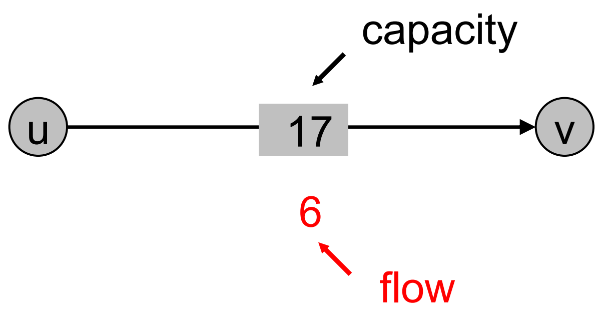
CSE 421

Network Flows, Matching

Shayan Oveis Gharan

Network Flows

Residual Graph



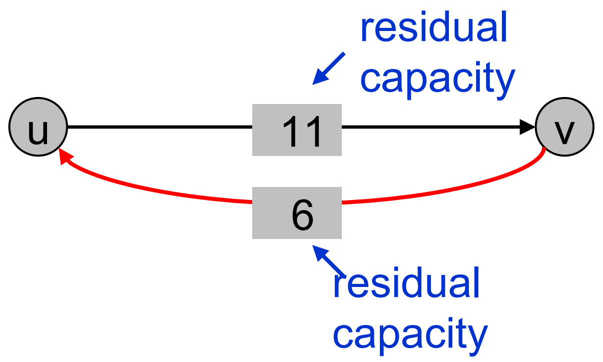
Original edge: e = (u, v) $\in$ E.

- Flow f(e), capacity c(e).

Residual edge.

- "Undo" flow sent.
- e = (u, v) and e^R = (v, u).
- Residual capacity:

$$c_f(e) = \begin{cases} c(e) - f(e) & if\ e \in E \\ f(e) & if\ e^R \in E \end{cases}$$

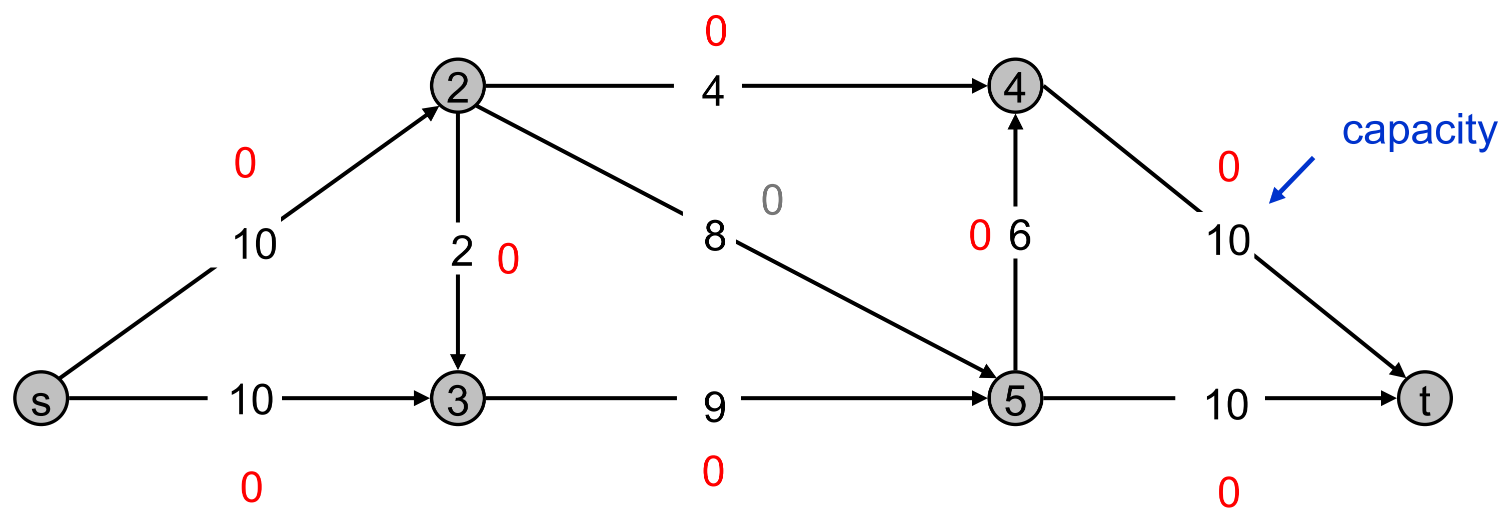


Residual graph: G_f = (V, E_f).

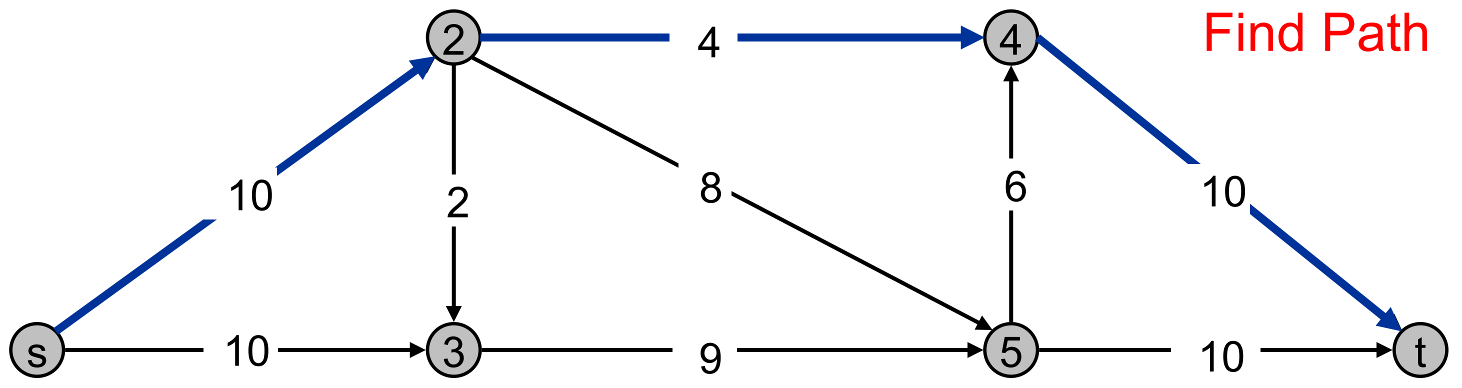
- Residual edges with positive residual capacity.
- $E_f \;=\; \{e : f(e) < \; c(e)\} \cup \{e :\; f(e^R) \;>\; 0\}$.

Ford-Fulkerson Alg: Greedy on G_f

G:

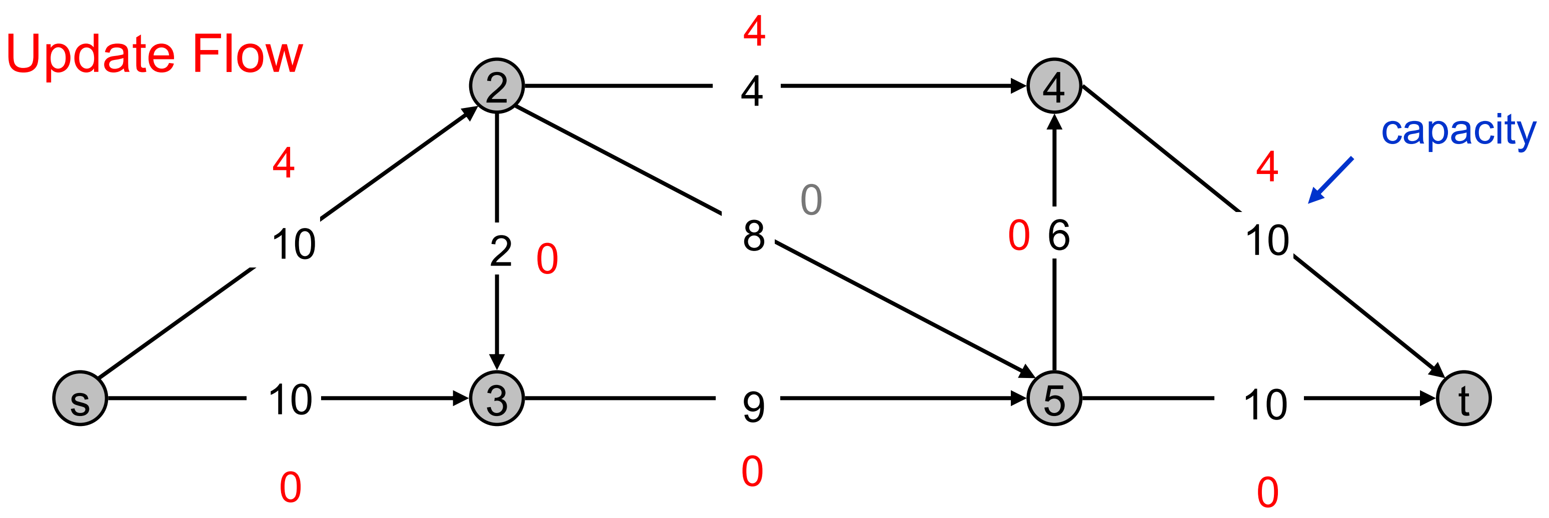


G_f:

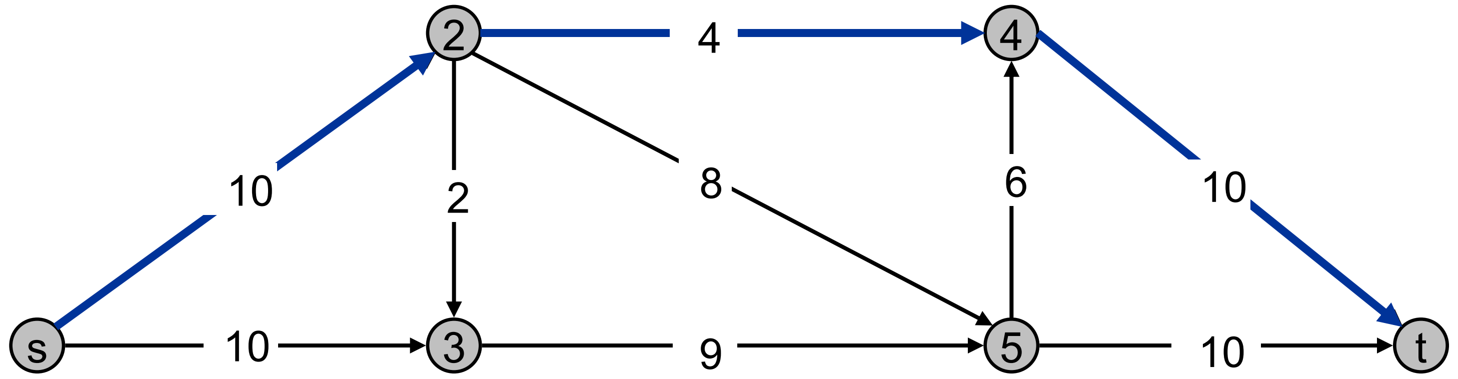


Ford-Fulkerson Alg: Greedy on G_f

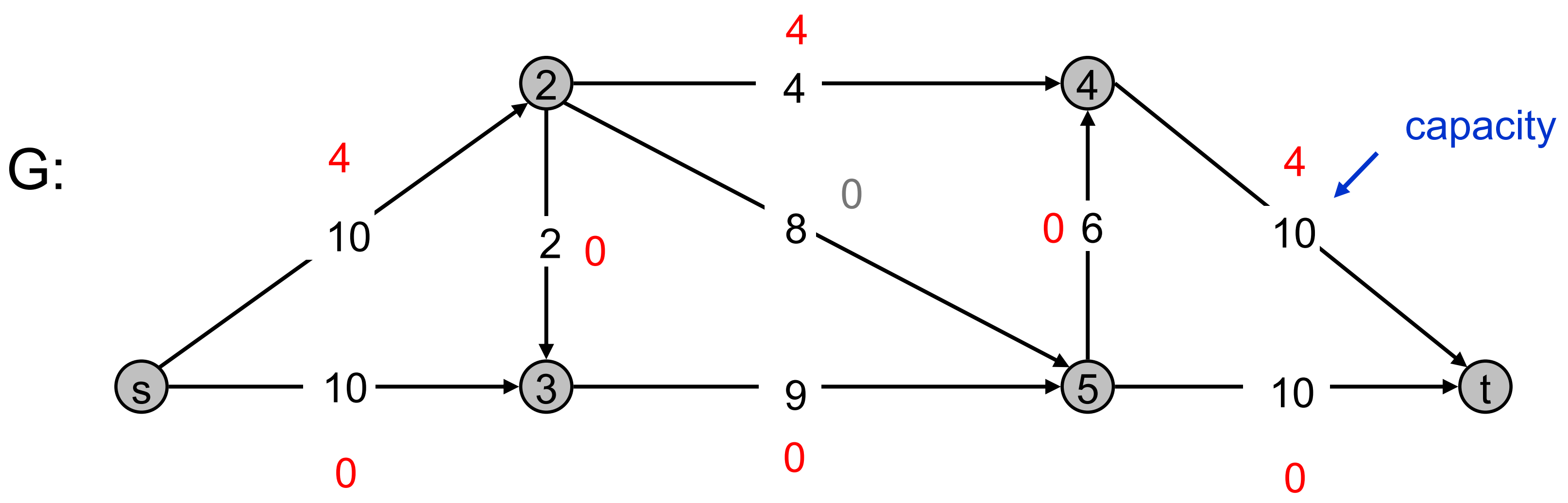
G:

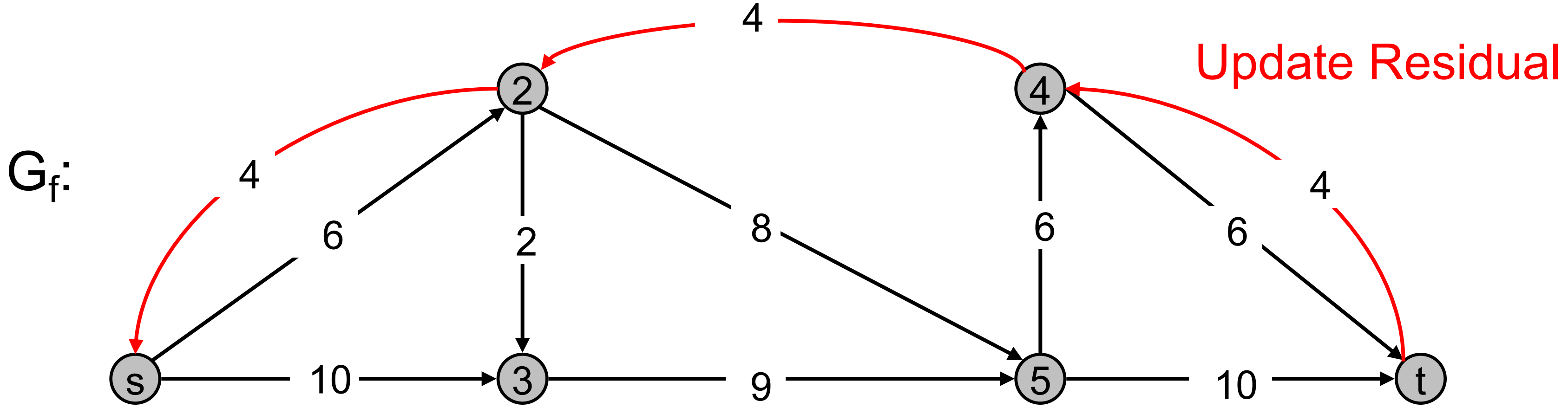


G_f:

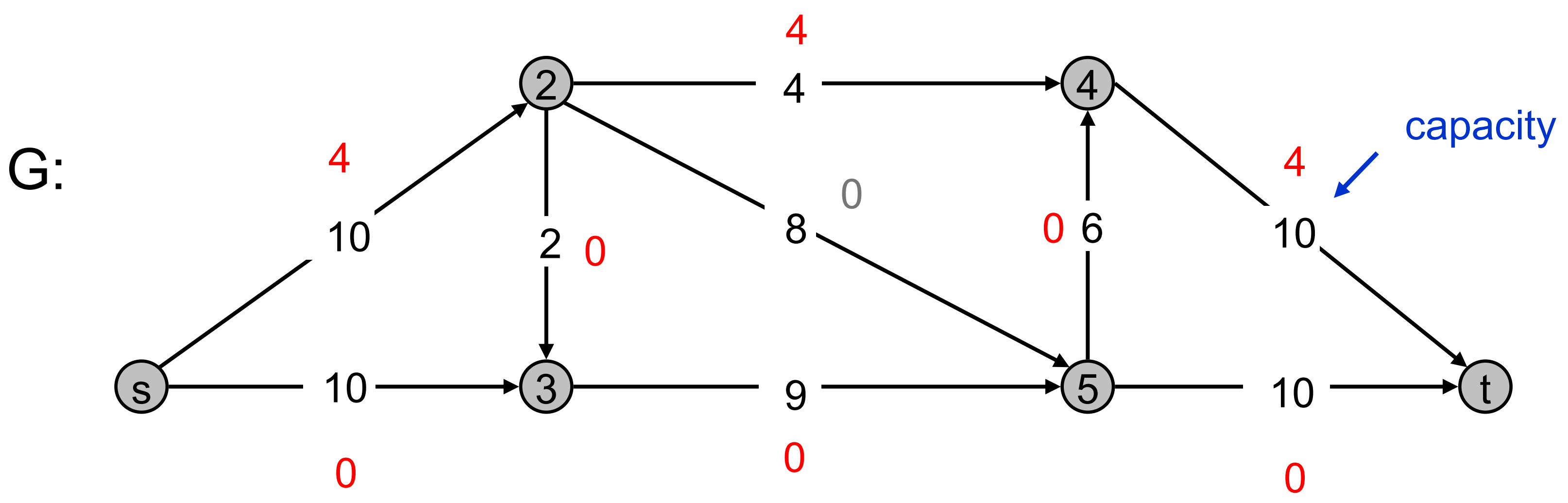


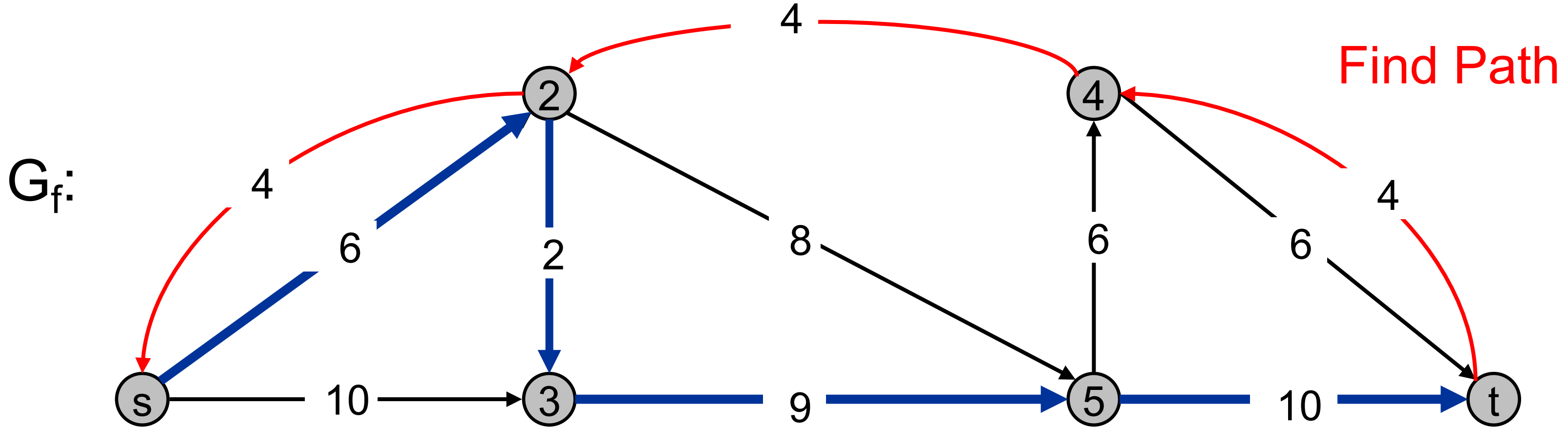
Ford-Fulkerson Alg: Greedy on G_f





Ford-Fulkerson Alg: Greedy on G_f

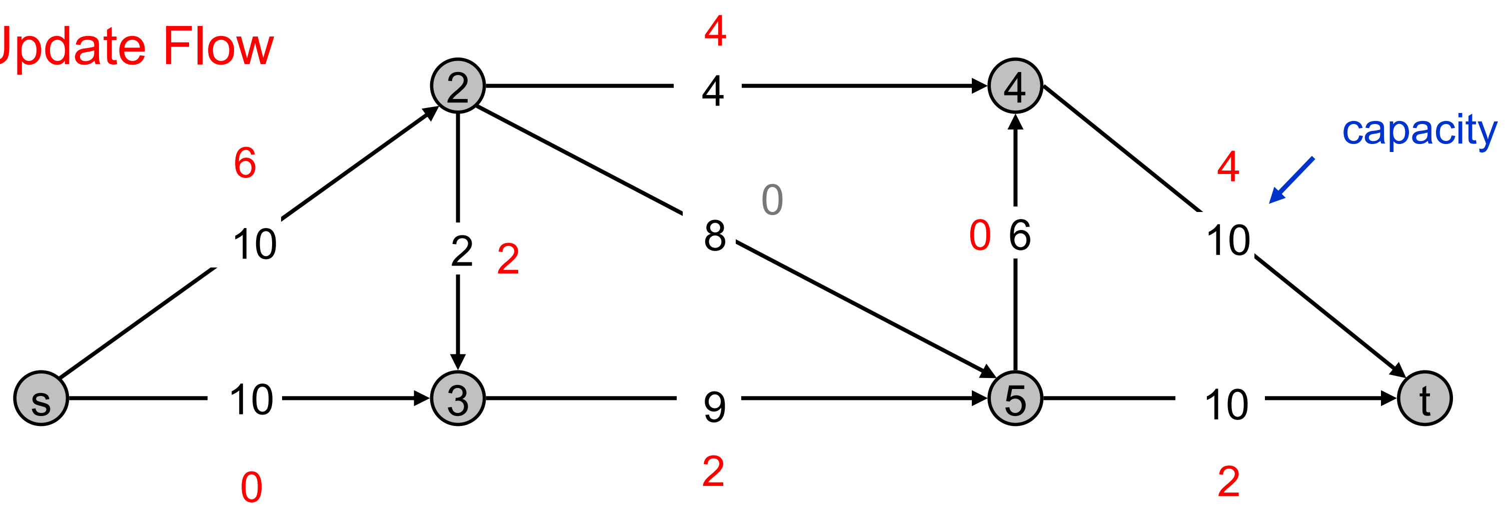




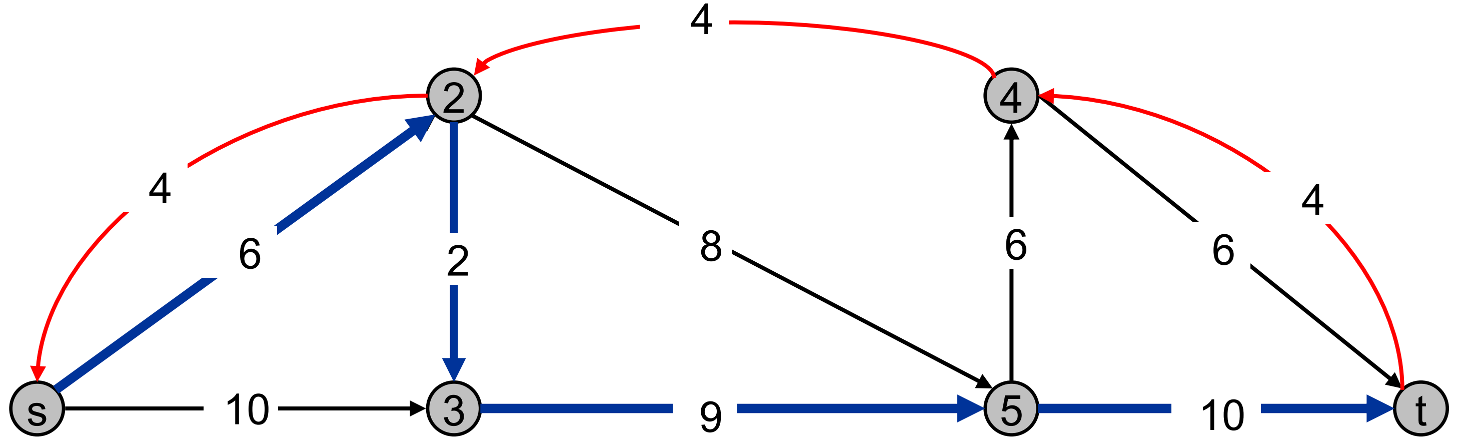
Ford-Fulkerson Alg: Greedy on G_f

Update Flow

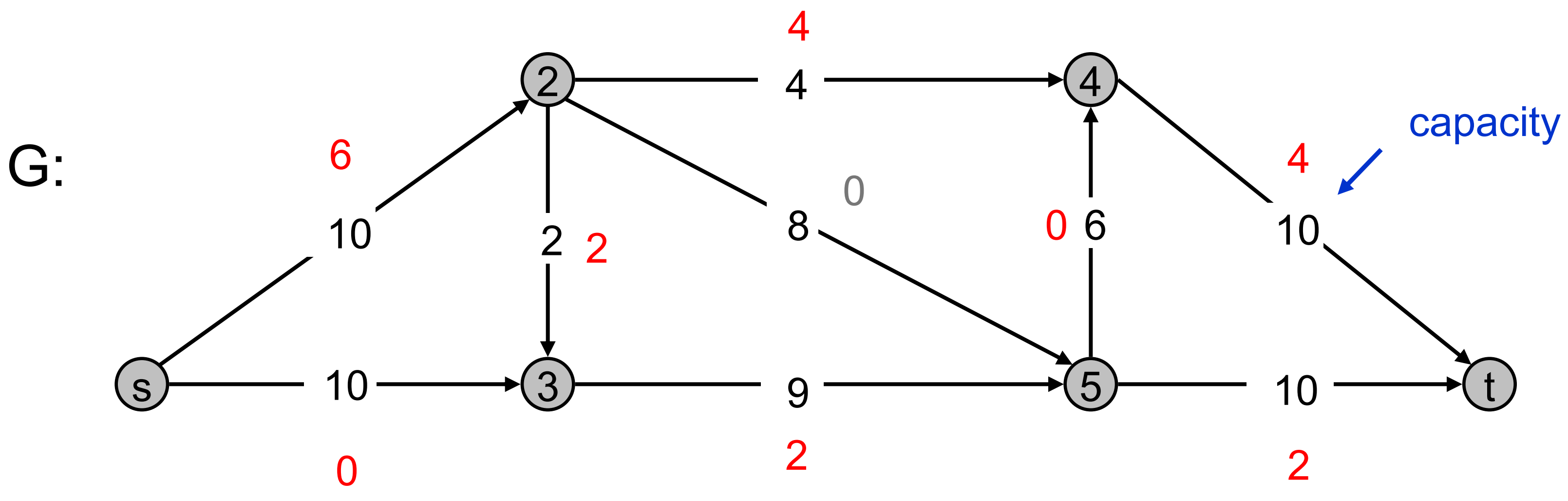
G:

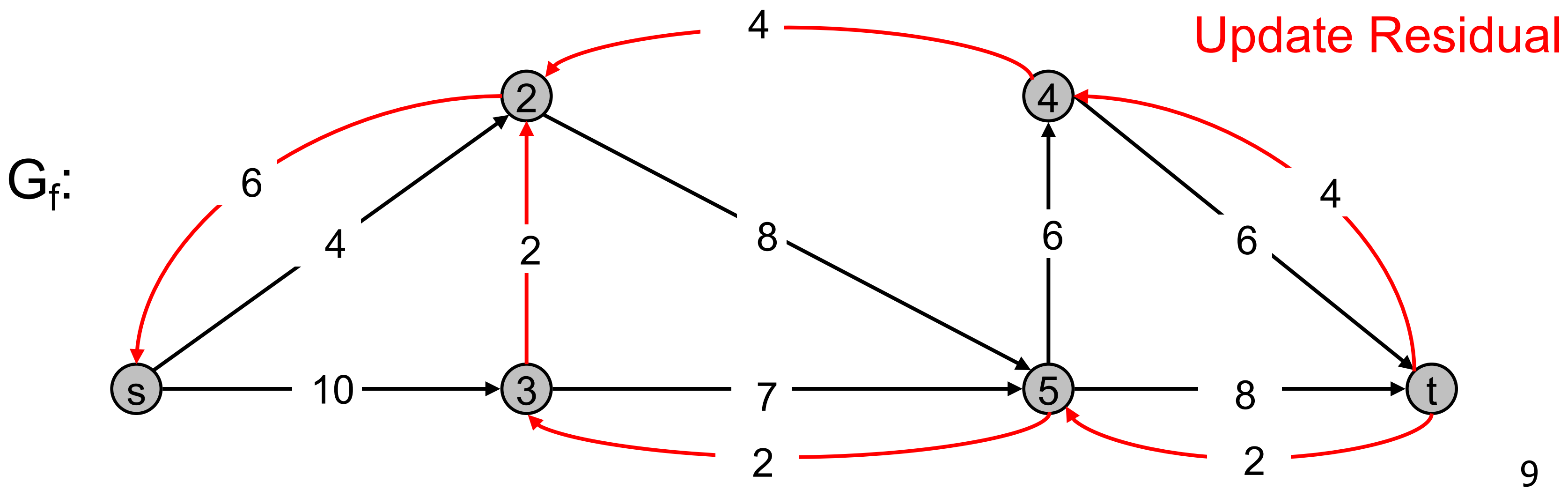


G_f:



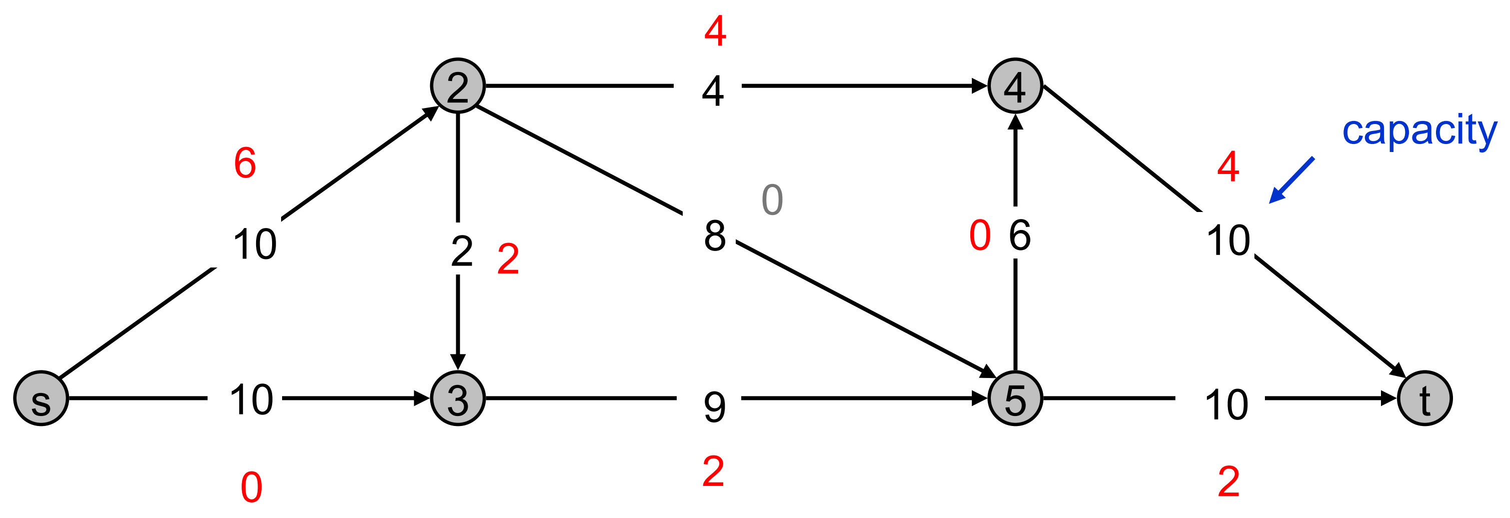
Ford-Fulkerson Alg: Greedy on G_f



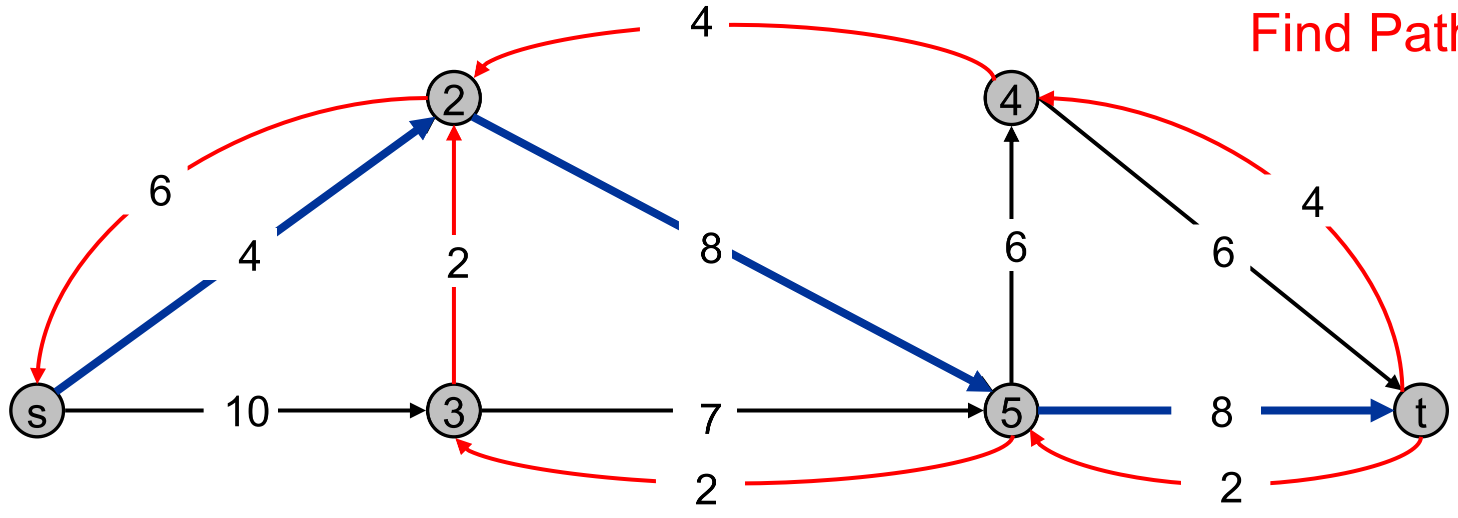


Ford-Fulkerson Alg: Greedy on G_f

G:



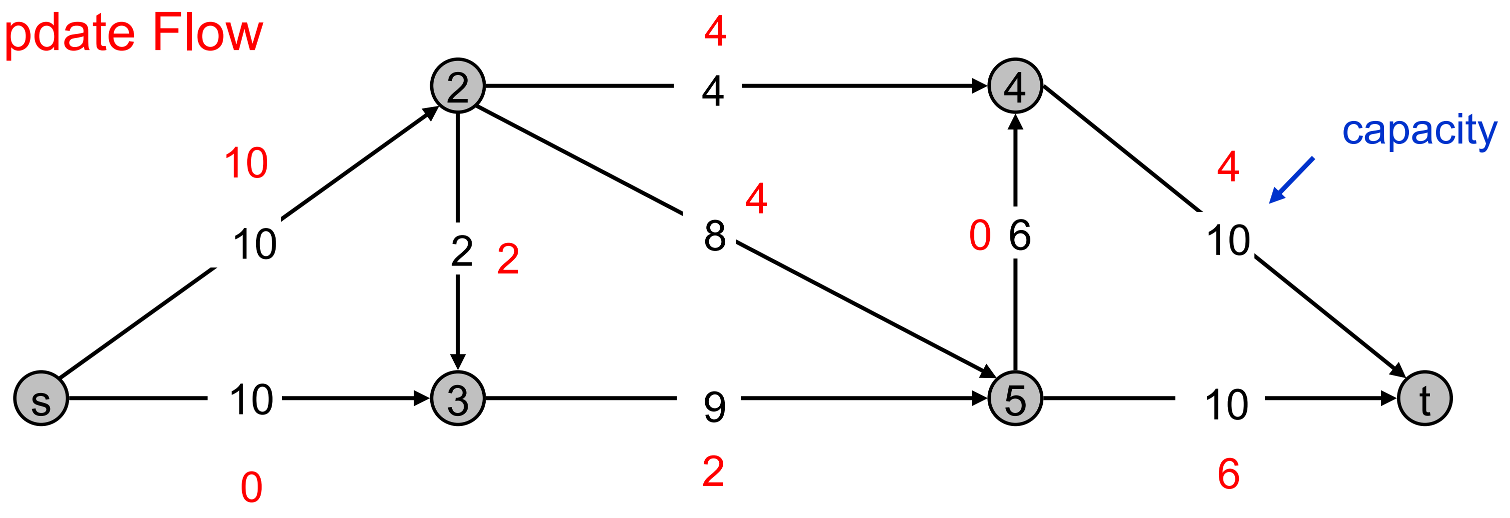
G_f:



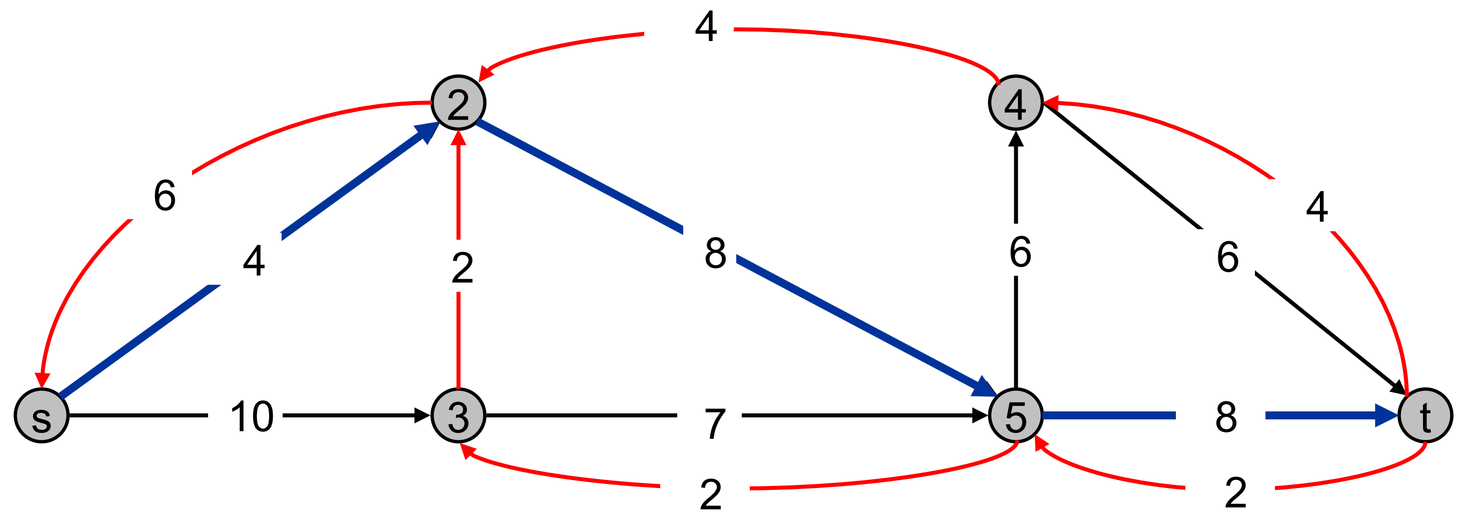
Ford-Fulkerson Alg: Greedy on G_f

Update Flow

G:

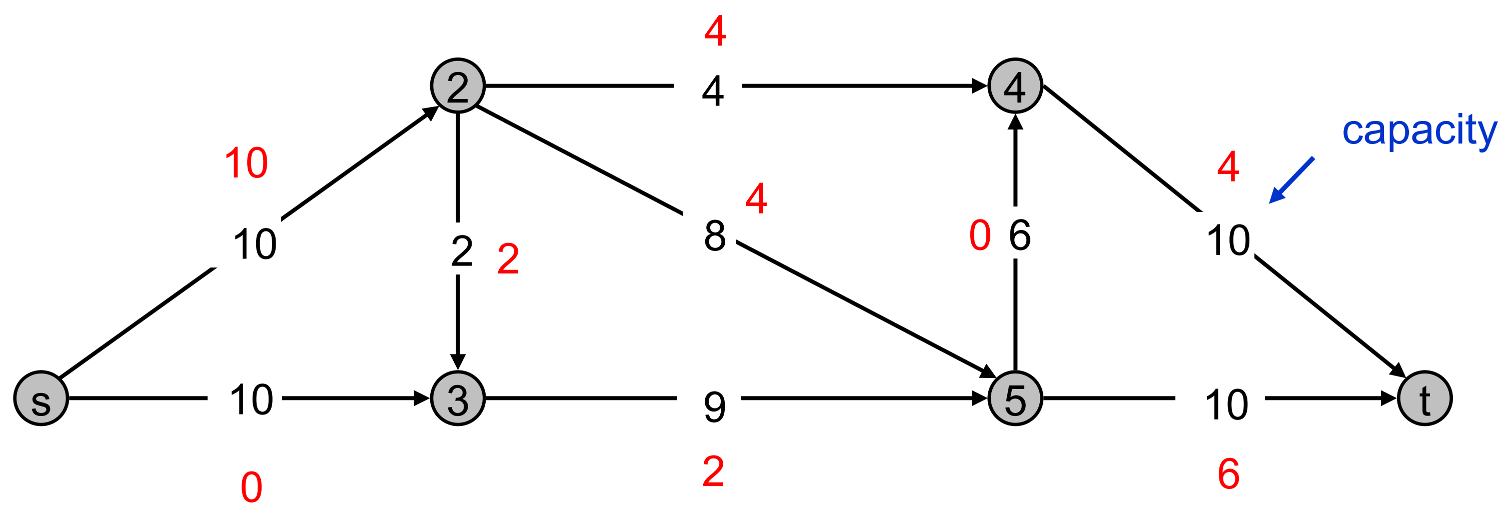


G_f:



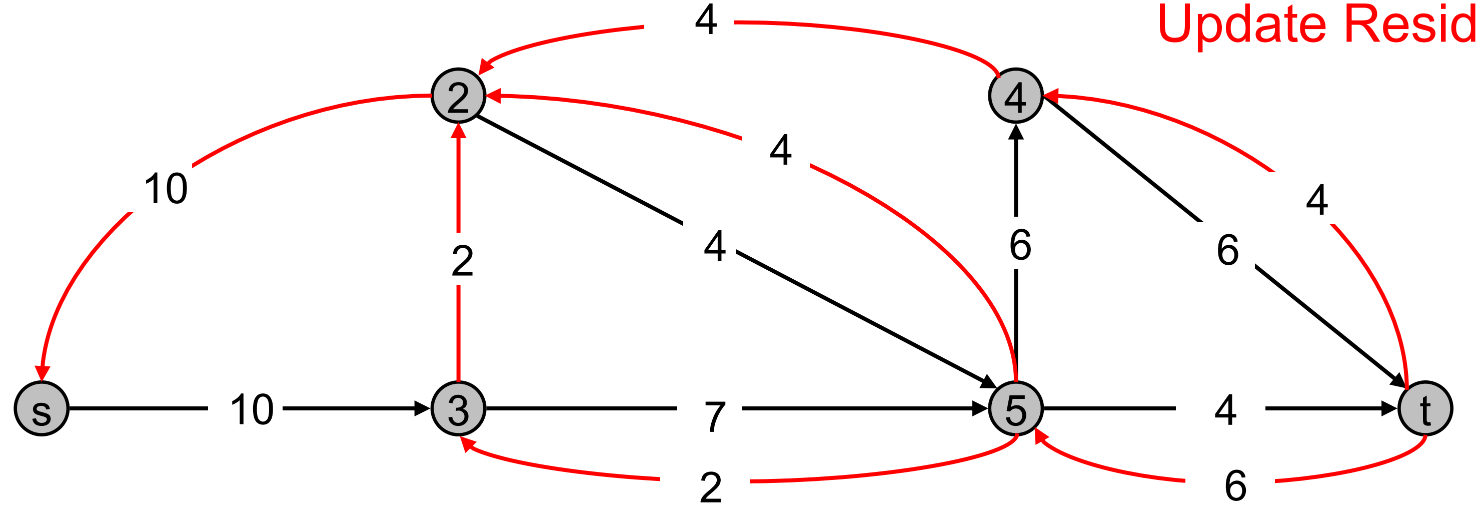
Ford-Fulkerson Alg: Greedy on G_f

G:

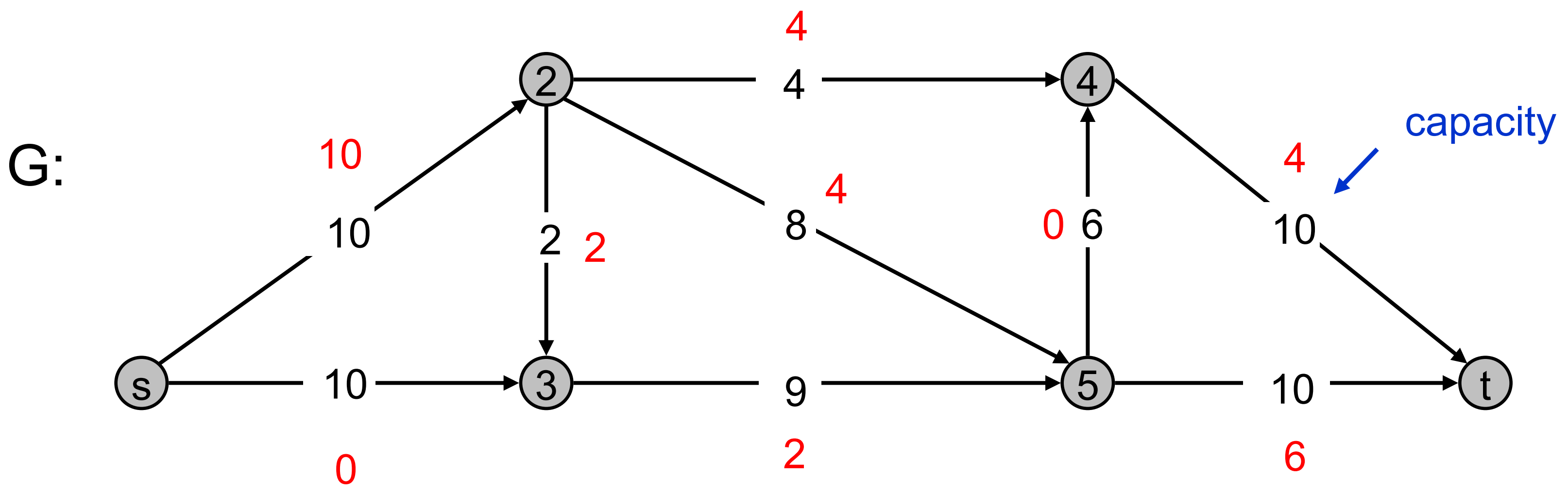


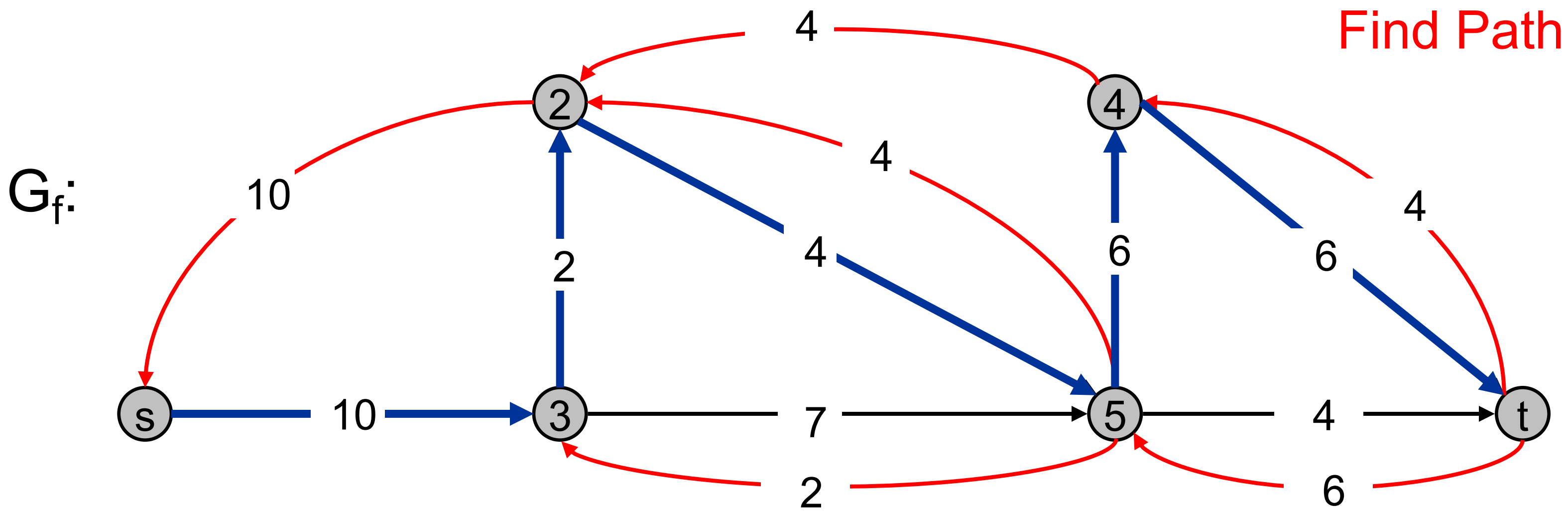
G_f:

Update Residual

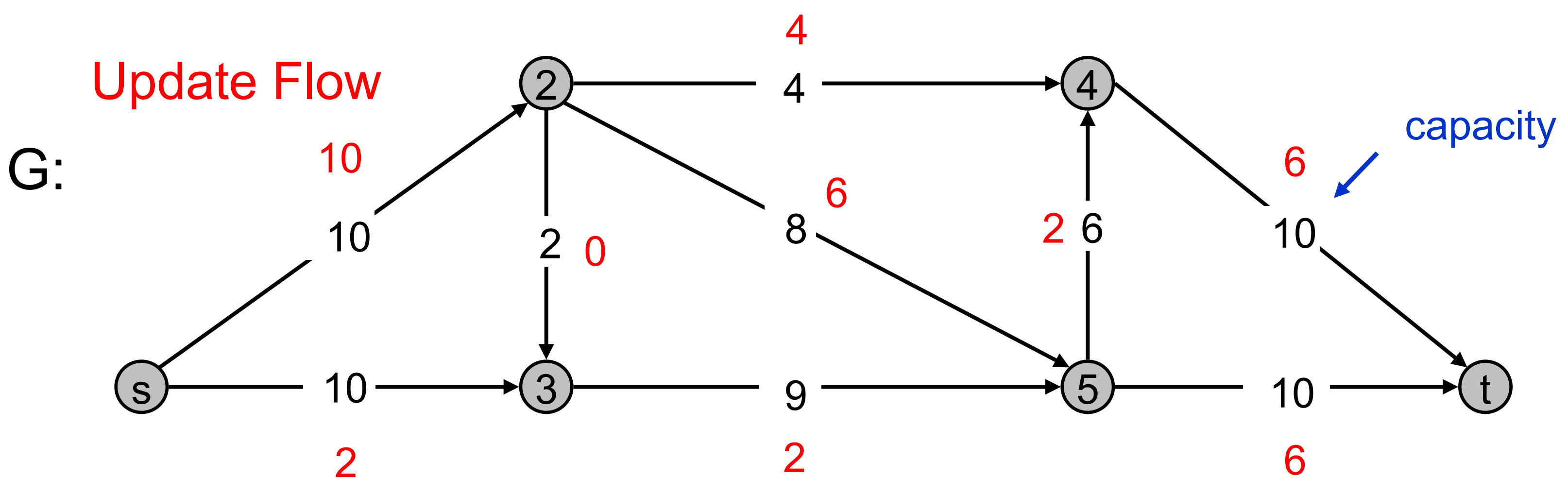


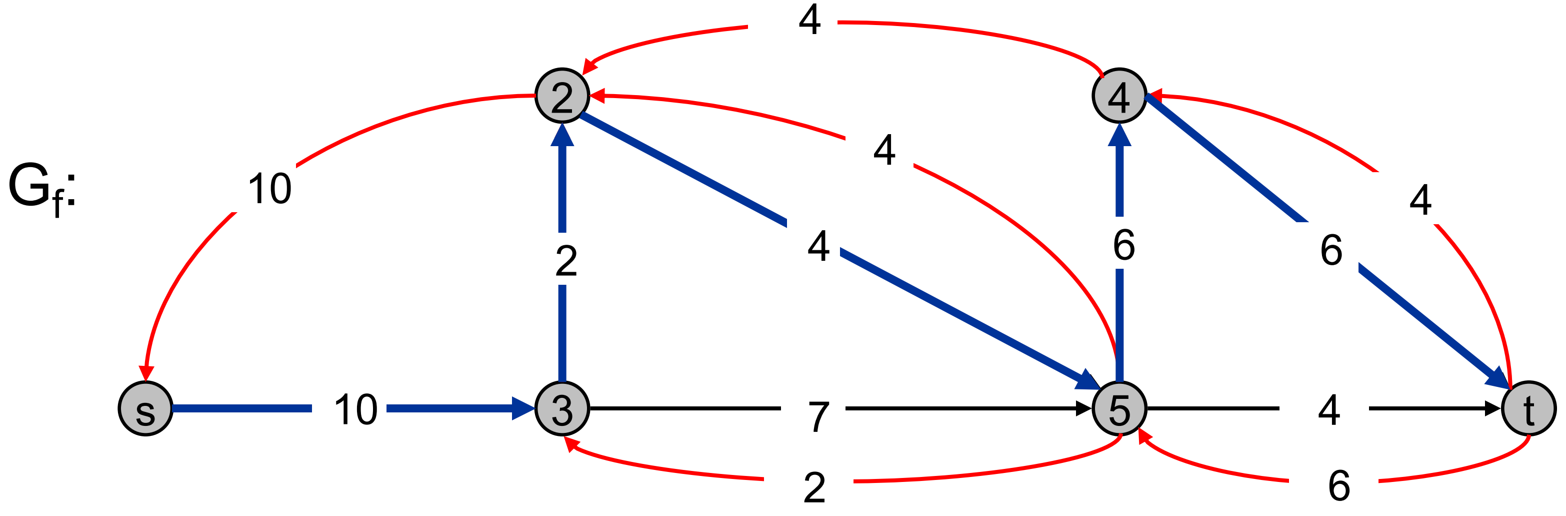
Ford-Fulkerson Alg: Greedy on G_f



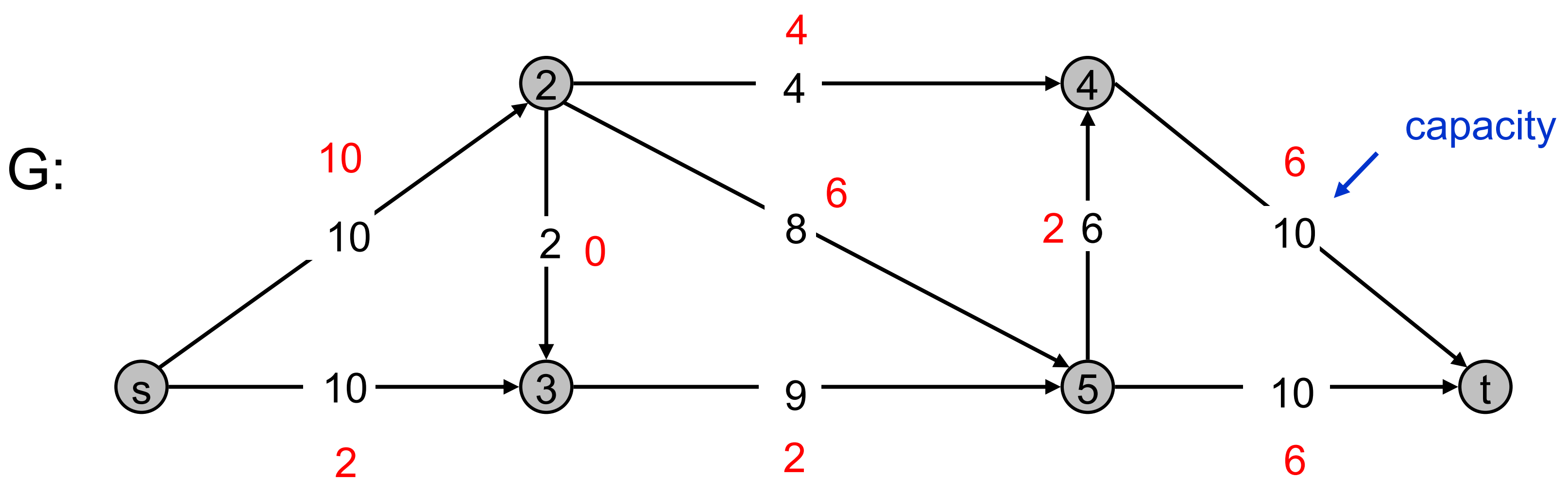


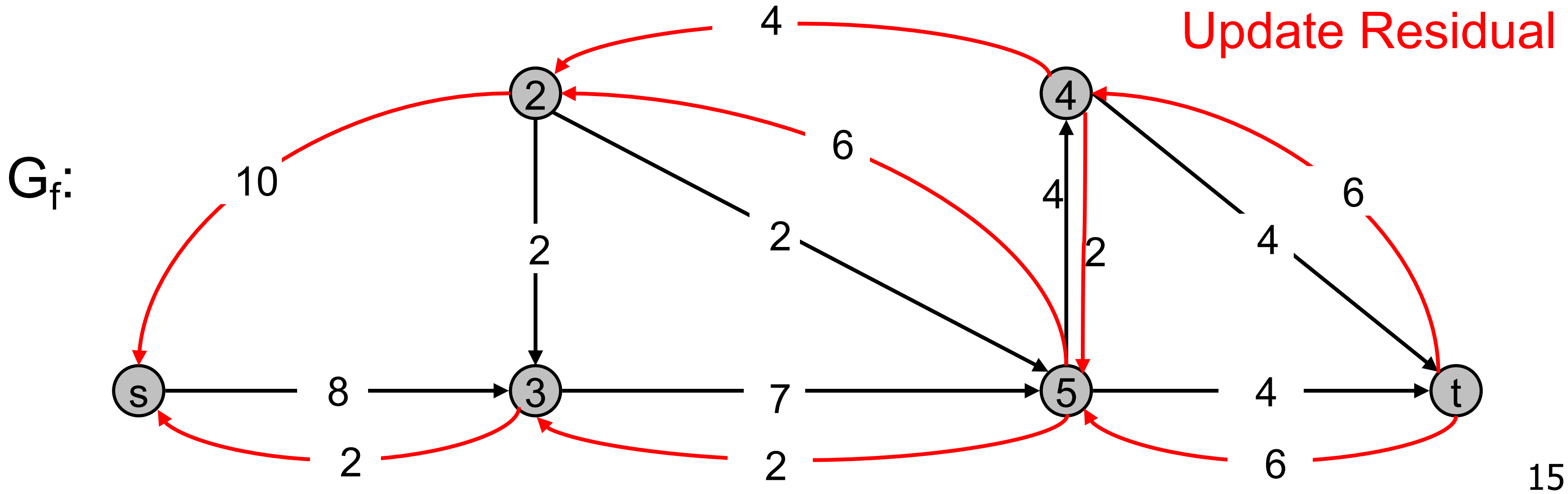
Ford-Fulkerson Alg: Greedy on G_f



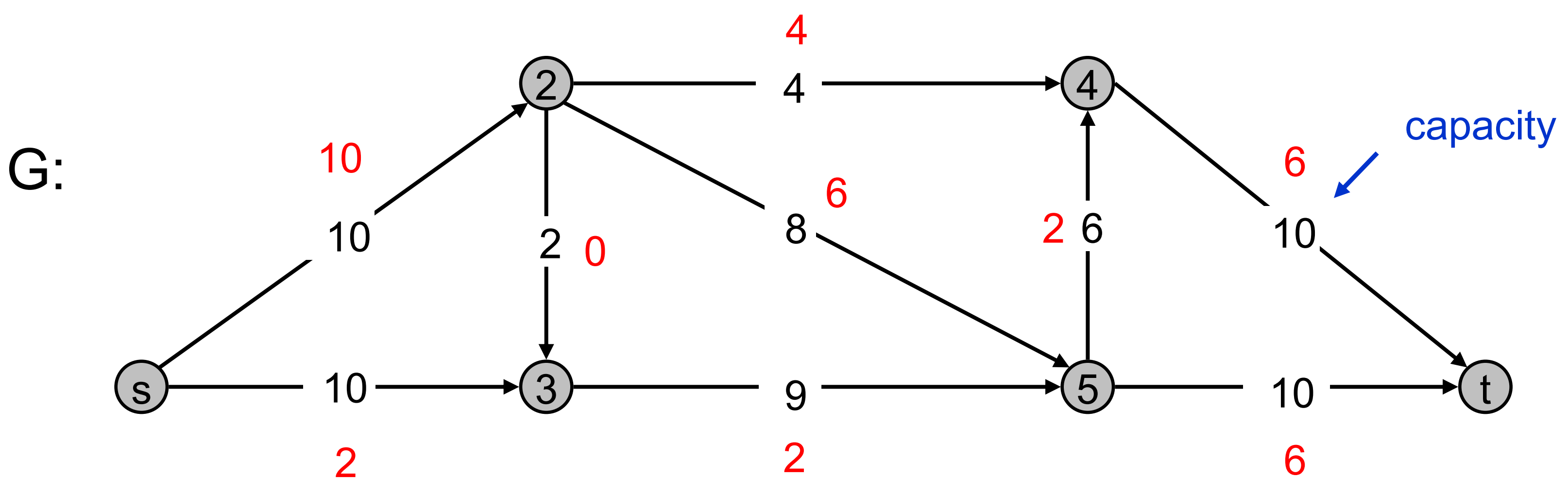


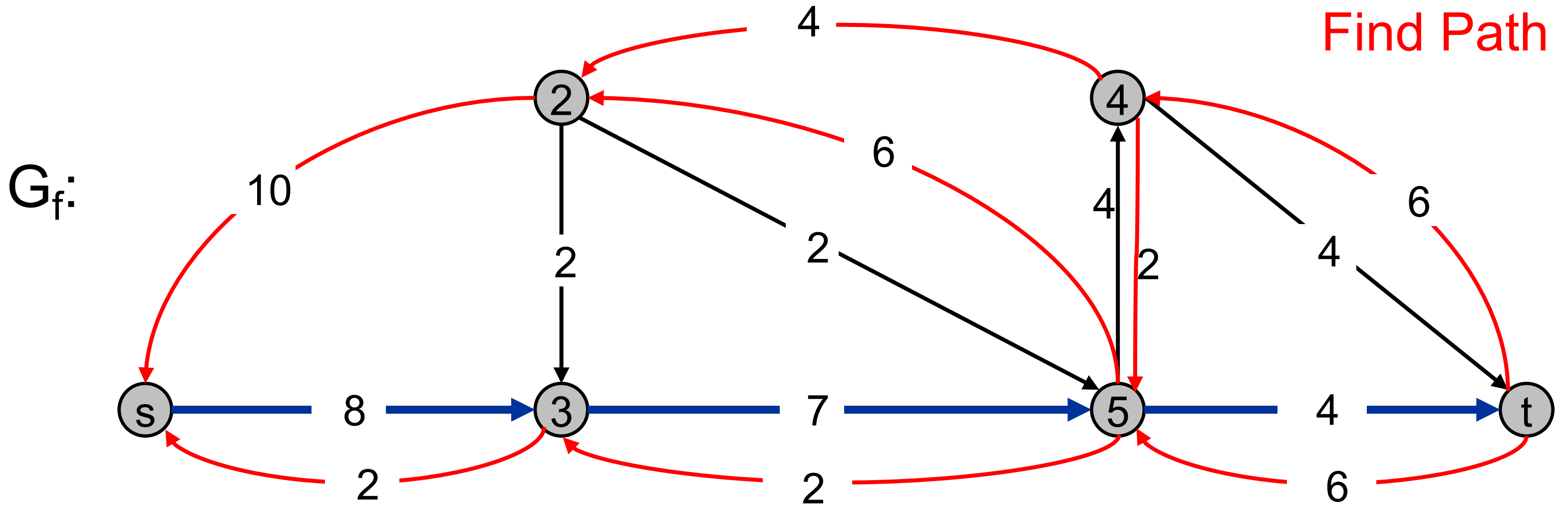
Ford-Fulkerson Alg: Greedy on G_f



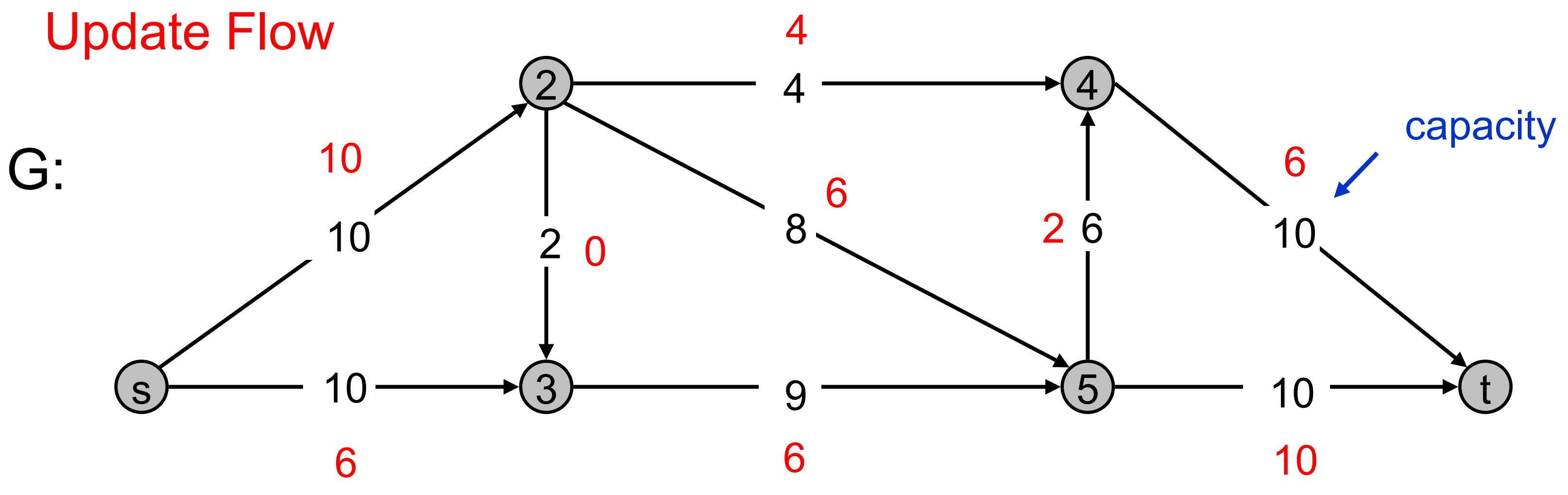


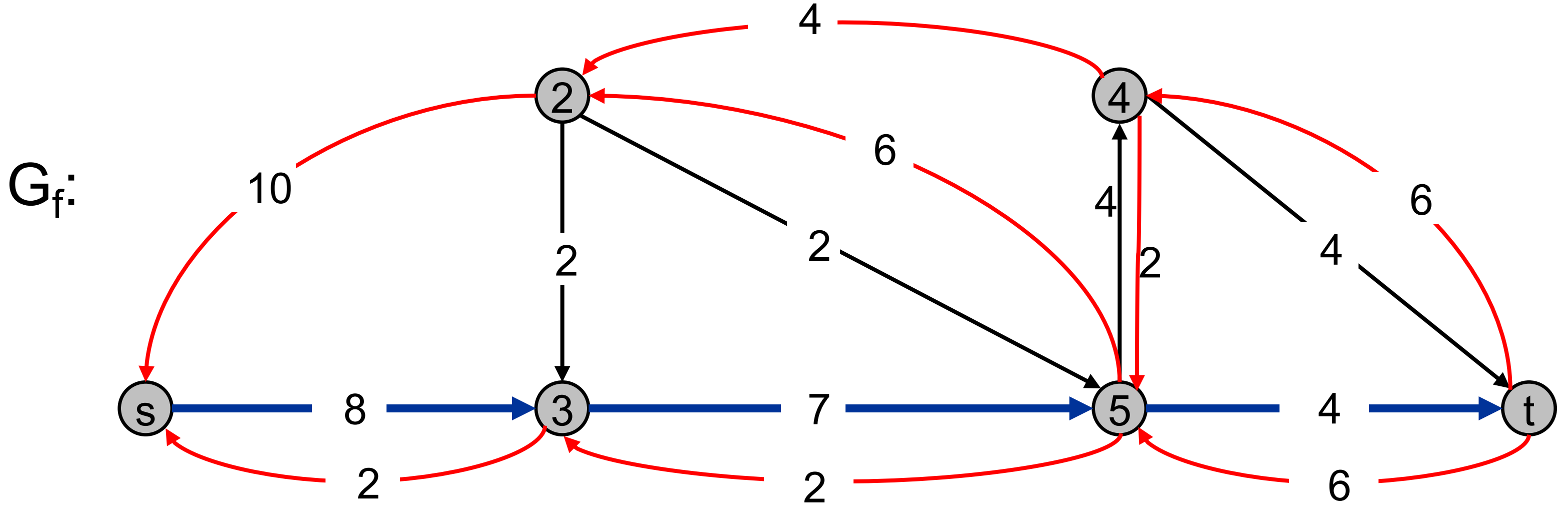
Ford-Fulkerson Alg: Greedy on G_f



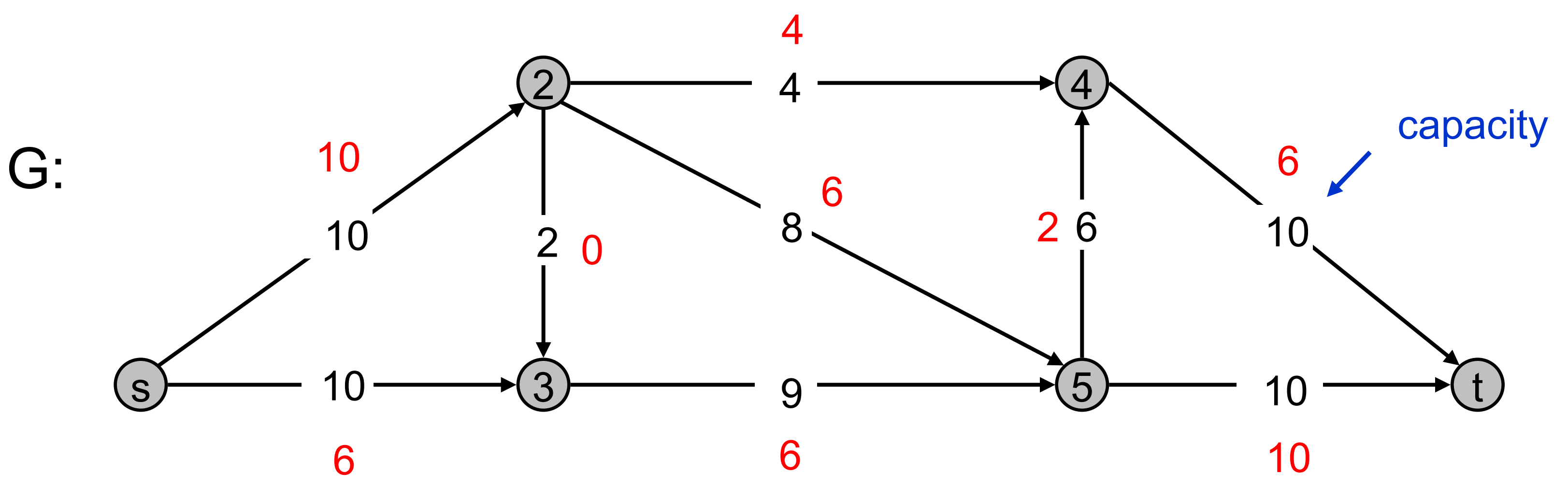


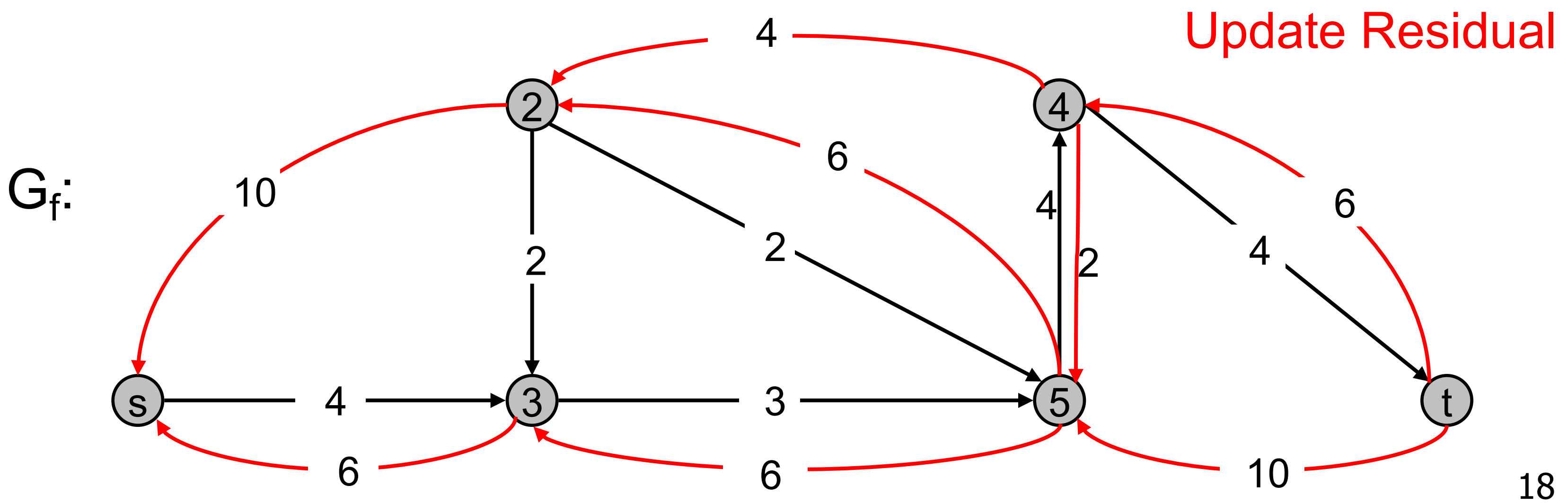
Ford-Fulkerson Alg: Greedy on G_f





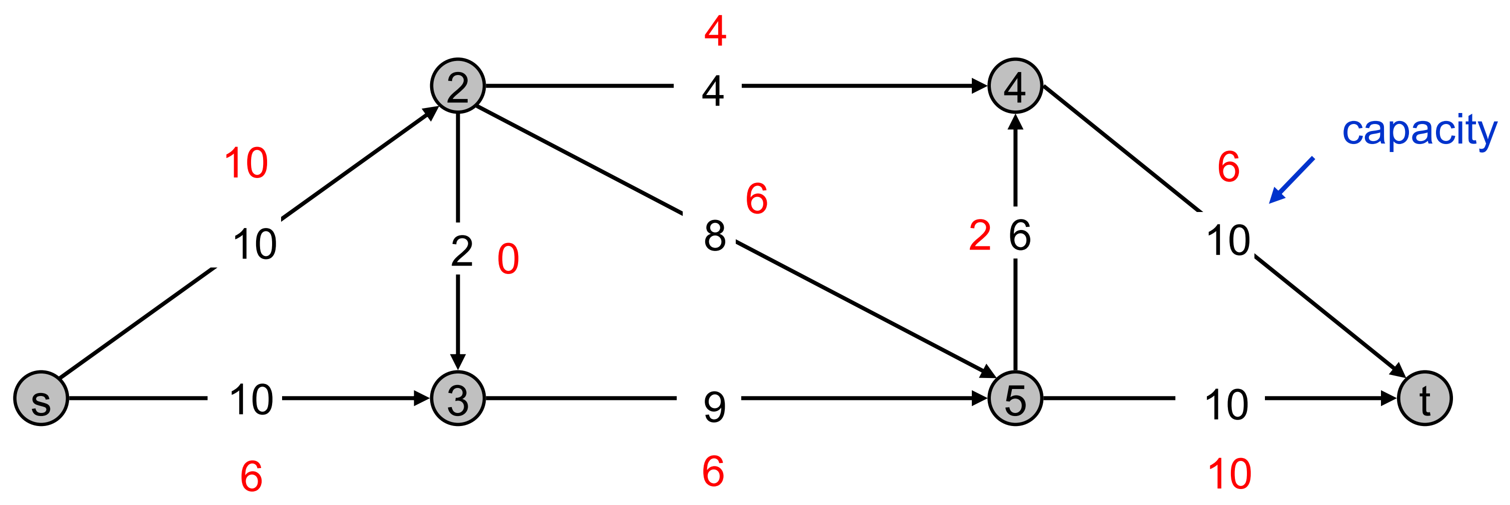
Ford-Fulkerson Alg: Greedy on G_f



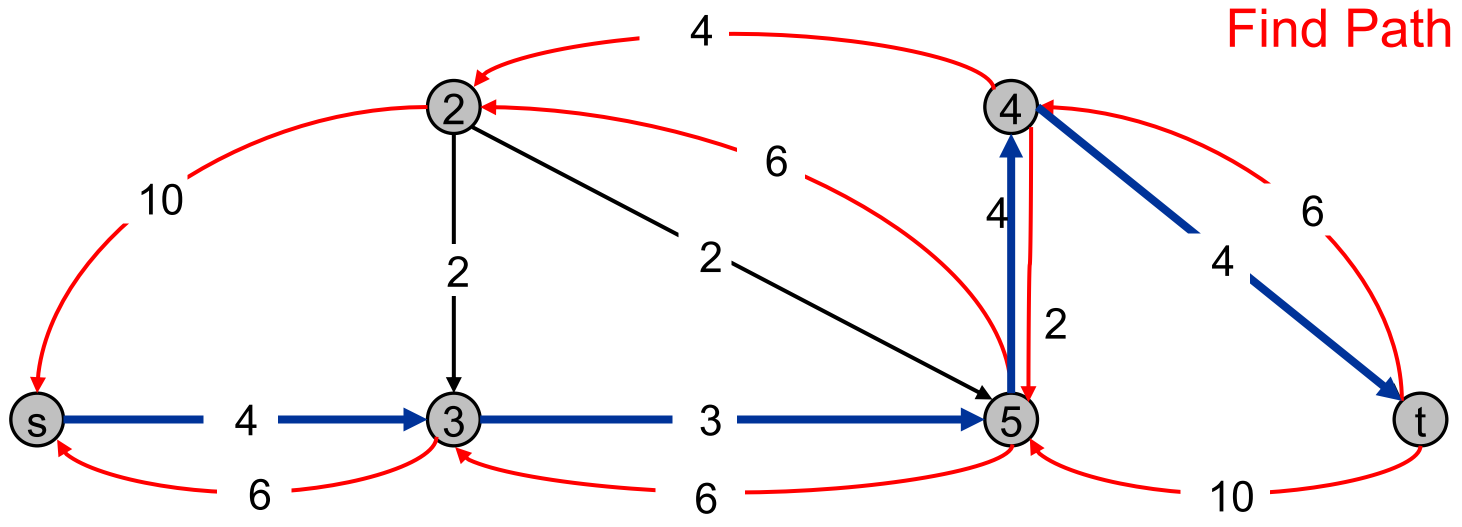


Ford-Fulkerson Alg: Greedy on G_f

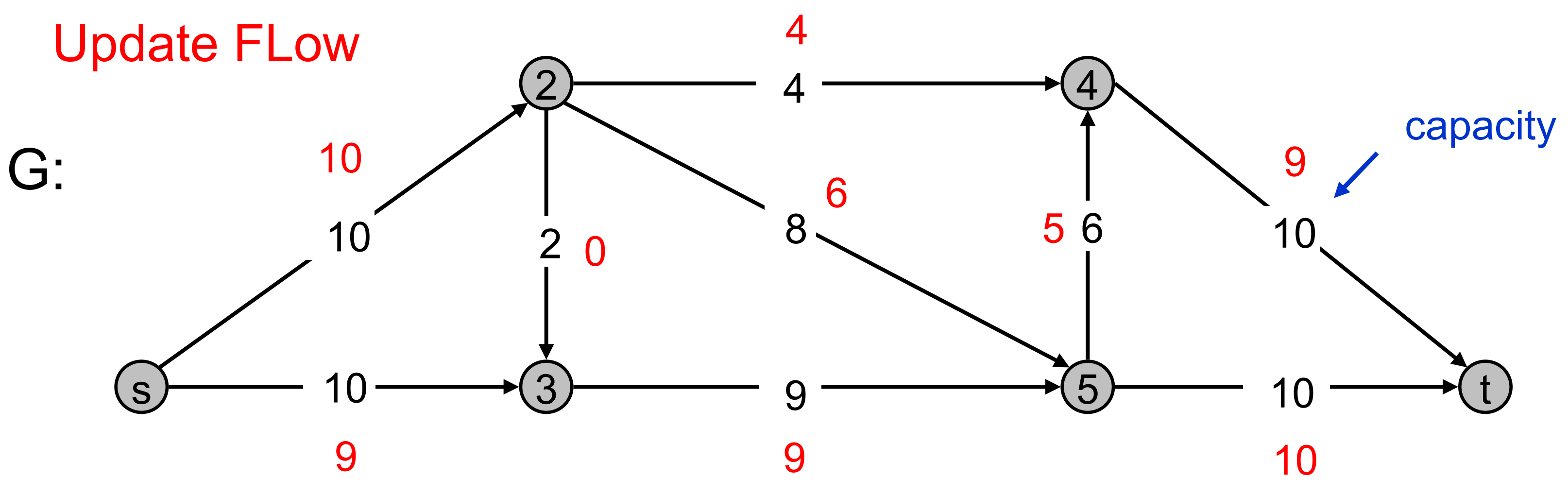
G:

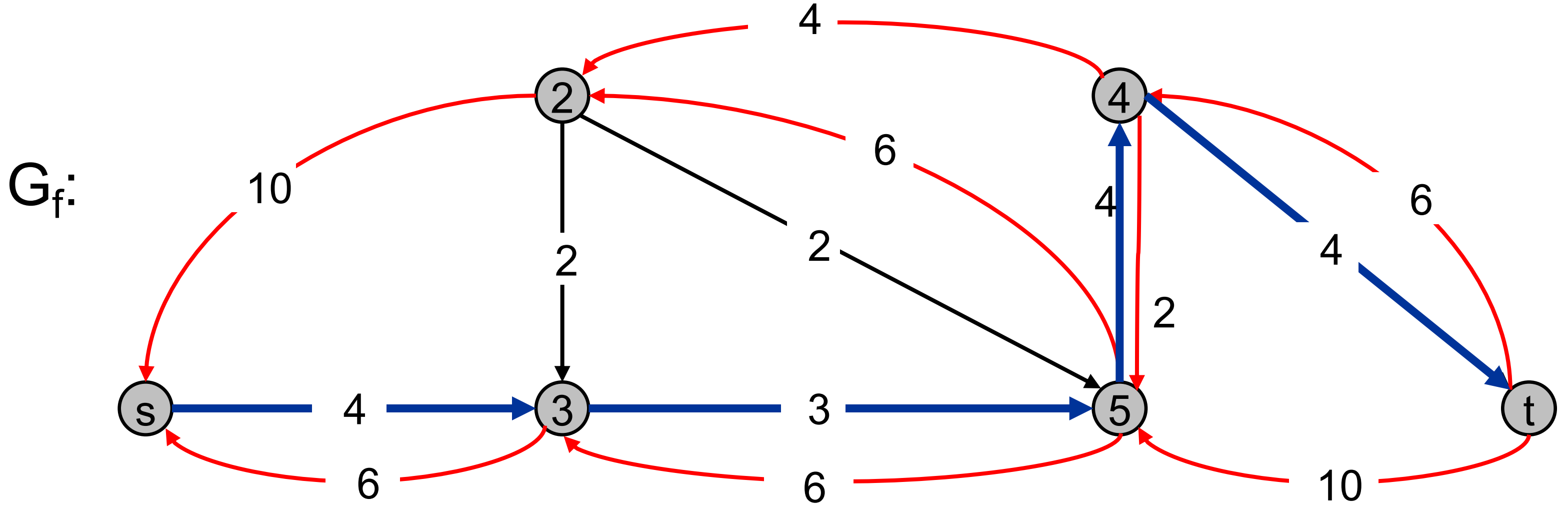


G_f:

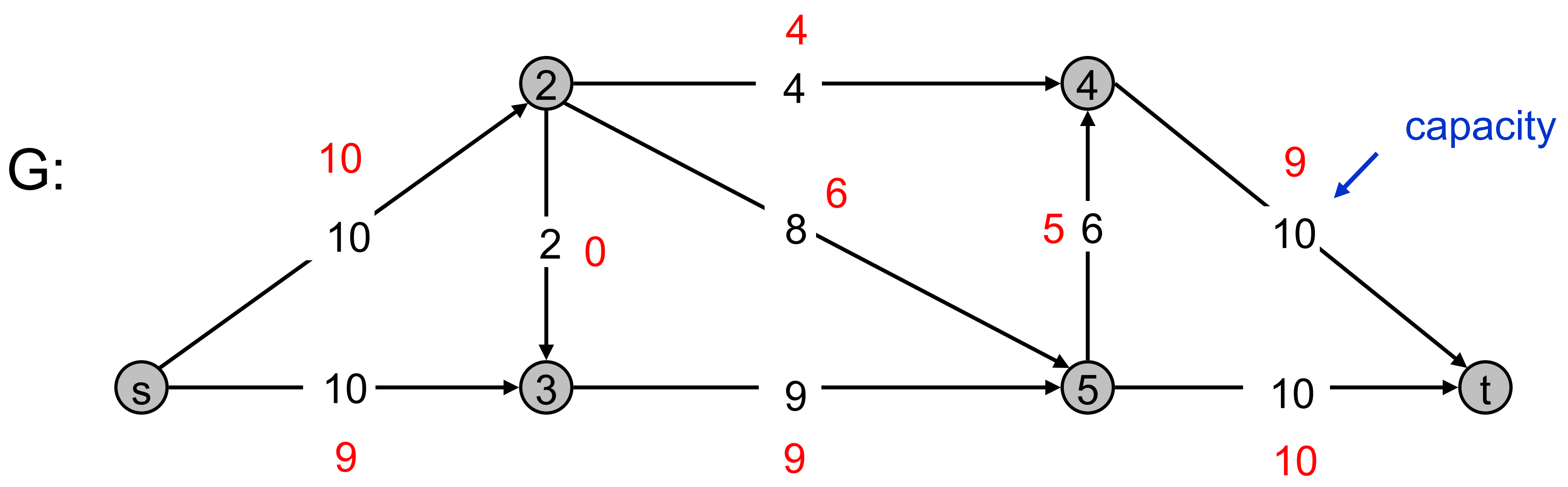


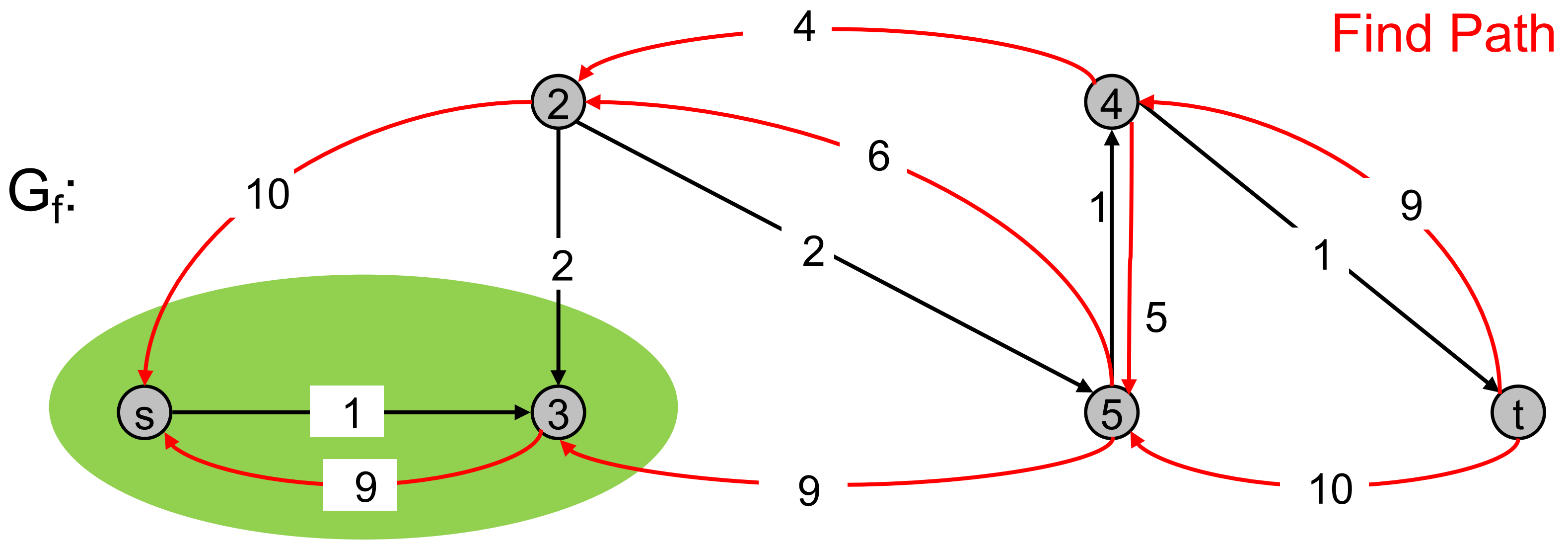
Ford-Fulkerson Alg: Greedy on G_f





Ford-Fulkerson Alg: Greedy on G_f





Augmenting Path Algorithm

```
Augment(f, c, P) {
   b ← bottleneck(P)
   foreach e ∈ P {
      if (e ∈ E) f(e) ← f(e) + b
                 c(e) ← c(e) - b
                 c(e^R) ← c(e^R) + b

      else       f(e^R) ← f(e^R) - b
                 c(e^R) ← c(e^R) + b
                 c(e) ← c(e) - b
   }
   return f
}
```

Smallest capacity edge on P

Forward edge

Reverse edge

```
Ford-Fulkerson(G, s, t, c) {
   foreach e ∈ E  f(e) ← 0. G_f is residual graph
   while (there exists augmenting path P) {
      f ← Augment(f, c, P)
}
   return f
}
```

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 s-t flow is equal to the value of the min s-t cut.

Proof strategy. We prove both simultaneously by showing the TFAE:

(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 that path.

Pf of Max Flow Min Cut Theorem

(iii) => (i)

No augmenting path for f => there is a cut (A,B): v(f)=cap(A,B)

- 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.

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

Running Time

Assumption. All capacities are integers between 1 and C.

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

Theorem. The algorithm terminates in at most $v(f^*) \leq nC$ iterations, if f^* is optimal flow.
Pf. Each augmentation increase value by at least 1.

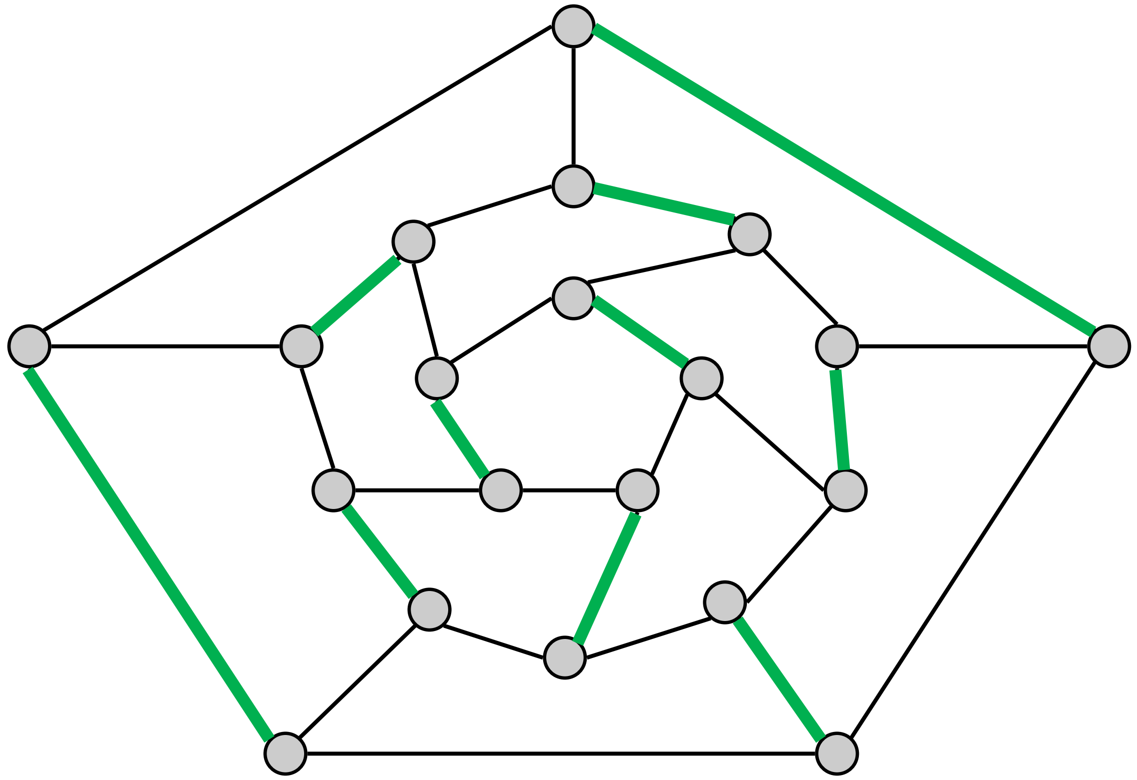
Corollary. If C = 1, Ford-Fulkerson runs in O(mn) time.

Integrality theorem. If all capacities are integers, then there exists a max flow f for which every flow value f(e) is an integer.
Pf. Since algorithm terminates, theorem follows from invariant.

Applications of Max Flow: Bipartite Matching

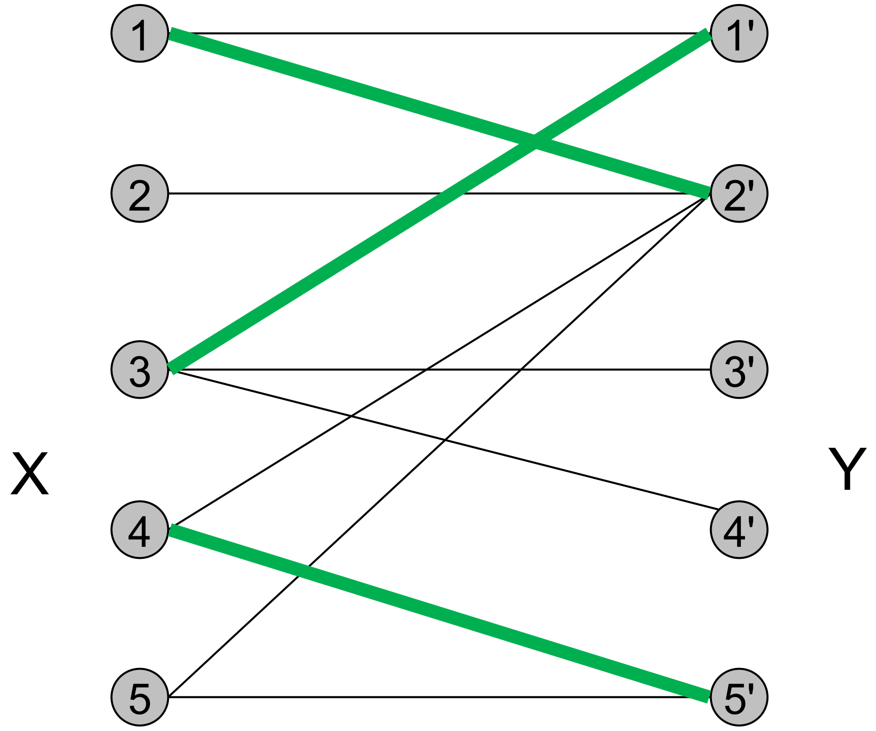
Maximum Matching Problem

Given an undirected graph G = (V, E).
A set $M \subseteq E$ is a matching if each node appears in at most one edge in M.
Goal: find a matching with largest cardinality.



Bipartite Matching Problem

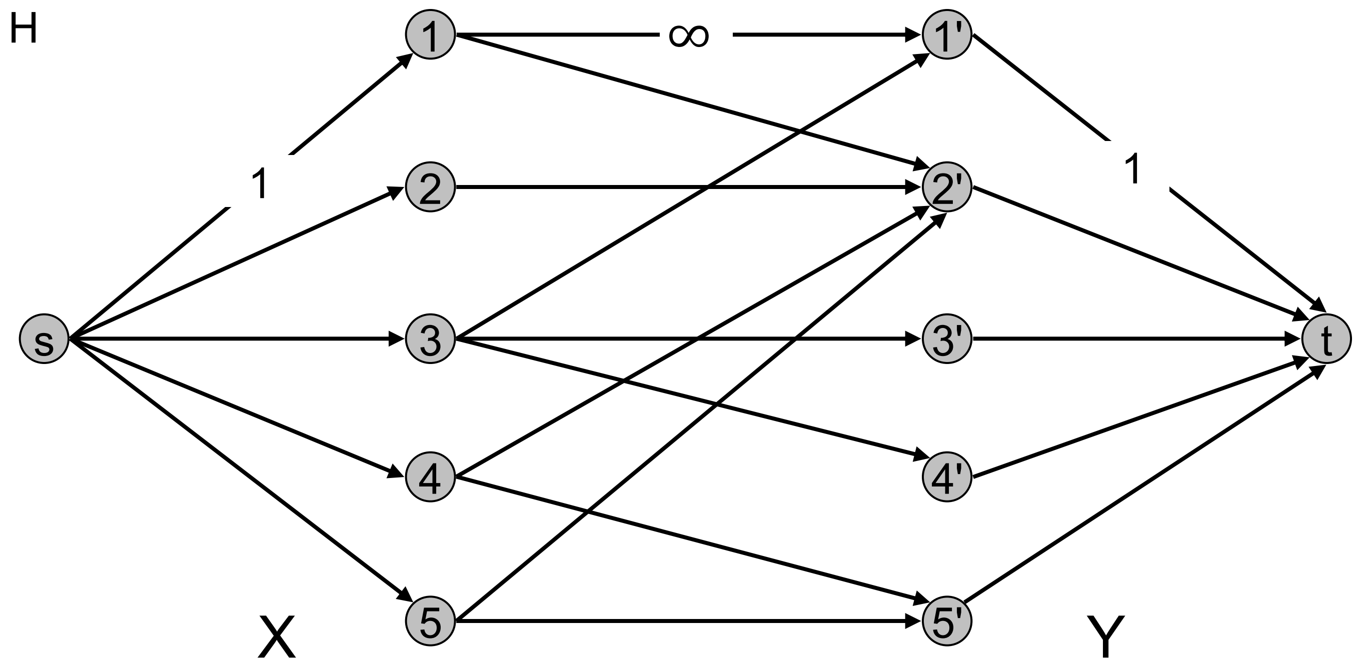
Given an undirected bibpartite graph $G = (X \cup Y, E)$
A set $M \subseteq E$ is a matching if each node appears in at most one edge in M.
Goal: find a matching with largest cardinality.



Bipartite Matching using Max Flow

Create digraph H as follows:

- Orient all edges from X to Y, and assign infinite (or unit) capacity.
- Add source s, and unit capacity edges from s to each node in L.
- Add sink t, and unit capacity edges from each node in R to t.



Bipartite Matching: Proof of Correctness

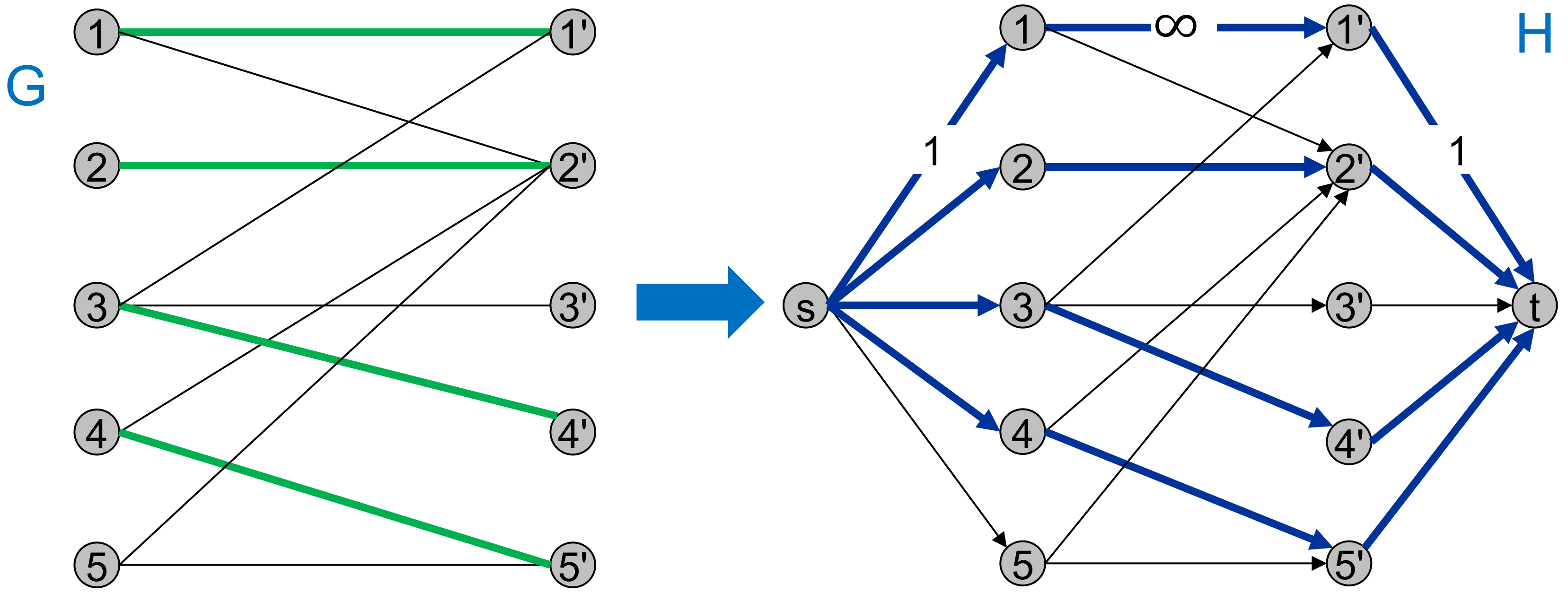
Thm. Max cardinality matching in G = value of max flow in H.

Pf. ≤

Given max matching M of cardinality k.

Consider flow f that sends 1 unit along each of k edges of M.

f is a flow, and has cardinality k. ▪



Bipartite Matching: Proof of Correctness

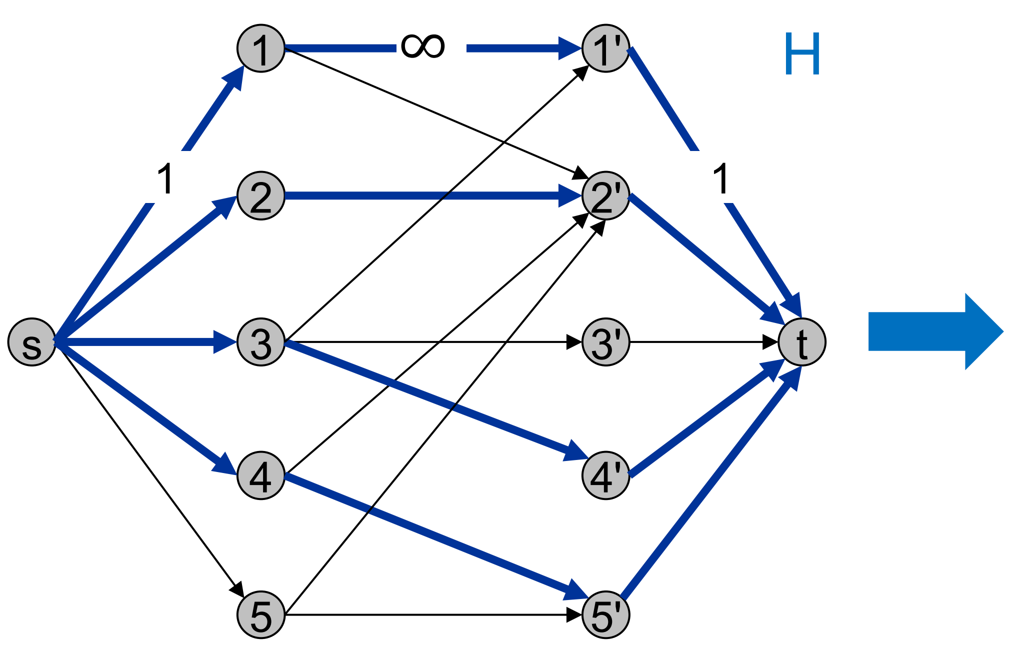
Thm. Max cardinality matching in G = value of max flow in H.

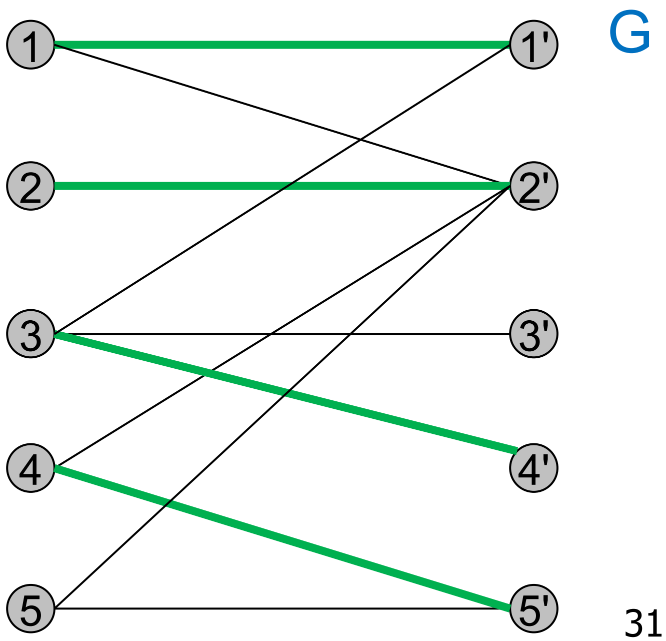
Pf. (of $\geq$) Let f be a max flow in H of value k.

Integrality theorem $\Rightarrow$ k is integral and we can assume f is 0-1.

Consider M = set of edges from X to Y with f(e) = 1.

- each node in X and Y participates in at most one edge in M
- |M| = k: consider s-t cut $(s \cup X, t \cup Y)$





Perfect Bipartite Matching

Perfect Bipartite Matching

Def. A matching $M \subseteq E$ is perfect if each node appears in exactly one edge in M.

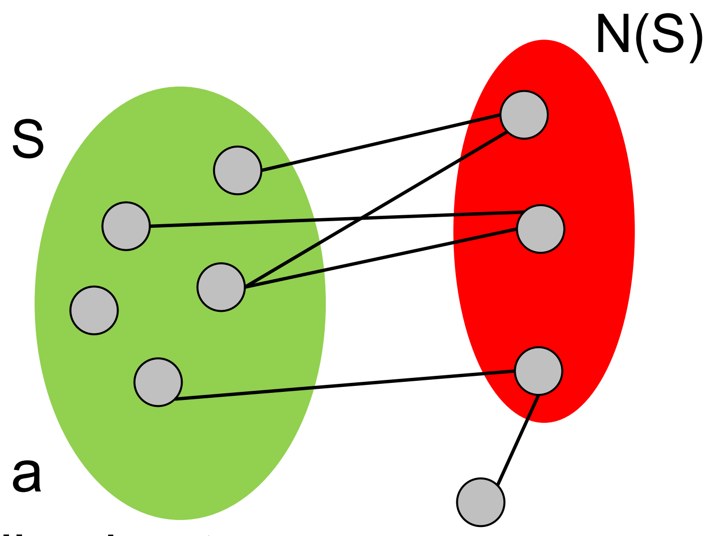
Q. When does a bipartite graph have a perfect matching?

Structure of bipartite graphs with perfect matchings:

- Clearly we must have $|X| = |Y|$.
- What other conditions are necessary?
- What conditions are sufficient?

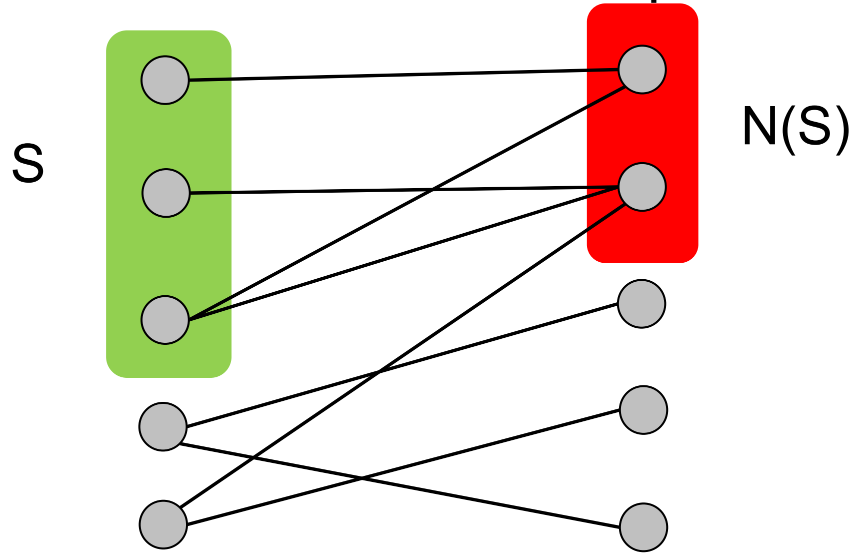
Perfect Bipartite Matching: N(S)

Def. Let S be a subset of nodes, and let N(S) be the set of nodes adjacent to nodes in S.



Observation. If a bipartite graph G has a perfect matching, then |N(S)| ≥ |S| for all subsets $S \subseteq X$.

Pf. Each $v \in S$ has to be matched to a unique node in N(S).



Marriage Theorem

Thm: [Frobenius 1917, Hall 1935] Let $G = (X \cup Y, E)$ be a bipartite graph with |X| = |Y|.

Then, G has a perfect matching iff $|N(S)| \geq |S|$ for all subsets $S \subseteq X$.

Pf. $\Rightarrow$

This was the previous observation.

If $|N(S)| < |S|$ for some S, then there is no perfect matching.

Marriage Theorem

Pf. $\exists S \subseteq X$ s.t., $|N(S)| < |S| \Leftarrow$ G does not a perfect matching

Formulate as a max-flow and let (A, B) be the min s-t cut

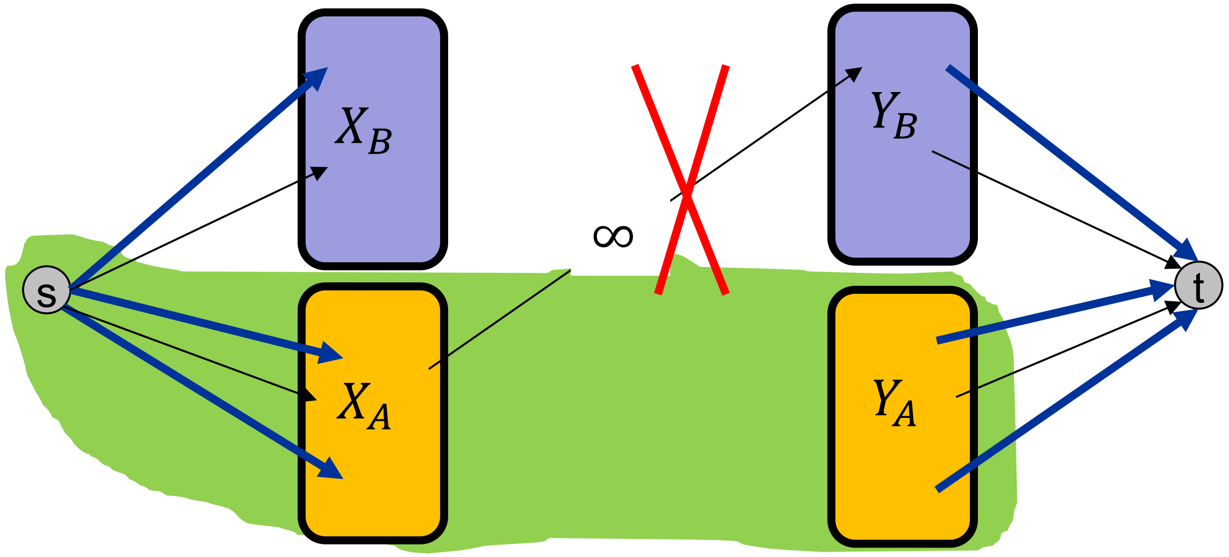
G has no perfect matching => $v(f^*) < |X|$. So, $cap(A, B) < |X|$

Define $X_A = X \cap A, X_B = X \cap B, Y_A = Y \cap A$

Then, $cap(A, B) = |X_B| + |Y_A|$

Since min-cut does not use ∞ edges, $N(X_A) \subseteq Y_A$

$|N(X_A)| \leq |Y_A| = cap(A, B) - |X_B| = cap(A, B) - |X| + |X_A| < |X_A|$



Bipartite Matching Running Time

Which max flow algorithm to use for bipartite matching?

- Generic augmenting path: $O(m\ val(f^*)) = O(mn)$.
- Capacity scaling: $O(m^2 \log C) = O(m^2)$.
- Shortest augmenting path: $O(m\ n^{1/2})$.

Non-bipartite matching.

- Structure of non-bipartite graphs is more complicated, but well-understood. [Tutte-Berge, Edmonds-Galai]
- Blossom algorithm: $O(n^4)$. [Edmonds 1965]
- Best known: $O(m\ n^{1/2})$. [Micali-Vazirani 1980]

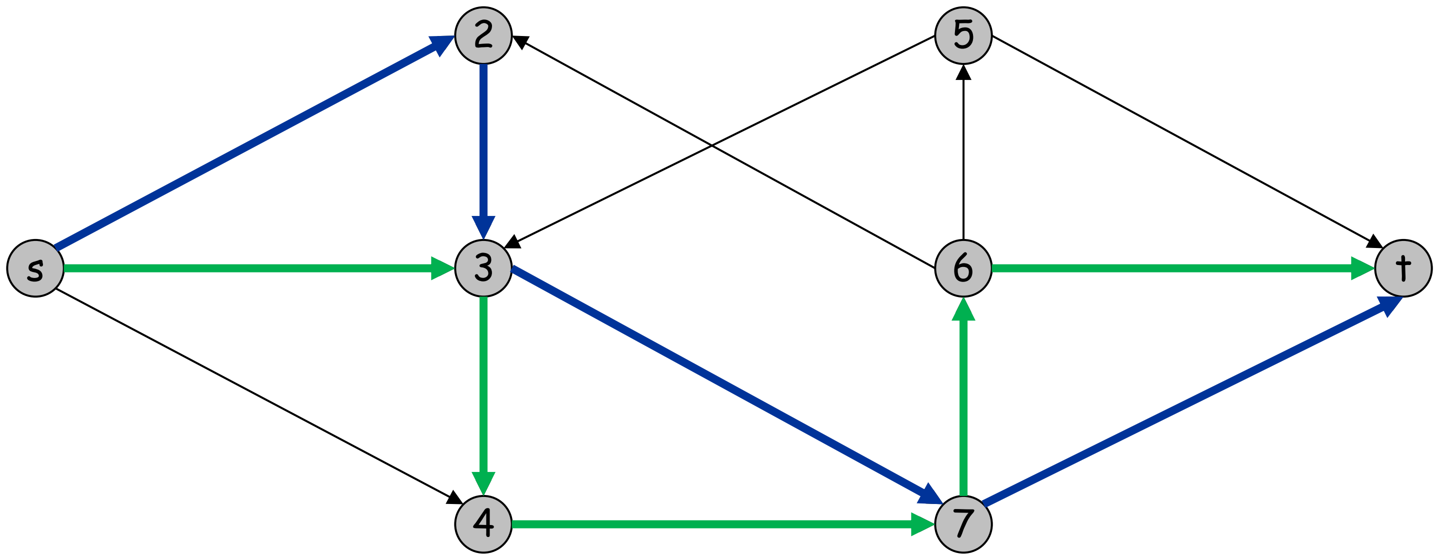
Edge Disjoint Paths

Edge Disjoint Paths Problem

Given a digraph G = (V, E) and two nodes s and t, find the max number of edge-disjoint s-t paths.

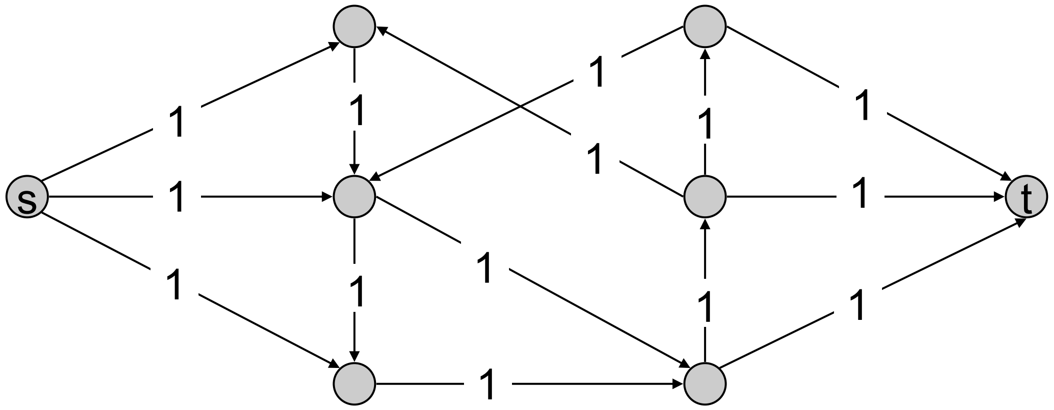
Def. Two paths are edge-disjoint if they have no edge in common.

Ex: communication networks.



Max Flow Formulation

Assign a unit capacitary to every edge. Find Max flow from s to t.



Thm. Max number edge-disjoint s-t paths equals max flow value.

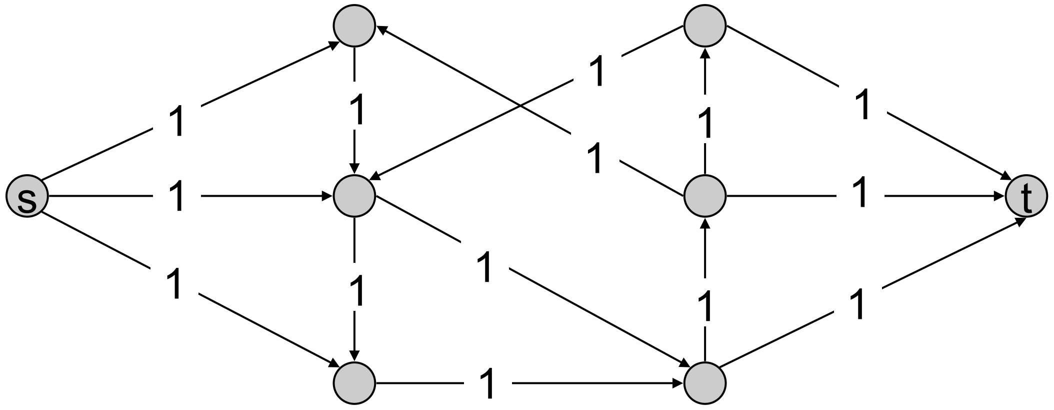
Pf. $\leq$

Suppose there are k edge-disjoint paths $P_1, \ldots, P_k$.

Set f(e) = 1 if e participates in some path P_i ; else set f(e) = 0.

Since paths are edge-disjoint, f is a flow of value k. ▪

Max Flow Formulation



Thm. Max number edge-disjoint s-t paths equals max flow value.

Pf. $\geq$ Suppose max flow value is k

Integrality theorem $\Rightarrow$ there exists 0-1 flow f of value k.

Consider edge (s, u) with f(s, u) = 1.

- by conservation, there exists an edge (u, v) with f(u, v) = 1
- continue until reach t, always choosing a new edge

This produces k (not necessarily simple) edge-disjoint paths. ▪

We can return to u so we can have cycles. But we can eliminate cycles if desired