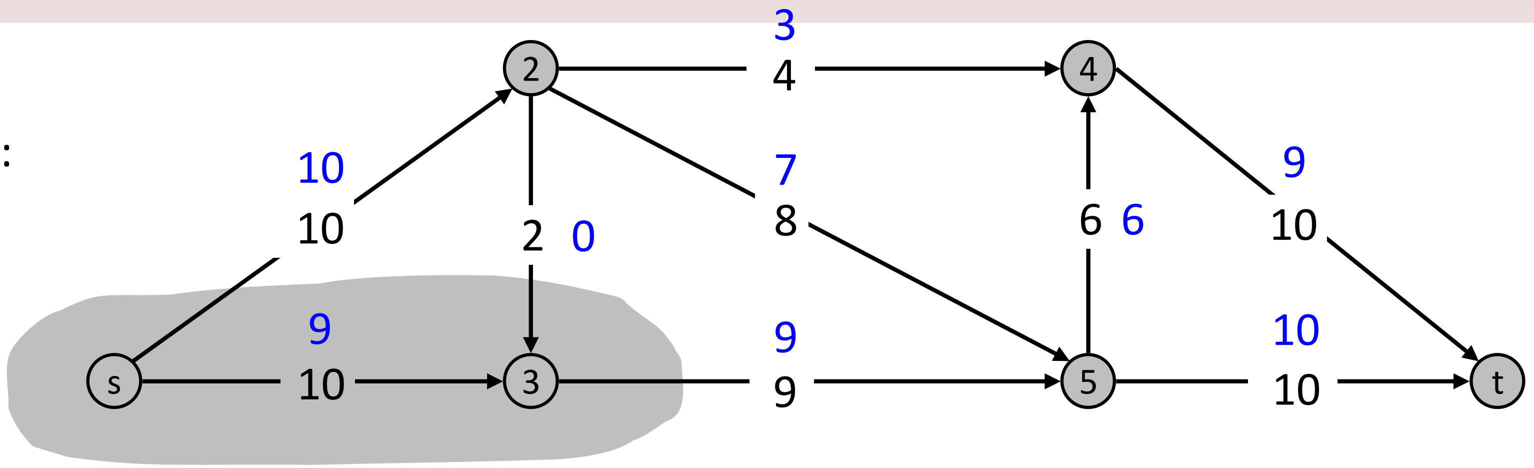
COMP 355
Advanced Algorithms

More on Network Flows
Section 7.1-7.3, 7.5-7.6 (KT)

Rhodes College
—1848—

Ford-Fulkerson Algorithm

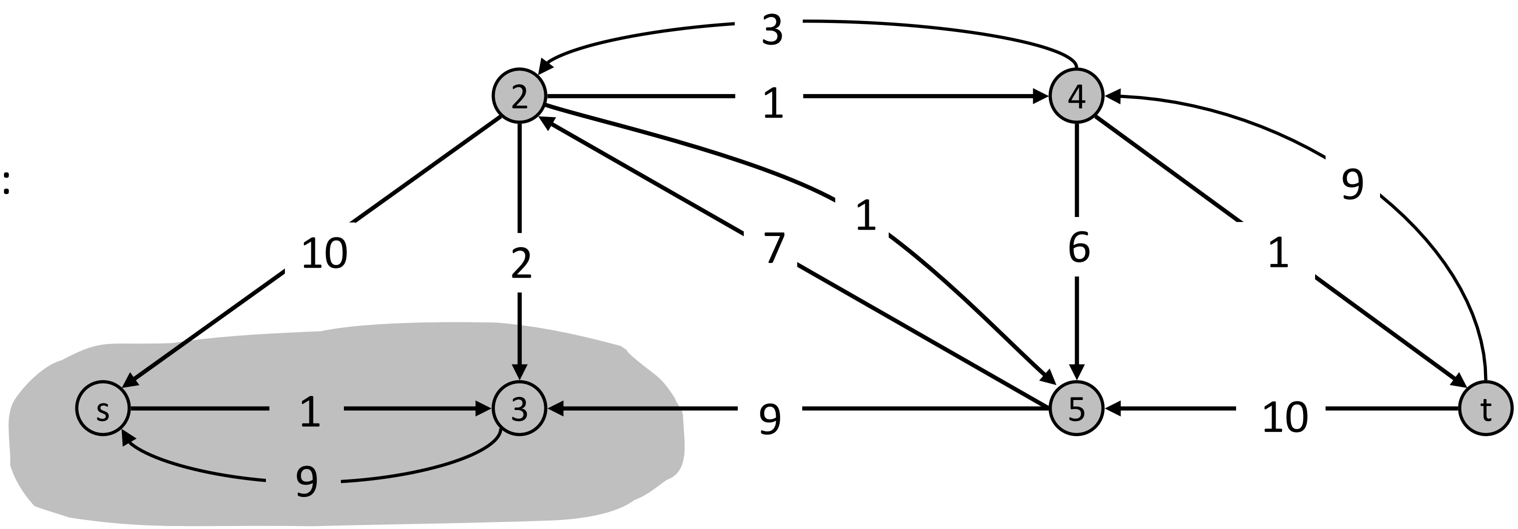
G:



Cut capacity = 19

Flow value = 19

G_f:



Augmenting Path Algorithm

```
Augment(f, c, P) {
   b ← bottleneck(P)
   foreach e ∈ P {
      if (e ∈ E) f(e) ← f(e) + b         forward edge
      else       f(eR) ← f(e) - b        reverse edge
   }
   return f
}
```

```
Ford-Fulkerson(G, s, t, c) {
   foreach e ∈ E  f(e) ← 0
   Gf ← residual graph

   while (there exists augmenting path P) {
      f ← Augment(f, c, P)
      update Gf
   }
   return f
}
```

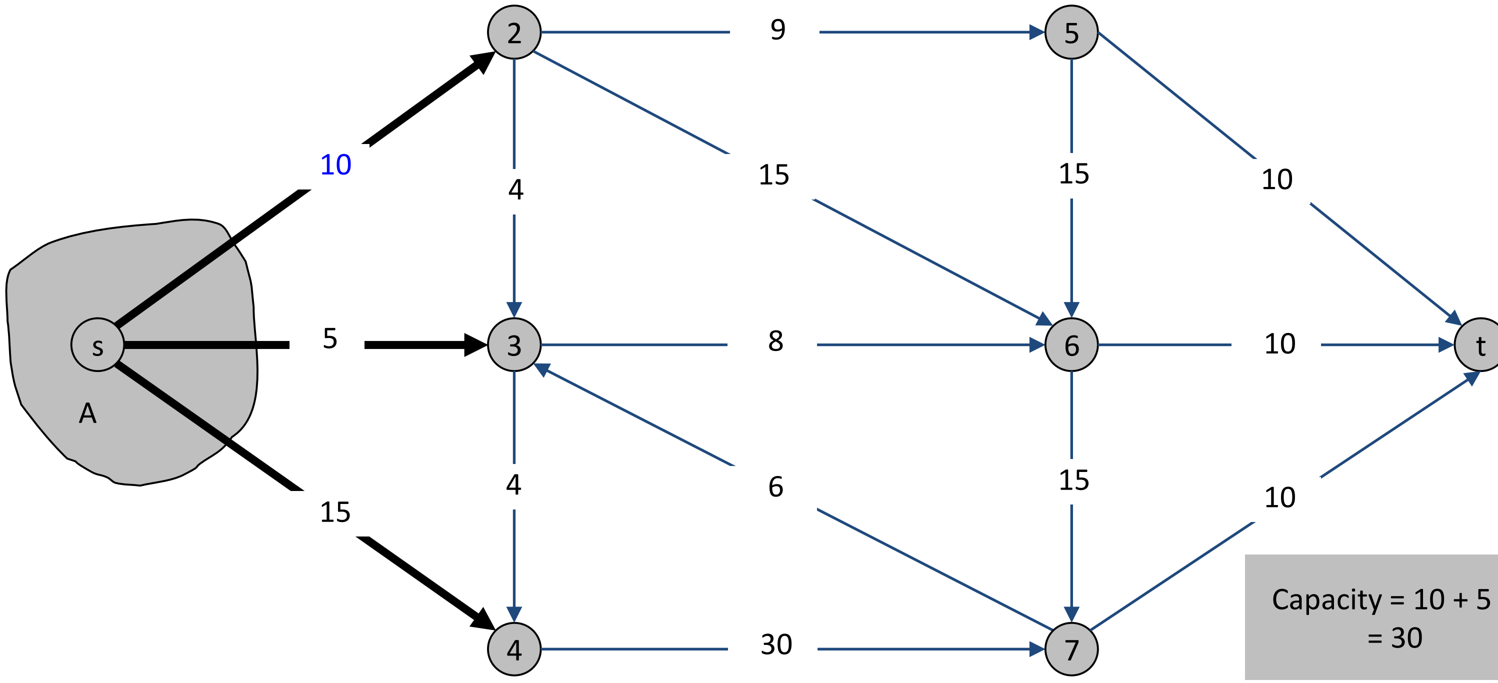
Remaining Issues

- How efficiently can we perform augmentation?
- How many augmentations might be required until converging?
- If no more augmentations can be performed, have we found the max-flow?

Cuts

Def. An *s-t cut* is a partition (A, B) of V with $s \in A$ and $t \in B$.

Def. The capacity of a cut (A, B) is: $cap(A, B) = \sum_{e \text{ out of } A} c(e)$

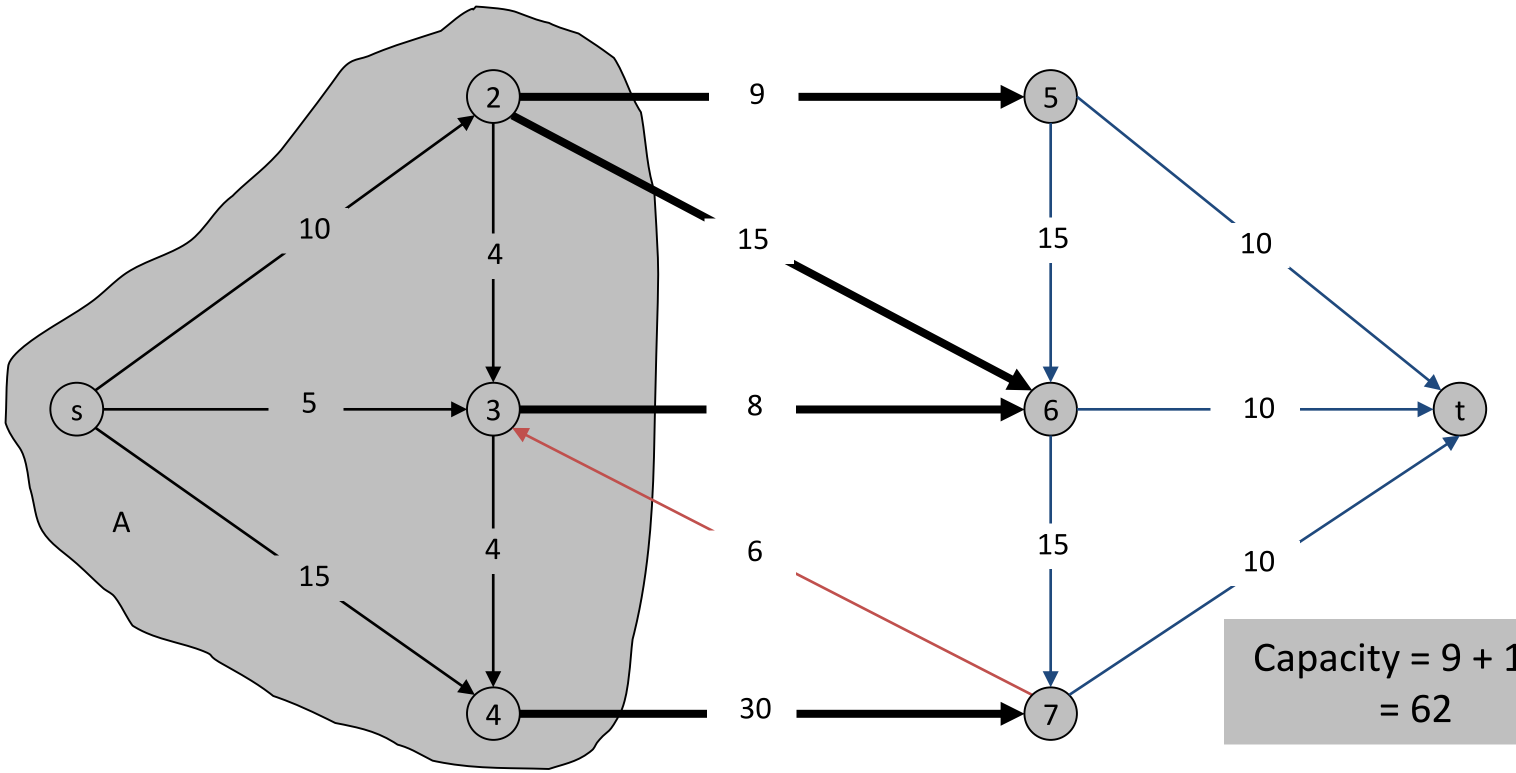


Capacity = 10 + 5 + 15
= 30

Cuts

Def. An *s-t cut* is a partition (A, B) of V with $s \in A$ and $t \in B$.

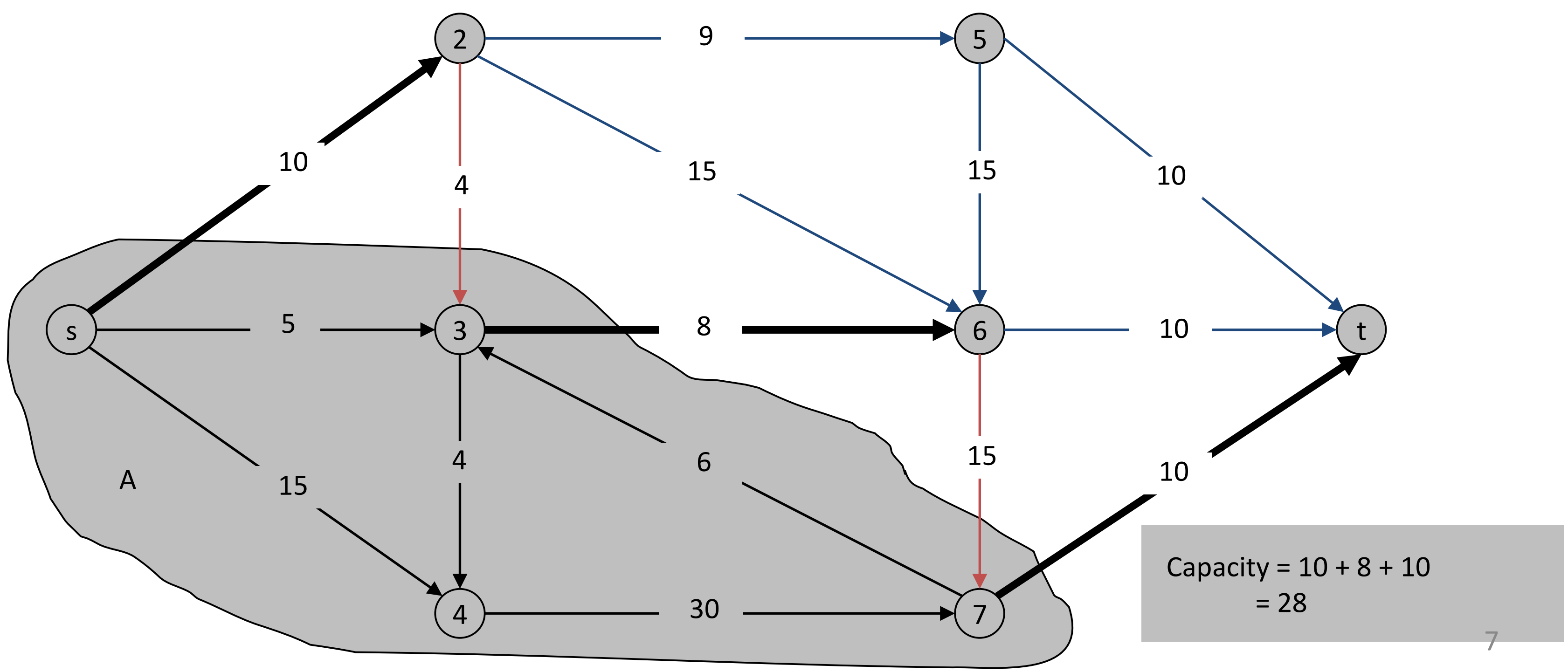
Def. The capacity of a cut (A, B) is: $cap(A, B) \; = \; \sum_{e \text{ out of } A} c(e)$



Capacity = 9 + 15 + 8 + 30
= 62

Minimum Cut Problem

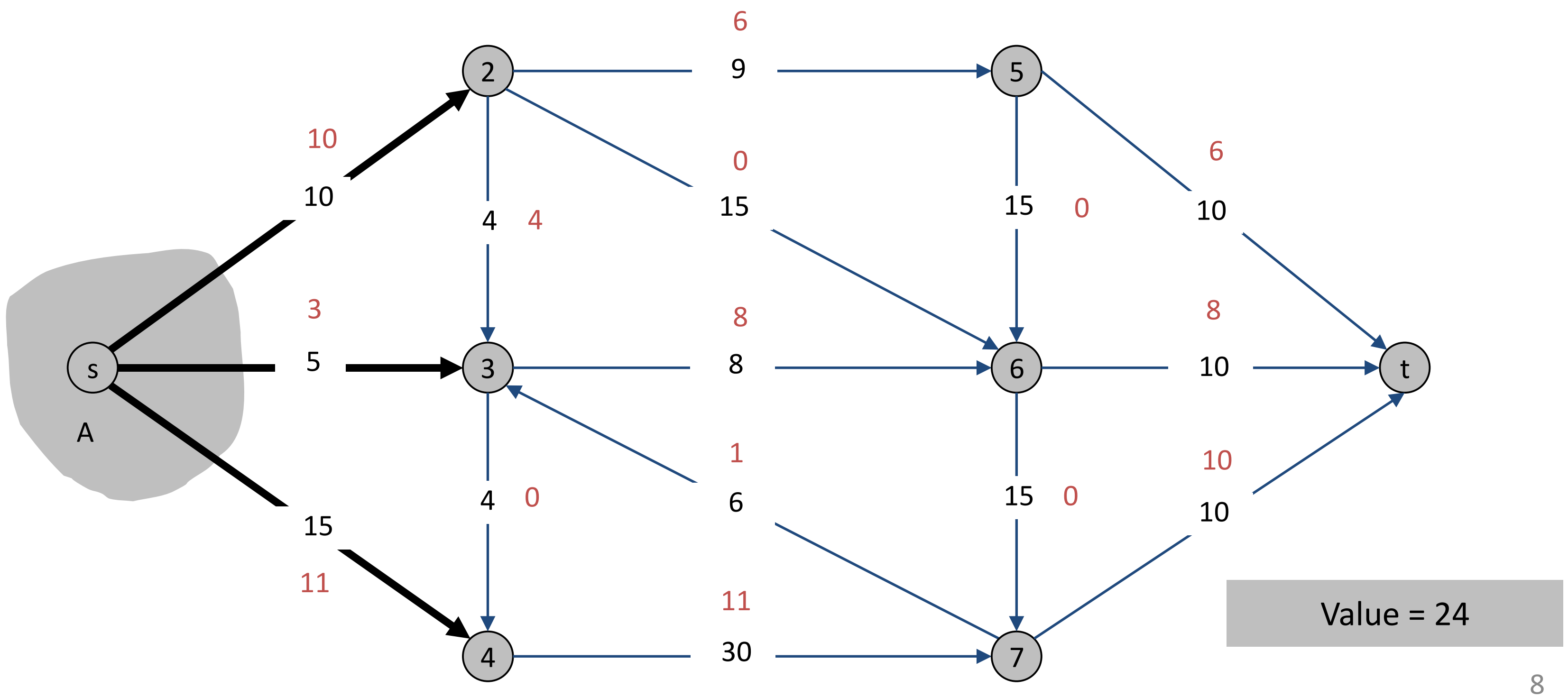
Min s-t cut problem. Find an s-t cut of minimum capacity.



Flows and Cuts

Flow value lemma. Let f be any flow, and let (A, B) be any s-t cut. Then, the net flow sent across the cut is equal to the amount leaving s.

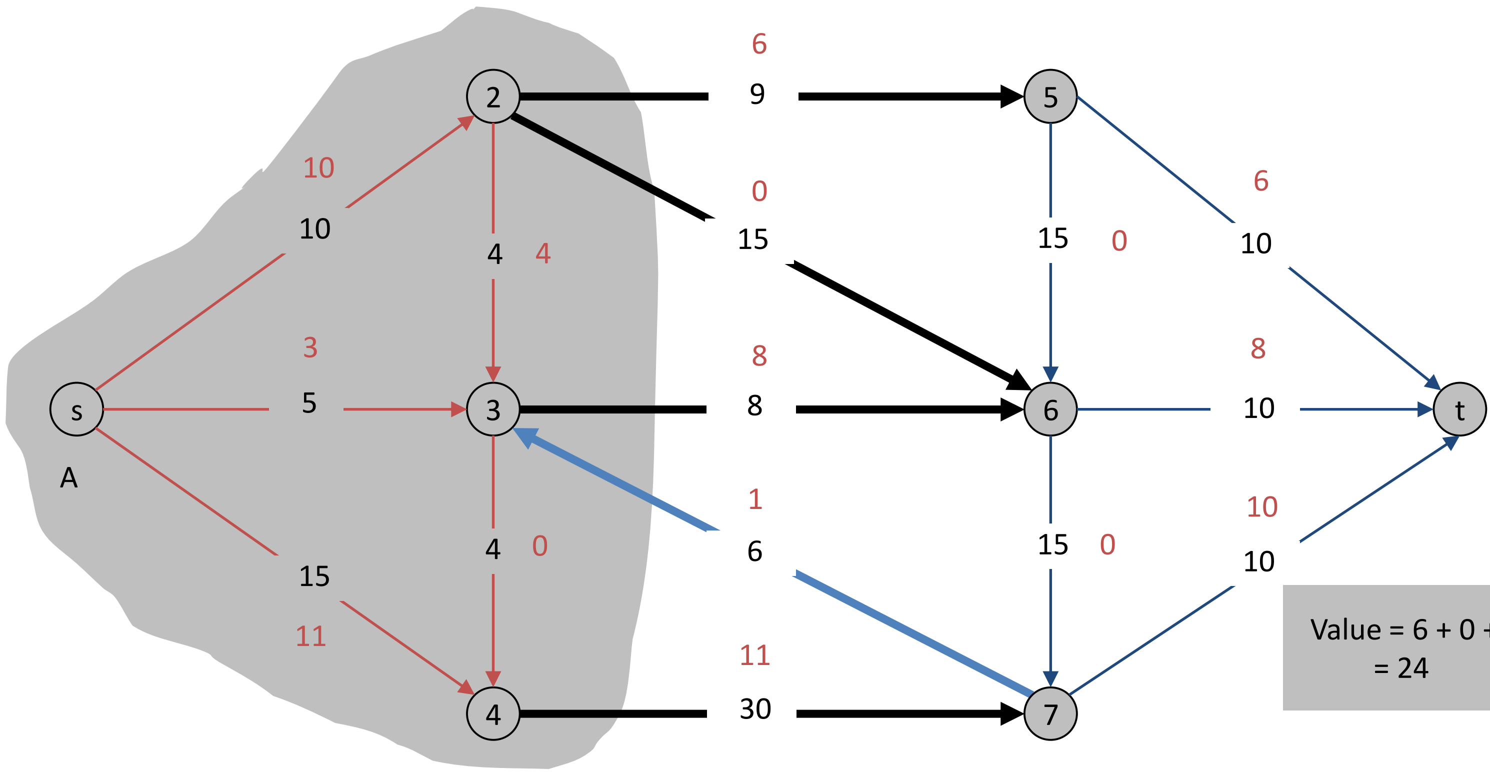
$$\sum_{e \text{ out of } A} f(e) \; - \sum_{e \text{ in to } A} f(e) \; = \; v(f)$$



Flows and Cuts

Flow value lemma. Let f be any flow, and let (A, B) be any s-t cut. Then, the net flow sent across the cut is equal to the amount leaving s.

$$\sum_{e \text{ out of } A} f(e) \; - \; \sum_{e \text{ in to } A} f(e) \; = \; v(f)$$

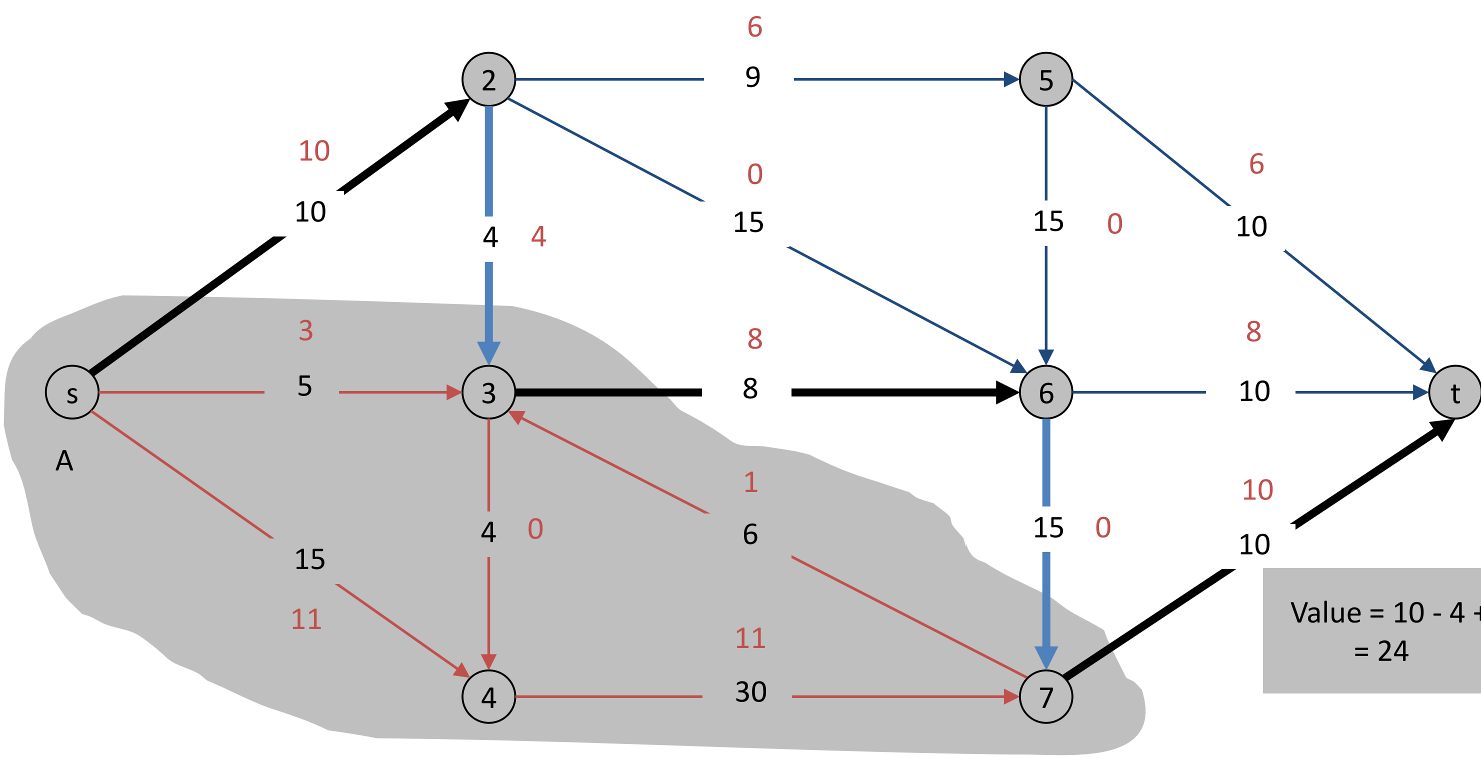


Value = 6 + 0 + 8 - 1 + 11
= 24

Flows and Cuts

Flow value lemma. Let f be any flow, and let (A, B) be any s-t cut. Then, the net flow sent across the cut is equal to the amount leaving s.

$$\sum_{e \text{ out of } A} f(e) \; - \sum_{e \text{ in to } A} f(e) \; = \; v(f)$$

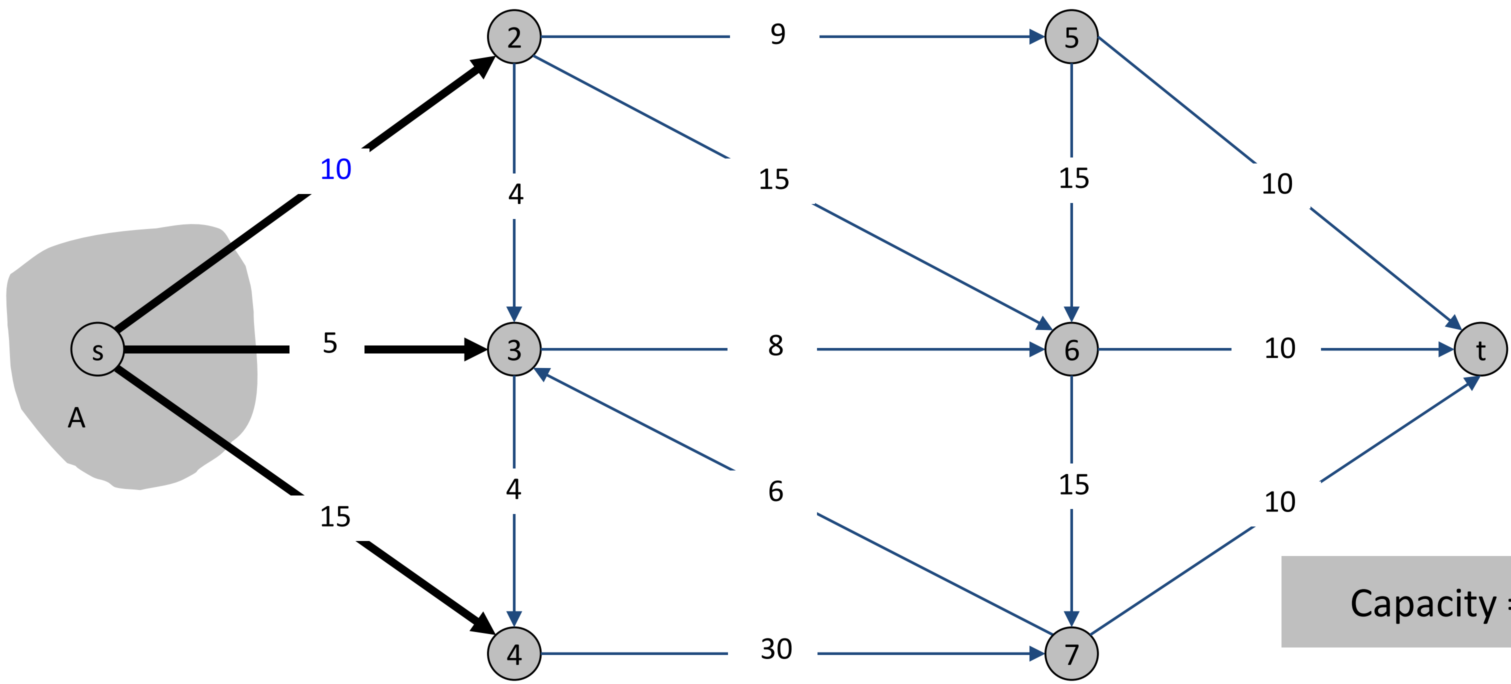


Value = 10 - 4 + 8 - 0 + 10
= 24

Flows and Cuts

Weak duality. Let f be any flow, and let (A, B) be any s-t cut. Then the value of the flow is at most the capacity of the cut.

Cut capacity = 30 $\Rightarrow$ Flow value $\leq$ 30

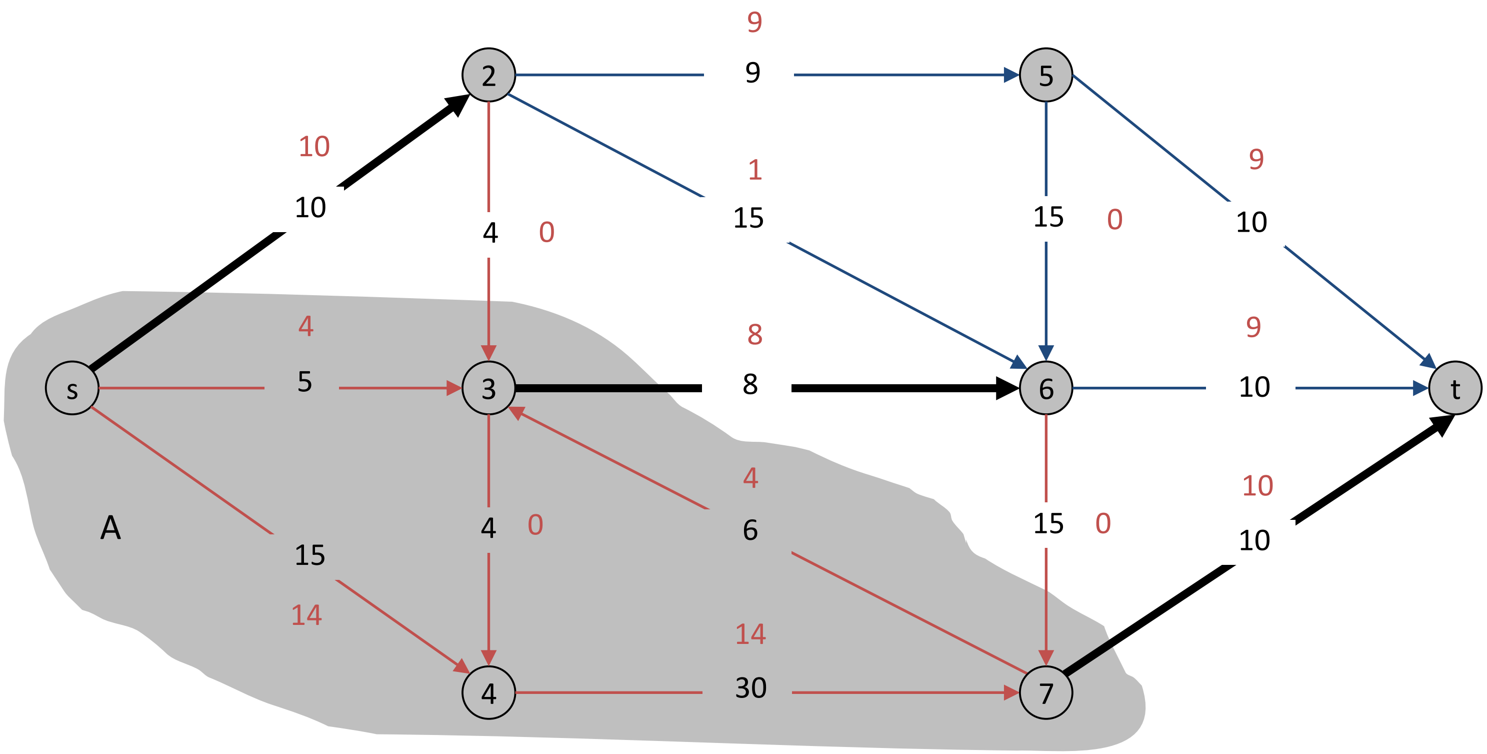


Capacity = 30

Certificate of Optimality

Max-Flow/Min-Cut Theorem.
Let f be any flow, and let (A, B) be any cut.
If v(f) = cap(A, B), then f is a max flow and (A, B) is a min cut.

Value of flow = 28
Cut capacity = 28 $\Rightarrow$ Flow value $\leq$ 28



Max-Flow Min-Cut Theorem

Augmenting path theorem. Flow f is a max flow iff there are no augmenting paths.

Max-flow min-cut theorem. [Ford-Fulkerson 1956] The value of the max flow is equal to the value of the min cut.

Proof strategy. We prove both simultaneously by showing the following are equivalent.

(i) There exists a cut (A, B) such that $v(f) = cap(A, B)$.

(ii) Flow f is a max flow.

(iii) There is no augmenting path relative to f.

(i) $\Rightarrow$ (ii) This was the corollary to weak duality lemma.

(ii) $\Rightarrow$ (iii) We show contrapositive.

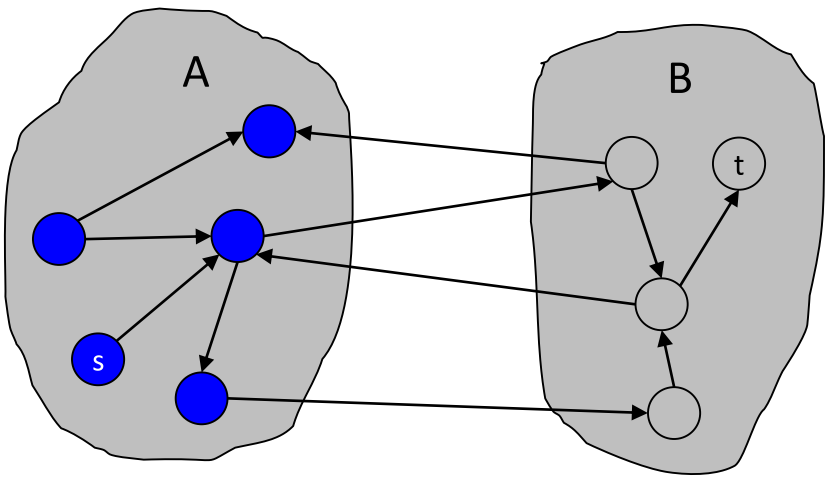
- Let f be a flow. If there exists an augmenting path, then we can improve f by sending flow along path.

Proof of Max-Flow Min-Cut Theorem

(iii) $\Rightarrow$ (i)

- Let f be a flow with no augmenting paths.
- Let A be set of vertices reachable from s in residual graph.
- By definition of A, $s \in A$.
- By definition of f, $t \notin A$.

$$
\begin{aligned}
v(f) &= \sum_{e \text{ out of } A} f(e) - \sum_{e \text{ in to A}} f(e) \\
&= \sum_{e \text{ out of } A} c(e) \\
&= cap(A, B)
\end{aligned}
$$



original network

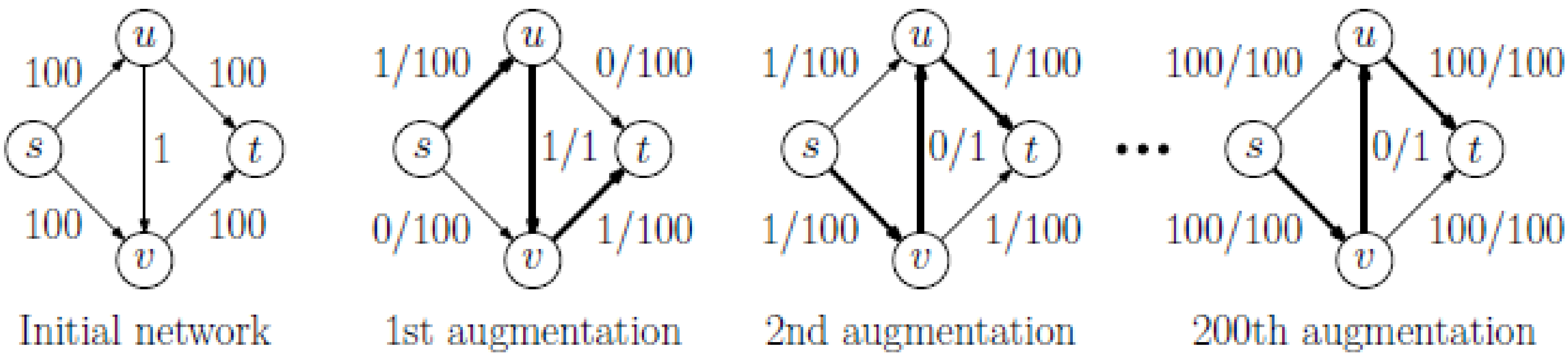
Analysis of Ford-Fulkerson

Assumption. All capacities are integers between 1 and C.

Invariant. Every flow value $f(e)$ and every residual capacity $c_f(e)$ remains an integer throughout the algorithm.

Lemma. Given an s-t network with integer capacities, the Ford-Fulkerson algorithm terminates. Furthermore, it produces an integer-valued flow function.

Bad Example for Ford-Fulkerson



If we let |f| denote the final maximum flow value, the number of augmentation steps can be as high as |f|.

Choosing Good Augmenting Paths

Use care when selecting augmenting paths.

- Some choices lead to exponential algorithms.
- Clever choices lead to polynomial algorithms.
- If capacities are irrational, algorithm not guaranteed to terminate!

Goal: choose augmenting paths so that:

- Can find augmenting paths efficiently.
- Few iterations.

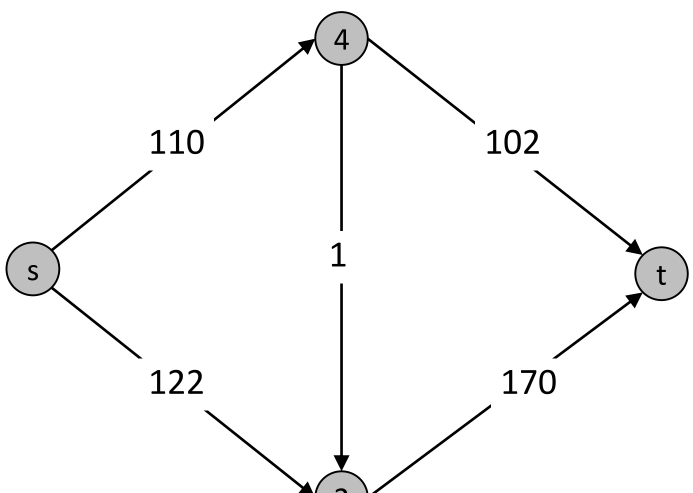
Choose augmenting paths with: [Edmonds-Karp 1972, Dinitz 1970]

- Max bottleneck capacity.
- Sufficiently large bottleneck capacity.
- Fewest number of edges.

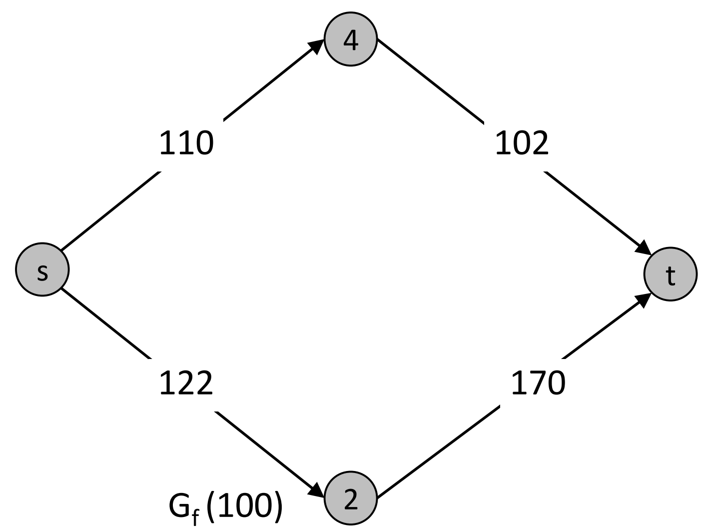
Capacity Scaling

Intuition. Choosing path with highest bottleneck capacity increases flow by max possible amount.

- The sum of capacities of the edges leaving s is $C = \sum_{(s,v)\in E} c(s,v)$
- Define Δ to be the largest power of 2, such that $\Delta \leq C$
- Let $G_f(\Delta)$ be the subgraph of the residual graph consisting of only arcs with capacity at least Δ.



G_f



$G_f(100)$

Capacity Scaling

```
Scaling-Max-Flow(G, s, t, c) {
   foreach e ∈ E  f(e) ← 0
   Δ ← smallest power of 2 greater than or equal to C
   G_f ← residual graph

   while (Δ ≥ 1) {
      G_f(Δ) ← Δ-residual graph
      while (there exists augmenting path P in G_f(Δ)) {
         f ← augment(f, c, P)
         update G_f(Δ)
      }
      Δ ← Δ / 2
   }
   return f
}
```

Edmonds-Karp Algorithm

- Neither of the algorithms we have seen so far runs in “truly” polynomial time
- Edmonds and Karp developed the first polynomial-time algorithm for flow networks.
  - Uses Ford-Fulkerson as basis
  - Modification: when finding the augmenting path, we compute the s-t path in the residual network having the smallest number of edges
    - Note that this can be accomplished by using BFS to compute the augmenting path
  - It can be shown that the total number of augmenting steps using this method is O(nm) (Proof in CLRS)
  - Overall runtime = $O(nm^2)$

Other Algorithms

- KT discusses pre-flow push algorithm
  - Number of variants of this algorithm
  - Simplest version runs in $O(n^3)$ time

- Another quite sophisticated algorithm runs in time $O(\min(n^{2/3}, m^{1/2}) m \log n \log U)$, where U is an upper bound on the largest capacity.