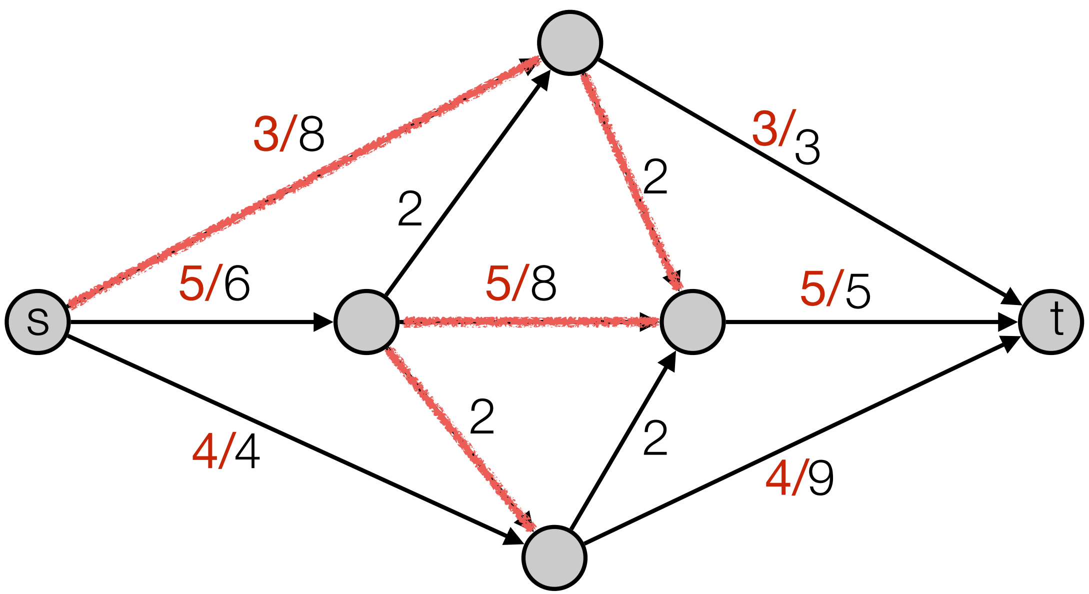




Ford Fulkerson

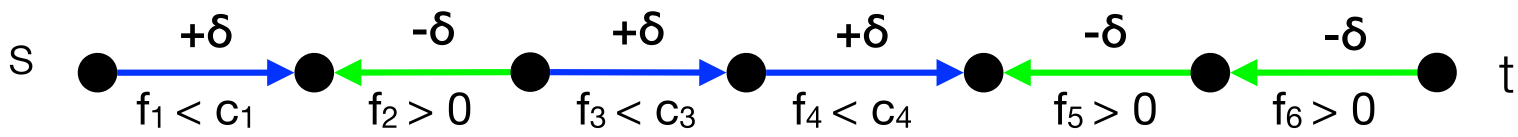
- Augmenting path (definition different than in CLRS): s-t path where
  - forward edges have leftover capacity
  - backwards edges have positive flow

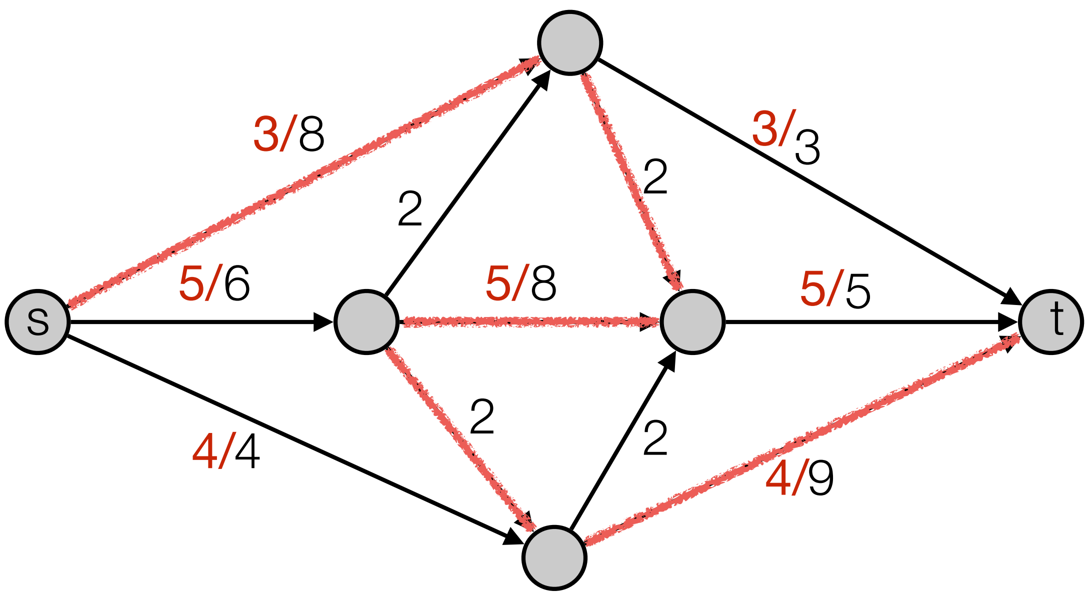




Ford Fulkerson

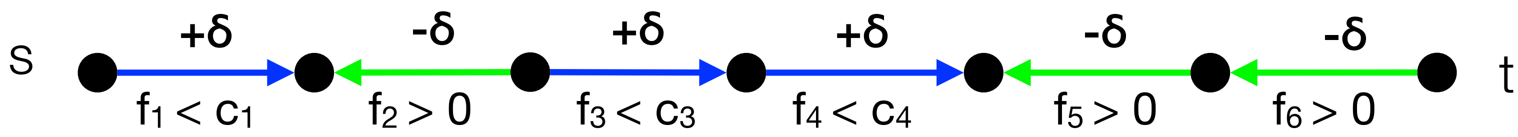
- Augmenting path (definition different than in CLRS): s-t path where
  - forward edges have leftover capacity
  - backwards edges have positive flow

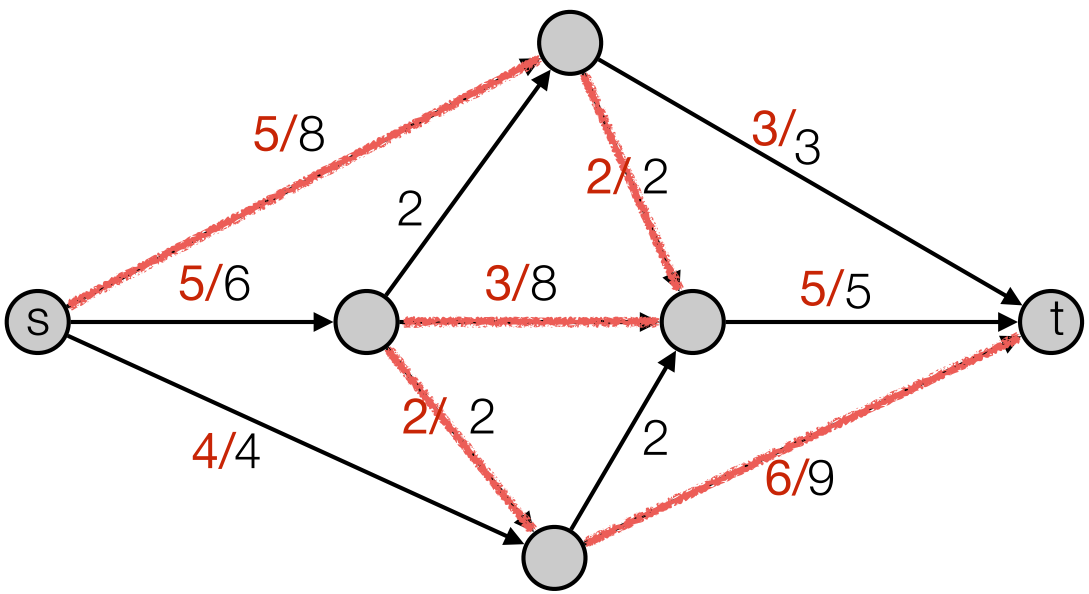




Ford Fulkerson

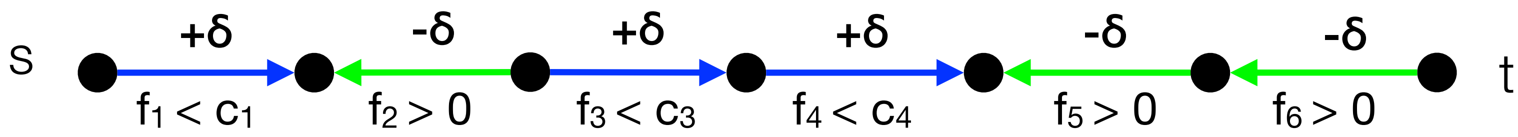
- Augmenting path (definition different than in CLRS): s-t path where
    - forward edges have leftover capacity
    - backwards edges have positive flow

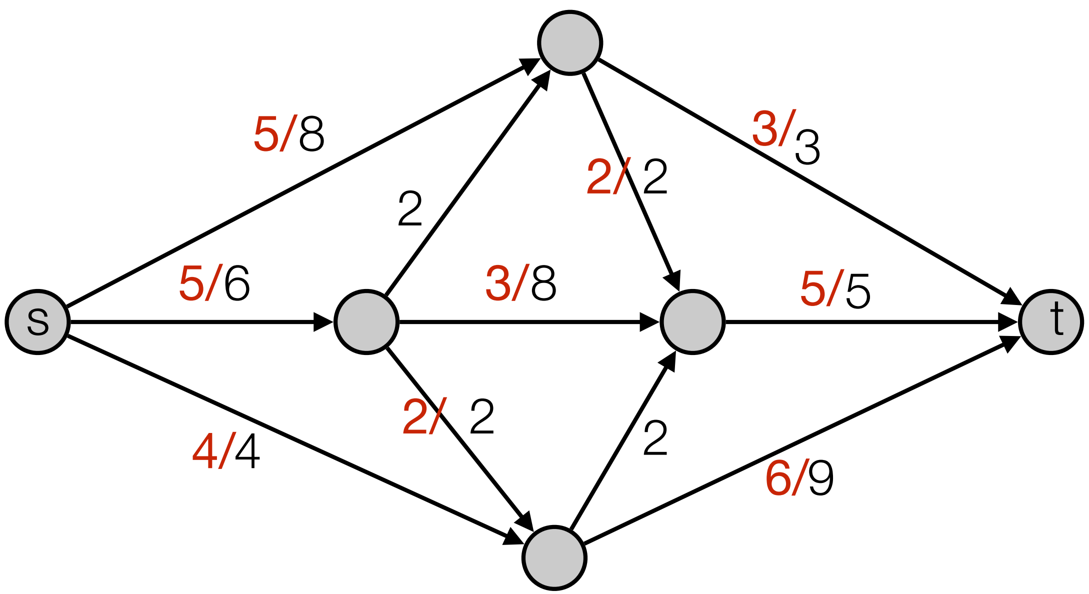




Ford Fulkerson

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





Analysis of Ford-Fulkerson

- Assume integral weights
- Number of iterations:

Analysis of Ford-Fulkerson

- Assume integral weights
- Number of iterations:
    - Always increment flow by at least 1: #iterations ≤ max flow value f*

Analysis of Ford-Fulkerson

- Assume integral weights

- Number of iterations:

    - Always increment flow by at least 1: #iterations ≤ max flow value f*

- Time for one iteration:

Analysis of Ford-Fulkerson

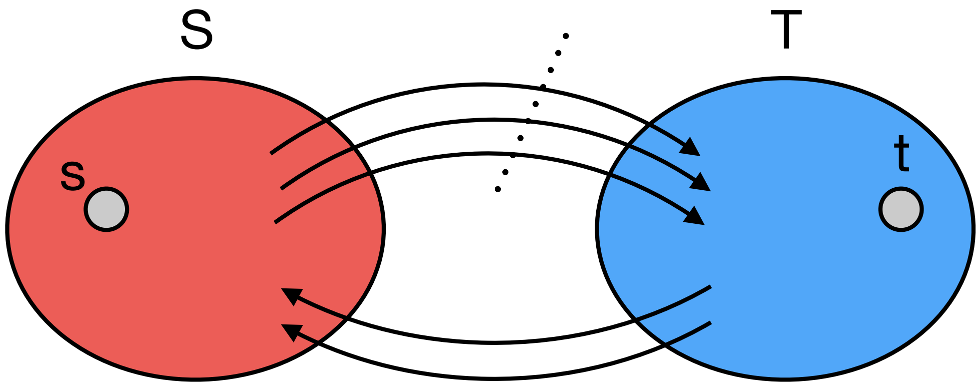
- Assume integral weights
- Number of iterations:
  - Always increment flow by at least 1: #iterations ≤ max flow value f*
- Time for one iteration:
  - Can find augmenting path in linear time: One iteration takes O(m) time.

Analysis of Ford-Fulkerson

- Assume integral weights
- 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).

s-t Cuts

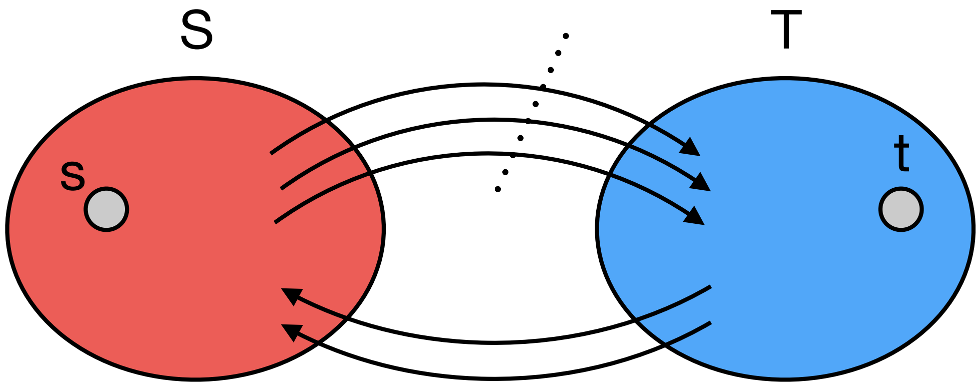
- Cut: Partition of vertices into S and T, such that s ∈ S and t ∈ T.



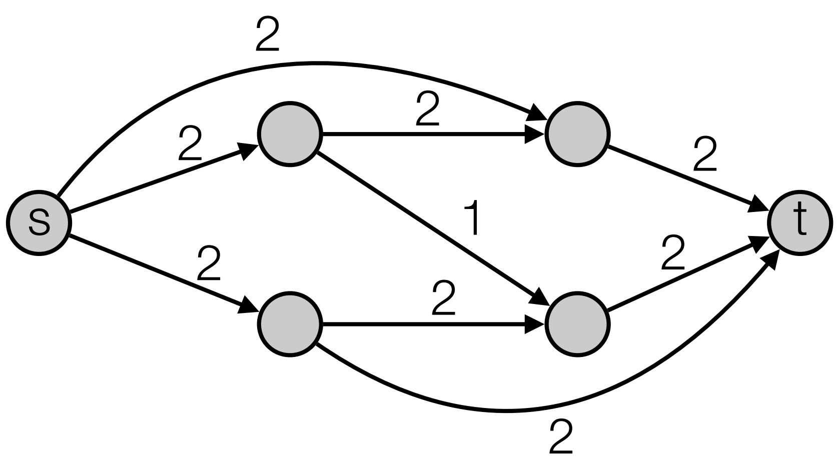
- Capacity of cut: total capacity of edges going *from* S to T.

s-t Cuts

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

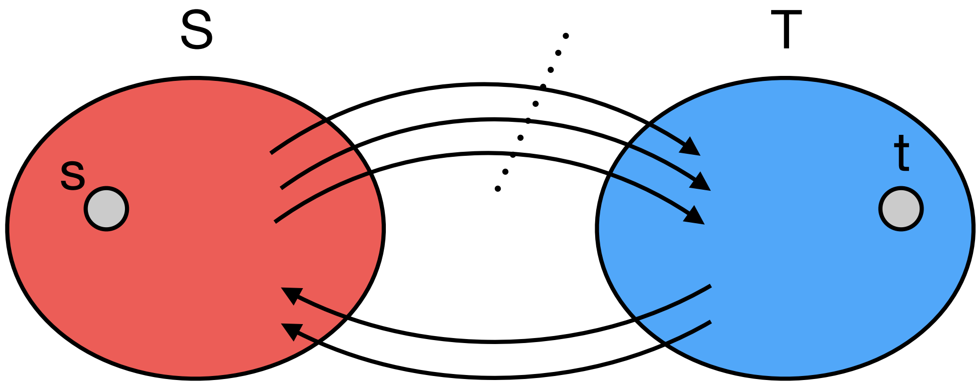


- Capacity of cut: total capacity of edges going *from* S to T.

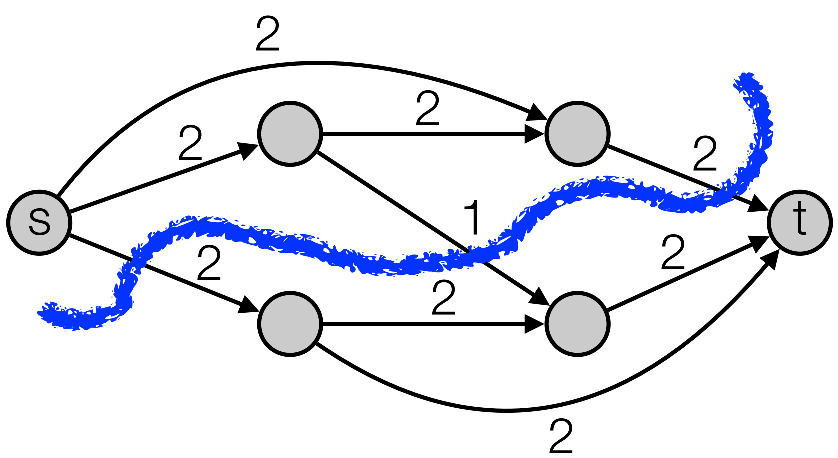


s-t Cuts

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

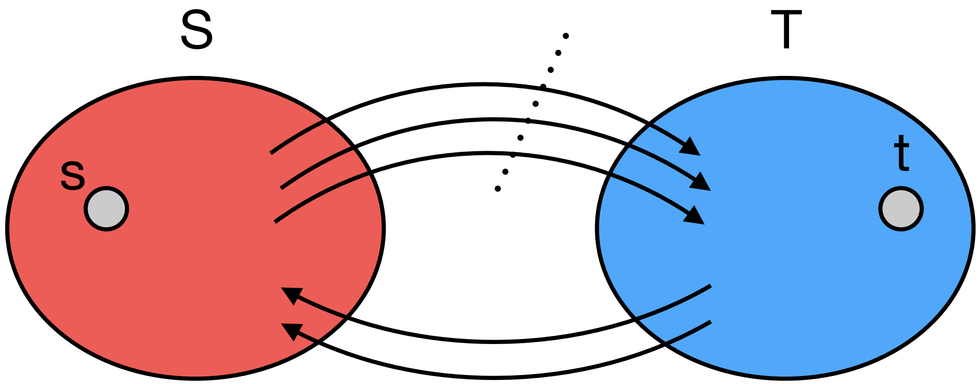


- Capacity of cut: total capacity of edges going *from* S to T.

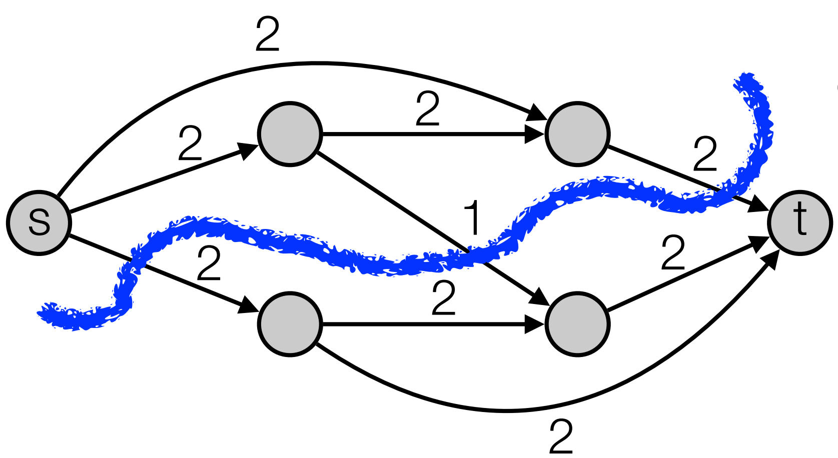


s-t Cuts

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



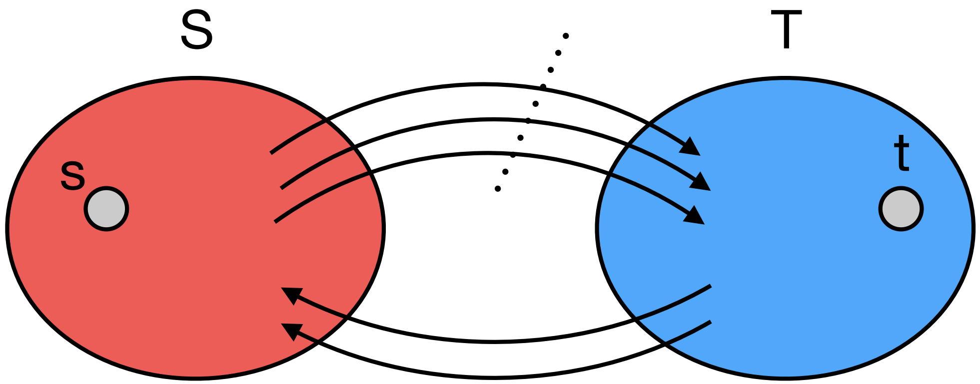
- Capacity of cut: total capacity of edges going *from* S to T.



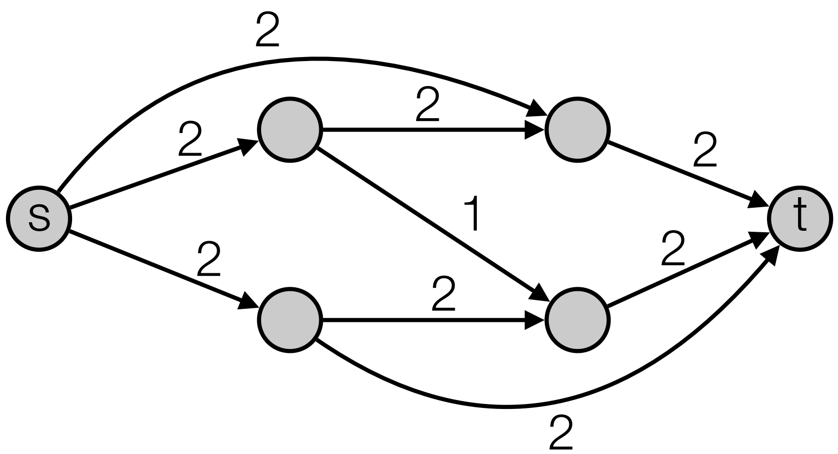
c(S,T) = 5

s-t Cuts

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

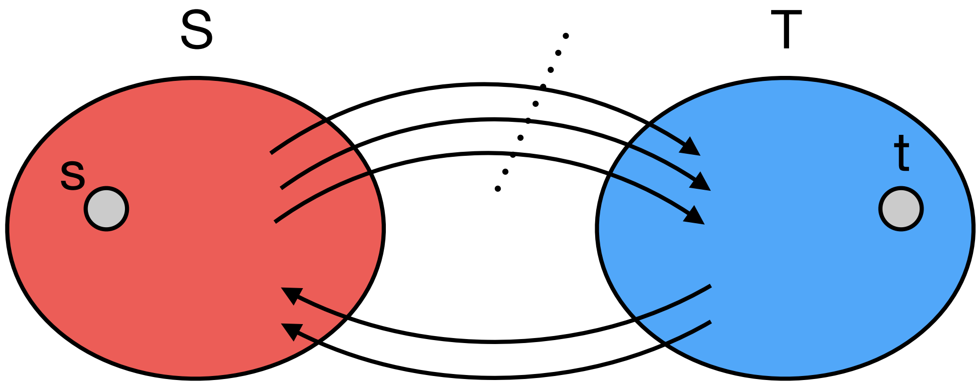


- Capacity of cut: total capacity of edges going *from* S to T.

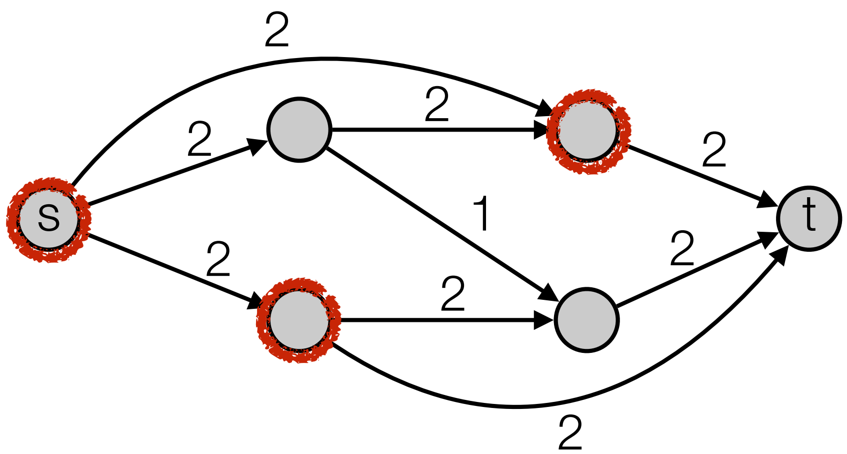


s-t Cuts

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

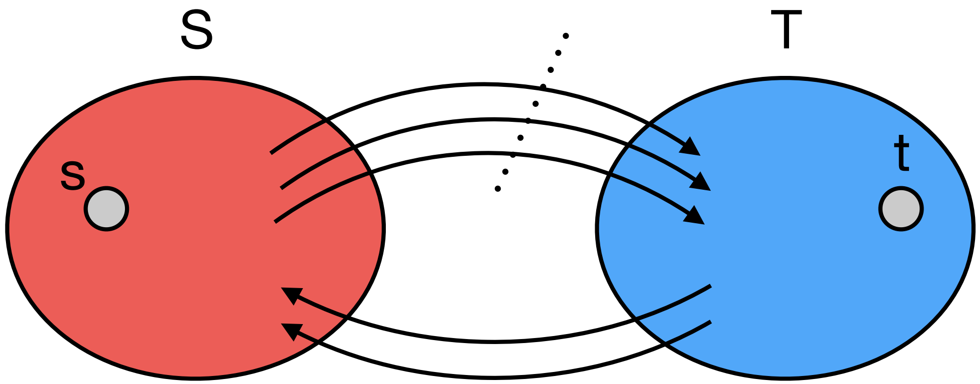


- Capacity of cut: total capacity of edges going *from* S to T.

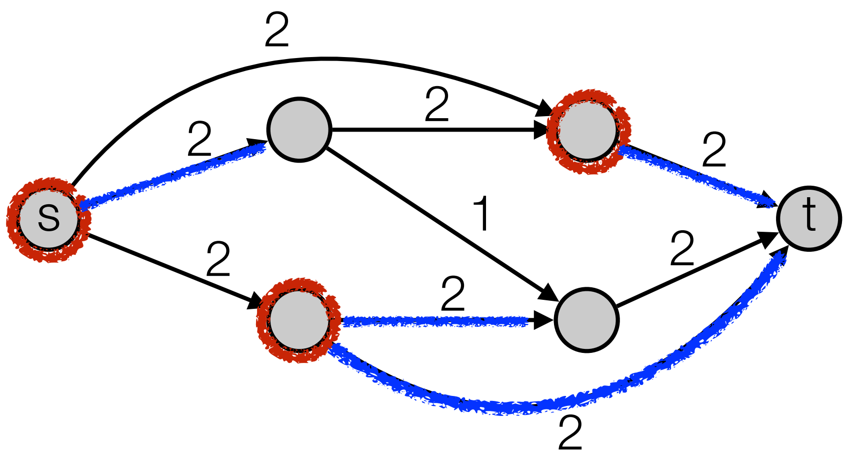


s-t Cuts

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

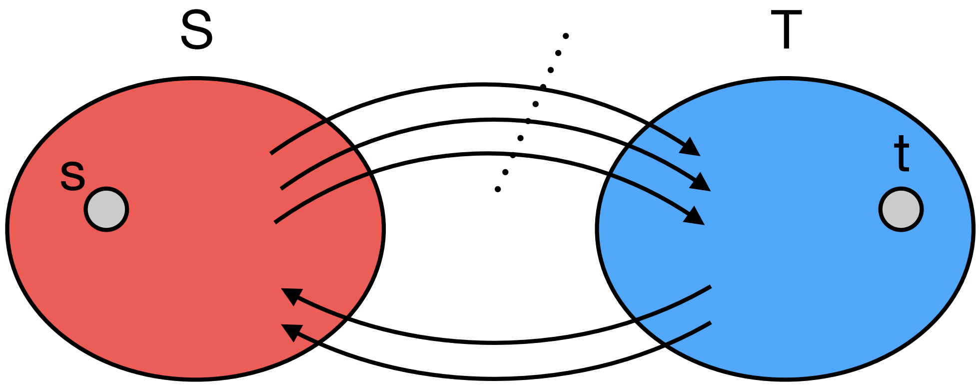


- Capacity of cut: total capacity of edges going *from* S to T.

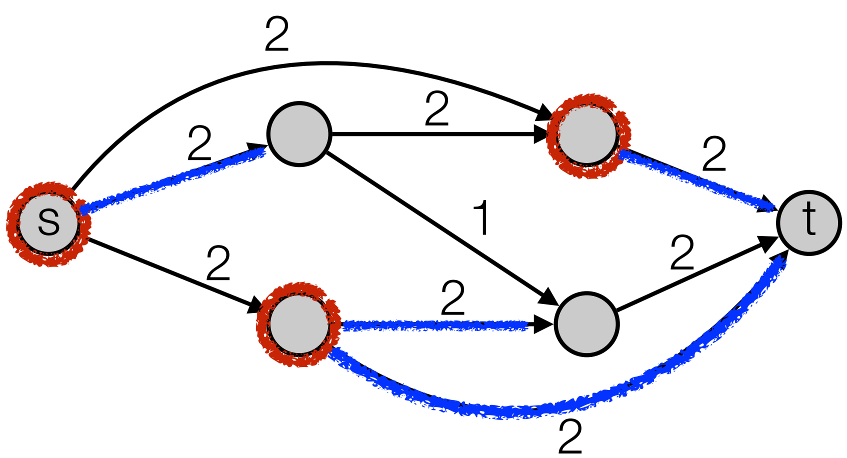


s-t Cuts

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



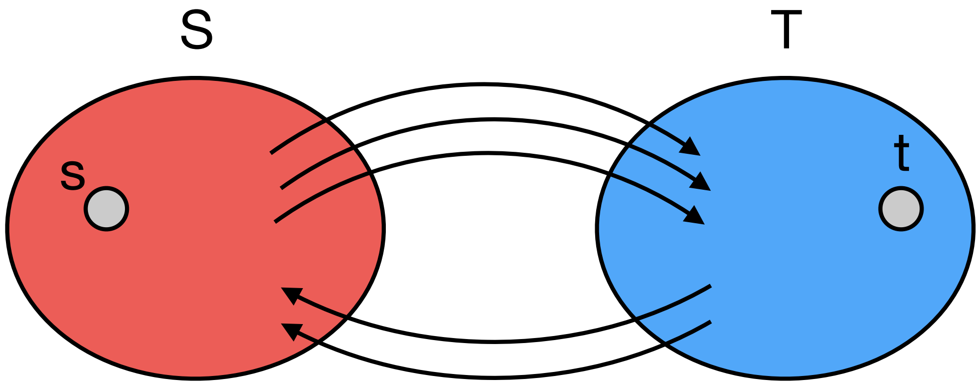
- Capacity of cut: total capacity of edges going *from* S to T.



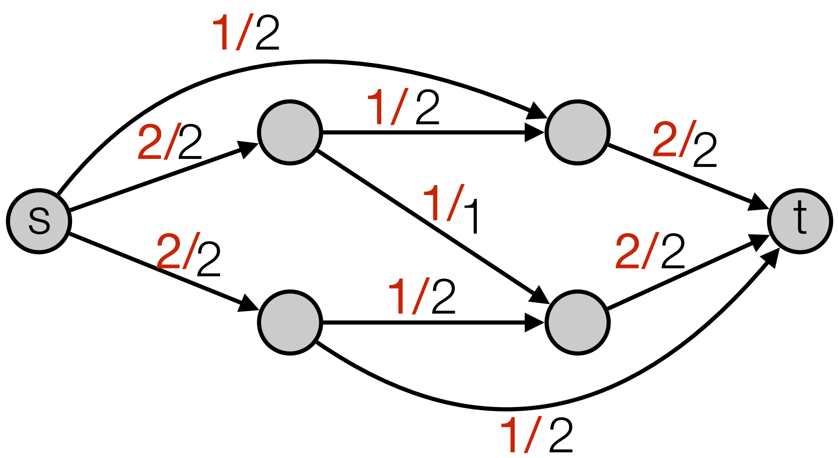
c(S,T) = 8

s-t Cuts

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

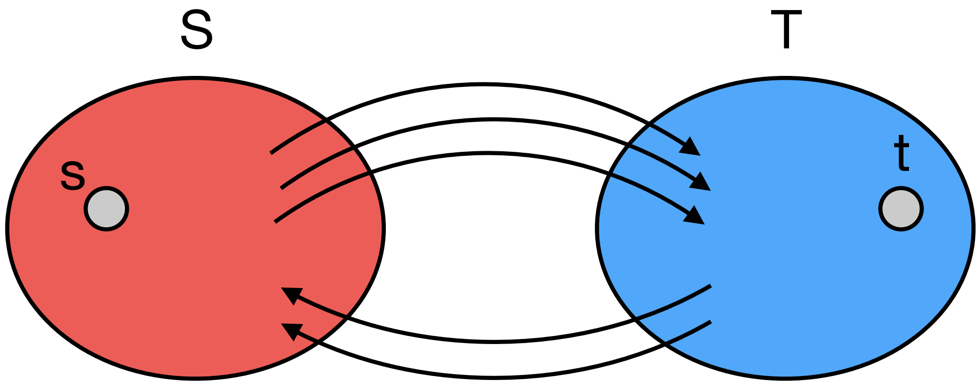


- Flow across cut: = flow *from* S to T *minus flow from* T to S.

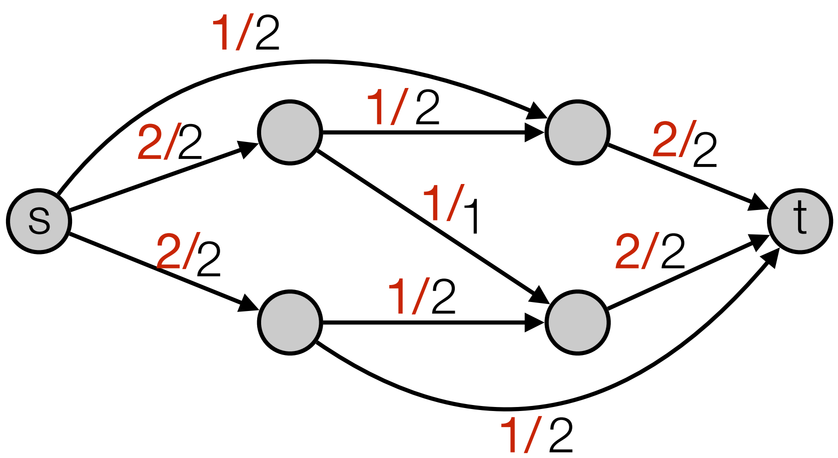


s-t Cuts

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

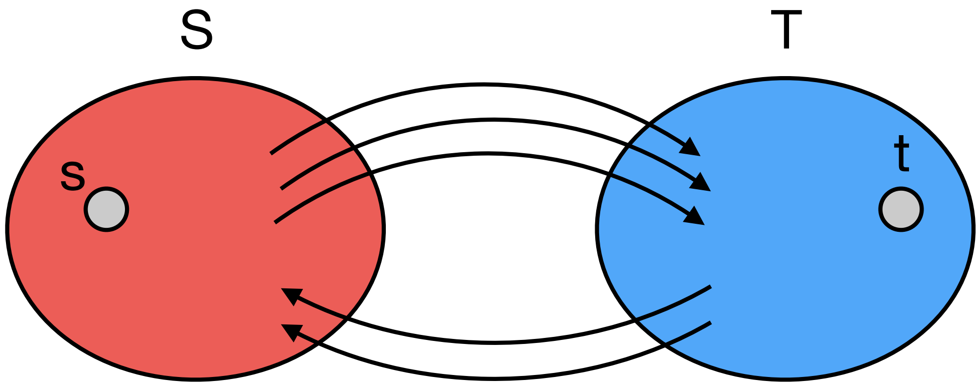


- Flow across cut: = flow *from* S to T *minus flow from* T to S.

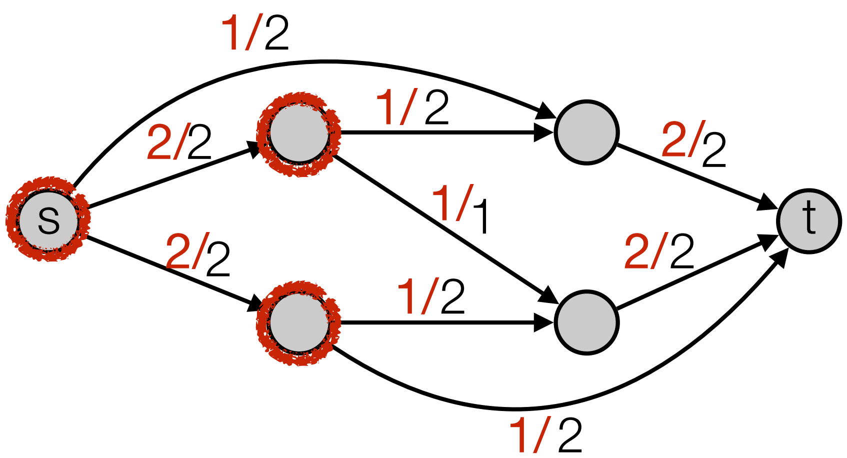


s-t Cuts

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

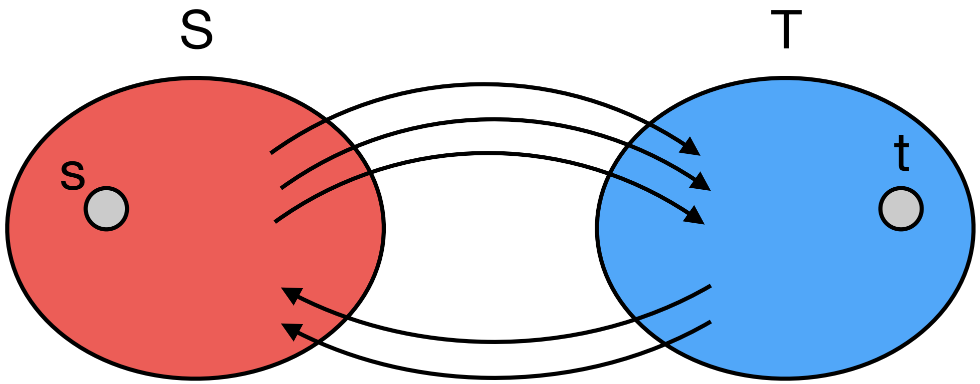


- Flow across cut: = flow *from* S to T *minus flow from* T to S.

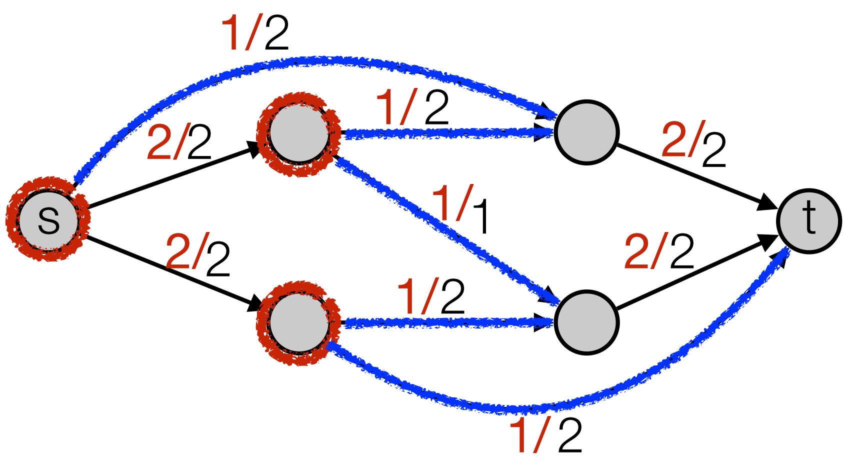


s-t Cuts

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

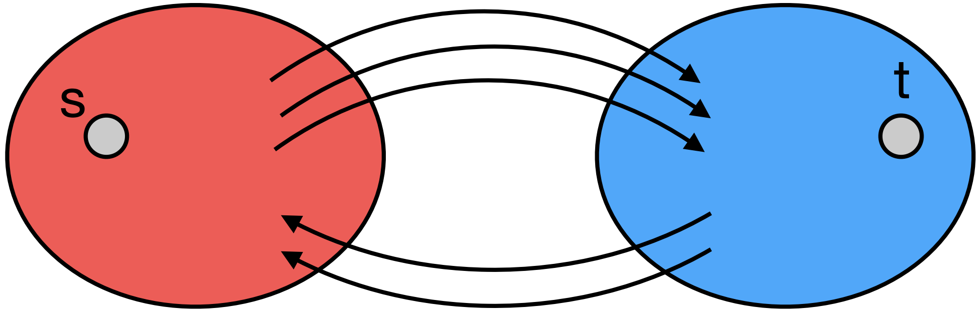


- Flow across cut: = flow *from* S to T *minus flow from* T to S.

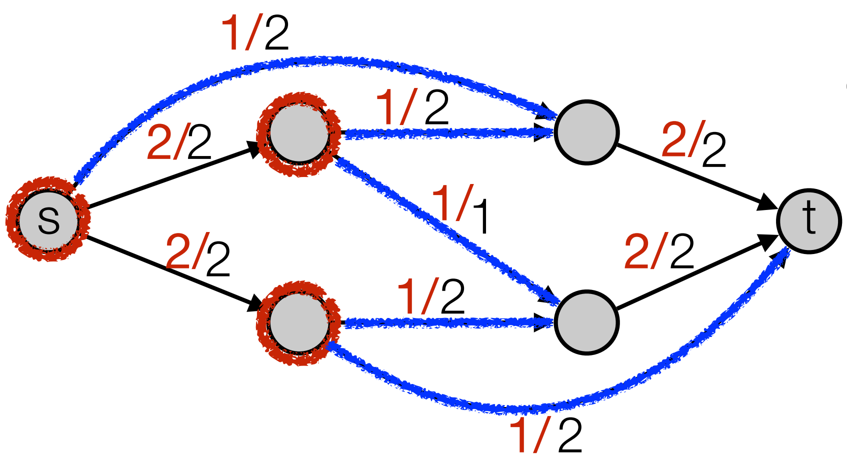


s-t Cuts

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



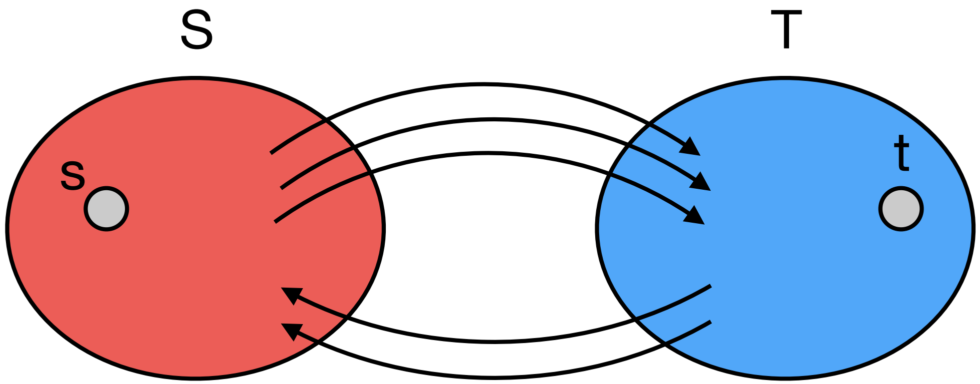
- Flow across cut: = flow *from* S to T *minus flow from* T to S.



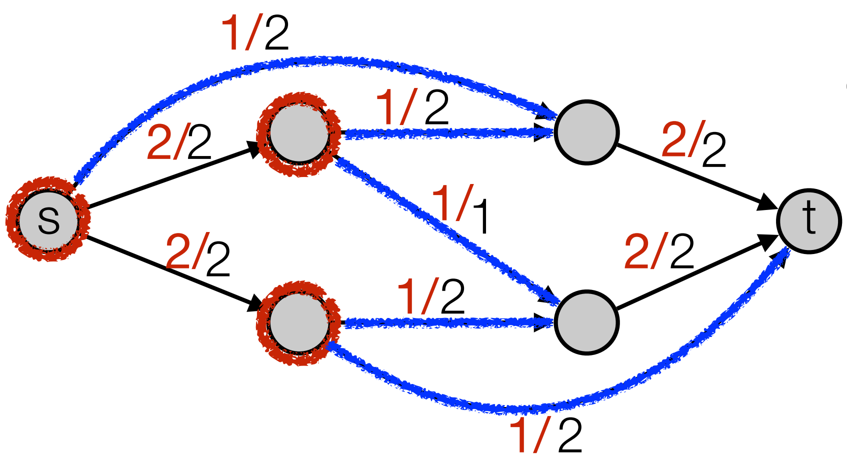
c(S,T) = 9

s-t Cuts

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



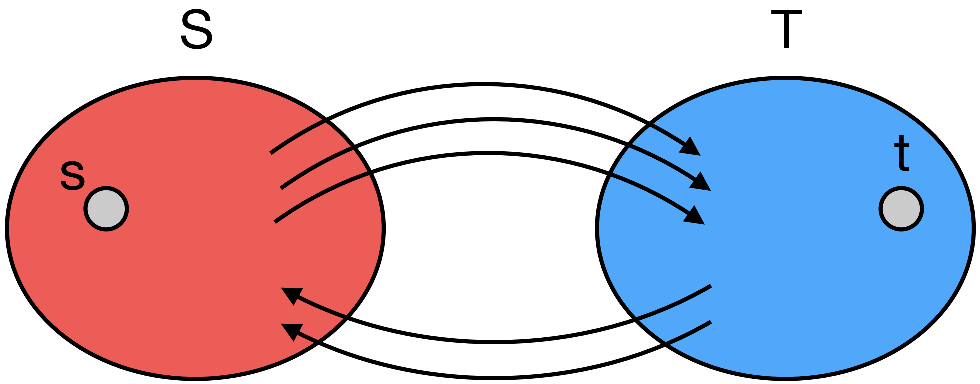
- Flow across cut: = flow *from* S to T *minus flow from* T to S.



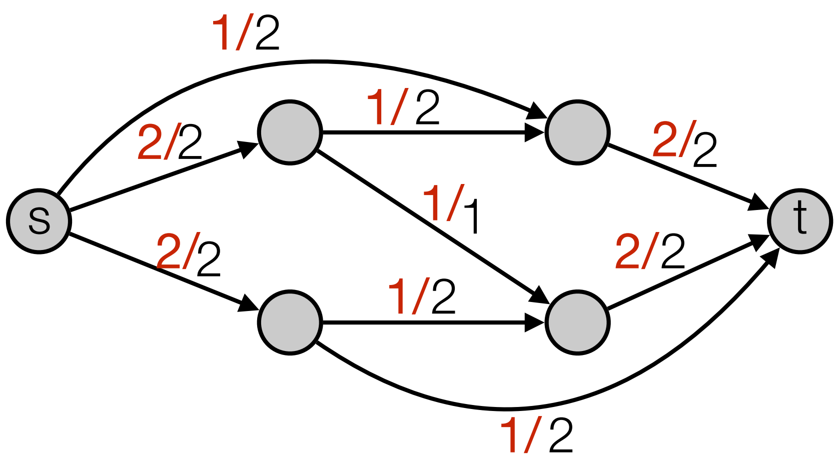
c(S,T) = 9   f(S,T) = 5

s-t Cuts

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

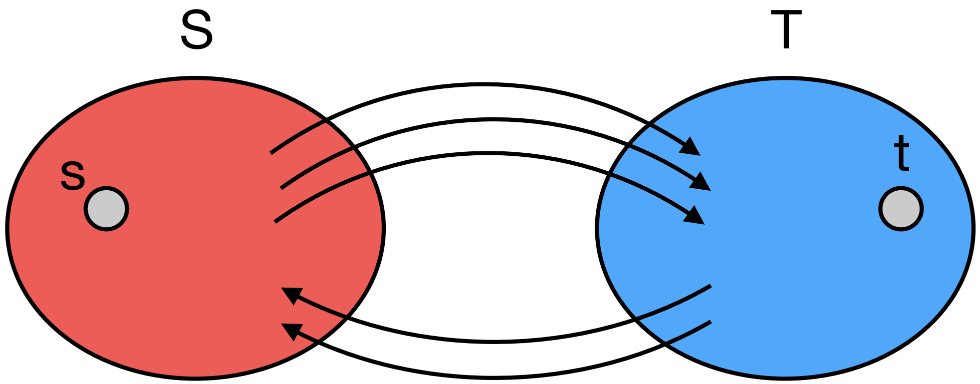


- Flow across cut: = flow *from* S to T *minus flow from* T to S.

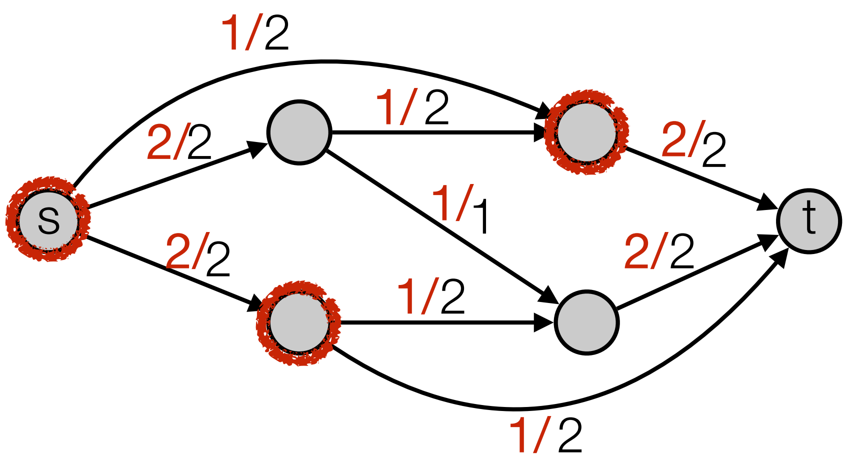


s-t Cuts

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

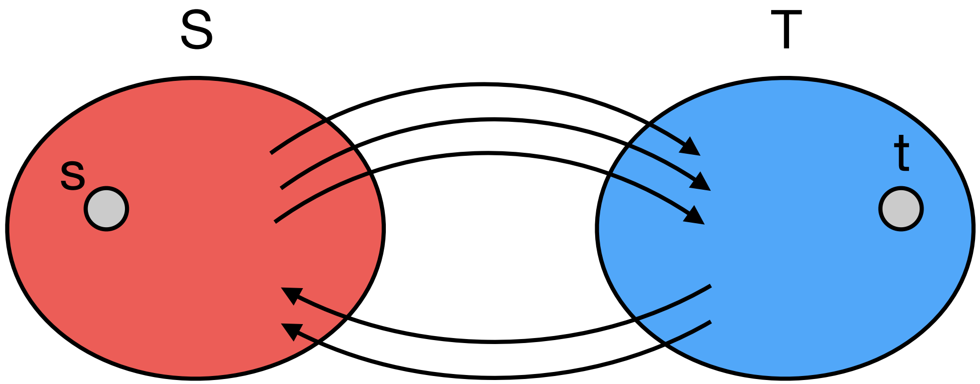


- Flow across cut: = flow *from* S to T *minus flow from* T to S.

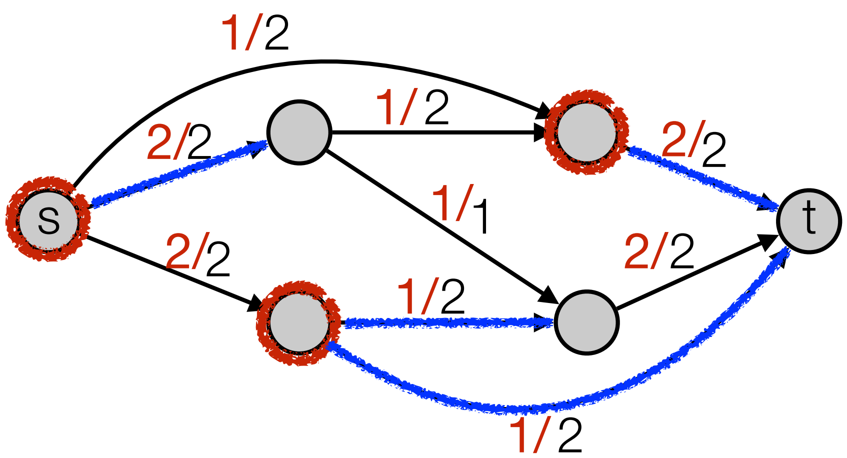


s-t Cuts

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

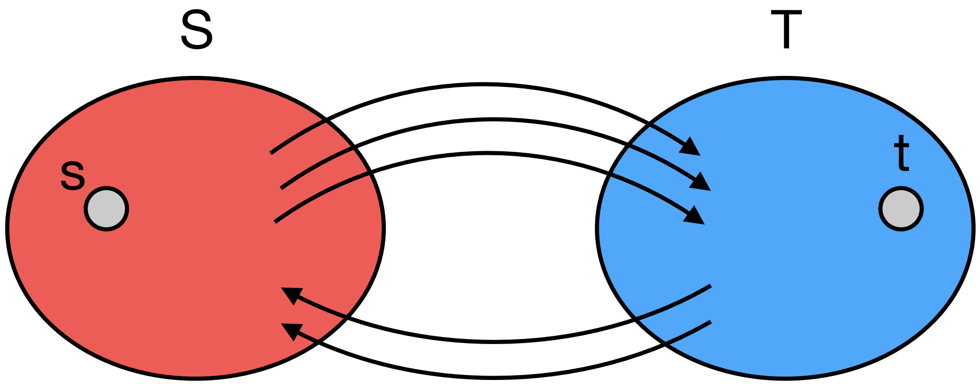


- Flow across cut: = flow *from* S to T *minus flow from* T to S.

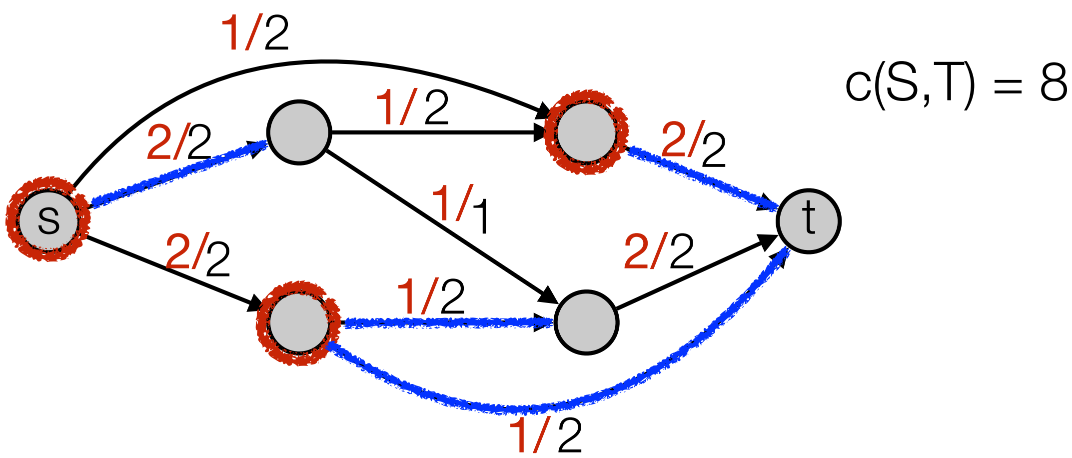


s-t Cuts

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

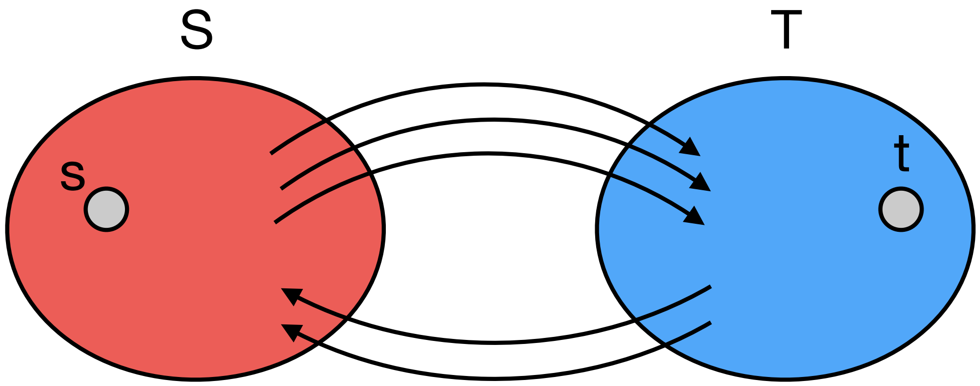


- Flow across cut: = flow *from* S to T *minus flow from* T to S.

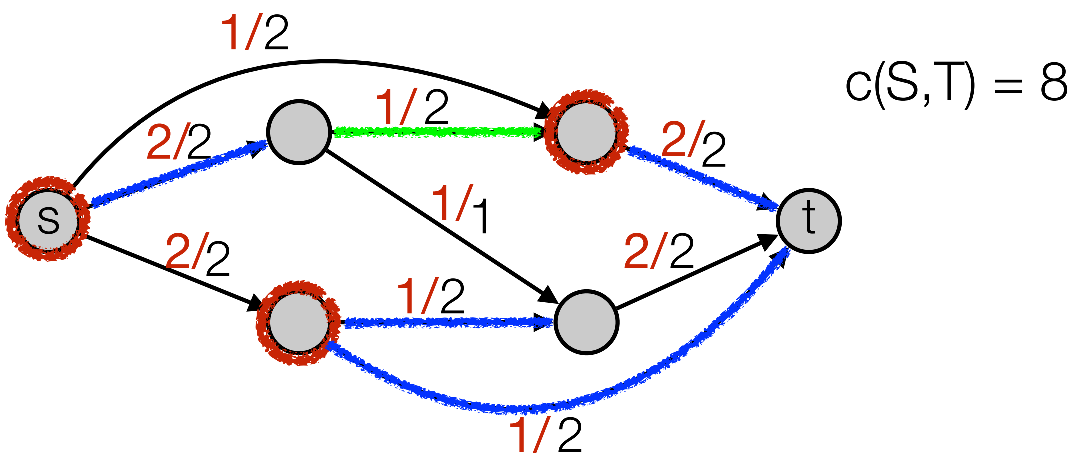


s-t Cuts

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

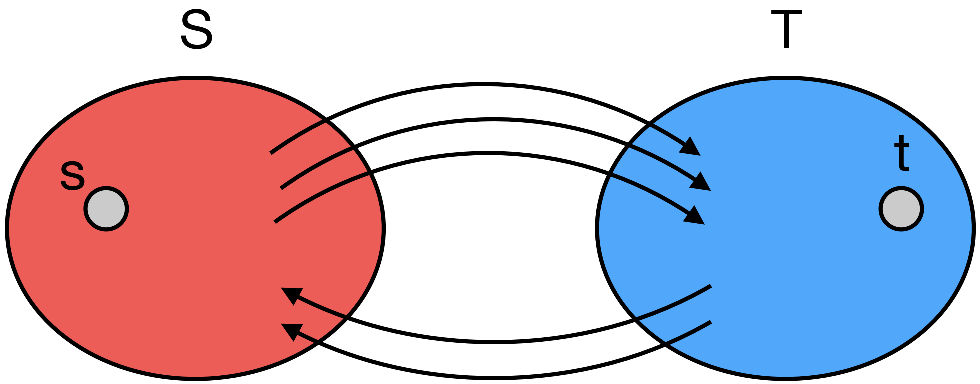


- Flow across cut: = flow *from* S to T *minus flow from* T to S.

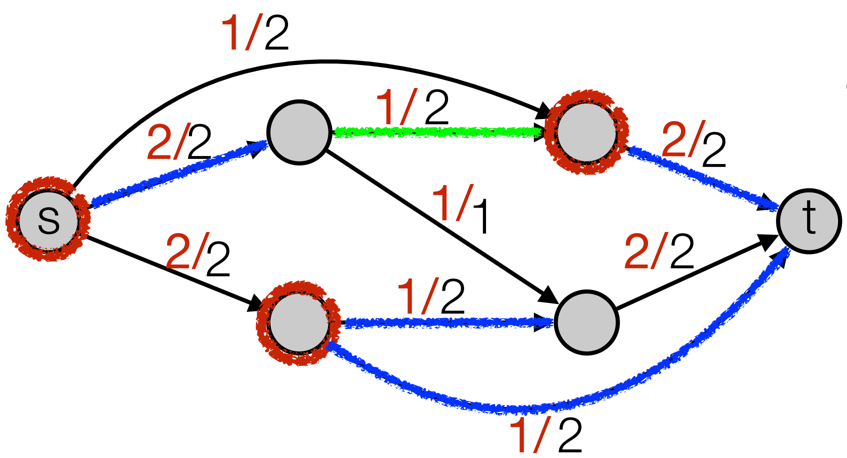


s-t Cuts

- 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.

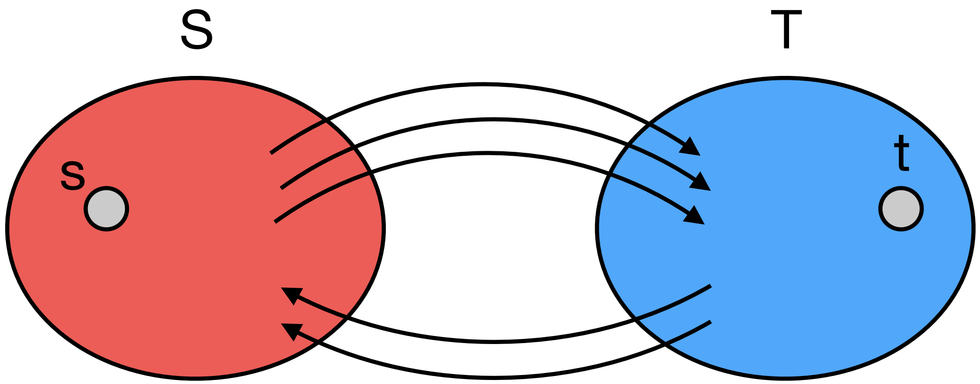


c(S,T) = 8

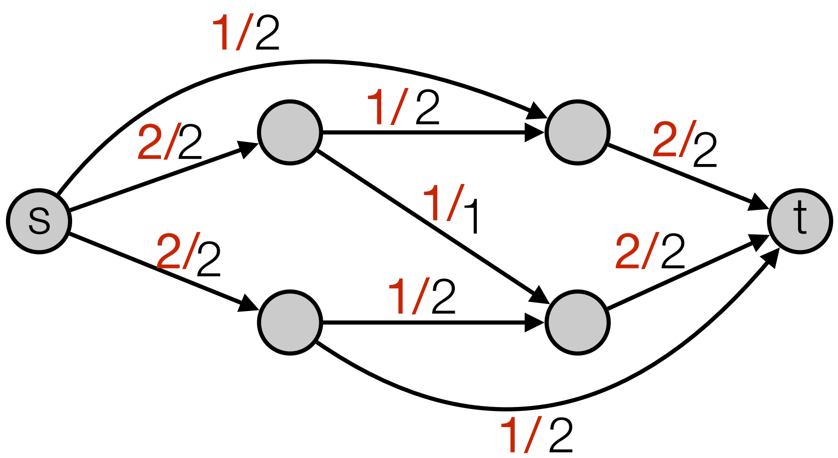
f(S,T) = 6 - 1 = 5

s-t Cuts

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

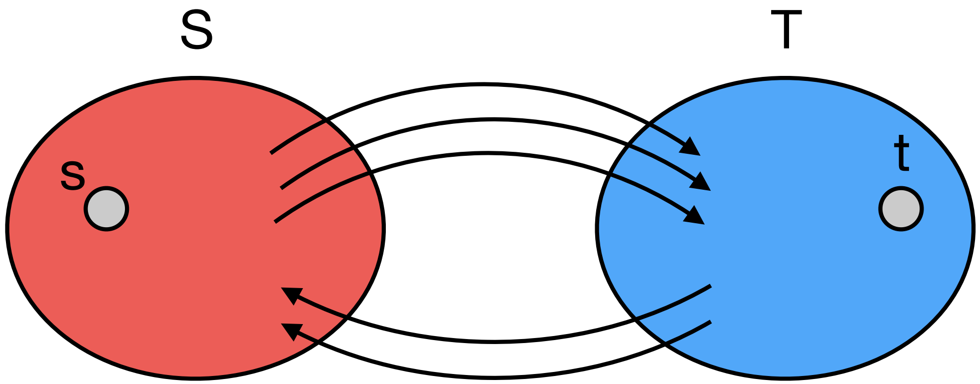


- Flow across cut: = flow *from* S to T *minus flow from* T to S.

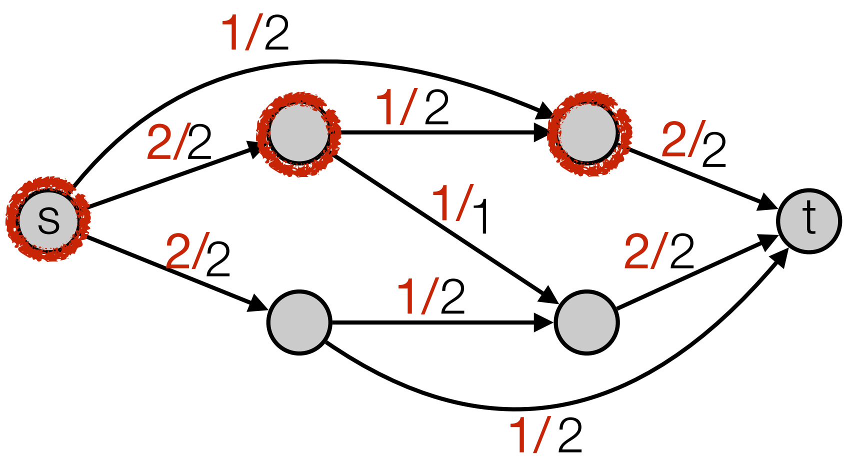


s-t Cuts

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

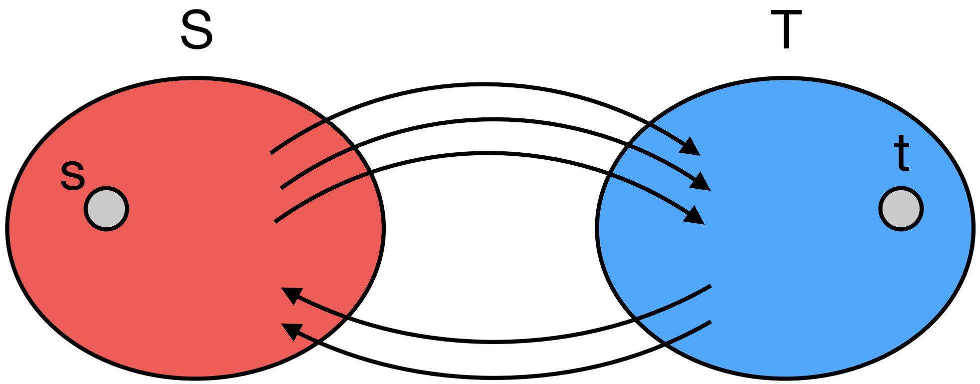


- Flow across cut: = flow *from* S to T *minus flow from* T to S.

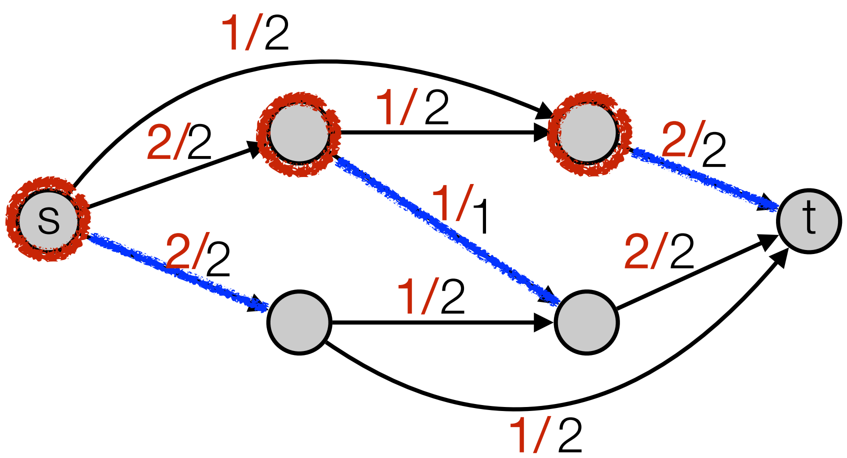


s-t Cuts

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

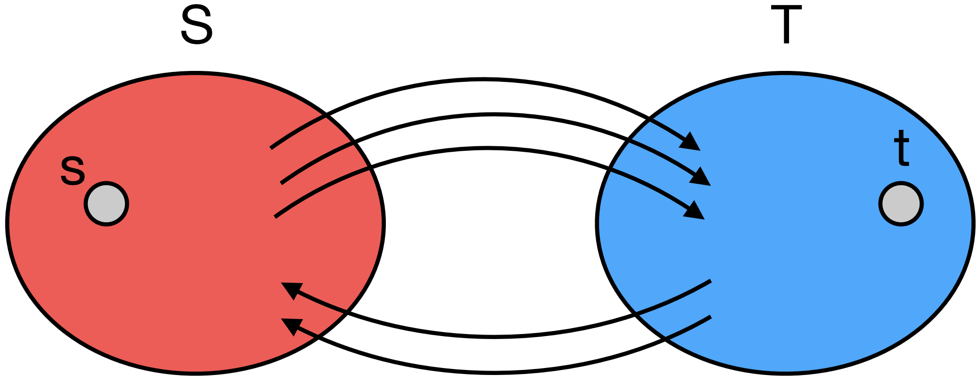


- Flow across cut: = flow *from* S to T *minus flow from* T to S.

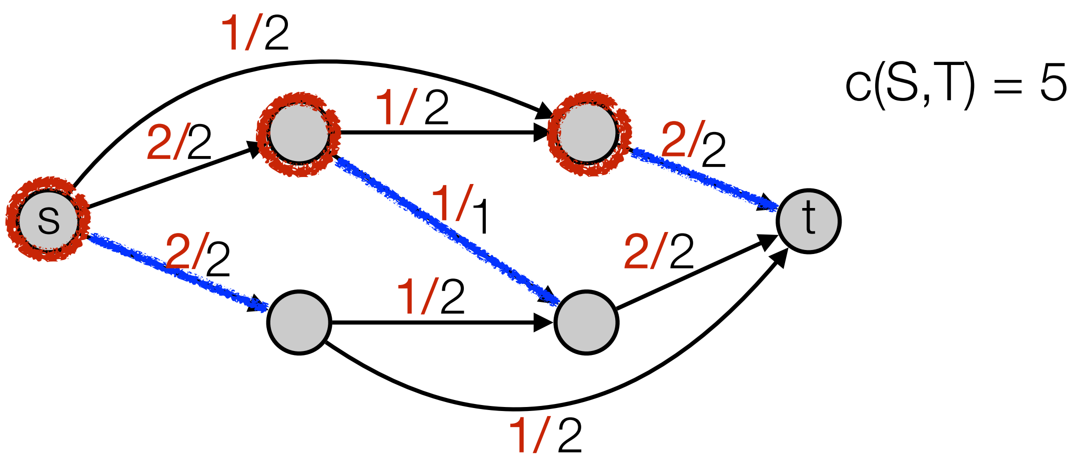


s-t Cuts

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

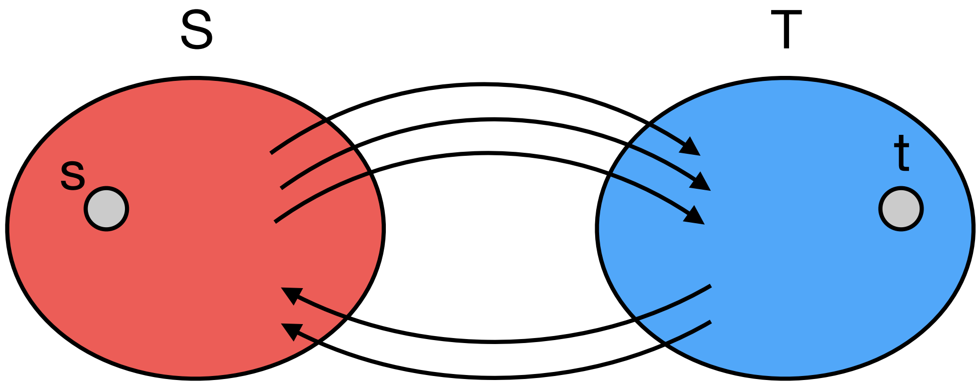


- Flow across cut: = flow *from* S to T *minus flow from* T to S.

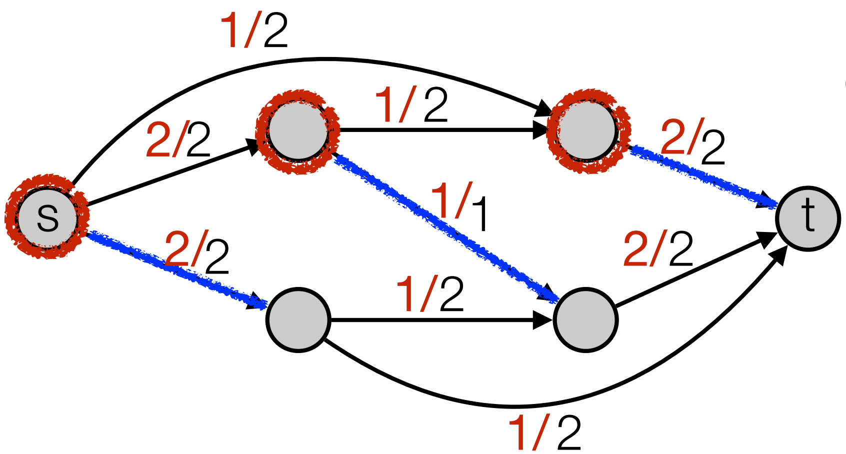


s-t Cuts

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



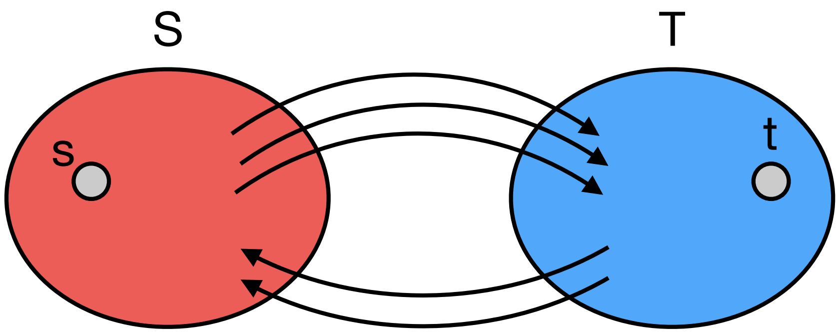
- Flow across cut: = flow *from* S to T *minus flow from* T to S.



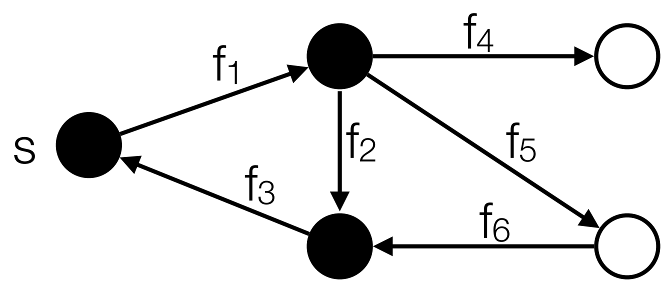
c(S,T) = 5 f(S,T) = 5

s-t Cuts

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

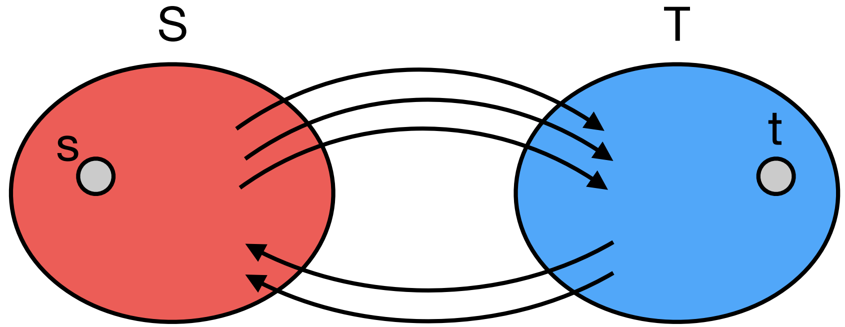


- Capacity of cut: total capacity of edges going *from* S to T.

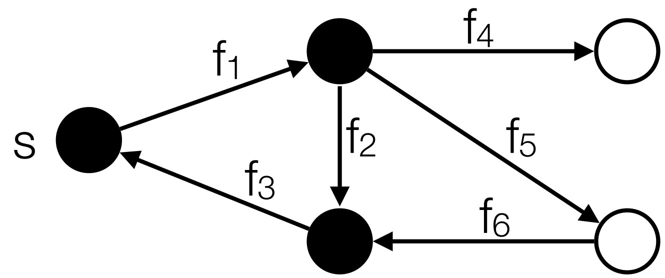


s-t Cuts

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

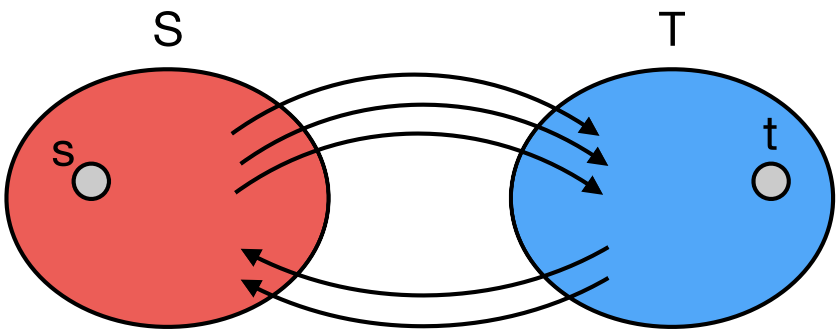


- Capacity of cut: total capacity of edges going *from* S to T.
- Flow across cut = flow *from* S to T *minus flow from* T to S.

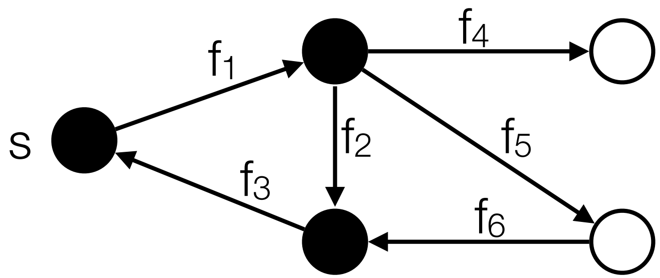


s-t Cuts

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

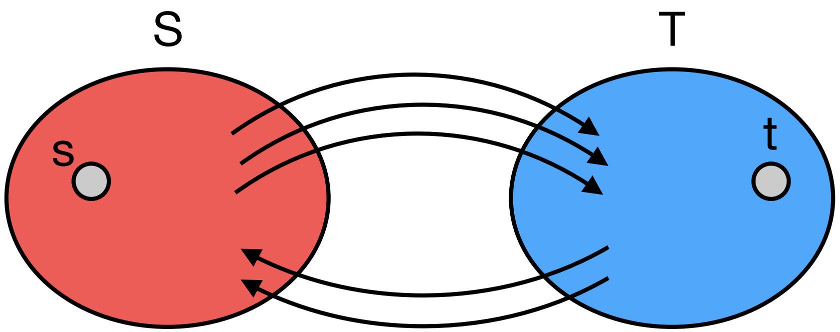


- Capacity of cut: total capacity of edges going *from* S to T.
- Flow across cut = flow *from* S to T *minus flow from* T to S.
- Flow across cut: $f_4 + f_5 - f_6 = ?$

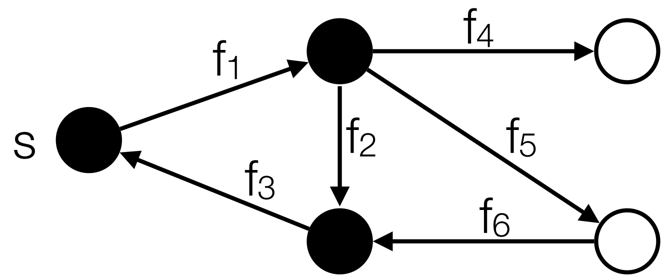


s-t Cuts

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

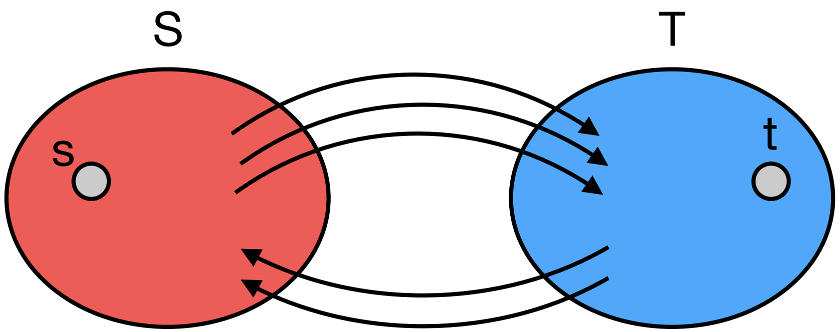


- Capacity of cut: total capacity of edges going *from* S to T.
- Flow across cut = flow *from* S to T *minus flow from* T to S.
- Flow across cut: $f_4 + f_5 - f_6 = ?$
  - $f_2 + f_4 + f_5 - f_1 = 0$

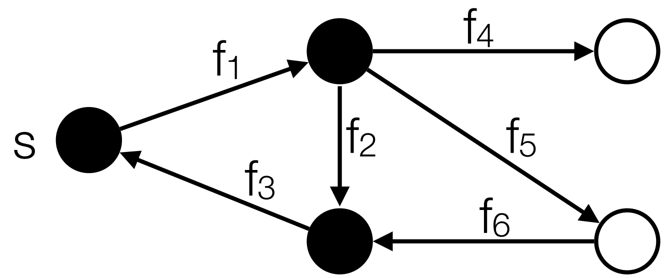


s-t Cuts

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

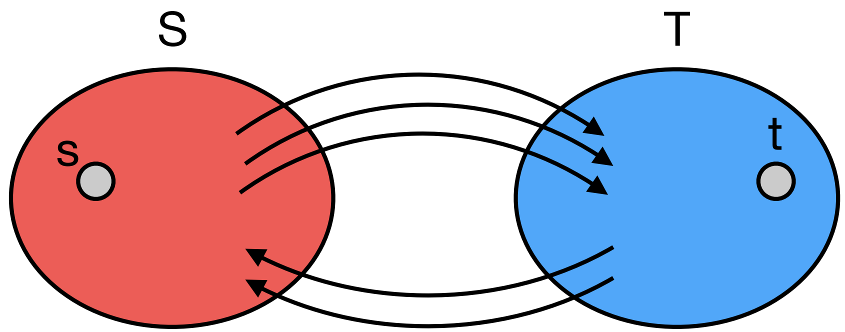


- Capacity of cut: total capacity of edges going *from* S to T.
- Flow across cut = flow *from* S to T *minus flow from* T to S.
- Flow across cut: $f_4 + f_5 - f_6 = ?$
  - $f_2 + f_4 + f_5 - f_1 = 0$
  - $f_3 - f_2 - f_6 = 0$

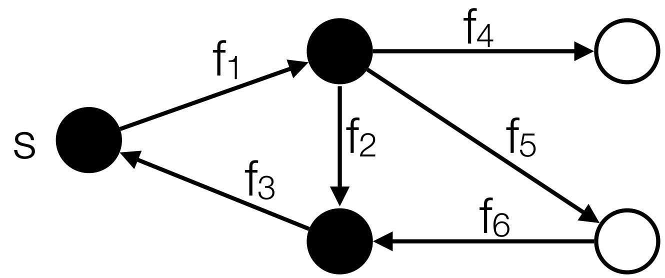


s-t Cuts

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

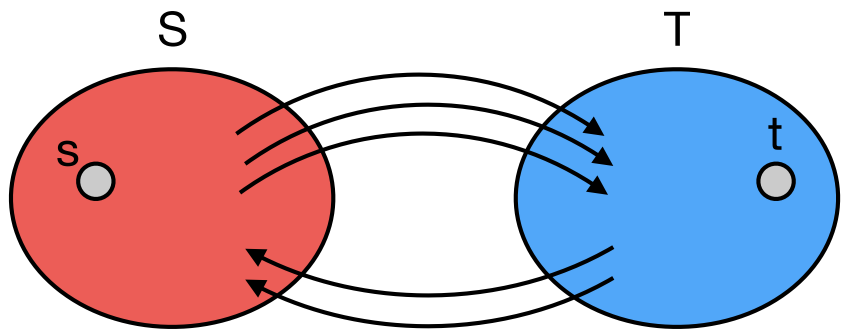


- Capacity of cut: total capacity of edges going *from* S to T.
- Flow across cut = flow *from* S to T *minus flow from* T to S.
- Flow across cut: $f_4 + f_5 - f_6 = ?$
  - $f_2 + f_4 + f_5 - f_1 = 0$
  - $f_3 - f_2 - f_6 = 0$
  - $f_1 - f_3 = |f|$

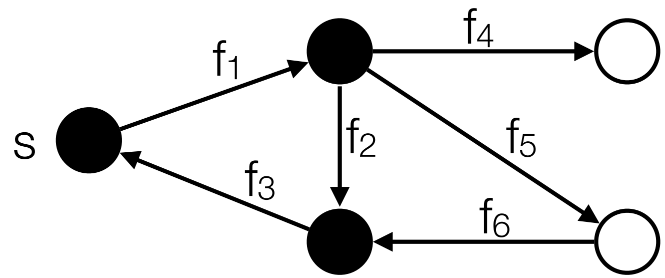


s-t Cuts

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

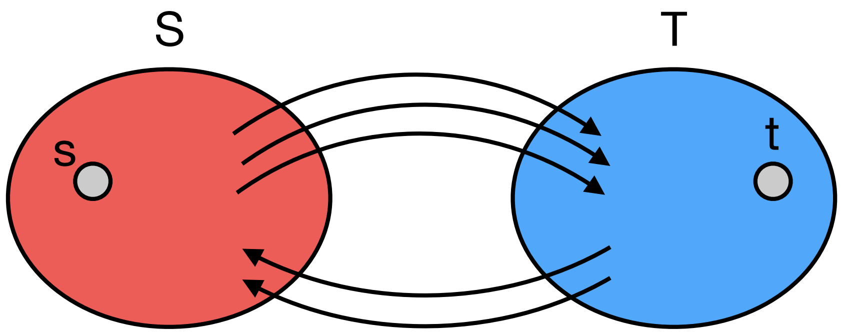


- Capacity of cut: total capacity of edges going *from* S to T.
- Flow across cut = flow *from* S to T *minus flow from* T to S.
- Flow across cut: $f_4 + f_5 - f_6 = ?$
    - $f_2 + f_4 + f_5 - f_1 = 0$
    - $f_3 - f_2 - f_6 = 0$
    - $f_1 - f_3 = |f|$
    - $(f_2 + f_4 - f_1 + f_5) + (f_3 - f_2 -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

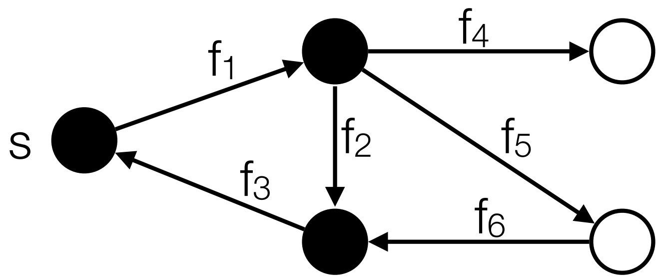


s-t Cuts

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

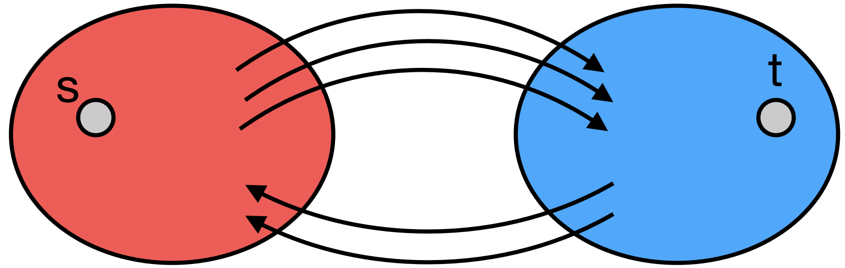


- Capacity of cut: total capacity of edges going *from* S to T.
- Flow across cut = flow *from* S to T *minus flow from* T to S.
- Flow across cut: $f_4 + f_5 - f_6 = ?$
  - $f_2 + f_4 + f_5 - f_1 = 0$
  - $f_3 - f_2 - f_6 = 0$
  - $f_1 - f_3 = |f|$
  - $(\cancel{f_2} + f_4 - f_1 + f_5) + (f_3 - \cancel{f_2} - f_6) + (f_1 - f_3) = |f|$

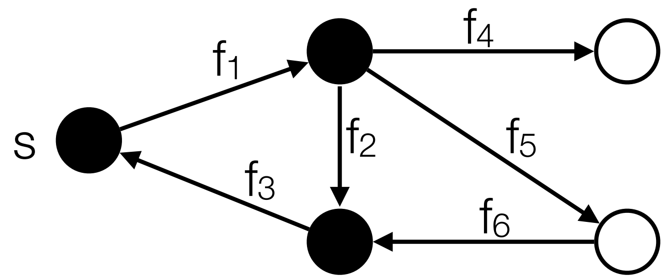


s-t Cuts

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

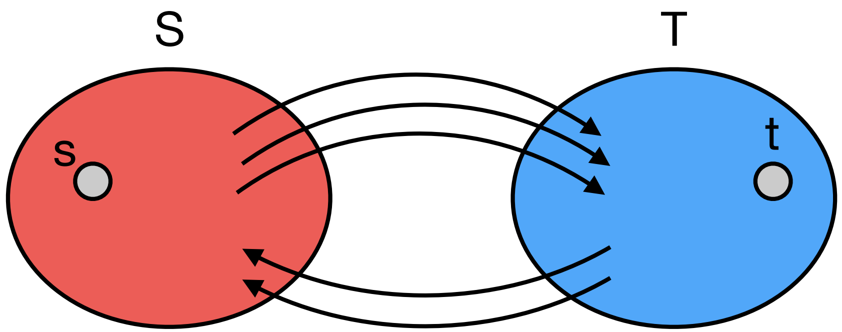


- Capacity of cut: total capacity of edges going *from* S to T.
- Flow across cut = flow *from* S to T *minus flow from* T to S.
- Flow across cut: $f_4 + f_5 - f_6 = ?$
  - $f_2 + f_4 + f_5 - f_1 = 0$
  - $f_3 - f_2 - f_6 = 0$
  - $f_1 - f_3 = |f|$
  - $(f_2 + f_4 - f_1 + f_5) + (f_3 - f_2 - f_6) +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

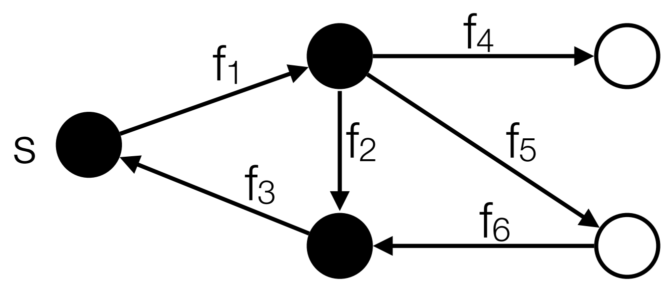


s-t Cuts

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

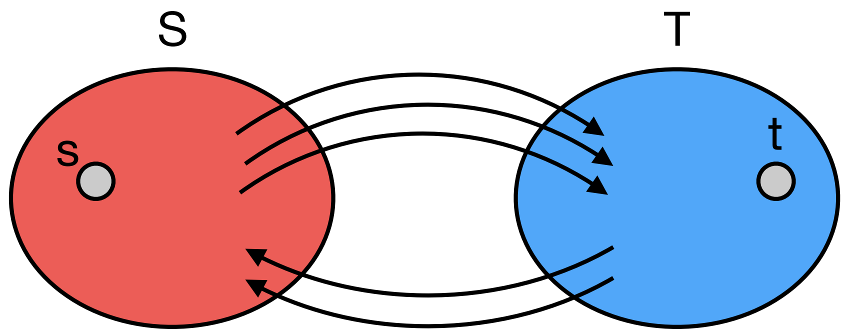


- Capacity of cut: total capacity of edges going *from* S to T.
- Flow across cut = flow *from* S to T *minus flow from* T to S.
- Flow across cut: $f_4 + f_5 - f_6 = ?$
  - $f_2 + f_4 + f_5 - f_1 = 0$
  - $f_3 - f_2 - f_6 = 0$
  - $f_1 - f_3 = |f|$
  - $(\cancel{f_2} + f_4 - \cancel{f_1} + f_5) + (\cancel{f_3} - \cancel{f_2} - f_6) + (\cancel{f_1} - \cancel{f_3}) = |f|$

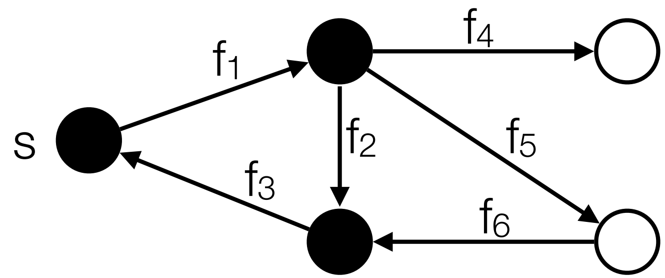


s-t Cuts

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

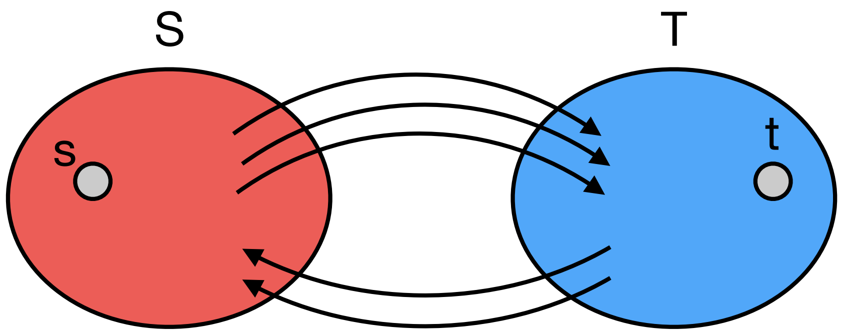


- Capacity of cut: total capacity of edges going *from* S to T.
- Flow across cut = flow *from* S to T *minus flow from* T to S.
- Flow across cut: $f_4 + f_5 - f_6 = ?$
  - $f_2 + f_4 + f_5 - f_1 = 0$
  - $f_3 - f_2 - f_6 = 0$
  - $f_1 - f_3 = |f|$
  - $(\cancel{f_2} + f_4 - \cancel{f_1} + f_5) + (\cancel{f_3} - \cancel{f_2} - f_6) + (\cancel{f_1} - \cancel{f_3}) = |f|$
  - $f_4 + f_5 - f_6 = |f|$

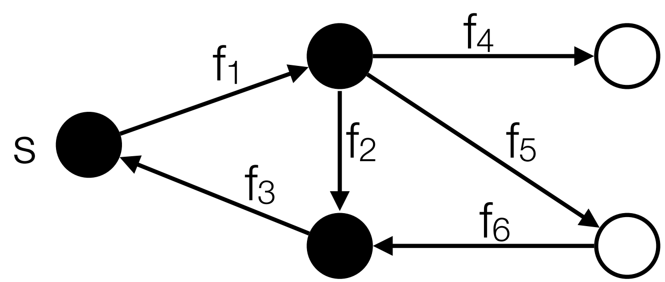


s-t Cuts

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



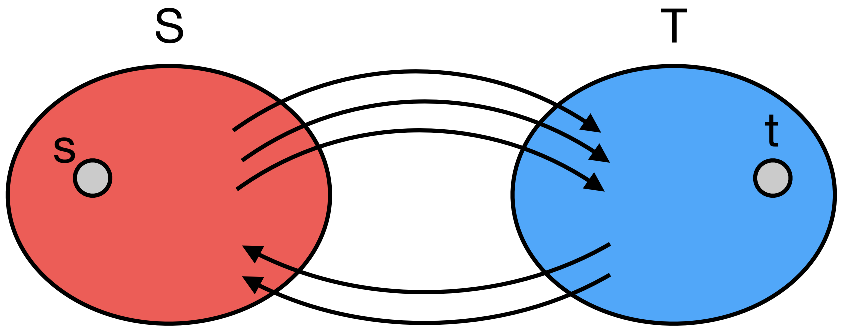
- Capacity of cut: total capacity of edges going *from* S to T.
- Flow across cut = flow *from* S to T *minus flow from* T to S.
- Flow across cut: $f_4 + f_5 - f_6 = ?$
    - $f_2 + f_4 + f_5 - f_1 = 0$
    - $f_3 - f_2 - f_6 = 0$
    - $f_1 - f_3 = |f|$
    - $(\cancel{f_2} + f_4 - \cancel{f_1} + f_5) + (\cancel{f_3} - \cancel{f_2} - f_6) + (\cancel{f_1} - \cancel{f_3}) = |f|$
    - $f_4 + f_5 - f_6 = |f|$



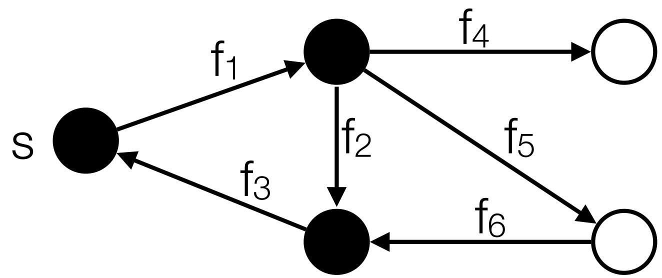
- Net flow across cut is |f| for all cuts => net flow out of s = net flow into t.

s-t Cuts

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

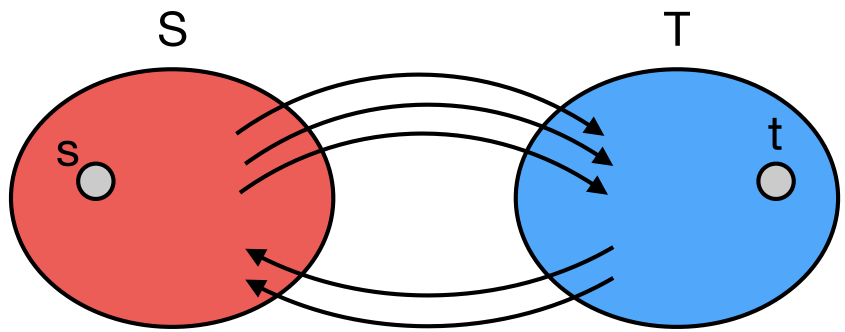


- Capacity of cut: total capacity of edges going from S to T.
- Flow across cut = flow from S to T minus flow from T to S.

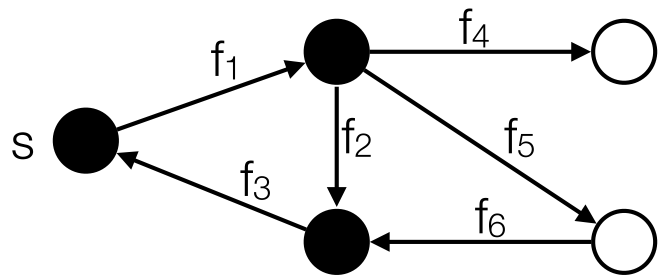


s-t Cuts

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

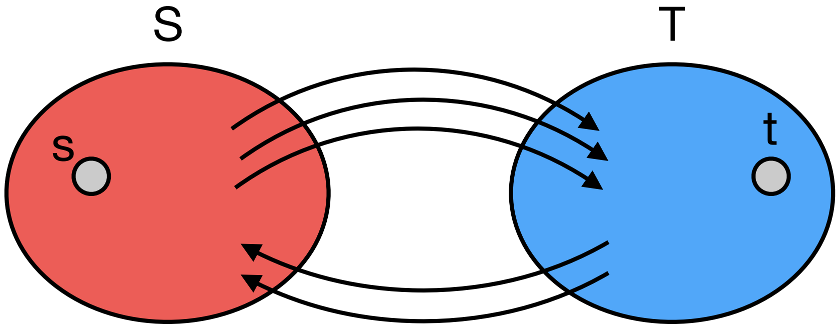


- Capacity of cut: total capacity of edges going from S to T.
- Flow across cut = flow from S to T minus flow from T to S.
- Net flow across cut is |f| for all cuts => net flow out of s = net flow into t.

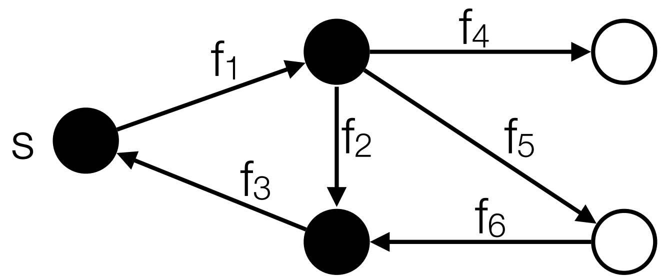


s-t Cuts

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

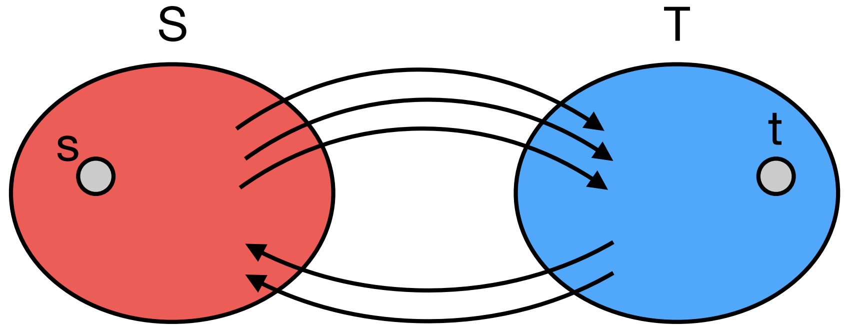


- Capacity of cut: total capacity of edges going from S to T.
- Flow across cut = flow from S to T minus flow from T to S.
- Net flow across cut is |f| for all cuts => net flow out of s = net flow into t.
- |f| ≤ c(S,T):

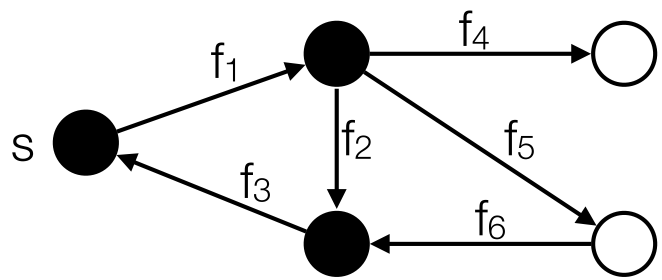


s-t Cuts

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

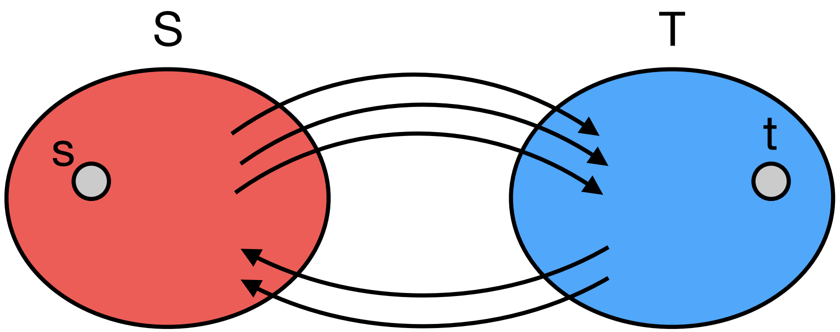


- Capacity of cut: total capacity of edges going from S to T.
- Flow across cut = flow from S to T minus flow from T to S.
- Net flow across cut is |f| for all cuts => net flow out of s = net flow into t.
- $|f| \leq c(S,T)$:
  - $|f| = f_4 + f_5 - f_6 \leq f_4 + f_5 \leq c_4 + c_5 = c(S,T)$



s-t Cuts

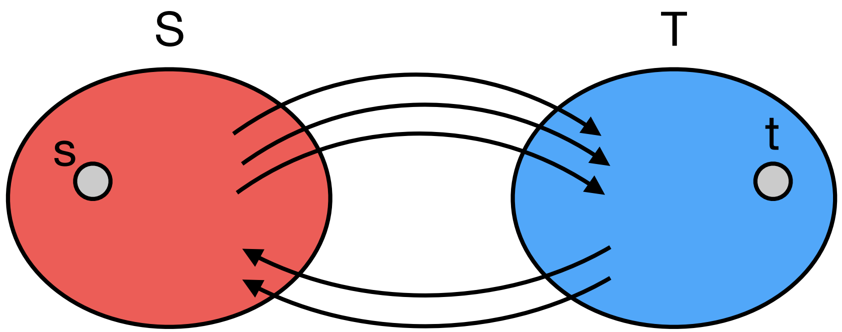
- Cut: Partition of vertices into S and T, such that s ∈ S and t ∈ T.



- Capacity of cut: total capacity of edges going from S to T.
- Flow across cut = flow from S to T minus flow from T to S.
- $|f| \leq c(S,T)$.

s-t Cuts

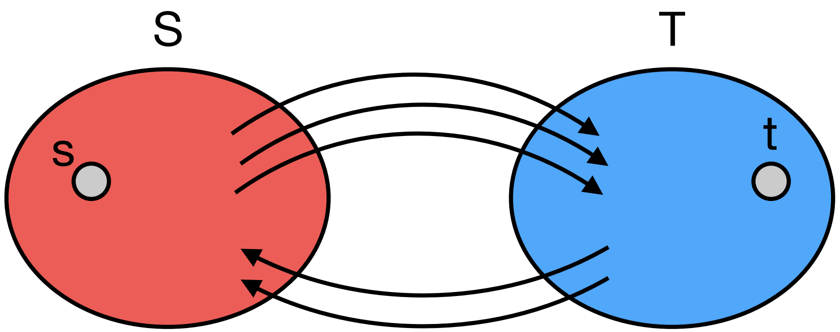
- Cut: Partition of vertices into S and T, such that s ∈ S and t ∈ T.



- Capacity of cut: total capacity of edges going from S to T.
- Flow across cut = flow from S to T minus flow from T to S.
- $|f| \leq c(S,T)$.
- 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

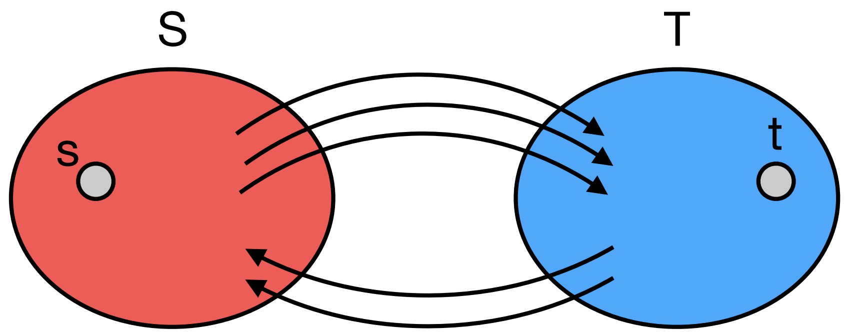
- Cut: Partition of vertices into S and T, such that s ∈ S and t ∈ T.



- Capacity of cut: total capacity of edges going from S to T.
- Flow across cut = flow from S to T minus flow from T to S.
- |f| ≤ c(S,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

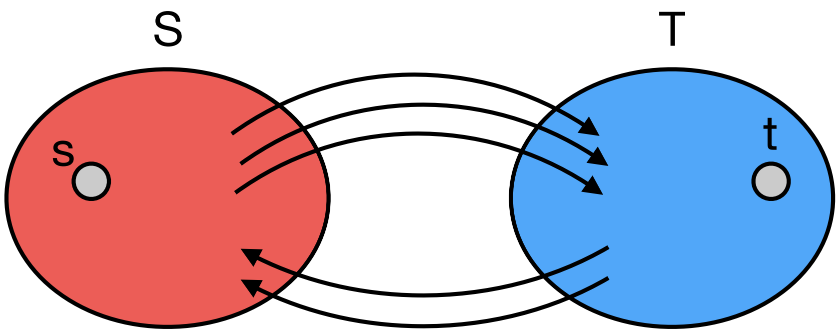
- Cut: Partition of vertices into S and T, such that s ∈ S and t ∈ T.



- Capacity of cut: total capacity of edges going from S to T.
- Flow across cut = flow from S to T minus flow from T to S.
- $|f| \leq c(S,T)$.
- 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

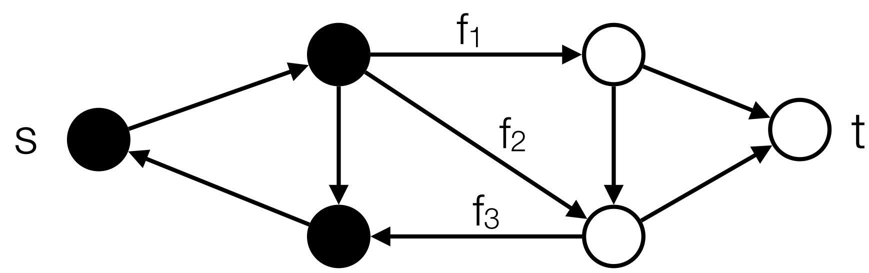
- Cut: Partition of vertices into S and T, such that s ∈ S and t ∈ T.



- Capacity of cut: total capacity of edges going from S to T.
- Flow across cut = flow from S to T minus flow from T to S.
- |f| ≤ c(S,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| ≤ |f*| ≤ c(S*,T*) ≤ c(S,T).
    - Since |f| = c(S,T) this implies |f| = |f*| and c(S,T) = c(S*,T*).

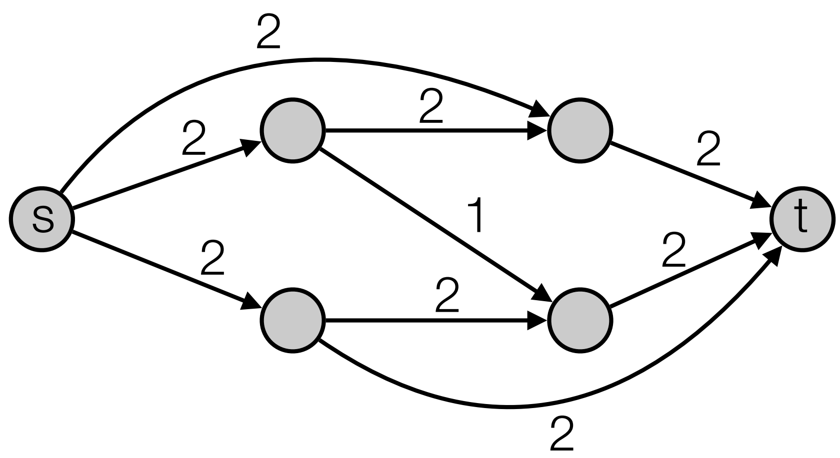
Max-flow min-cut theorem

- 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.



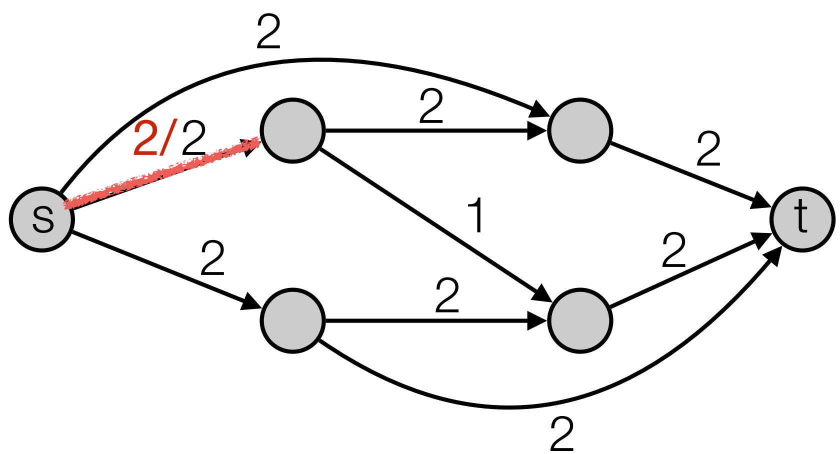
Finding minimum cuts

- 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.



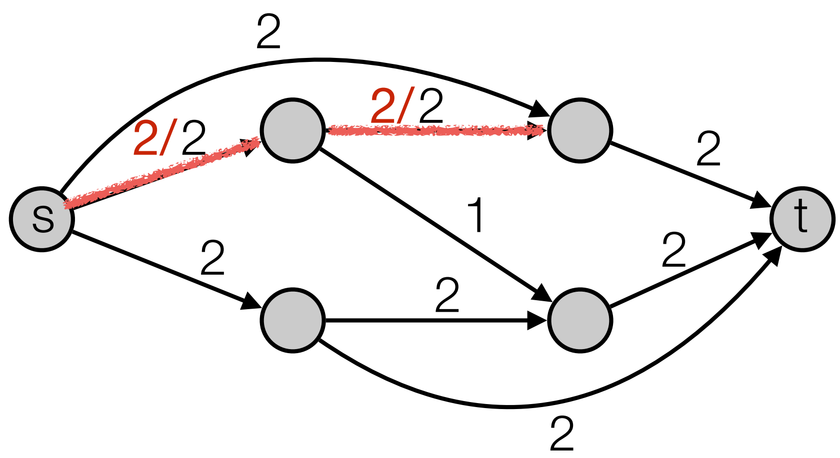
Finding minimum cuts

- 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.



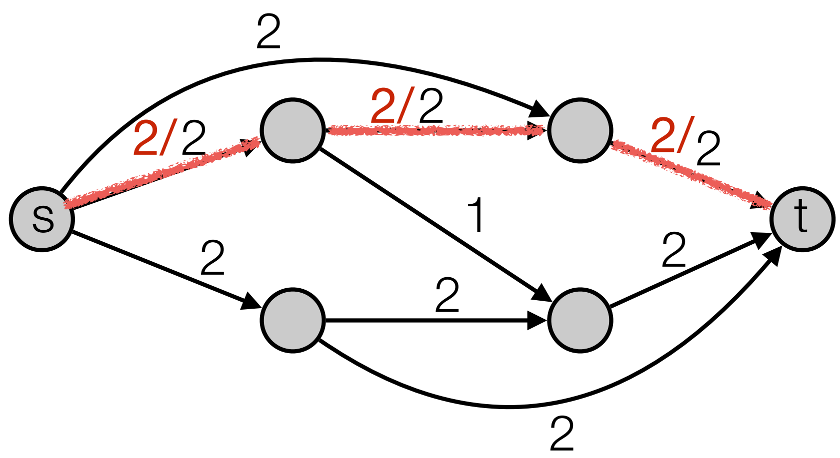
Finding minimum cuts

- 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.



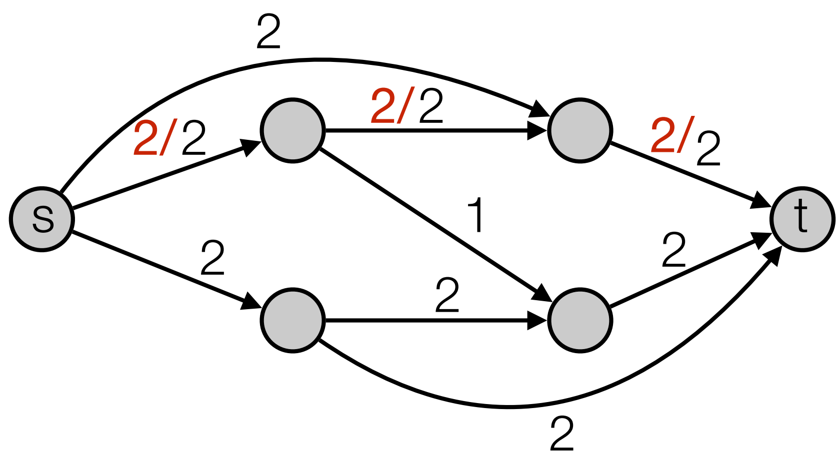
Finding minimum cuts

- 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.



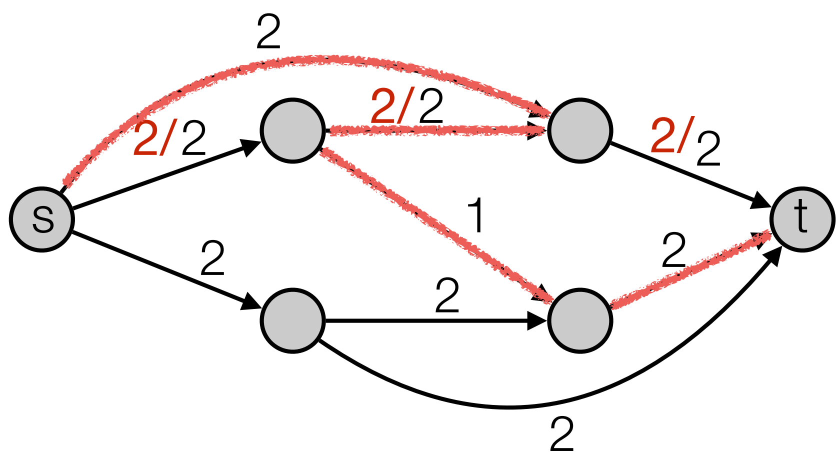
Finding minimum cuts

- 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.



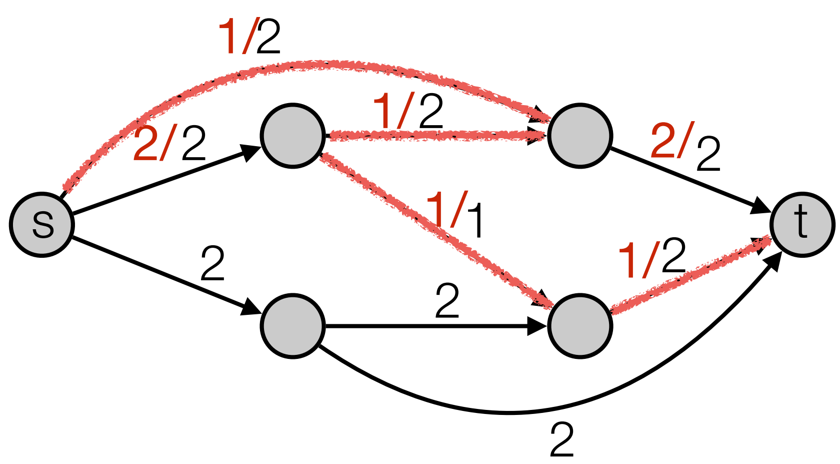
Finding minimum cuts

- 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.



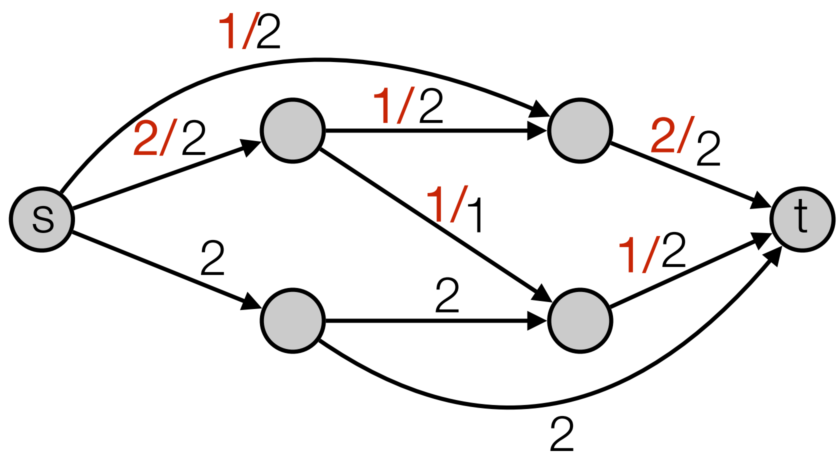
Finding minimum cuts

- 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.



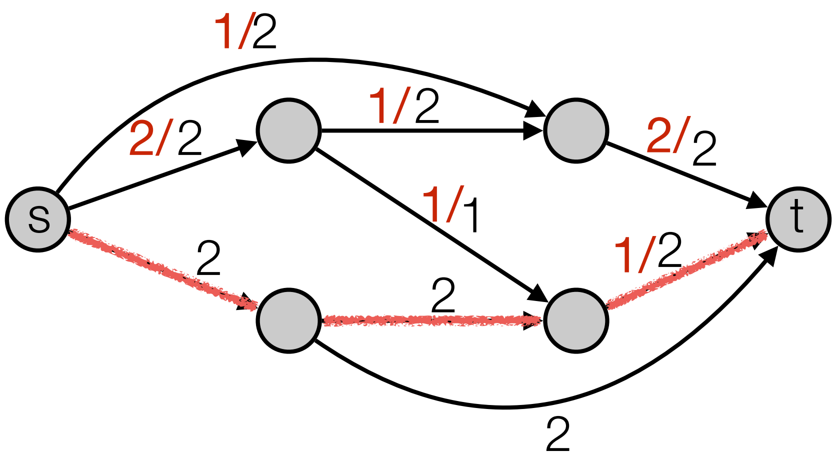
Finding minimum cuts

- 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.



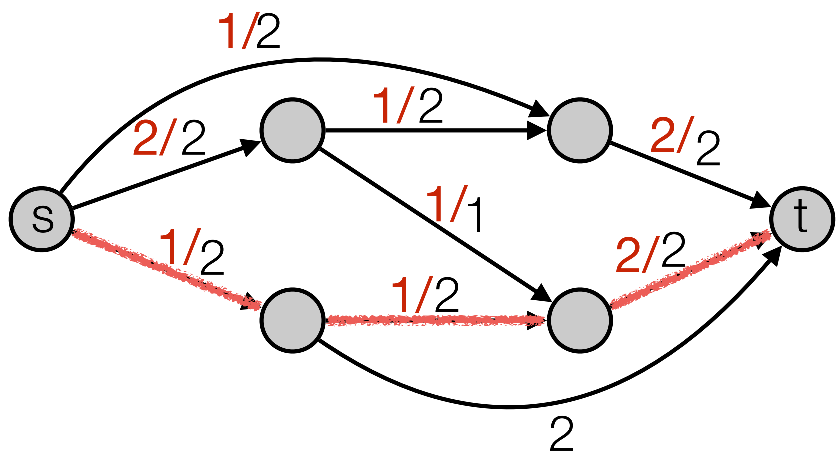
Finding minimum cuts

- 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.



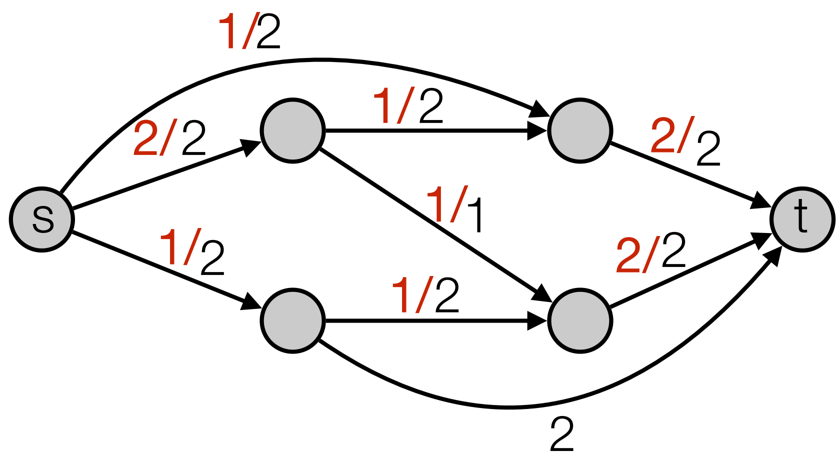
Finding minimum cuts

- 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.



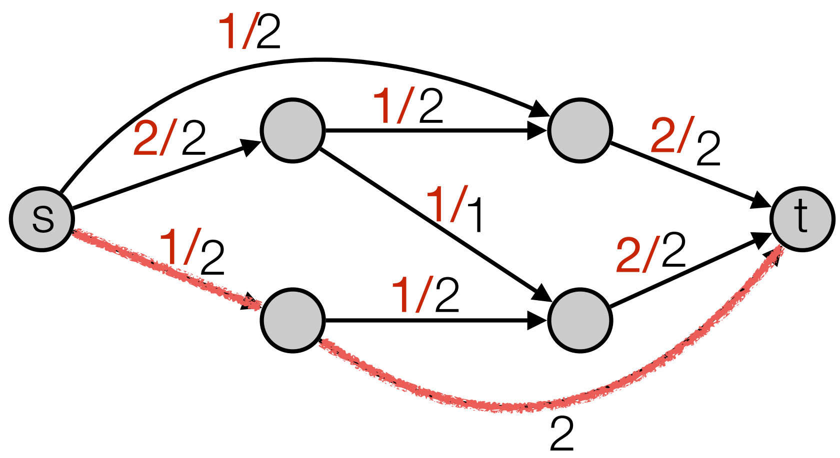
Finding minimum cuts

- 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.



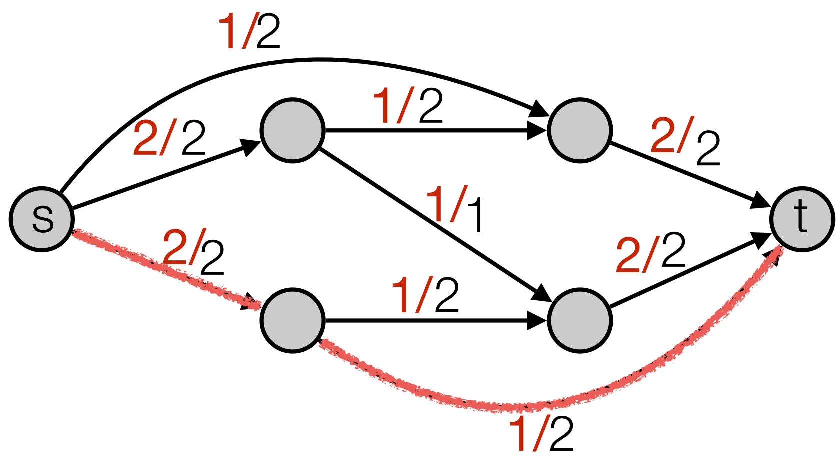
Finding minimum cuts

- 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.



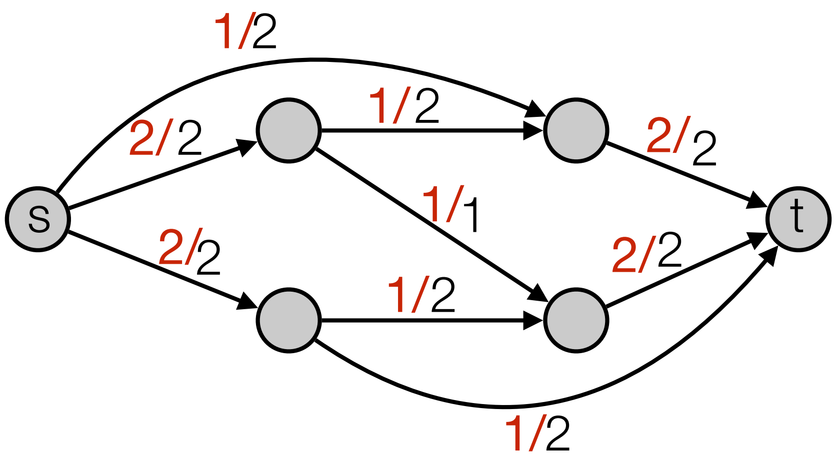
Finding minimum cuts

- 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.



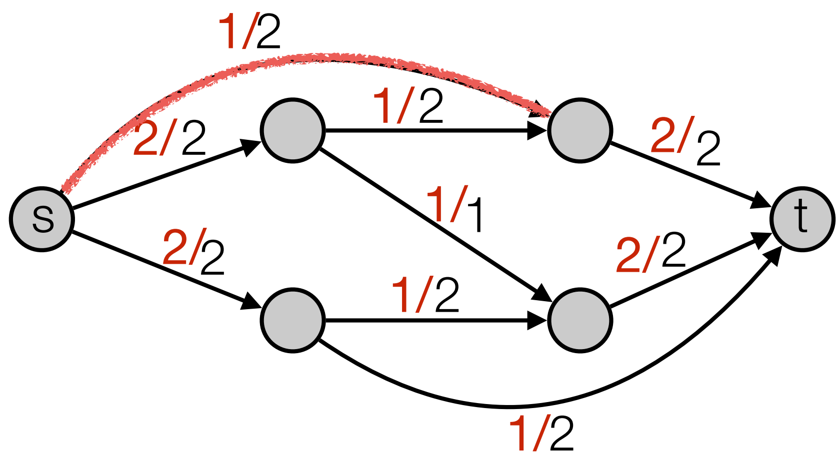
Finding minimum cuts

- 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.



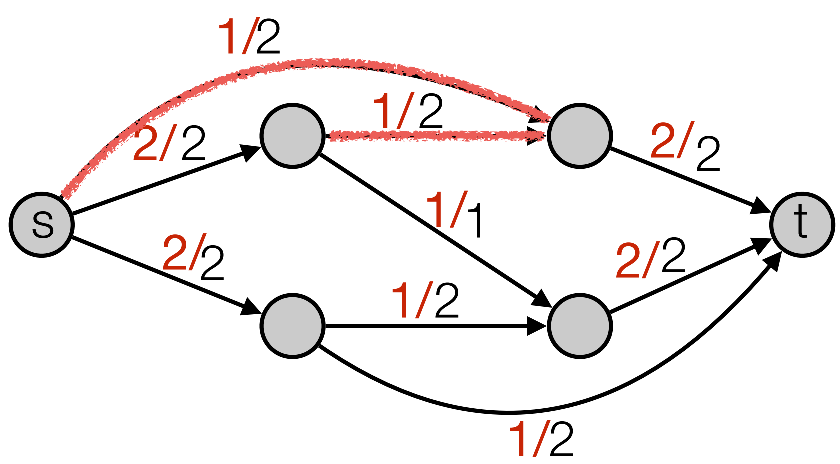
Finding minimum cuts

- 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.



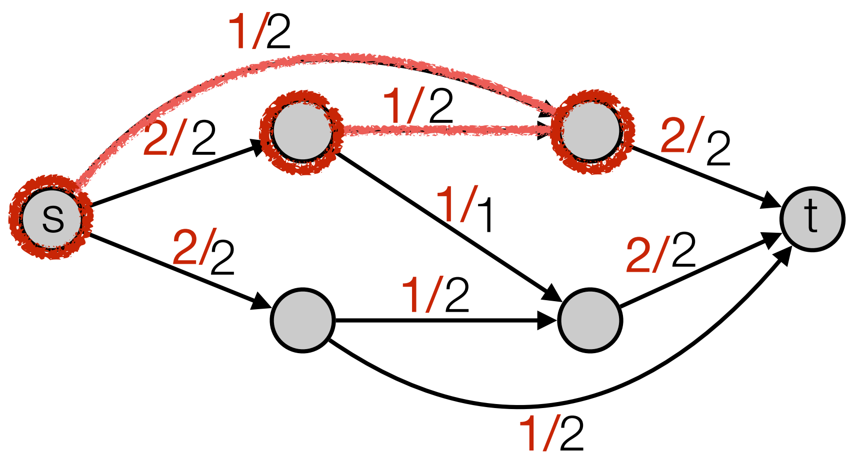
Finding minimum cuts

- 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.



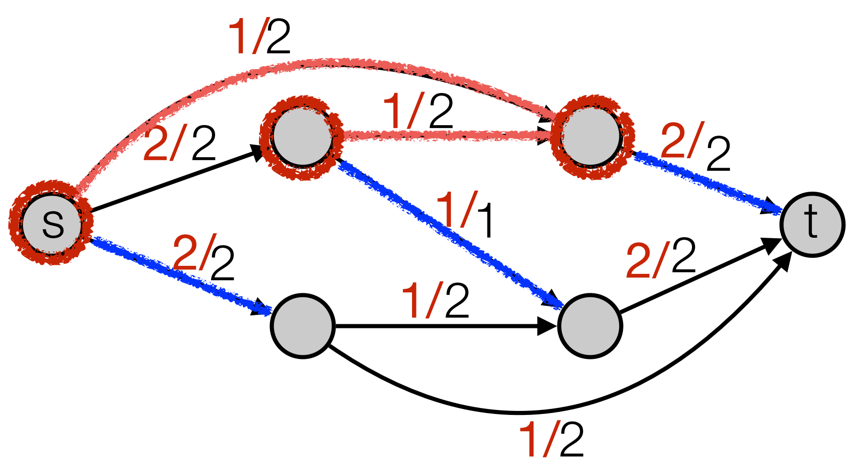
Finding minimum cuts

- 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.



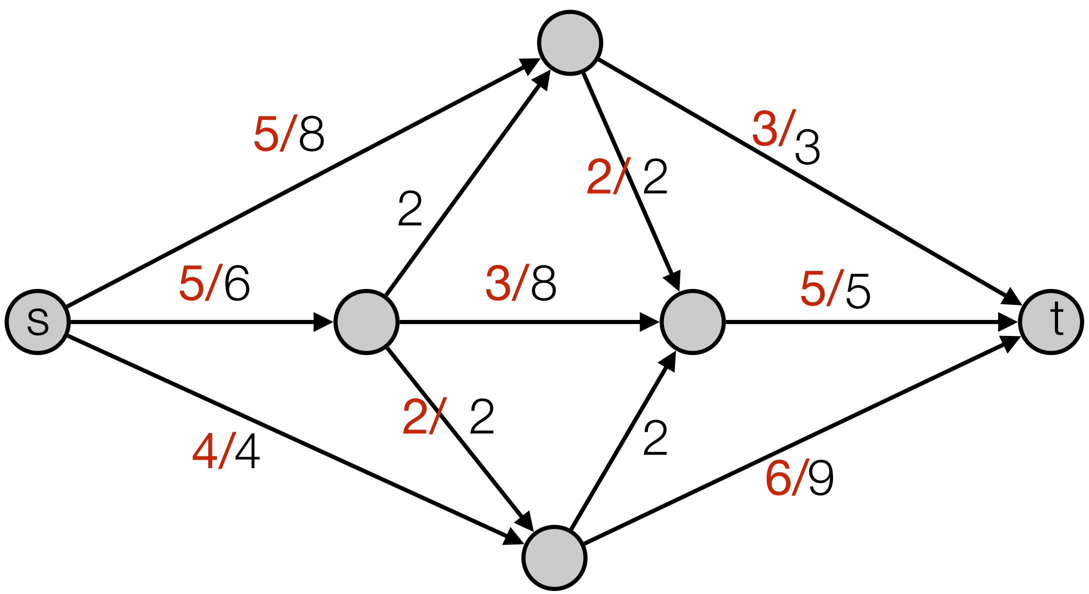
Finding minimum cuts

- 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.



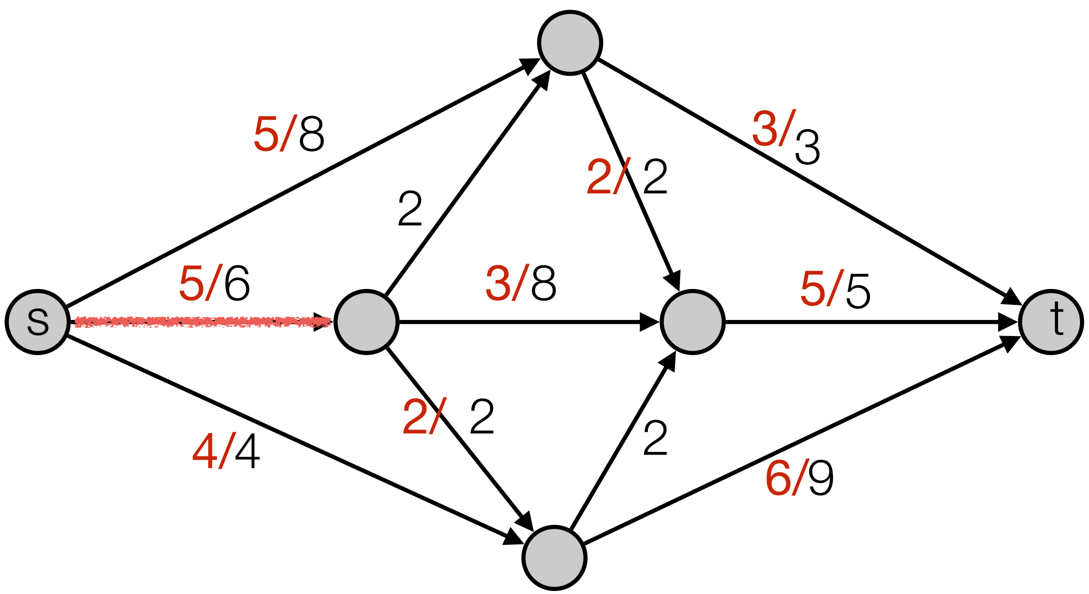
Finding minimum cuts

- 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.



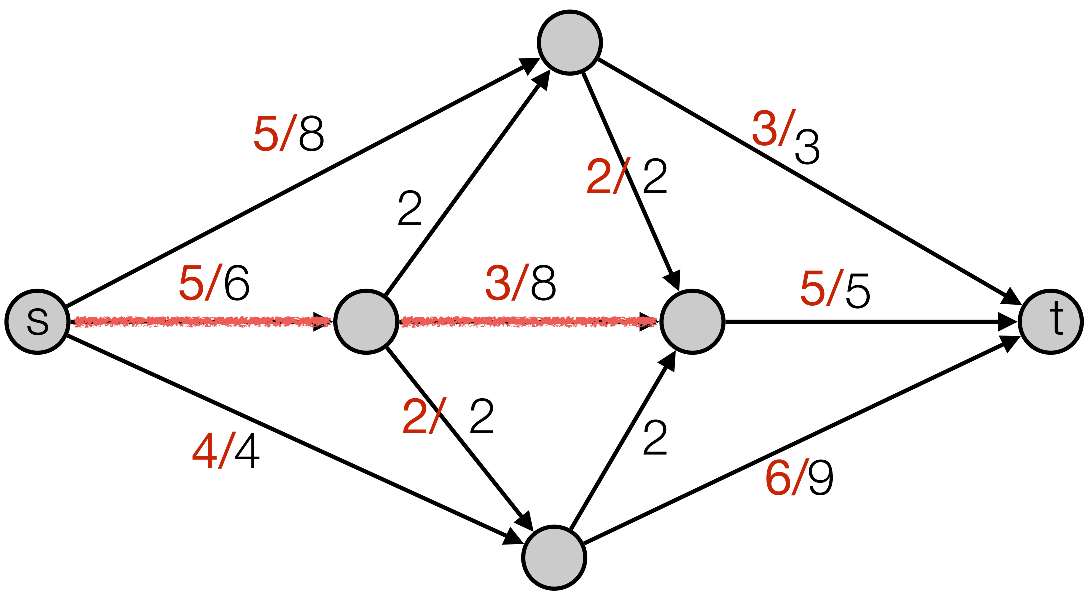
Finding minimum cuts

- 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.



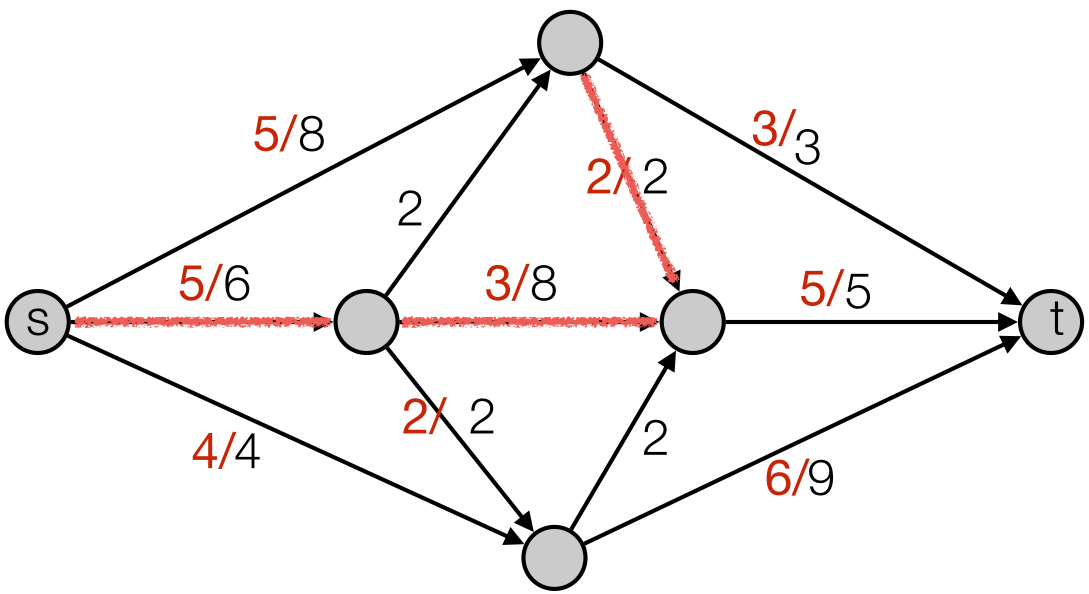
Finding minimum cuts

- 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.



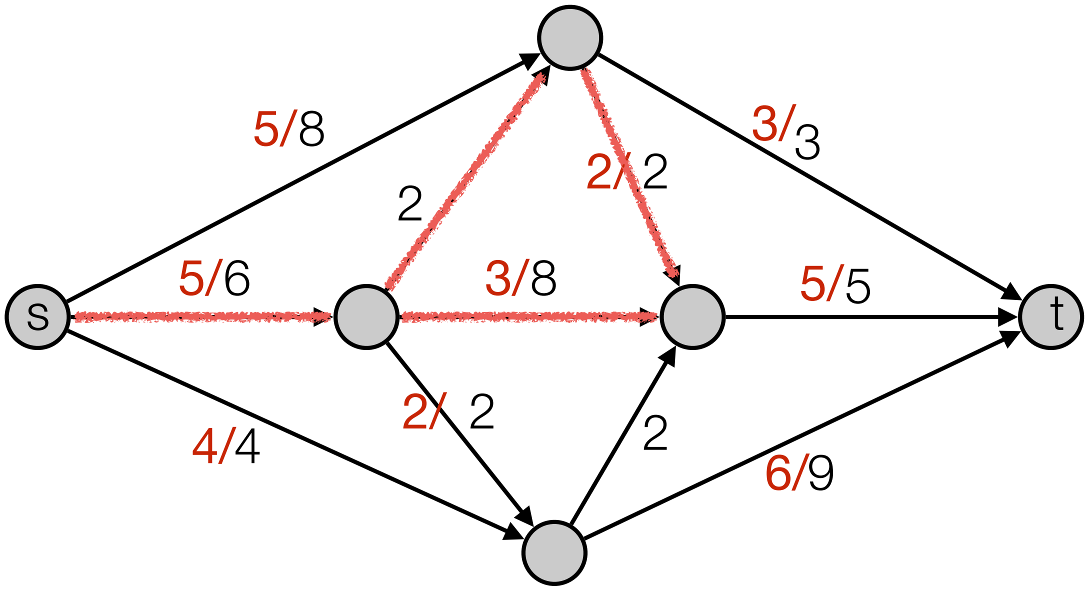
Finding minimum cuts

- 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.



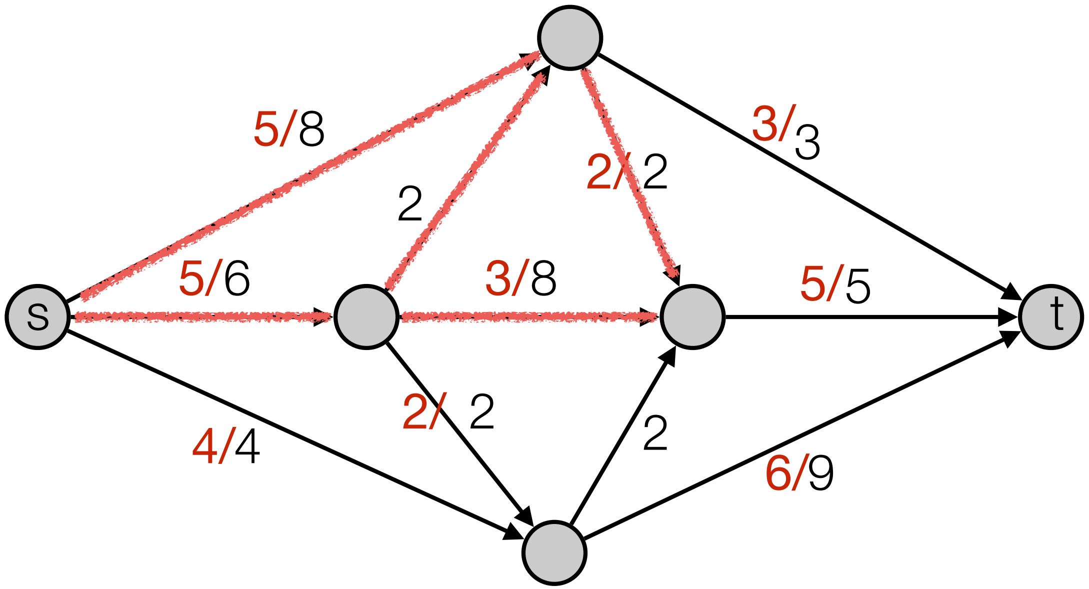
Finding minimum cuts

- 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.



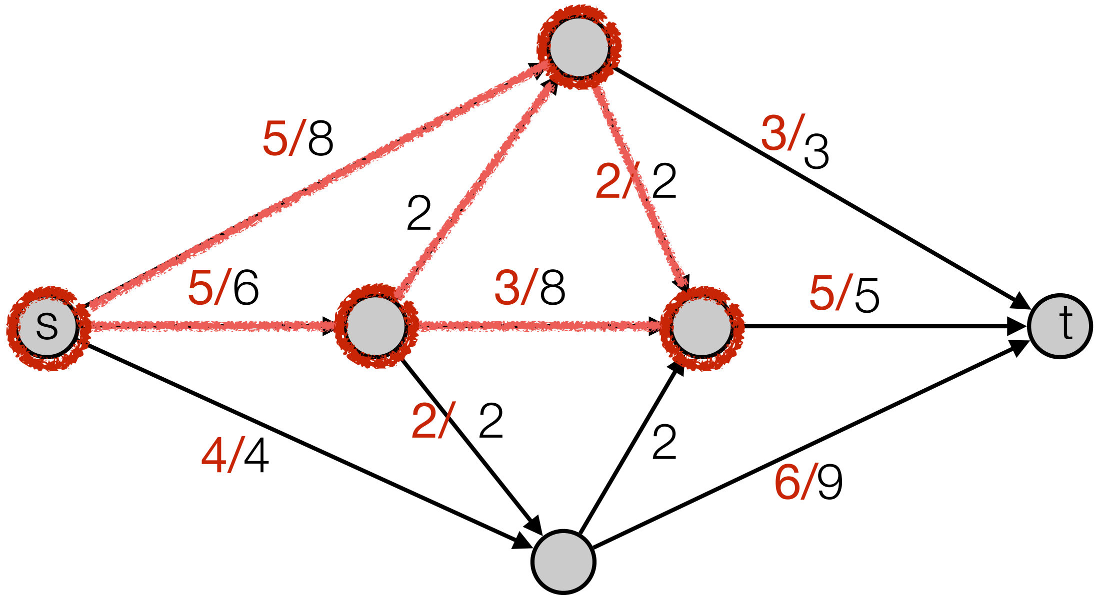
Finding minimum cuts

- 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.



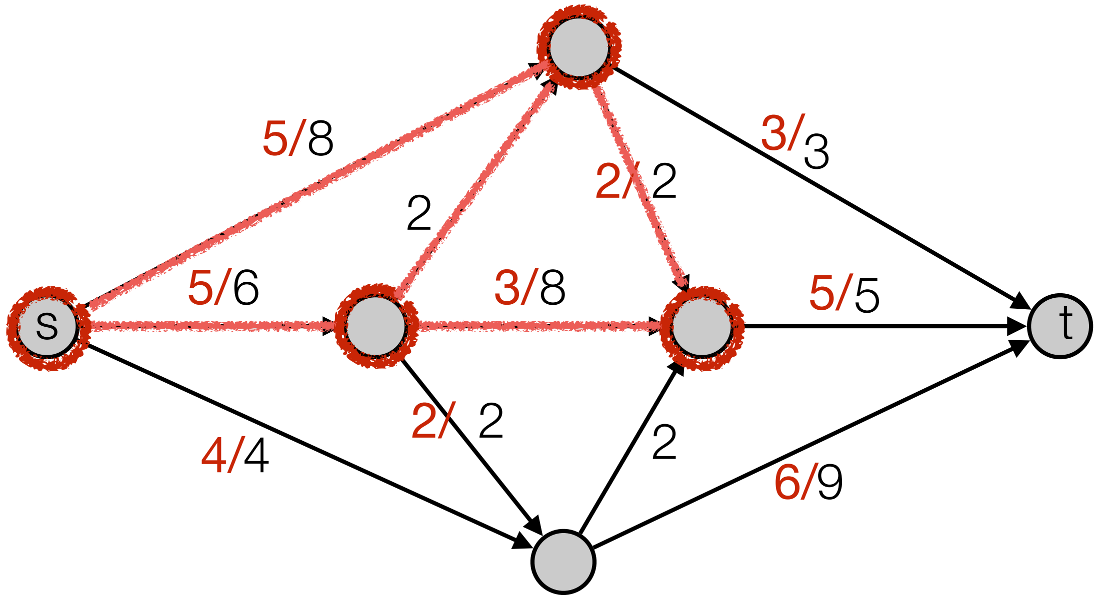
Finding minimum cuts

- 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.



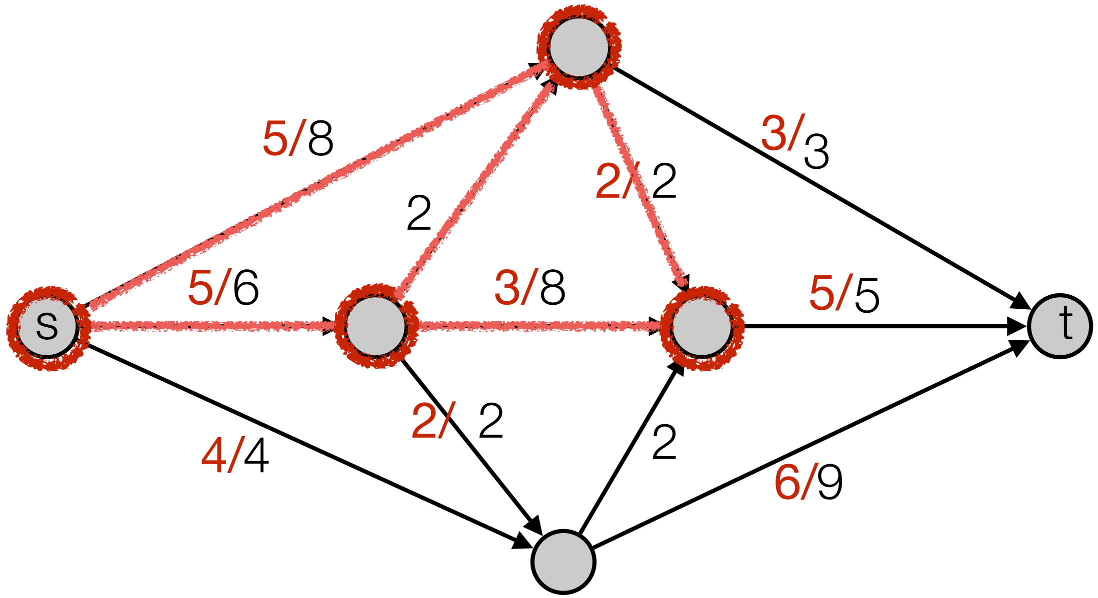
Finding minimum cuts

- 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.
- Remember:



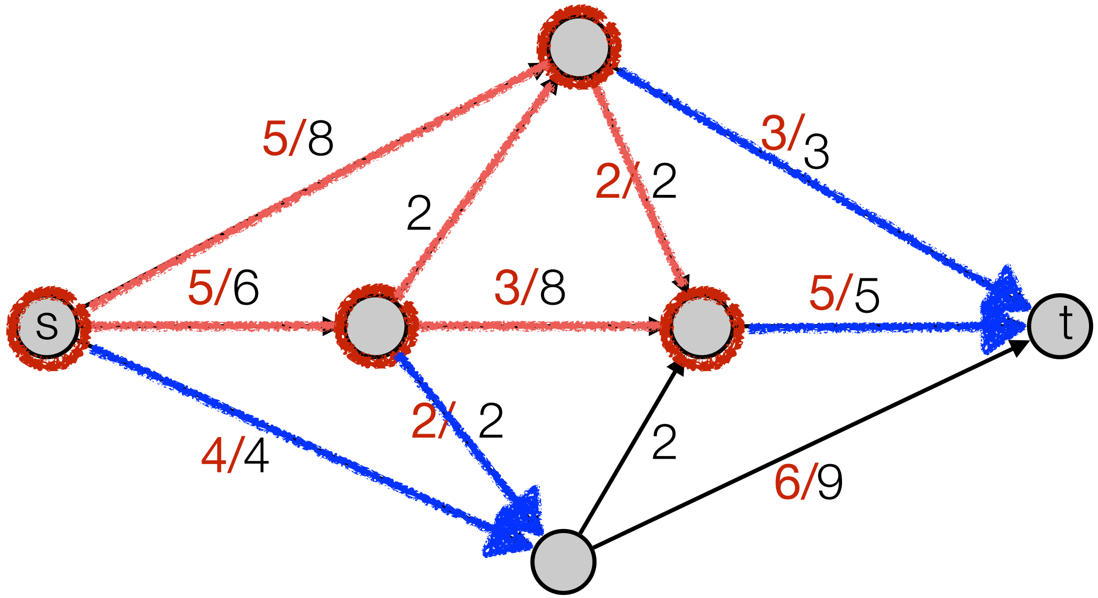
Finding minimum cuts

- 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.
- Remember:
  - All forward edges in the minimum cut are "full" (flow = capacity)



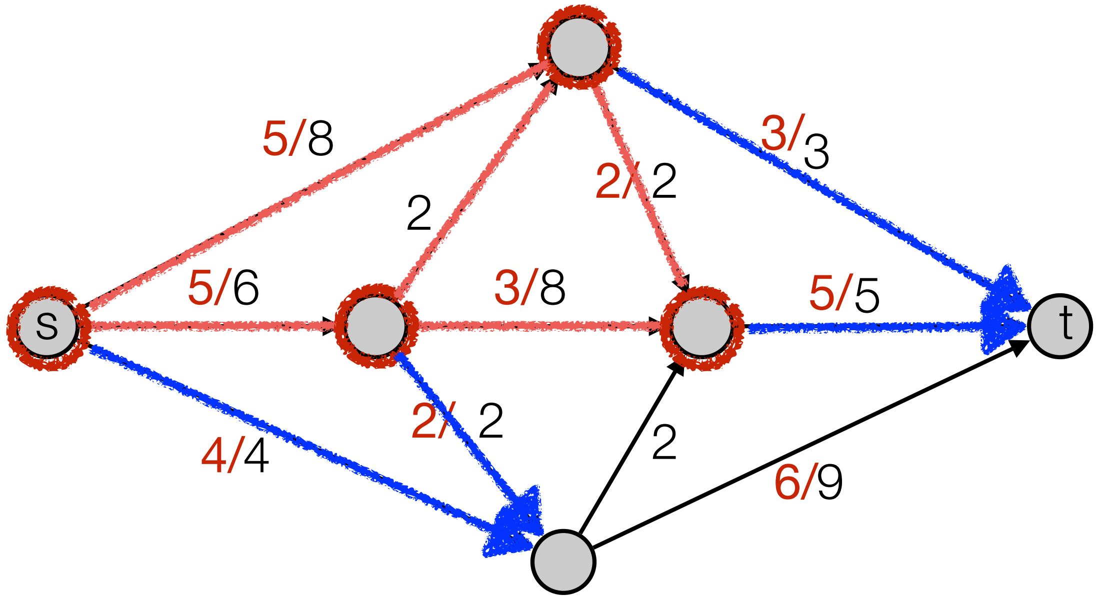
Finding minimum cuts

- 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.
- Remember:
  - All forward edges in the minimum cut are “full” (flow = capacity)



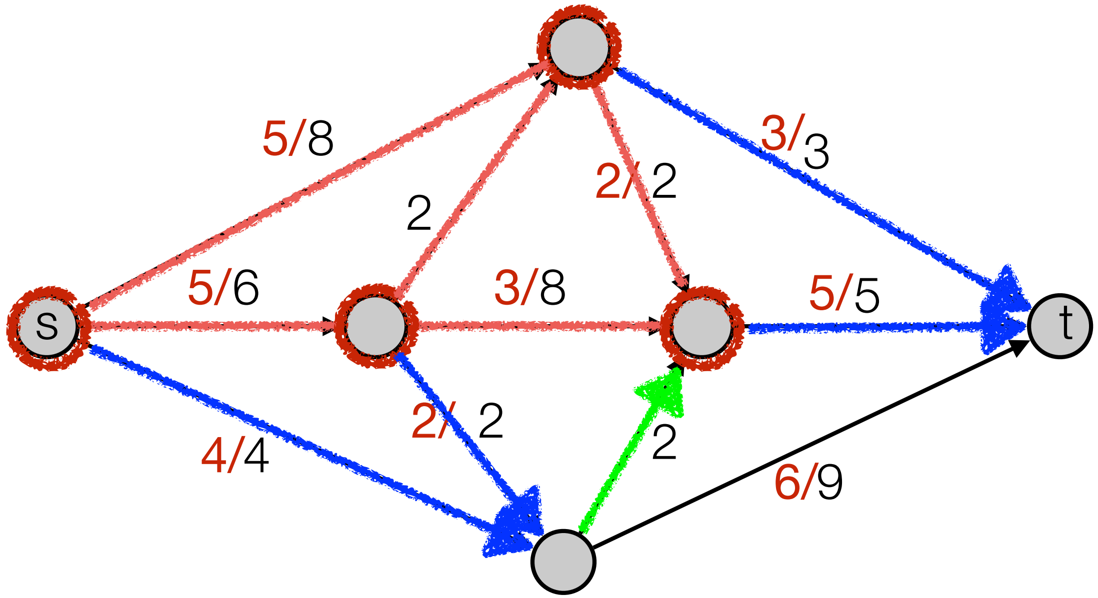
Finding minimum cuts

- 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.
- Remember:
  - All forward edges in the minimum cut are “full” (flow = capacity)
  - All backwards edges in minimum cut have 0 flow.

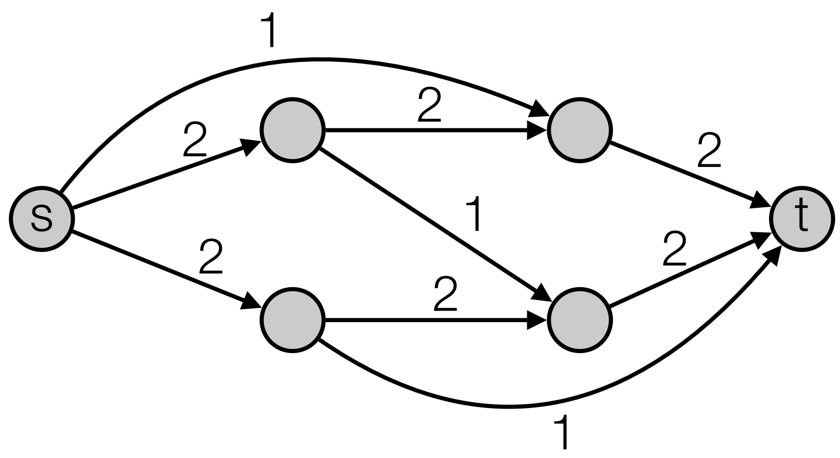


Finding minimum cuts

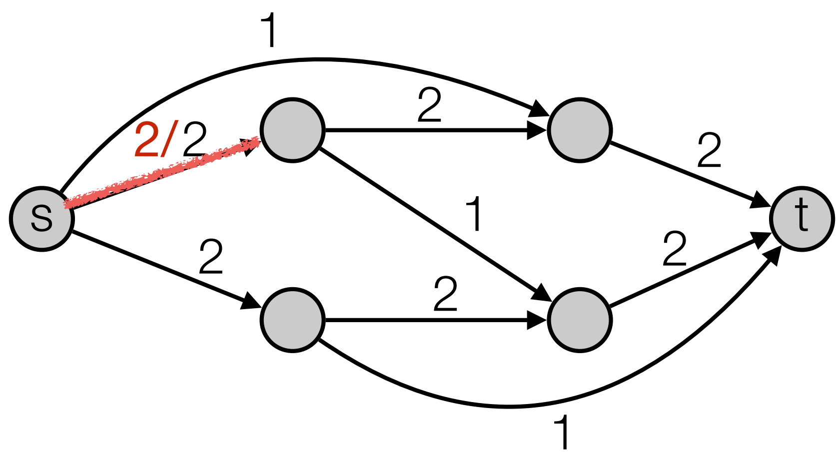
- Use Ford-Fulkerson to find a max-flow (finding augmenting paths).
- When no augmenting s-t path:
  - Let S be all vertices to which there exists an augmenting path from s.
- Remember:
  - All forward edges in the minimum cut are "full" (flow = capacity)
  - All backwards edges in minimum cut have 0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



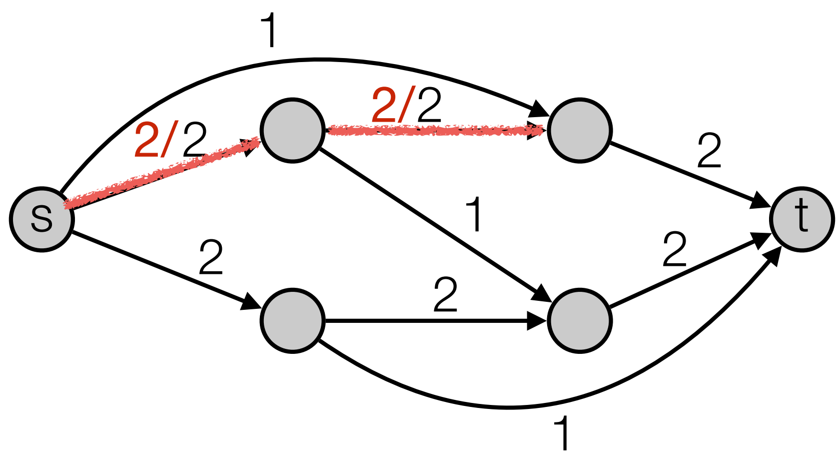
Residual networks



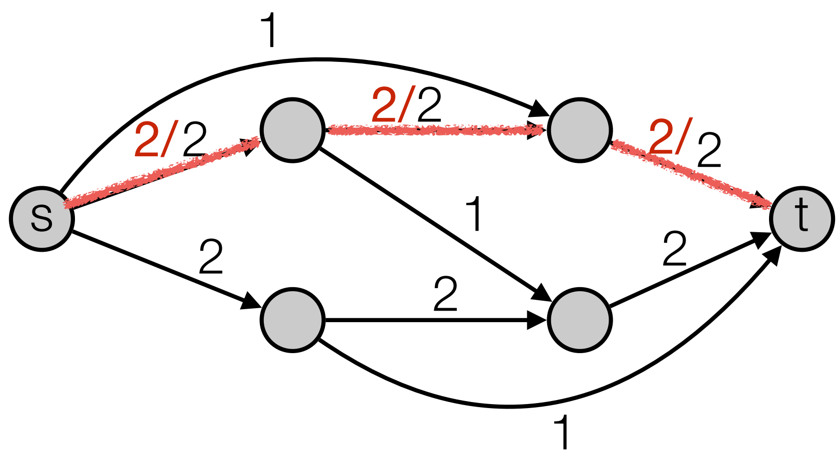
Residual networks



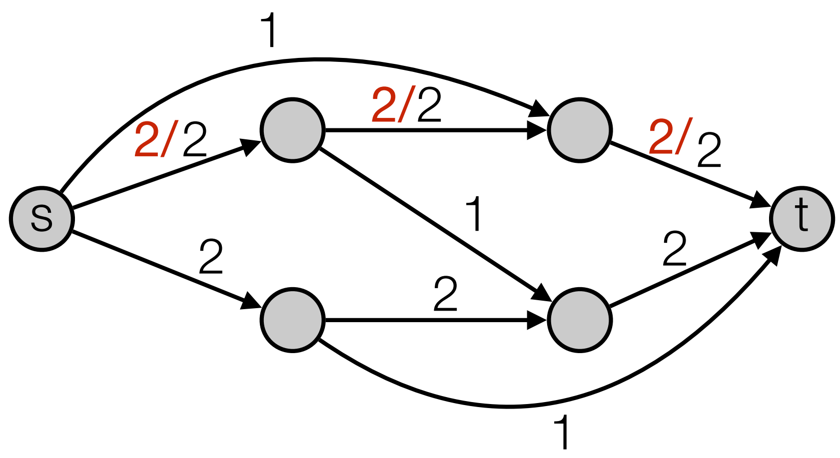
Residual networks



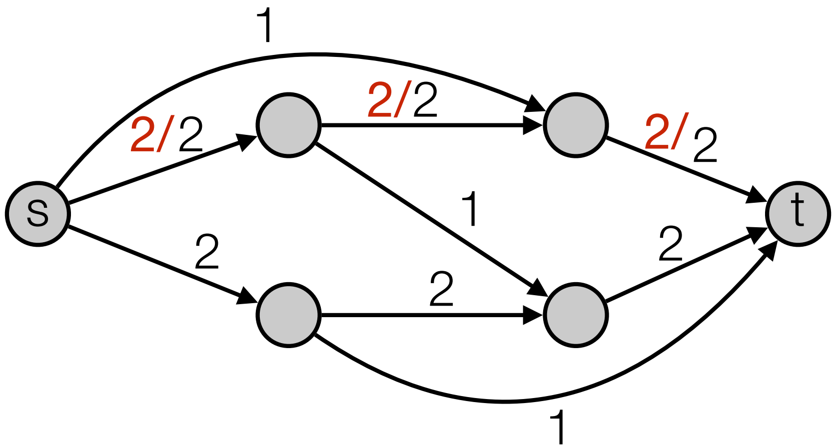
Residual networks



Residual networks

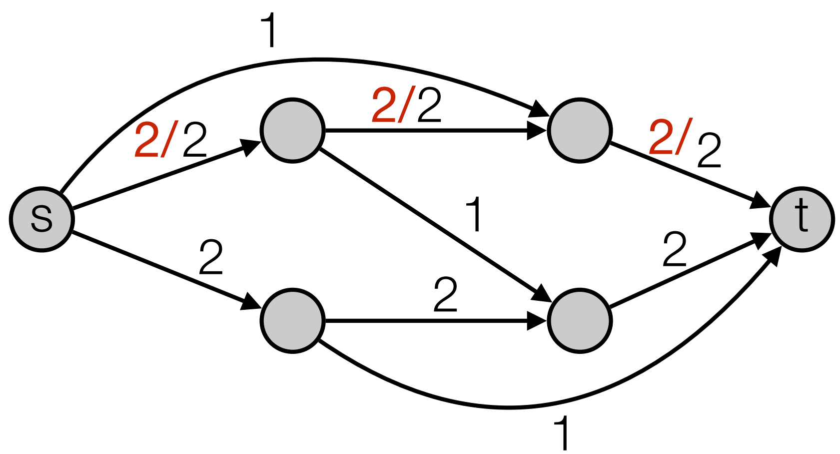


Residual networks

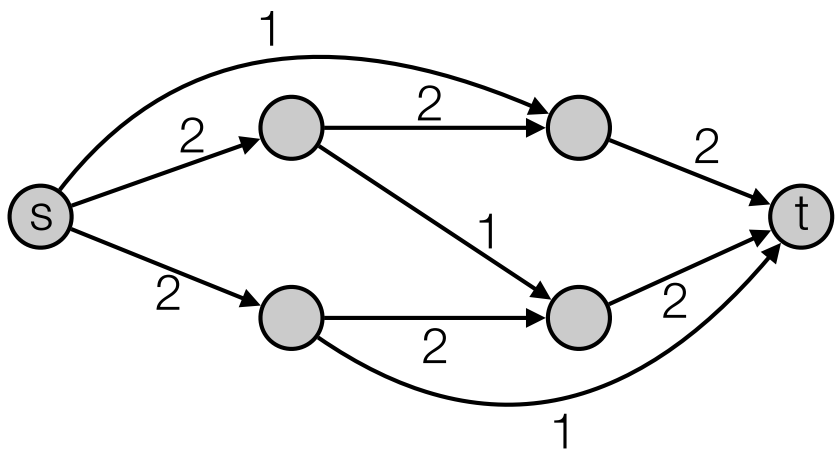


- Residual network

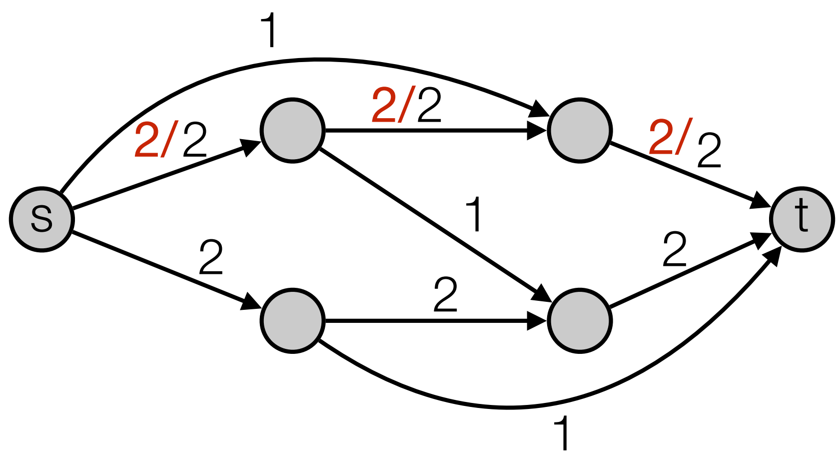
Residual networks



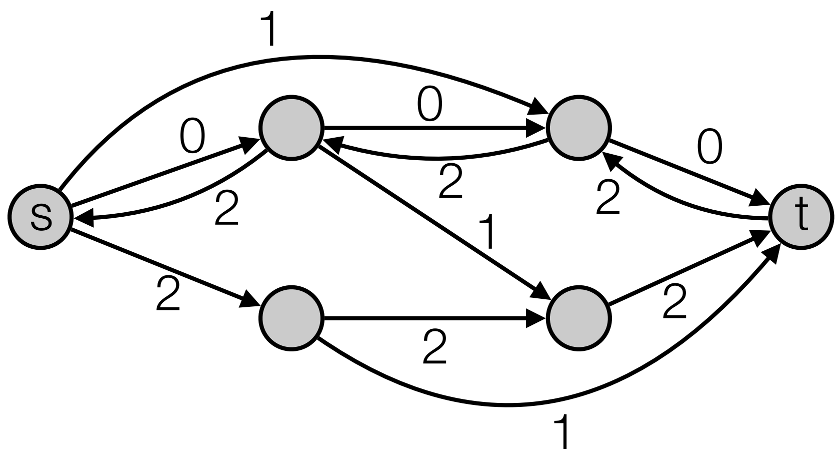
- Residual network



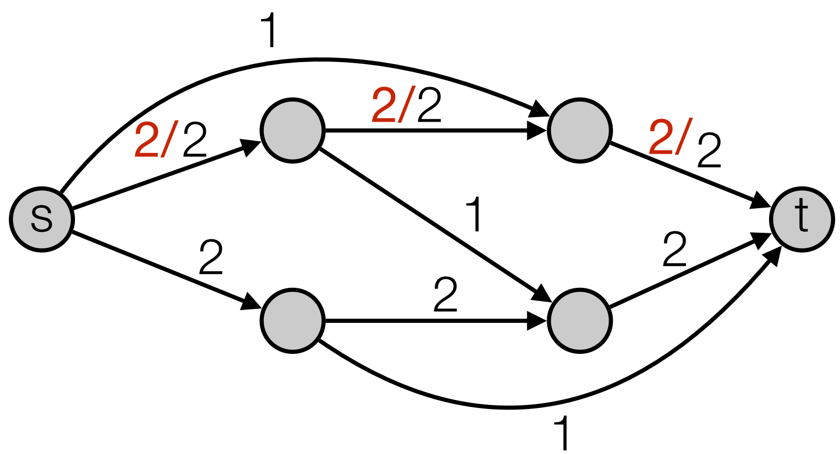
Residual networks



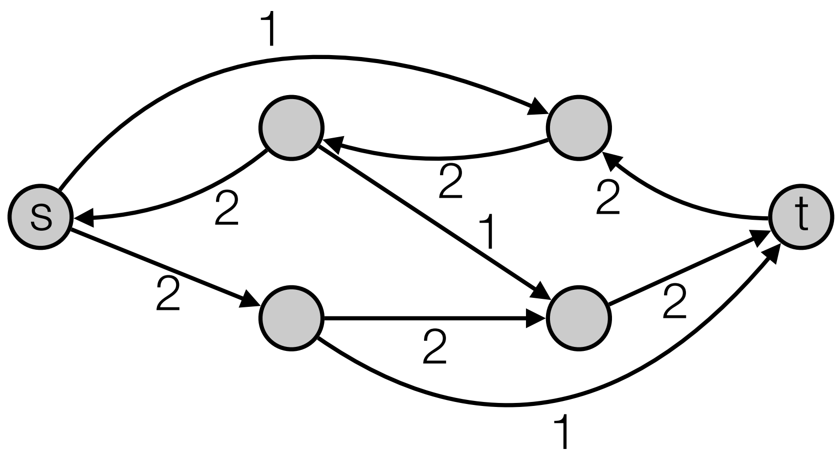
- Residual network



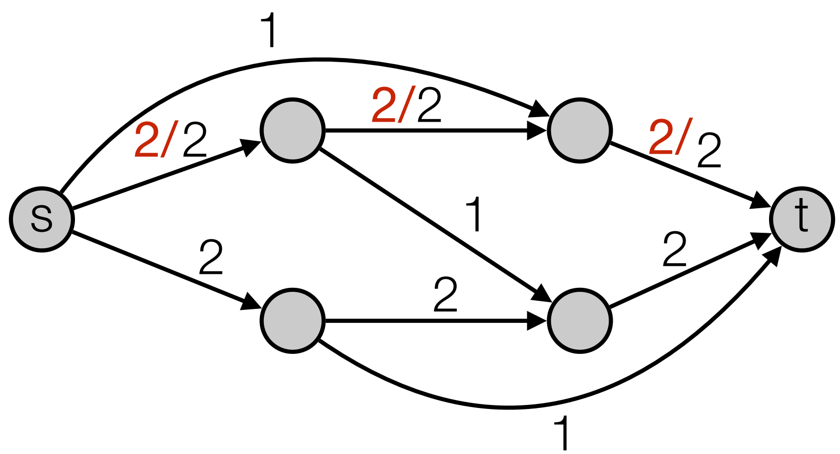
Residual networks



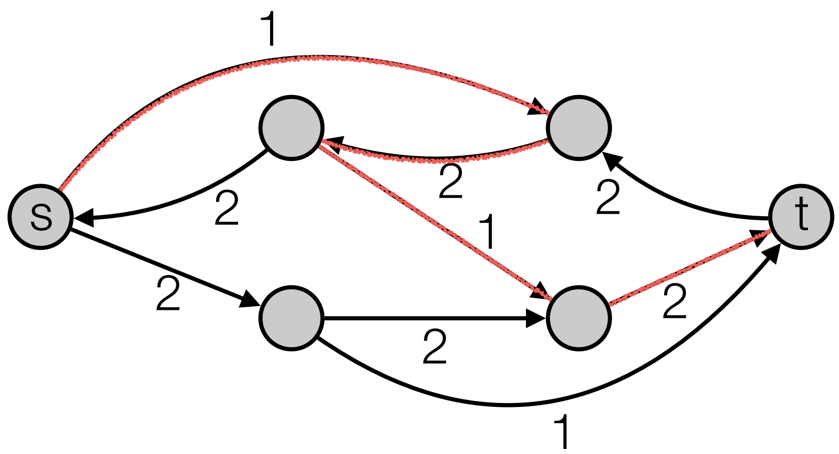
- Residual network



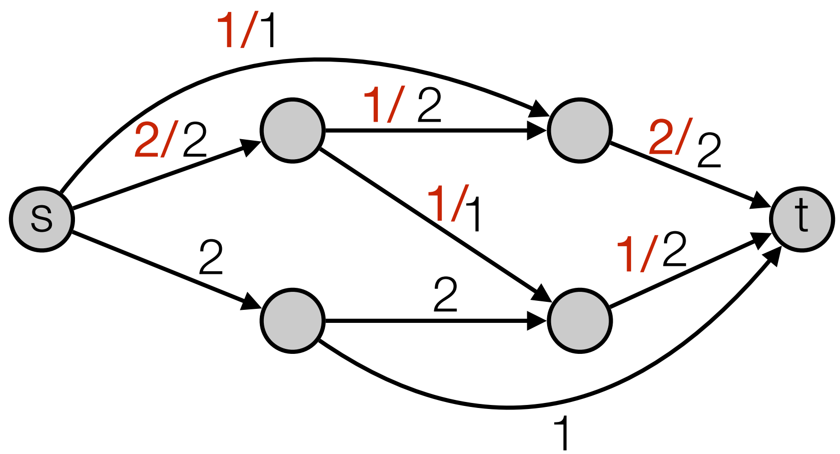
Residual networks



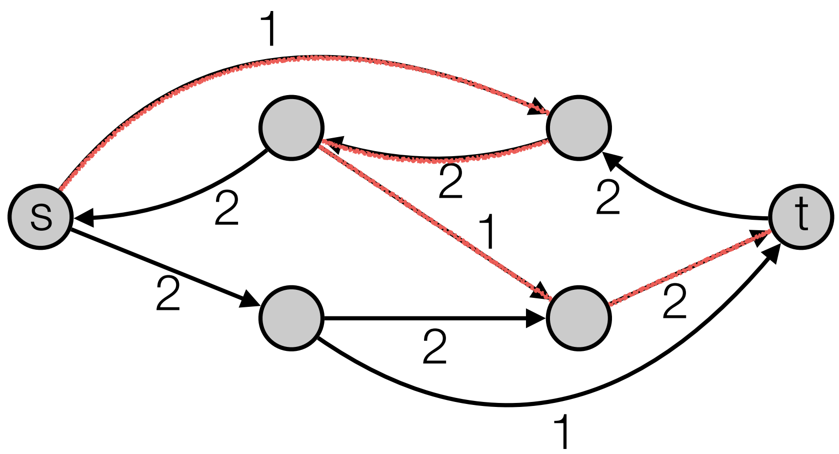
- Residual network



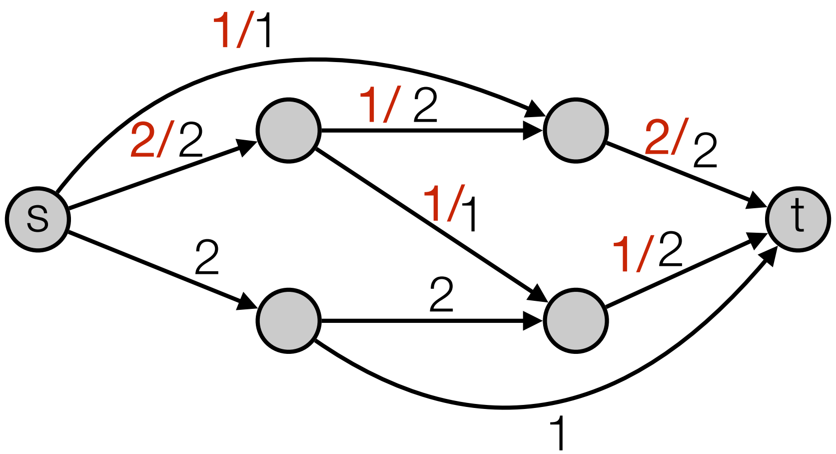
Residual networks



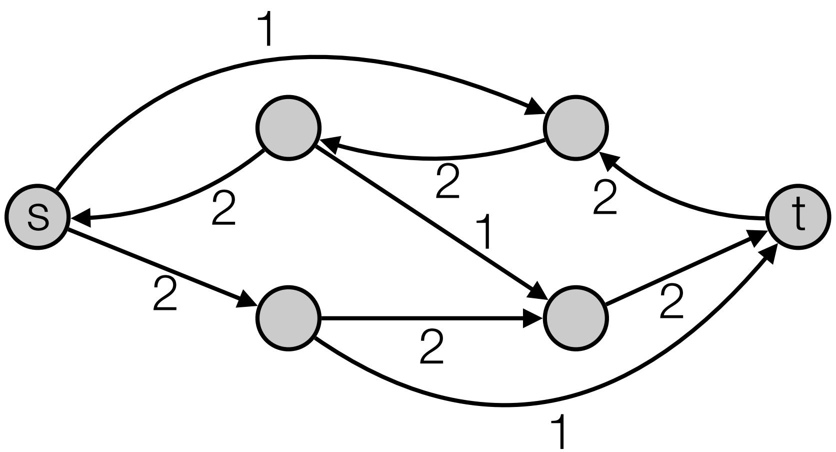
- Residual network



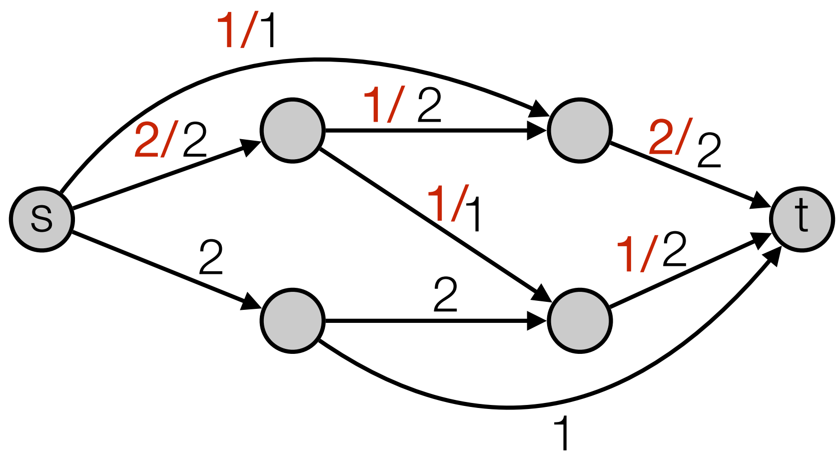
Residual networks



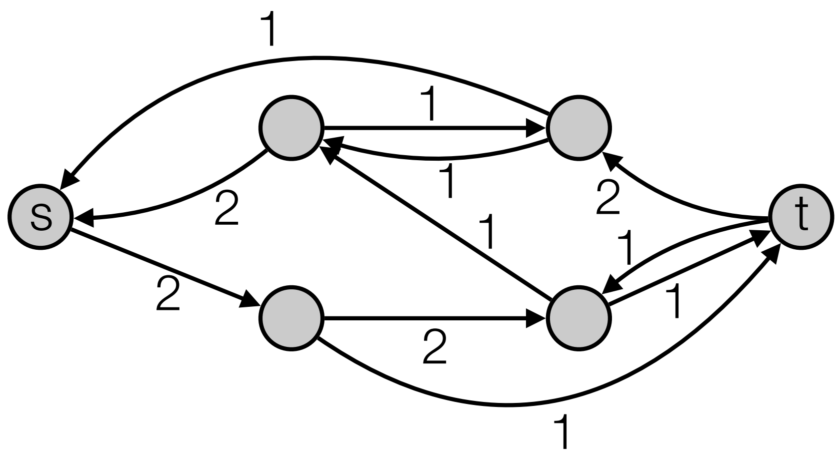
- Residual network



Residual networks



• Residual network



Network Flow

- Multiple sources and sinks: