CSE 421

Bellman-Ford ALG, Network Flows

Shayan Oveis Gharan

DP for Shortest Path

Def: Let $OPT(v, i)$ be the length of the shortest s - v path with at most i edges.

$$OPT(v,i) = \begin{cases} 0 & \text{if } v = s \\ \infty & \text{if } v \neq s, i = 0 \\ \min\left(OPT(v, i-1), \min_{u:(u,v) \text{ an edge}} OPT(u, i-1) + c_{u,v}\right) \end{cases}$$

So, for every v, $OPT(v, ?)$ is the shortest path from s to v.

But how long do we have to run?

Since G has no negative cycle, it has at most $n - 1$ edges. So, $OPT(v, n-1)$ is the answer.

Bellman Ford Algorithm

```
for v=1 to n
   if v ≠ s then
      M[v,0]=∞
M[s,0]=0.

for i=1 to n-1
   for v=1 to n
      M[v,i]=M[v,i-1]
      for every edge (u,v)
         M[v,i]=min(M[v,i], M[u,i-1]+c_{u,v})
```

Running Time: $O(nm)$

Can we test if G has negative cycles?

Bellman Ford Algorithm

```
for v=1 to n
   if v ≠ s then
      M[v,0]=∞
M[s,0]=0.

for i=1 to n-1
   for v=1 to n
      M[v,i]=M[v,i-1]
      for every edge (u,v)
         M[v,i]=min(M[v,i], M[u,i-1]+c_{u,v})
```

Running Time: $O(nm)$

Can we test if G has negative cycles?

Yes, run for i=1…2n and see if the M[v,n-1] is different from M[v,2n]

DP Techniques Summary

Recipe:

- Follow the natural induction proof.
- Find out additional assumptions/variables/subproblems that you need to do the induction
- Strengthen the hypothesis and define w.r.t. new subproblems

Dynamic programming techniques.

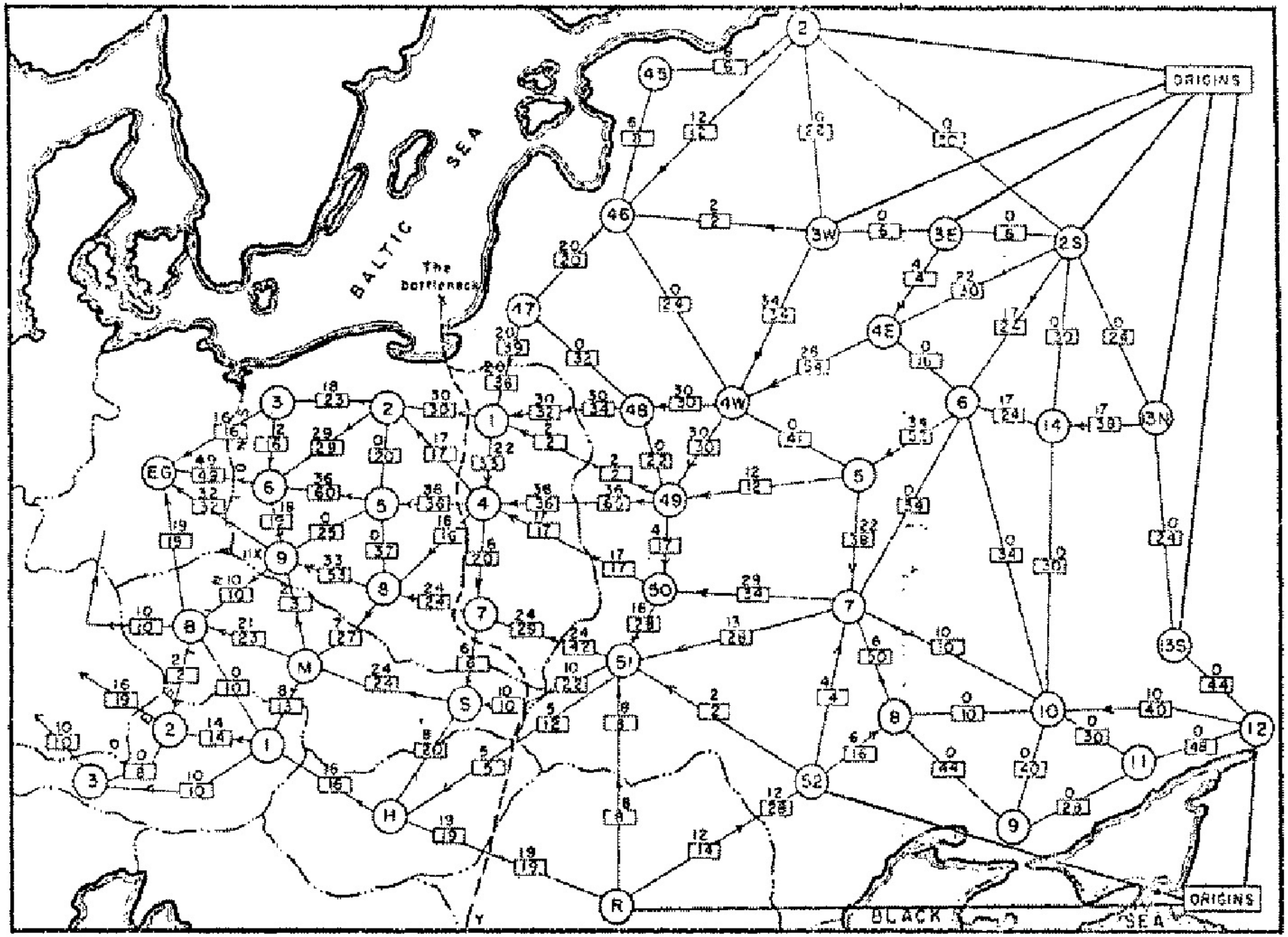
- Whenever a problem is a special case of an NP-hard problem an ordering is important:
- Adding a new variable: knapsack.
- Dynamic programming over intervals: RNA secondary structure.

Top-down vs. bottom-up:

- Different people have different intuitions
- Bottom-up is useful to optimize the memory

Network Flows

Soviet Rail Network



Reference: *On the history of the transportation and maximum flow problems*. Alexander Schrijver in Math Programming, 91: 3, 2002.

Network Flow Applications

Max flow and min cut.

- Two very rich algorithmic problems.
- Cornerstone problems in combinatorial optimization.
- Beautiful mathematical duality.

Nontrivial applications / reductions.

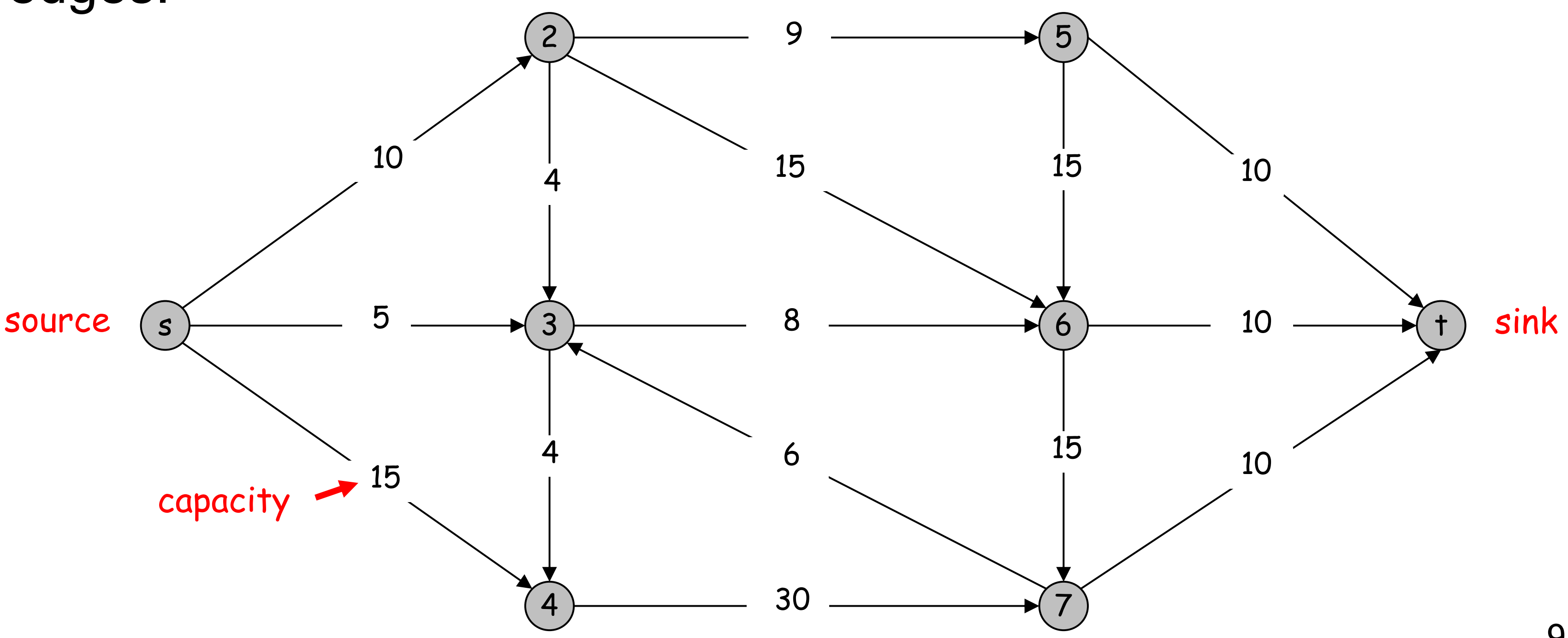
- Data mining.
- Open-pit mining.
- Project selection.
- Airline scheduling.
- Bipartite matching.
- Baseball elimination.
- Image segmentation.
- Network connectivity.

Minimum s-t Cut Problem

Given a directed graph G = (V, E) = directed graph and two distinguished nodes: s = source, t = sink.

Suppose each directed edge e has a nonnegative capacity $c(e)$

Goal: Find a cut separating s, t that cuts the minimum capacity of edges.

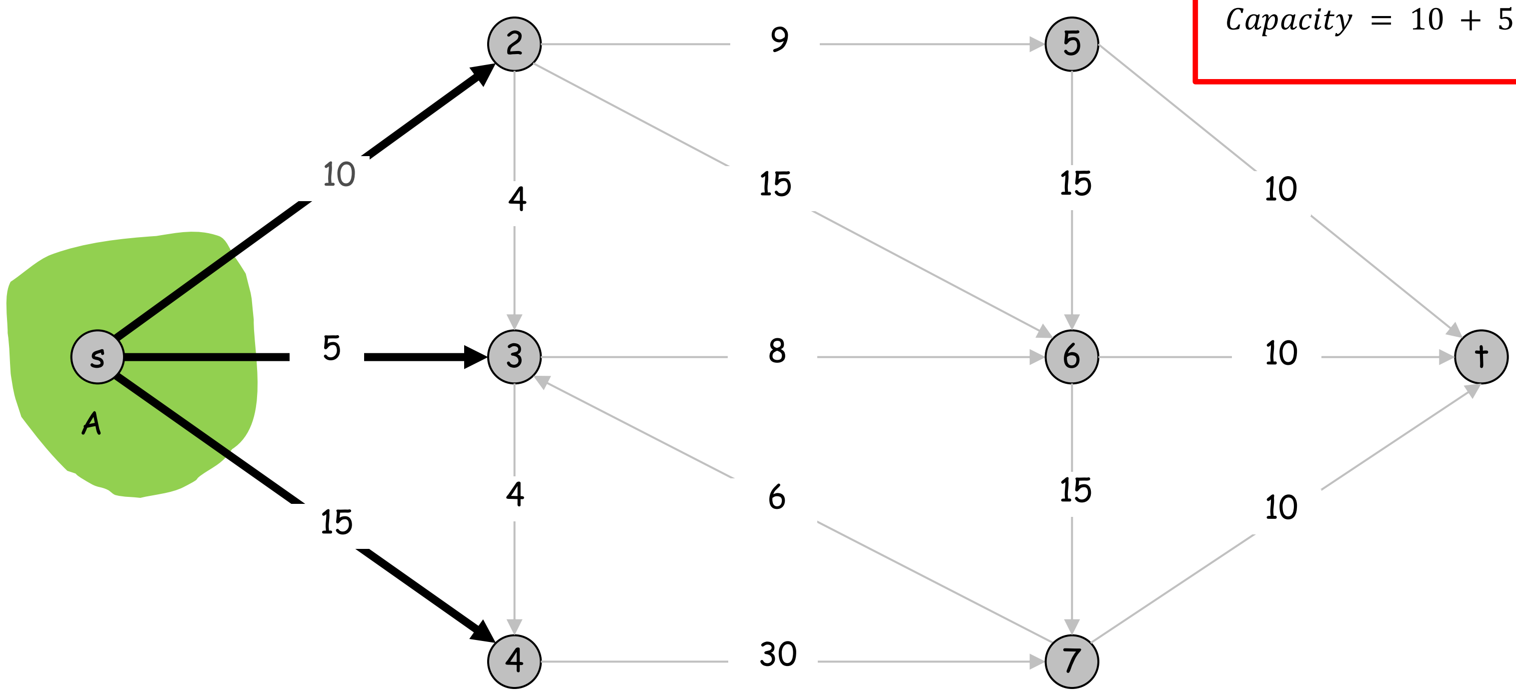


s-t cuts

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): $cap(A,B) = \sum_{e \text{ out of } A} c(e)$

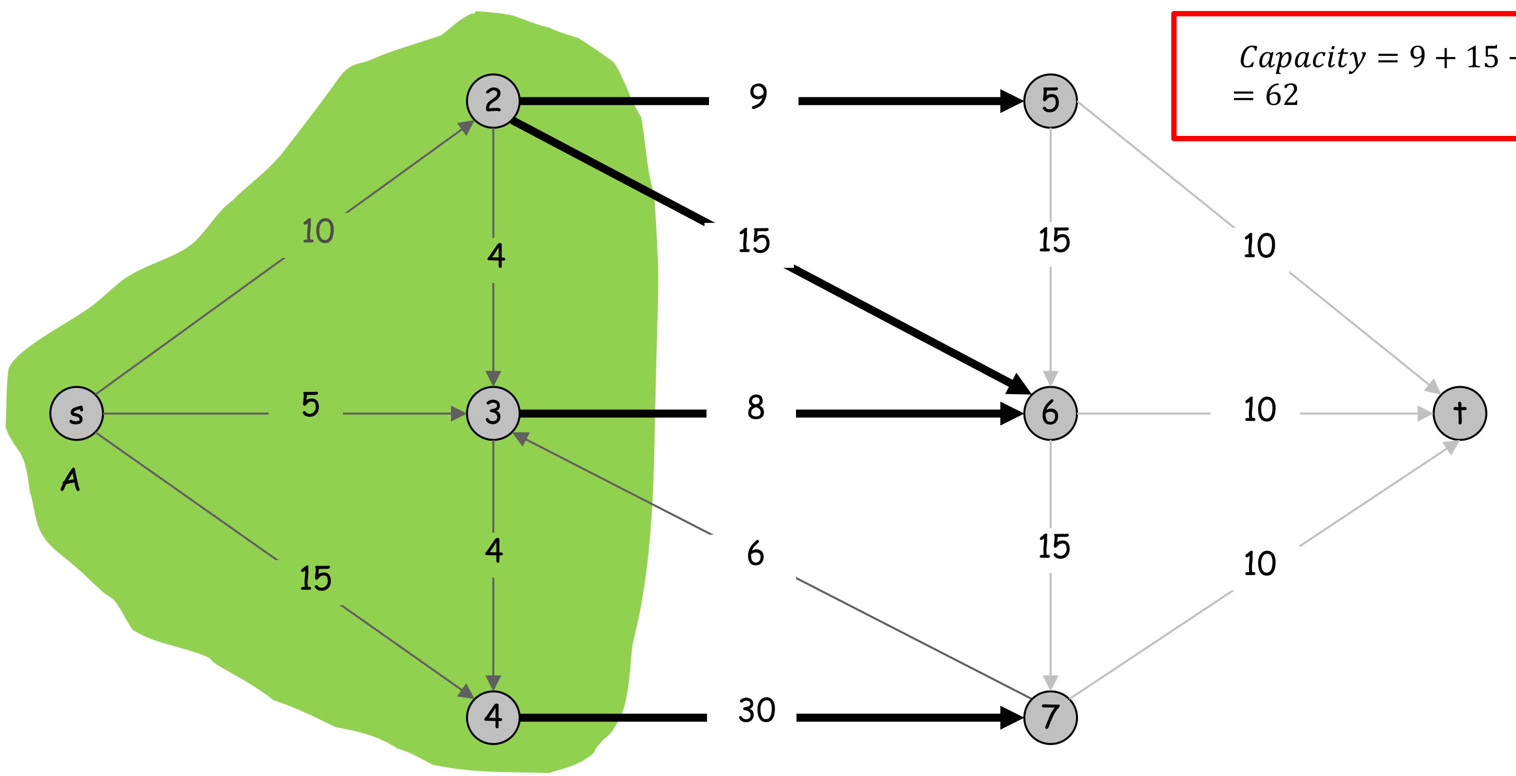
$Capacity = 10 + 5 + 15 = 30$



s-t 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): $cap(A,B) = \sum_{(u,v):u \in A, v \in B} c(u,v)$



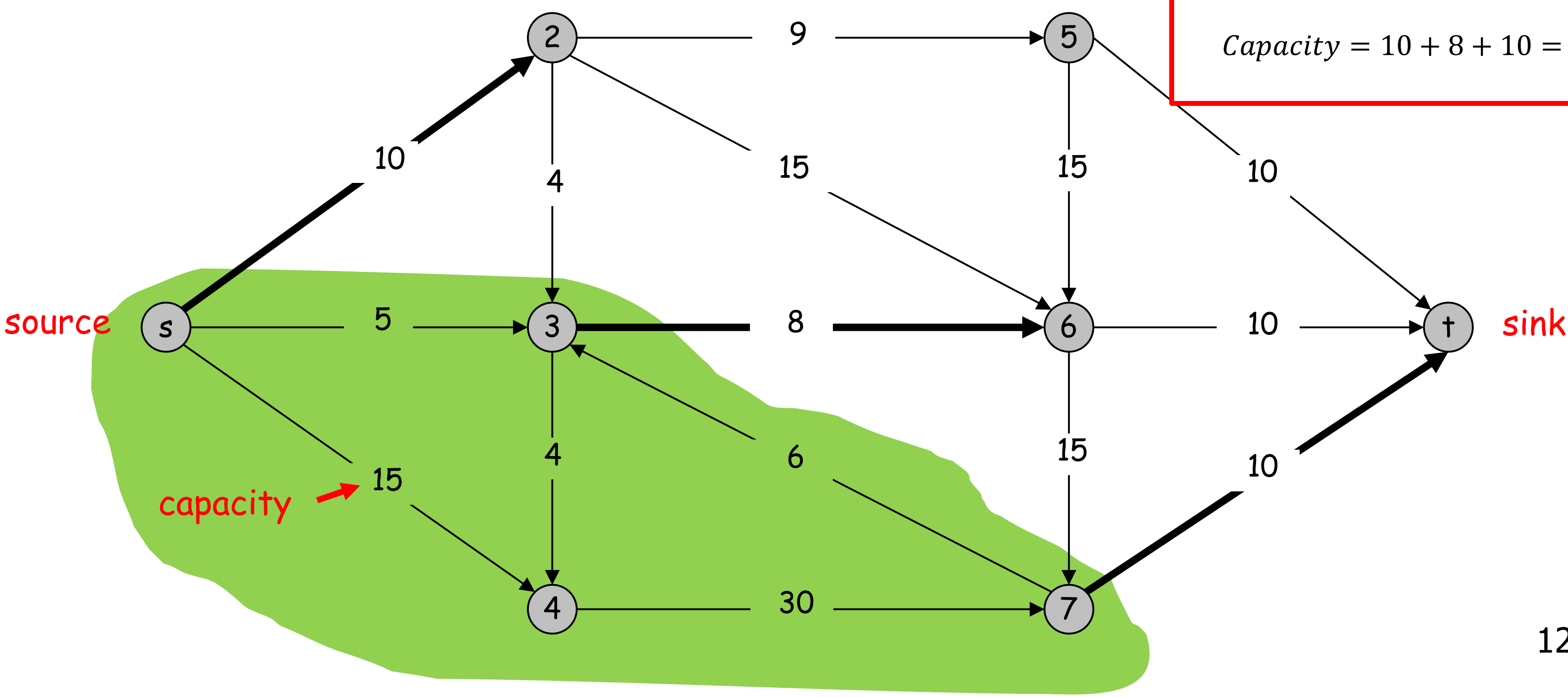
$Capacity = 9 + 15 + 8 + 30$
$= 62$

Minimum s-t Cut Problem

Given a directed graph G = (V, E) = directed graph and two distinguished nodes: s = source, t = sink.

Suppose each directed edge e has a nonnegative capacity $c(e)$

Goal: Find a s-t cut of minimum capacity



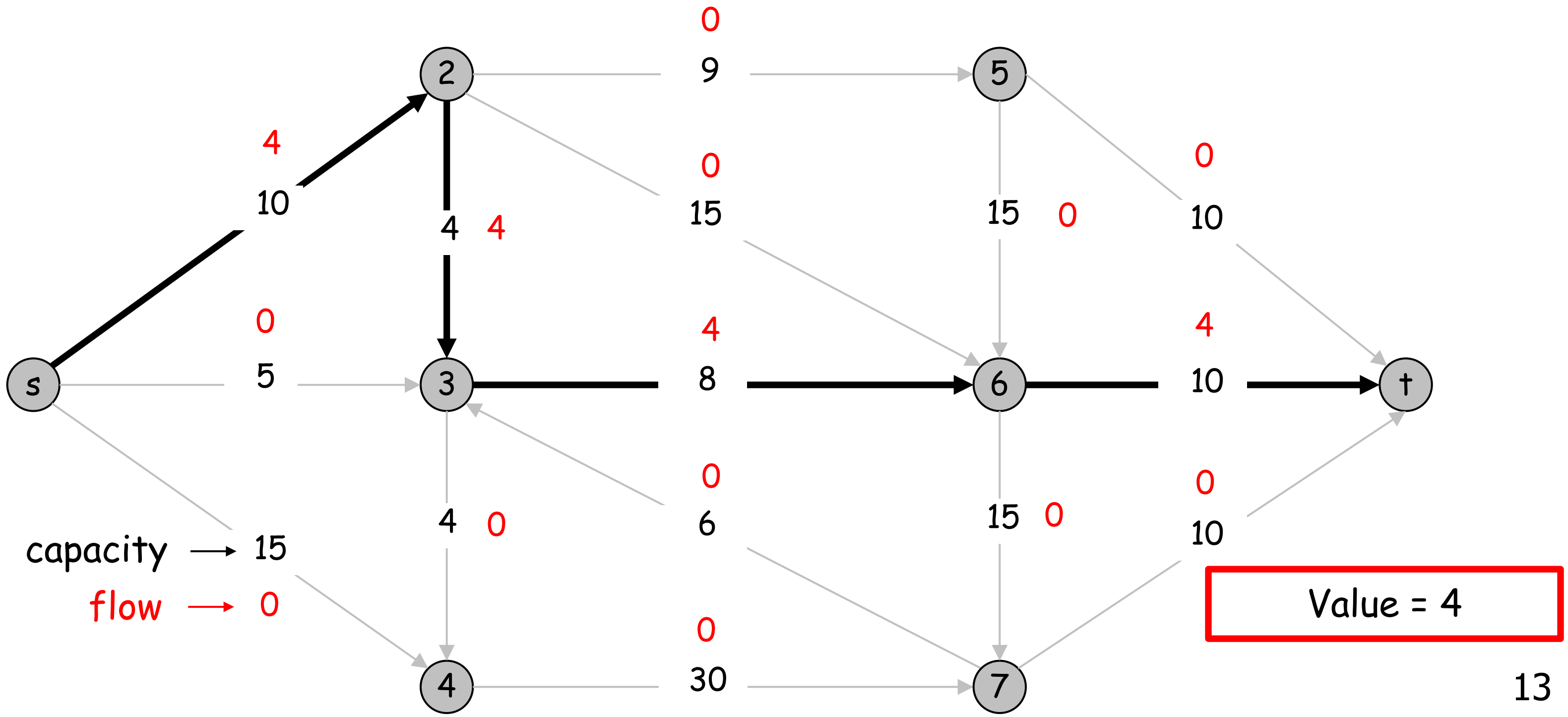
$Capacity = 10 + 8 + 10 = 28$

s-t Flows

Def. An s-t flow is a function that satisfies:

- For each $e \in E$: $0 \le f(e) \le c(e)$ (capacity)
- For each $v \in V - \{s, t\}$: $\sum_{e \text{ in to } v} f(e) = \sum_{e \text{ out of } v} f(e)$ (conservation)

Def. The value of a flow f is: $v(f) = \sum_{e \text{ out of } s} f(e)$

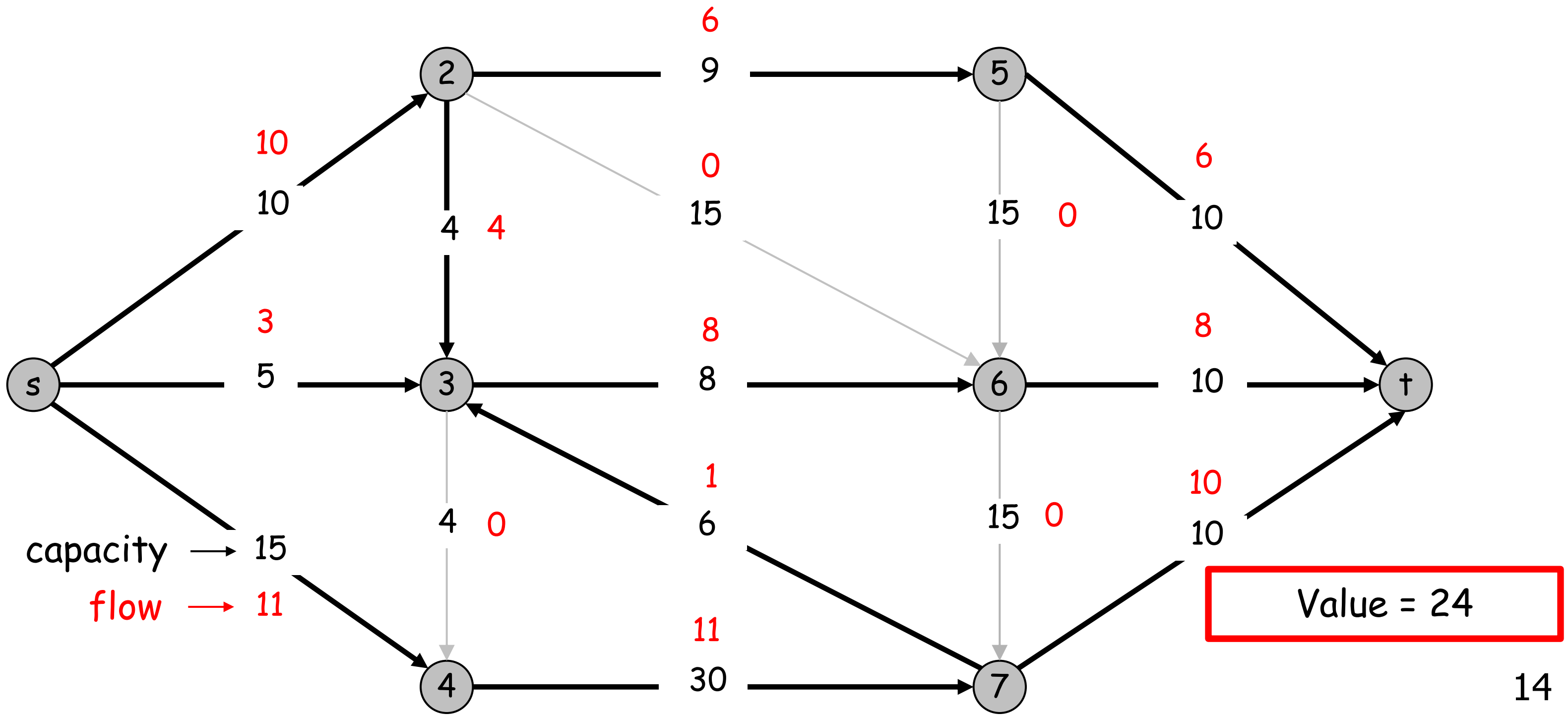


s-t Flows

Def. An s-t flow is a function that satisfies:

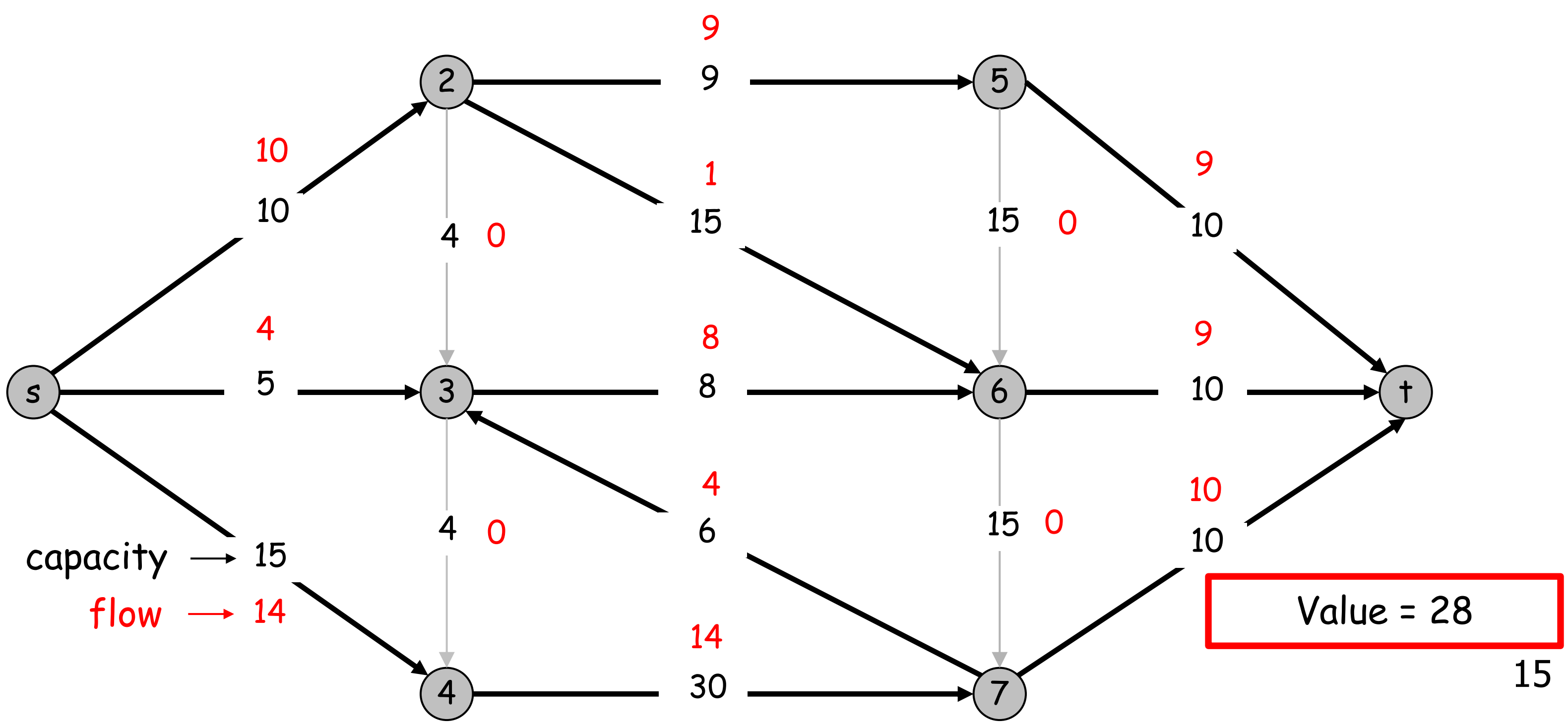
- For each $e \in E$: $0 \le f(e) \le c(e)$ (capacity)
- For each $v \in V - \{s, t\}$: $\sum_{e \text{ in to } v} f(e) = \sum_{e \text{ out of } v} f(e)$ (conservation)

Def. The value of a flow f is: $v(f) = \sum_{e \text{ out of } s} f(e)$



Maximum s-t Flow Problem

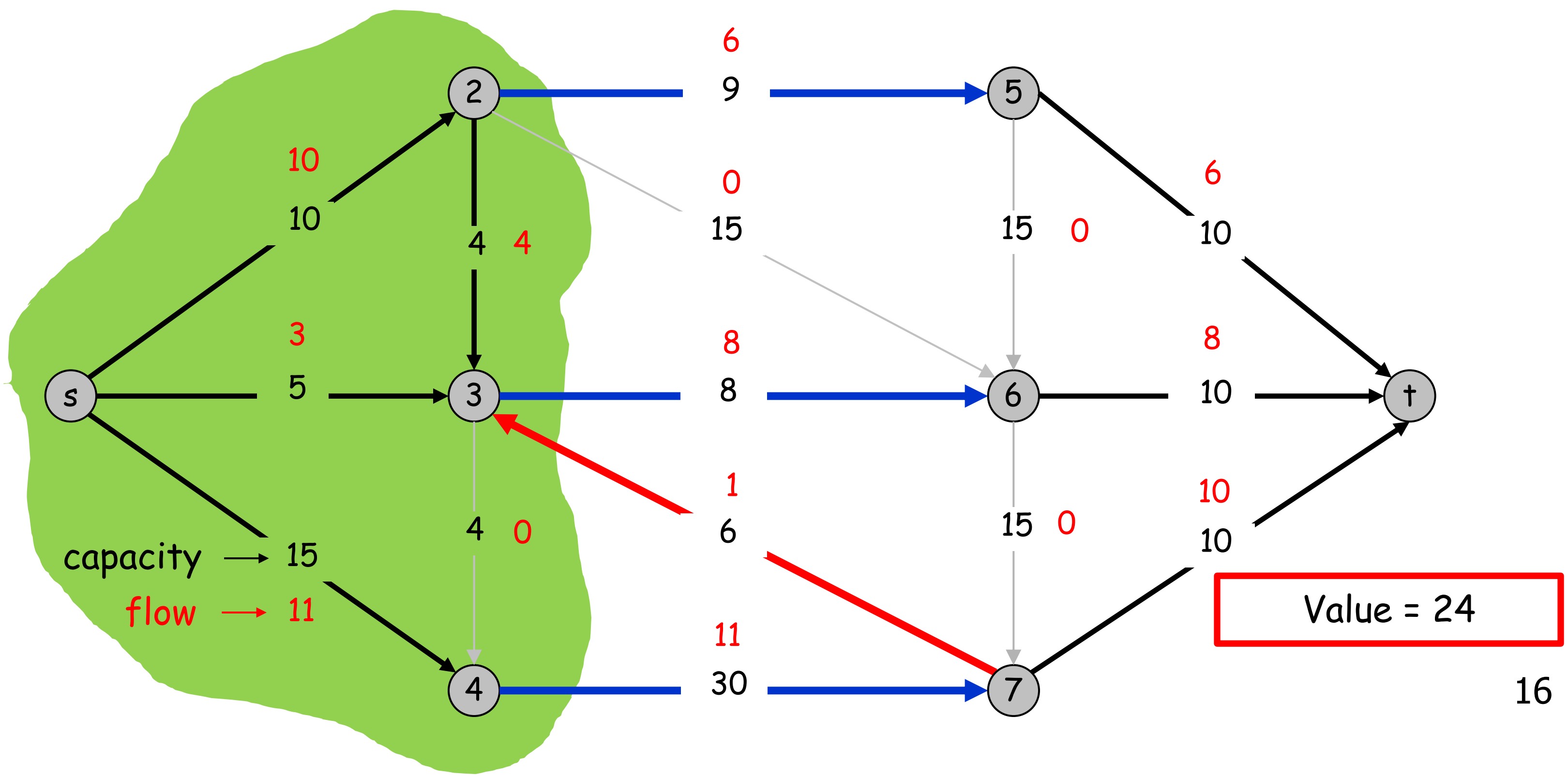
Goal: Find a s-t flow of largest value.



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.

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



Pf of Flow value Lemma

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.

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

Pf.

$$v(f) = \sum_{e \text{ out of } s} f(e)$$

By conservation of flow, all terms except v=s are0

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

All contributions due to internal edges cancel out

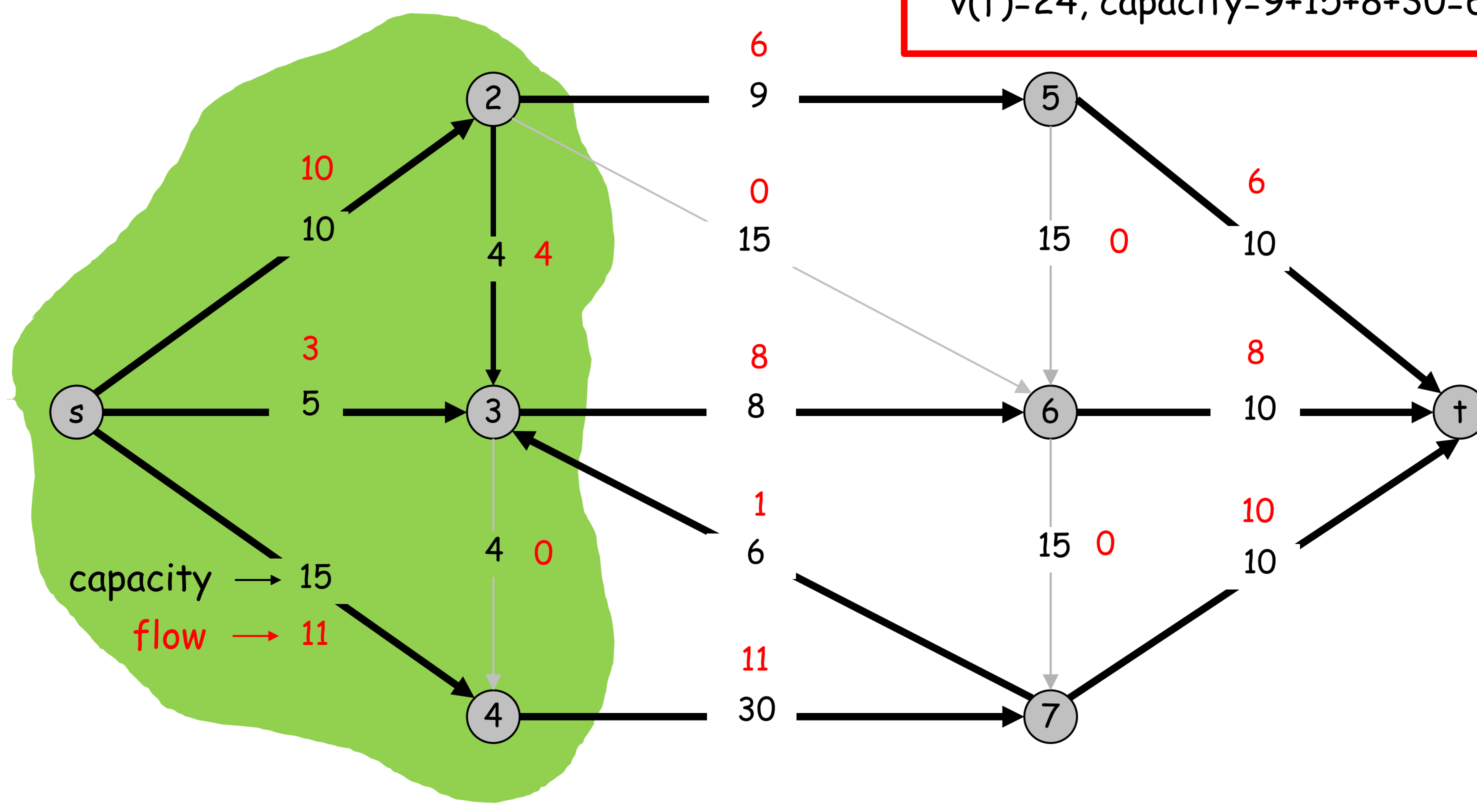
$$= \sum_{e \text{ out of } A} f(e) - \sum_{e \text{ in to } A} f(e)$$

Weak Duality of Flows and Cuts

Cut Capacity lemma. 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.

$$v(f) \leq cap(A, B)$$

v(f)=24, capacity=9+15+8+30=62



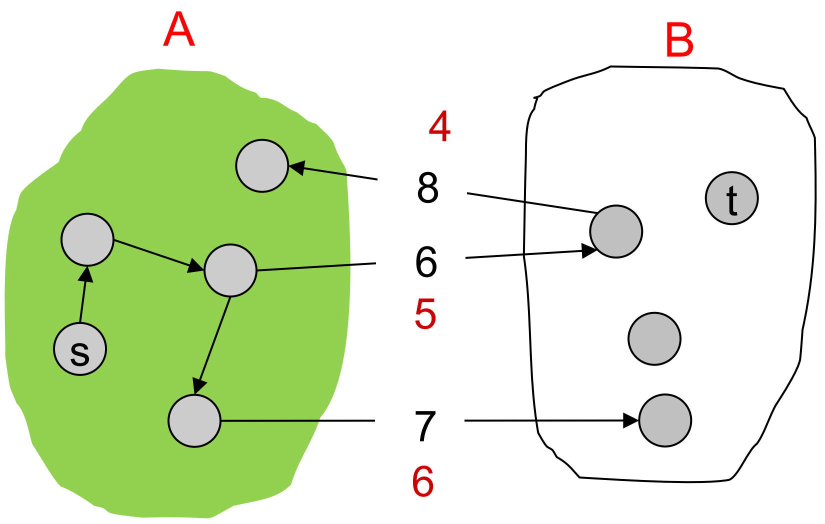
Weak Duality of Flows and Cuts

Cut capacity lemma. 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.

$$v(f) \leq cap(A, B)$$

Pf.

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



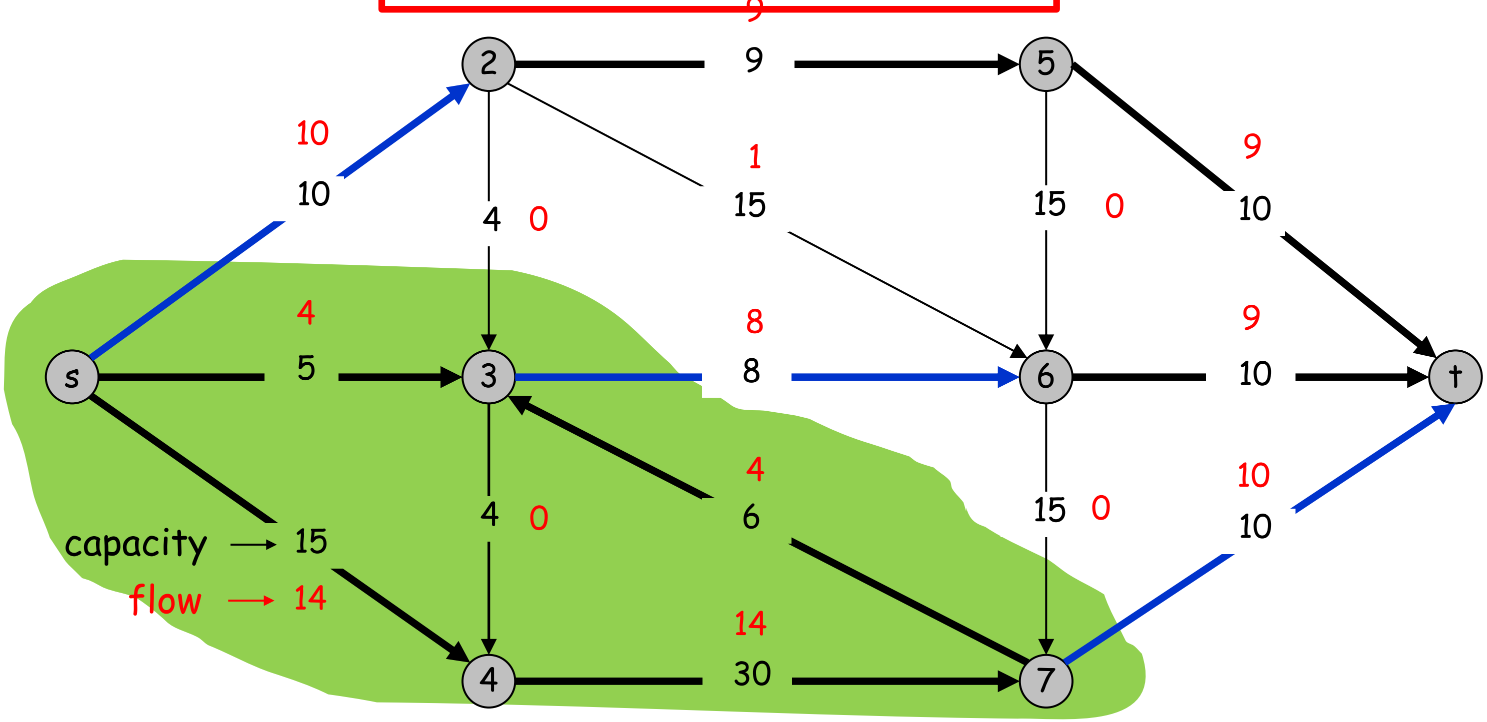
Certificate of Optimality

Corollary: Suppose there is a s-t cut (A,B) such that

$$v(f) = cap(A, B)$$

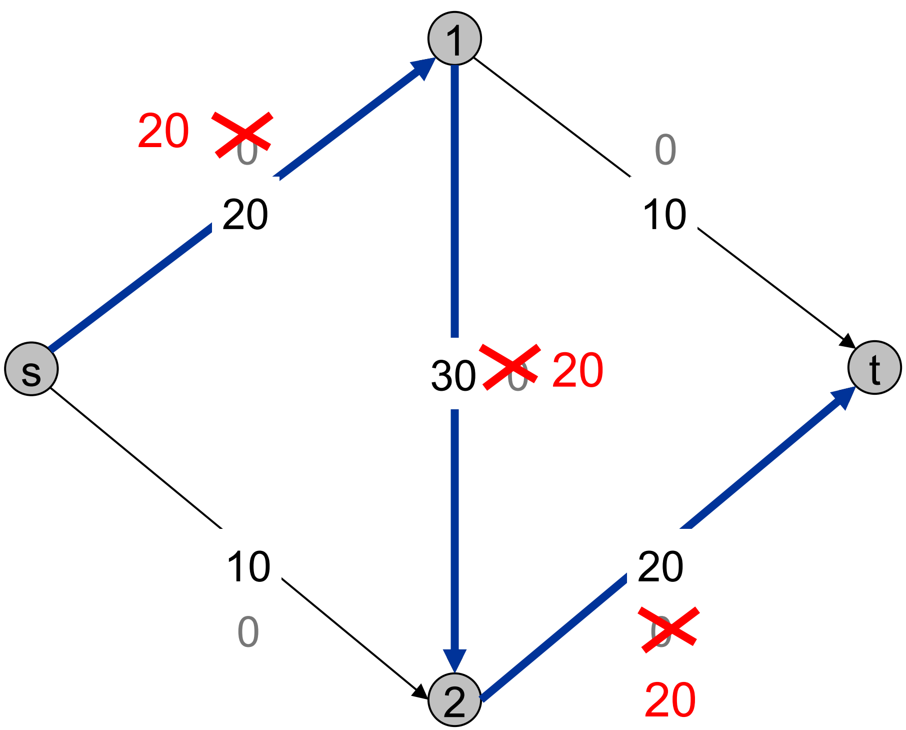
Then, f is a maximum flow and (A,B) is a minimum cut.

v(f)=28, cap(A,B)=28



A Greedy Algorithm for Max Flow

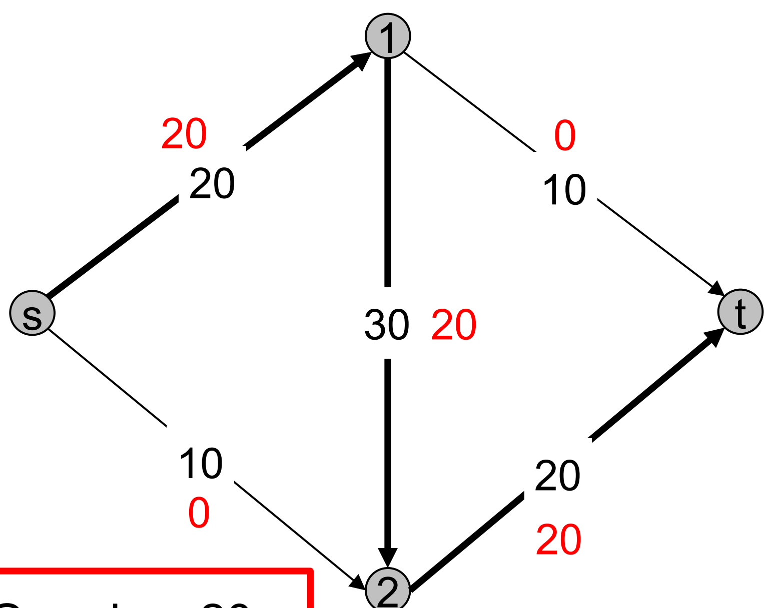
- Start with f(e) = 0 for all edge e $\in$ E.
- Find an s-t path P where each edge has f(e) < c(e).
- Augment flow along path P.
- Repeat until you get stuck.



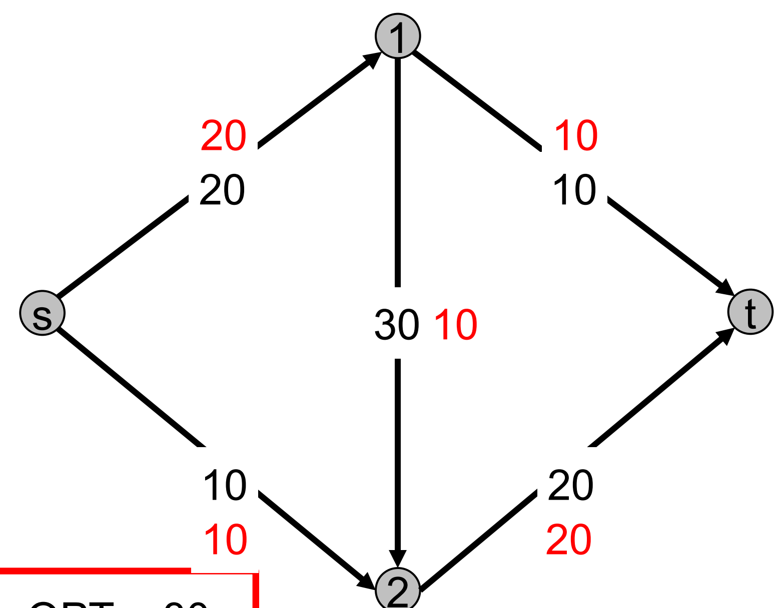
A Greedy Algorithm for Max Flow

- Start with f(e) = 0 for all edge e $\in$ E.
- Find an s-t path P where each edge has f(e) < c(e).
- Augment flow along path P.
- Repeat until you get stuck.

Local Optimum $\neq$ Global Optimum

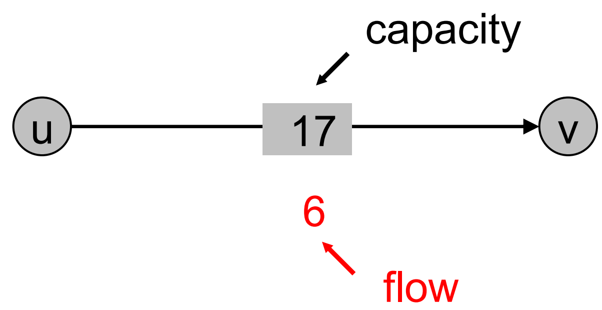


Greedy = 20



OPT = 30

Residual Graph



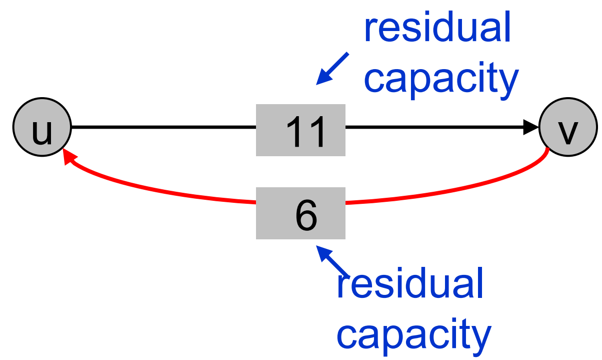
Original edge: e = (u, v) $\in$ E.

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

Residual edge.

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

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

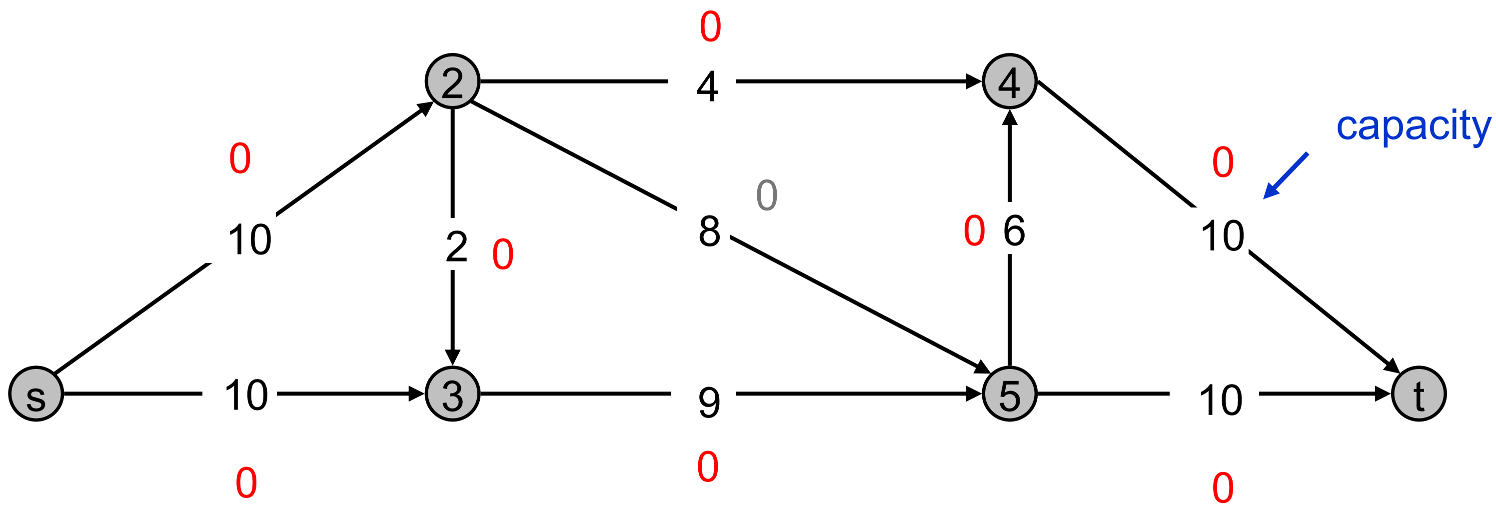


Residual graph: $G_f = (V, E_f)$.

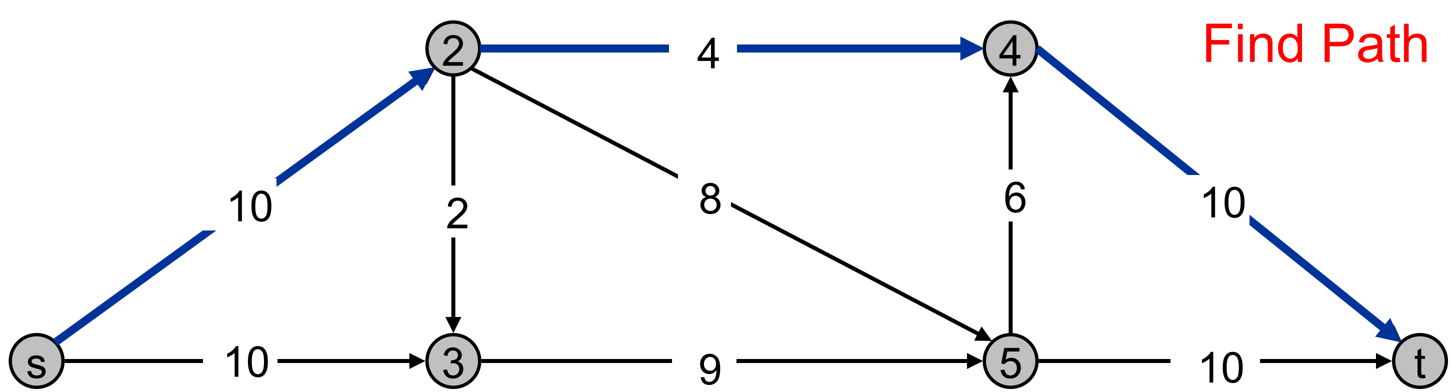
- Residual edges with positive residual capacity.
- $E_f = \{e : f(e) < c(e)\} \cup \{e : f(e^R) > 0\}$.

Ford-Fulkerson Alg: Greedy on G_f

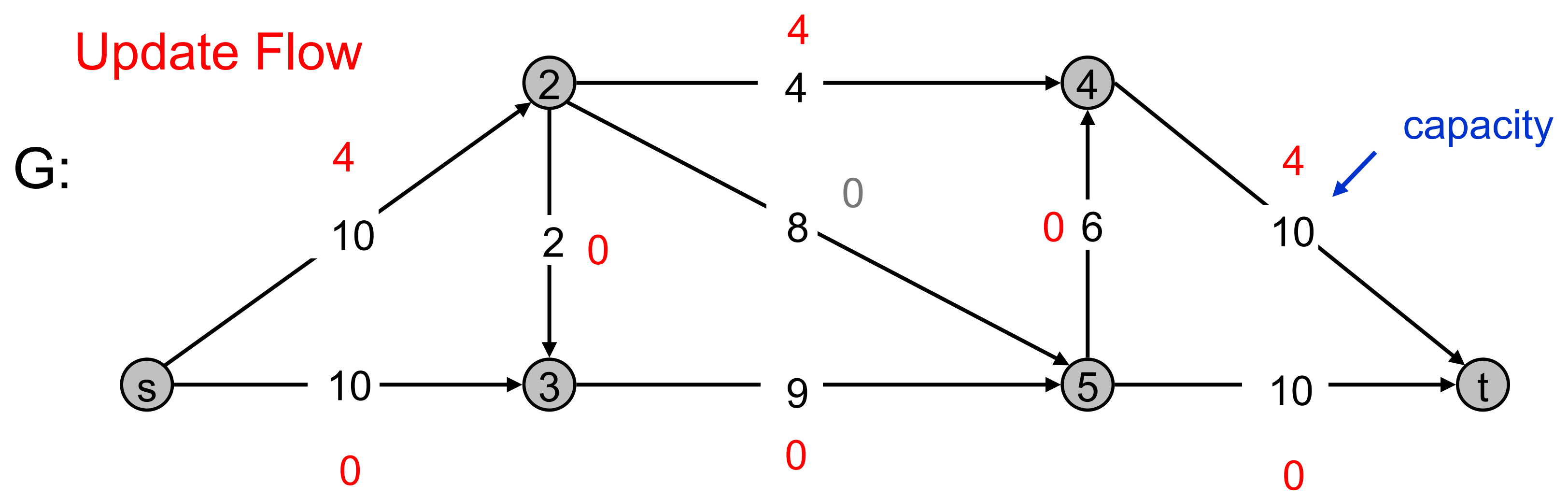
G:

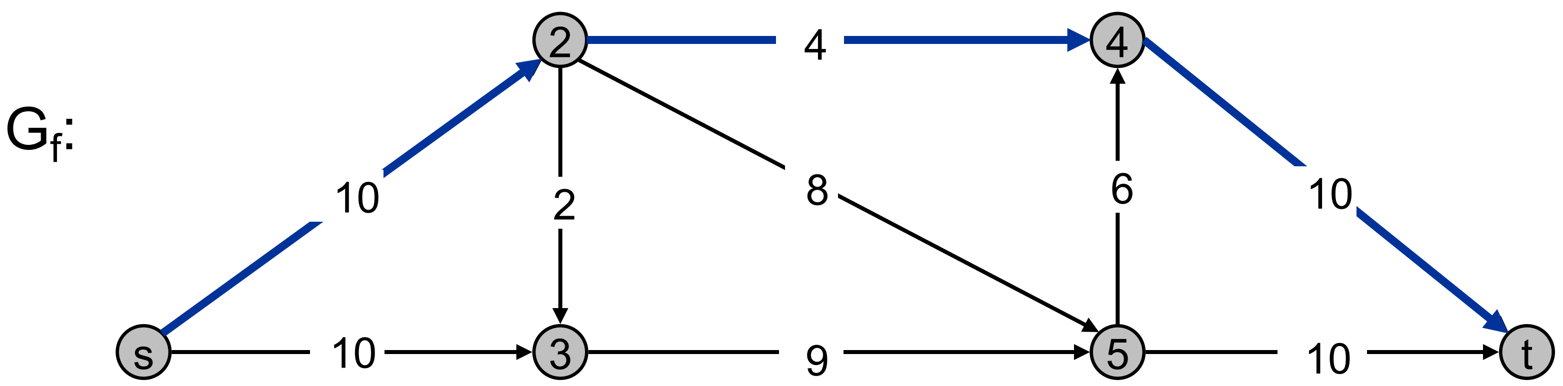


G_f:

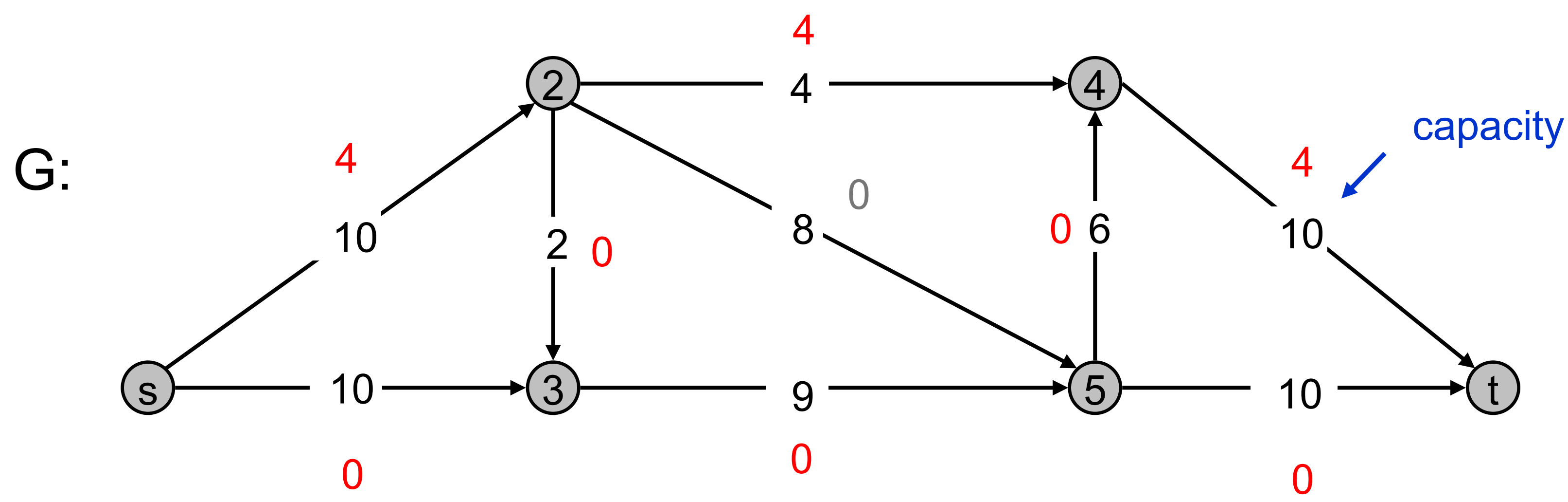


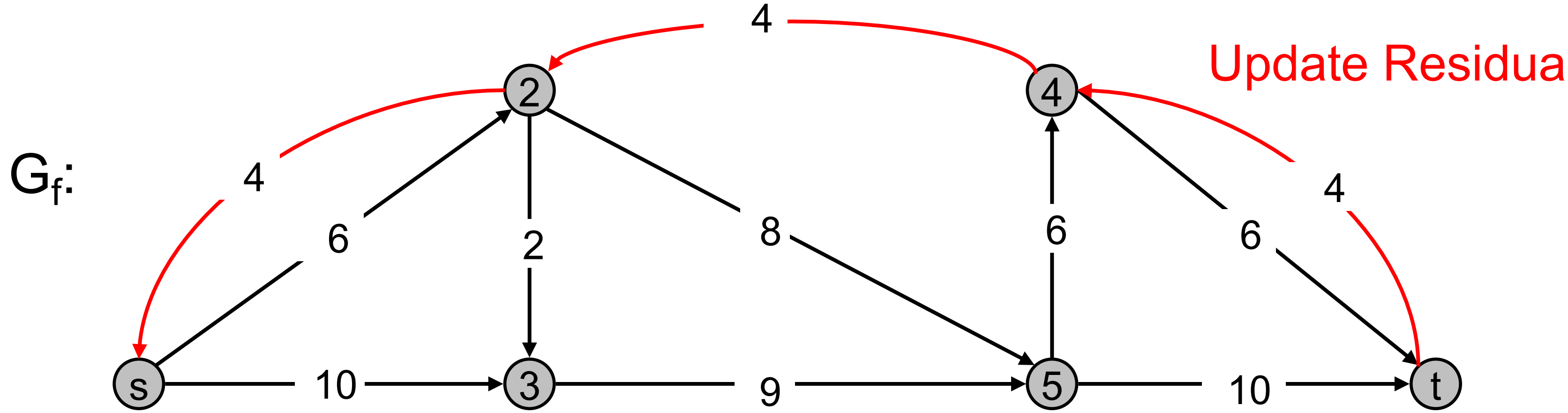
Ford-Fulkerson Alg: Greedy on G_f



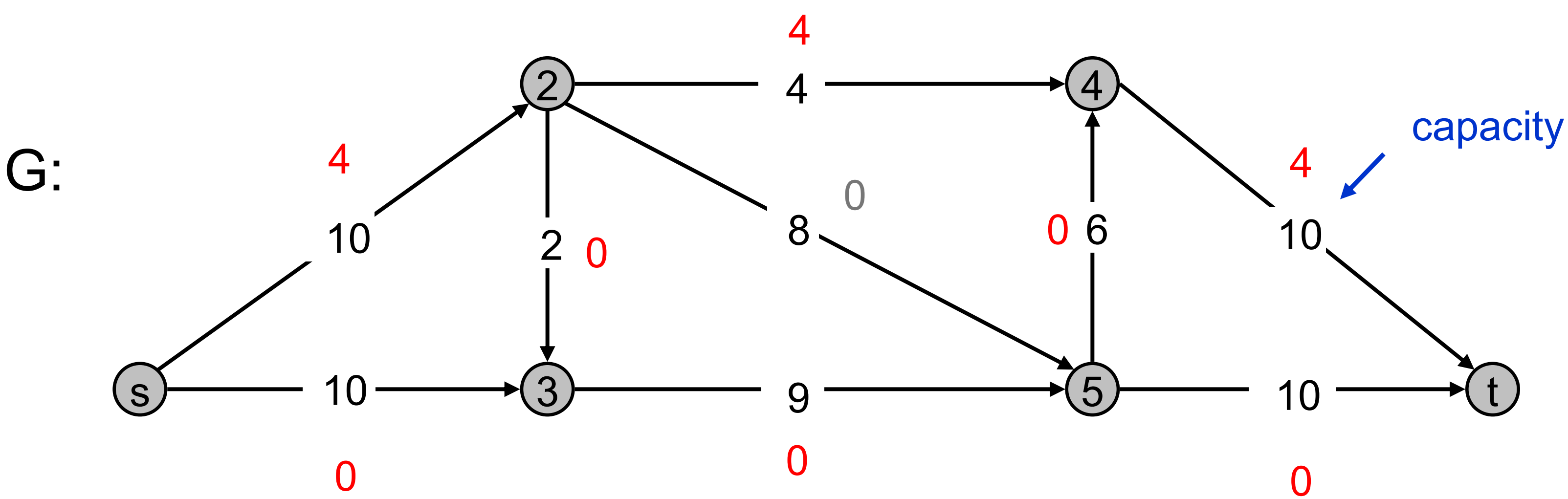


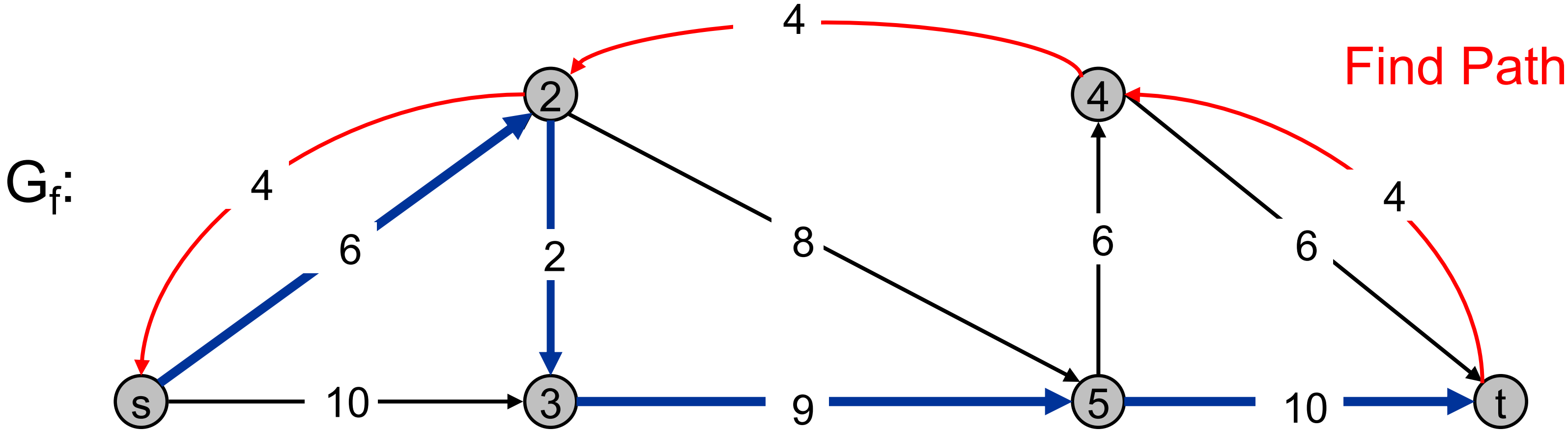
Ford-Fulkerson Alg: Greedy on G_f



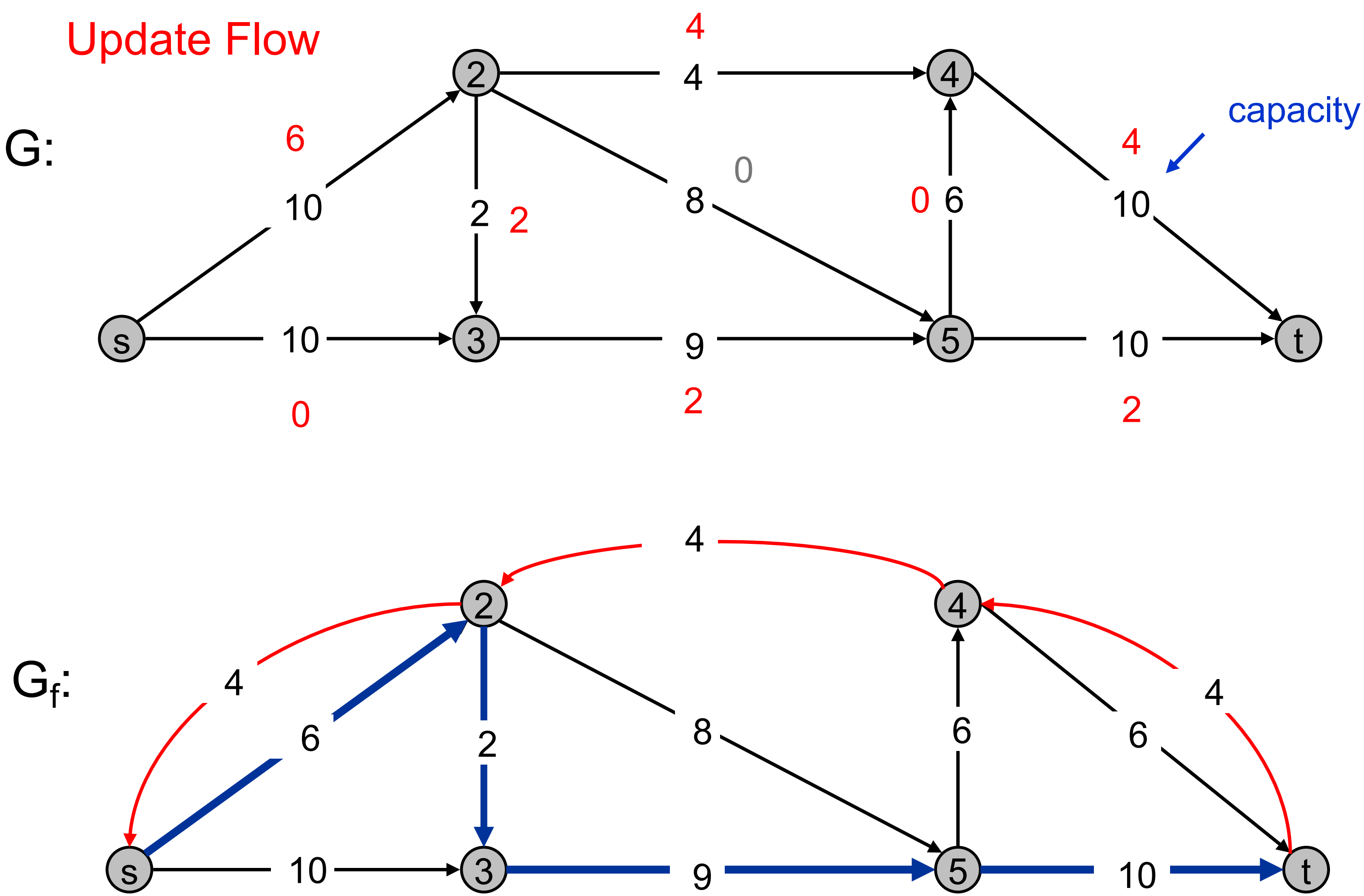


Ford-Fulkerson Alg: Greedy on G_f

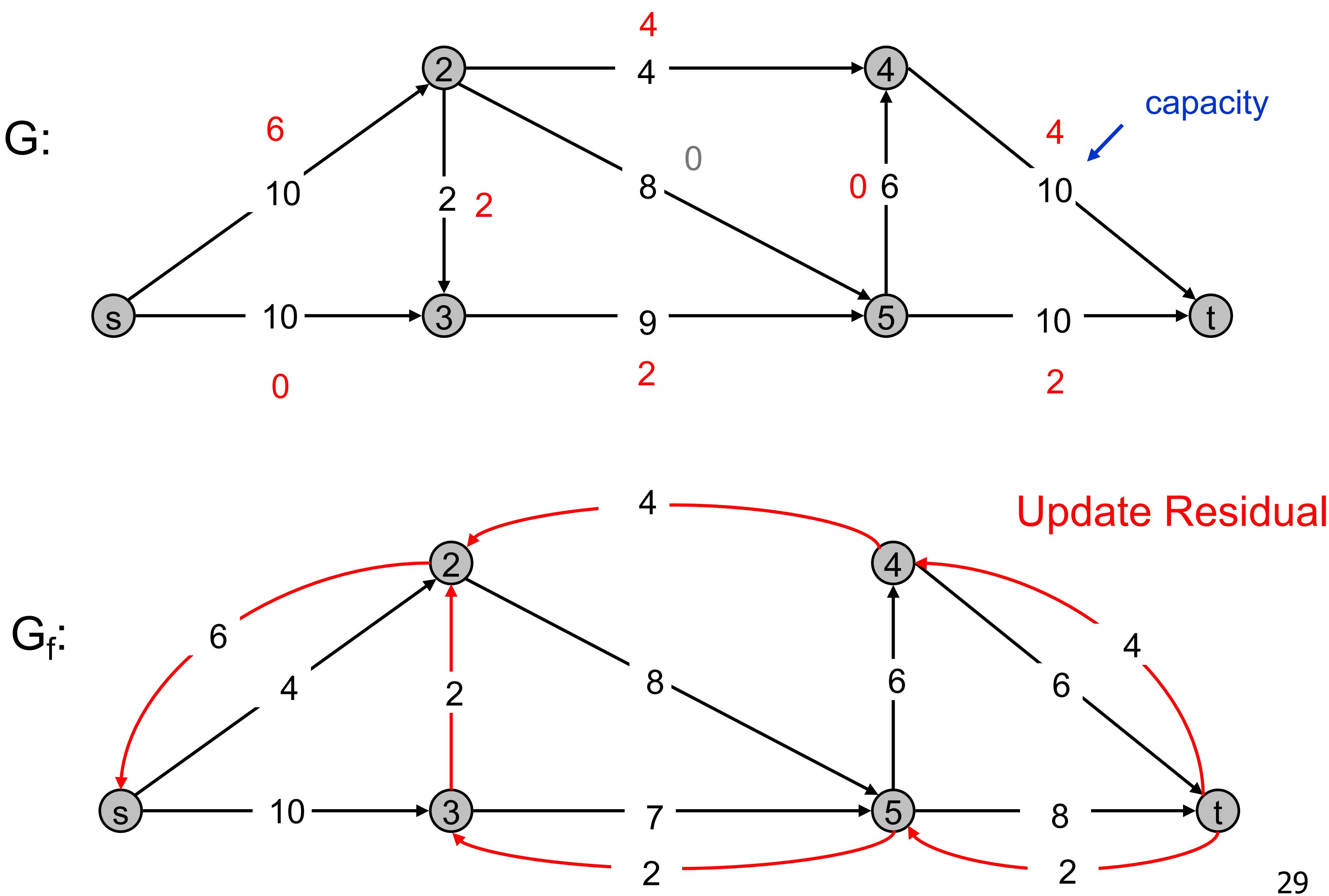




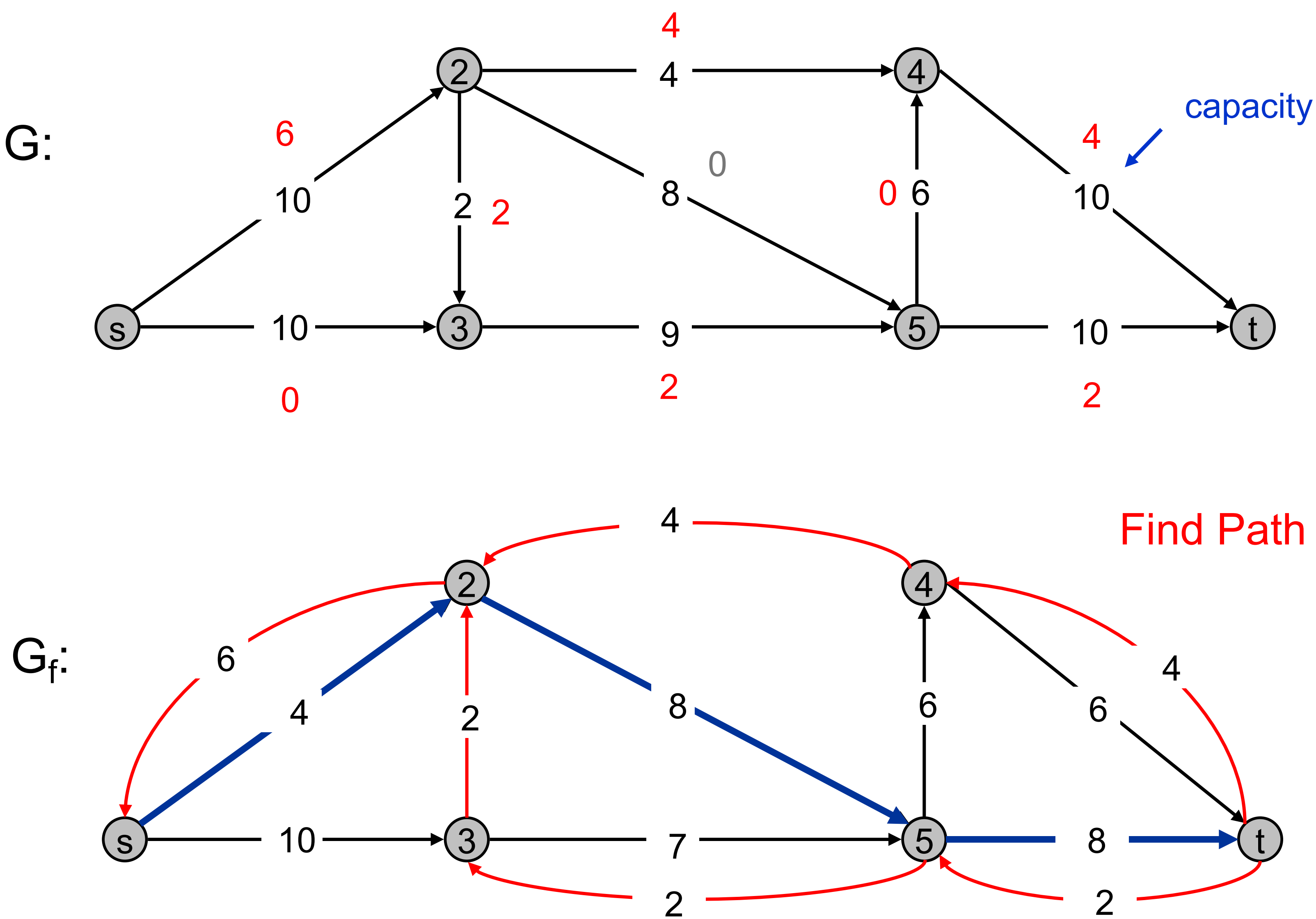
Ford-Fulkerson Alg: Greedy on G_f



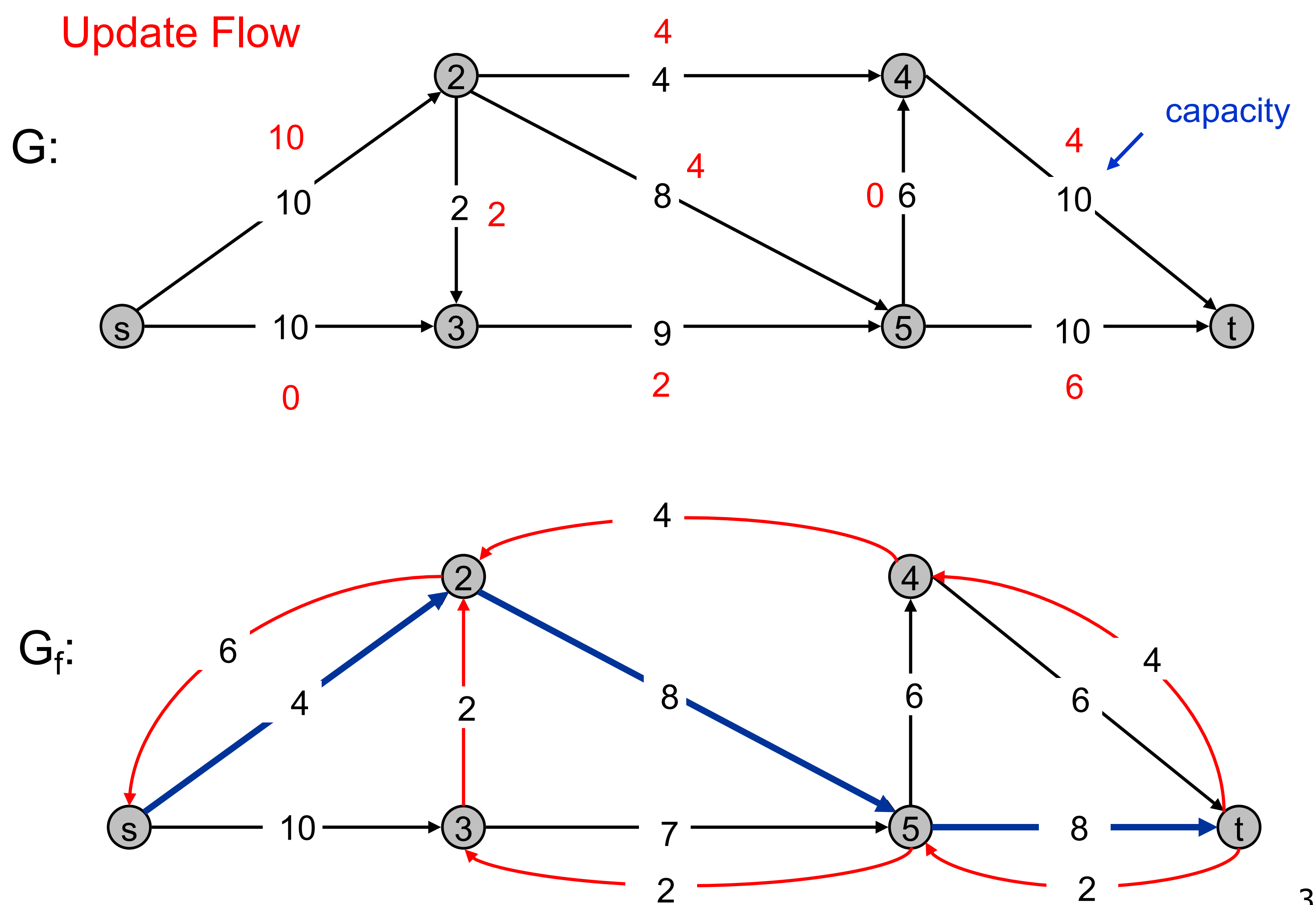
Ford-Fulkerson Alg: Greedy on G_f



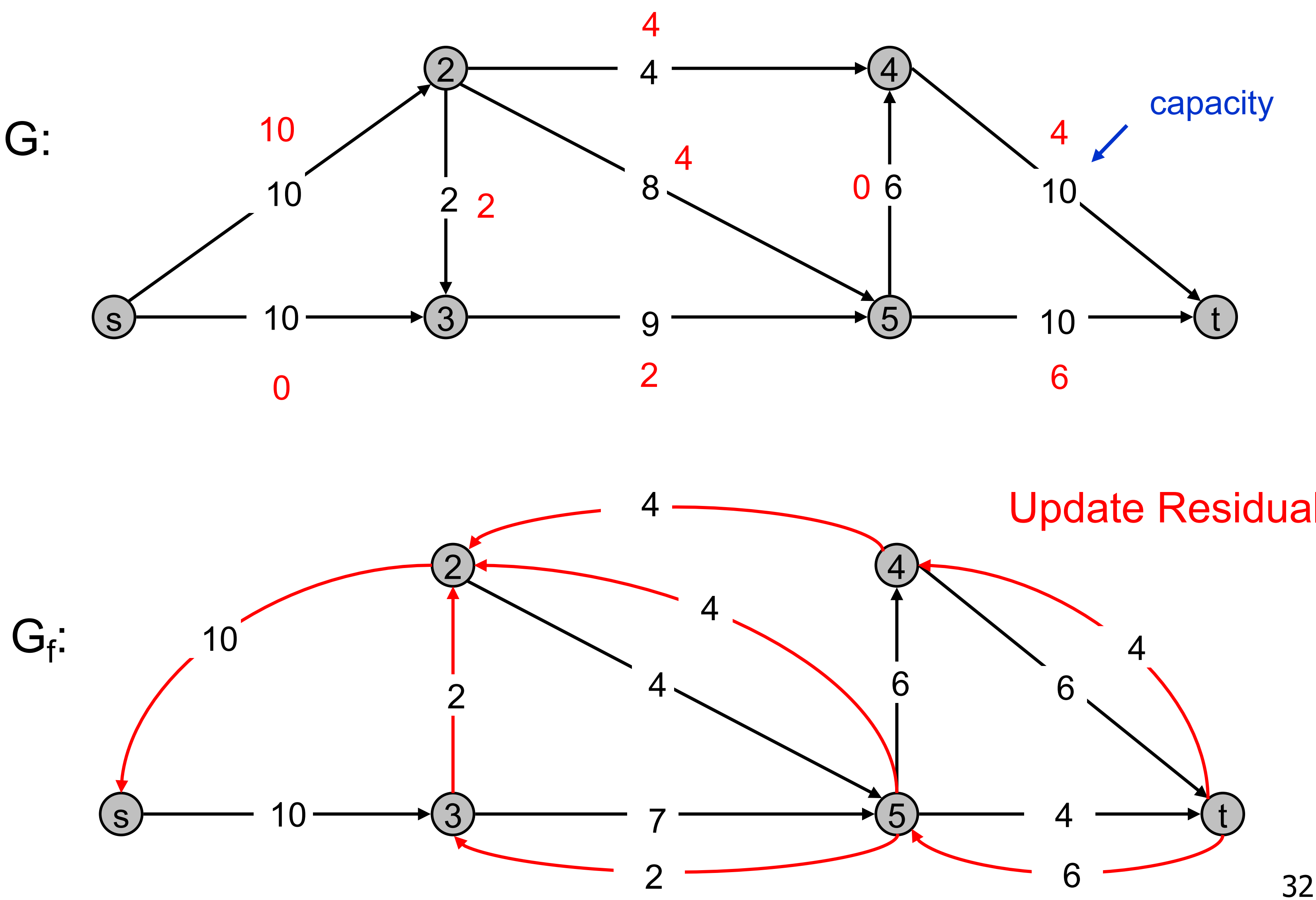
Ford-Fulkerson Alg: Greedy on G_f



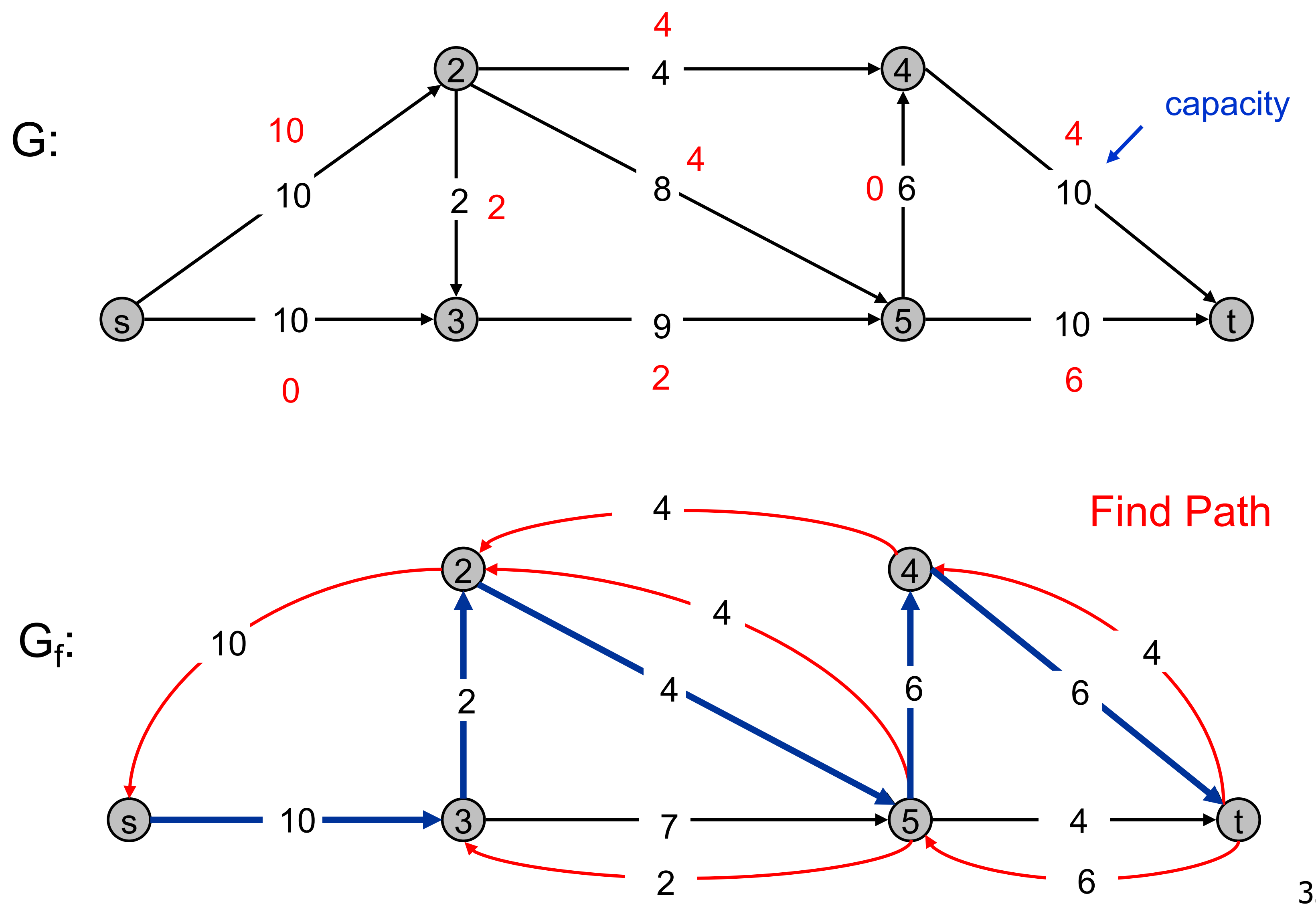
Ford-Fulkerson Alg: Greedy on G_f



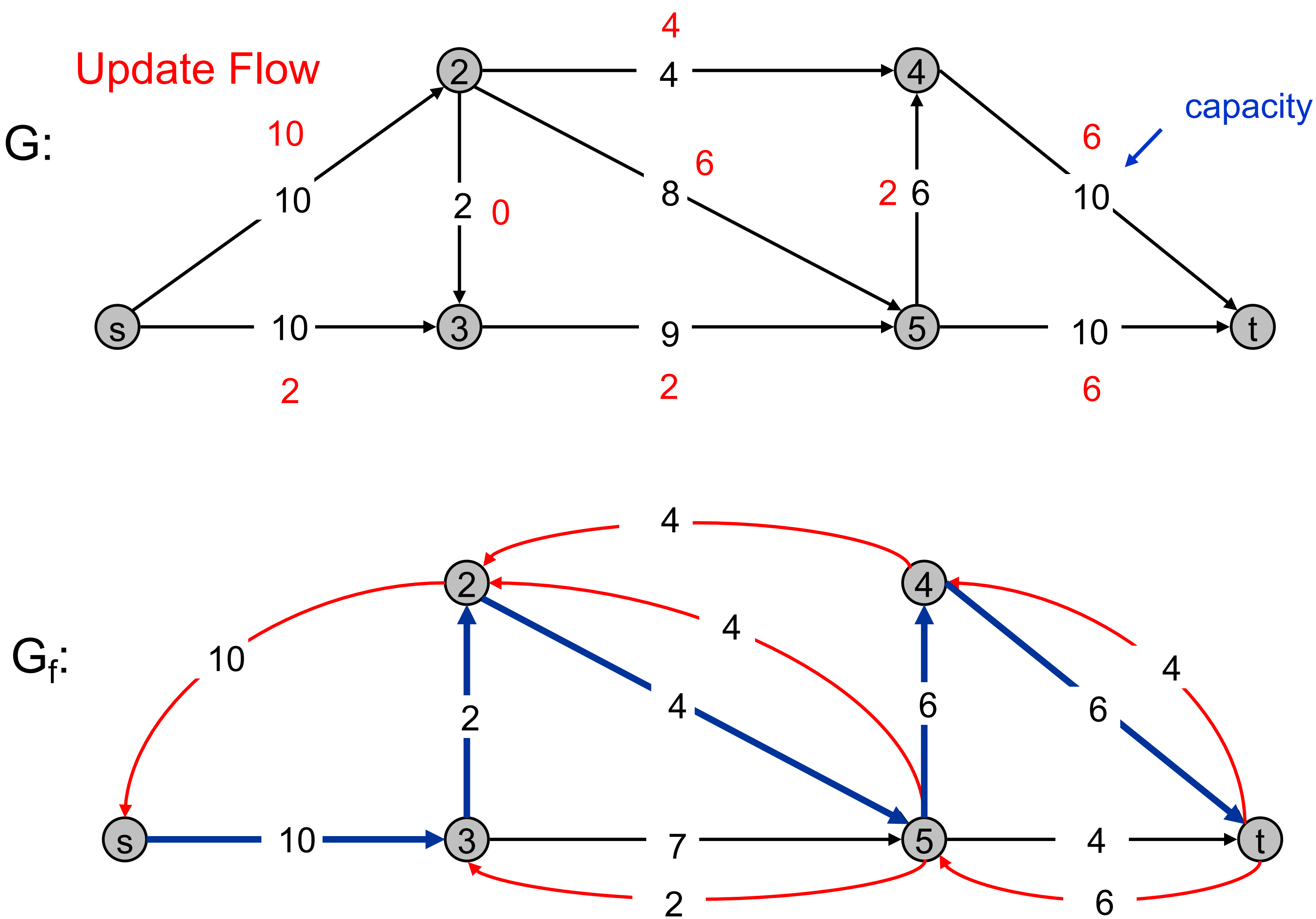
Ford-Fulkerson Alg: Greedy on G_f



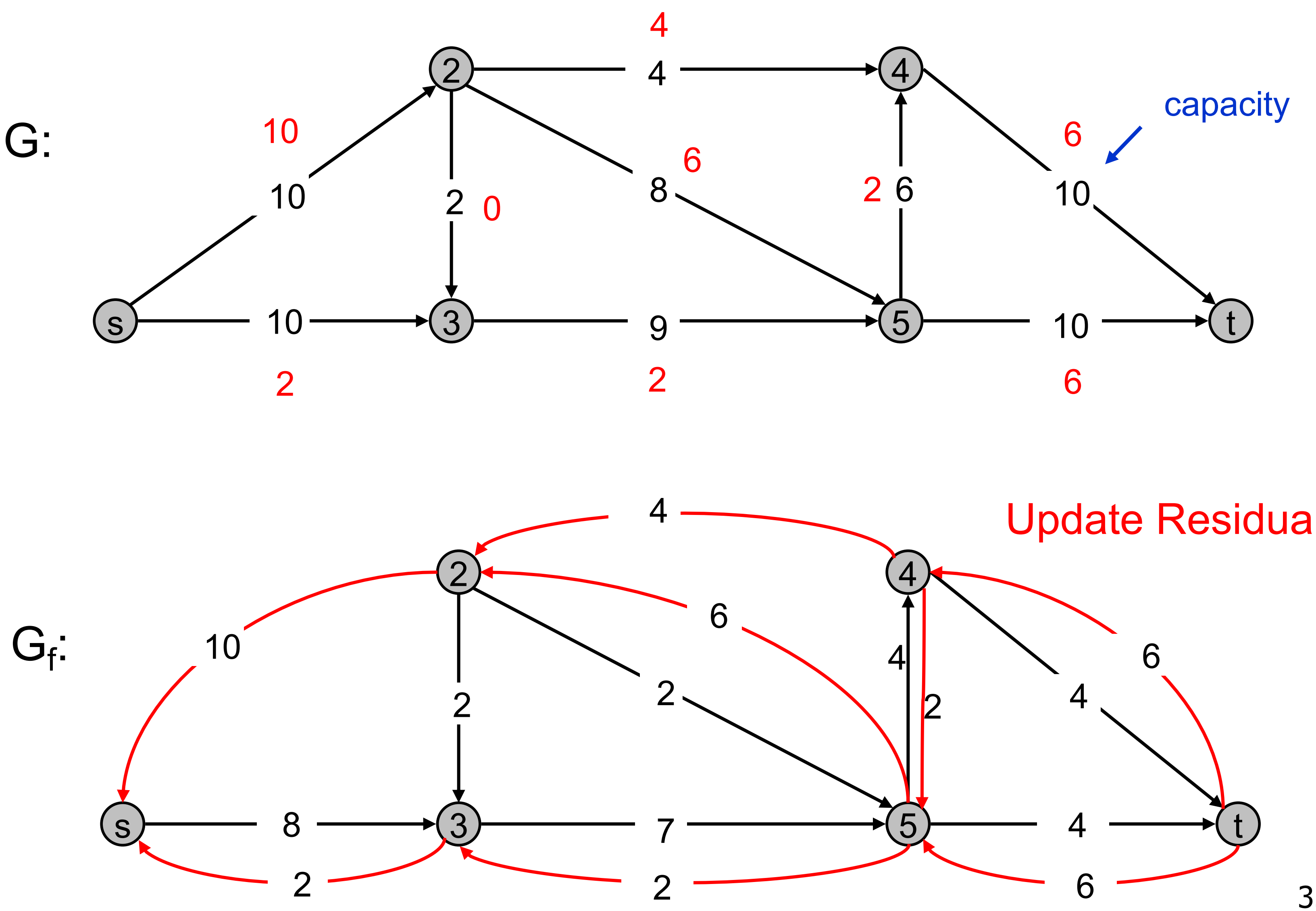
Ford-Fulkerson Alg: Greedy on G_f



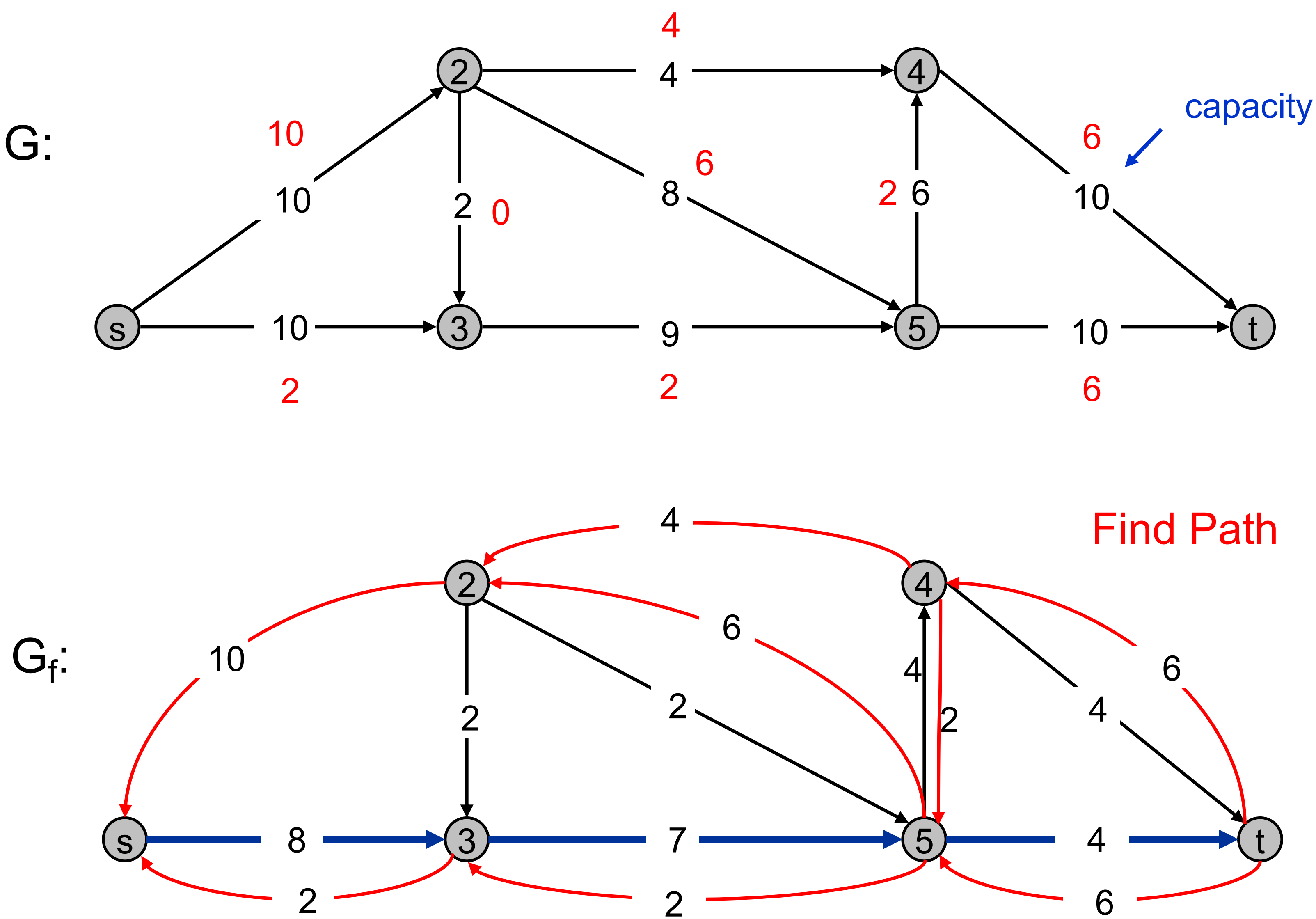
Ford-Fulkerson Alg: Greedy on G_f



Ford-Fulkerson Alg: Greedy on G_f



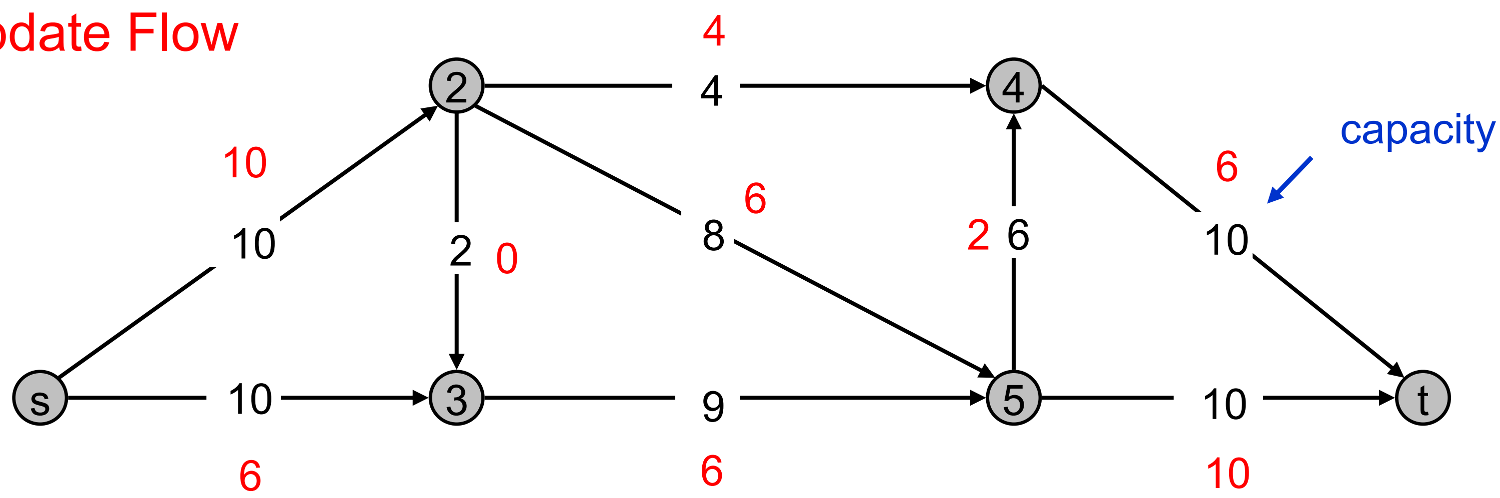
Ford-Fulkerson Alg: Greedy on G_f



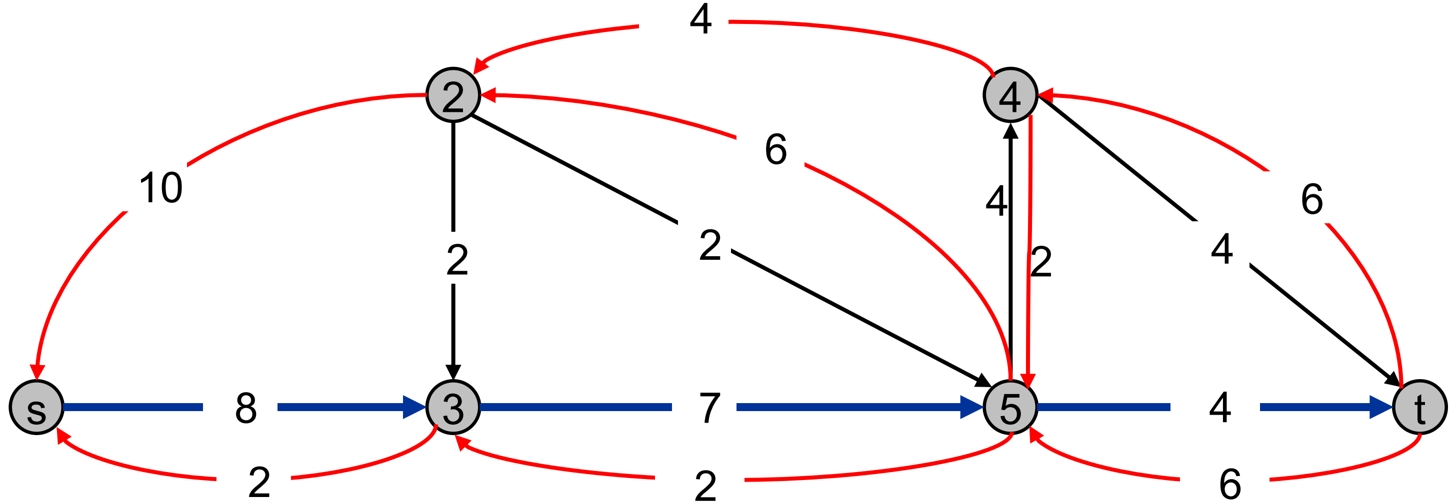
Ford-Fulkerson Alg: Greedy on G_f

Update Flow

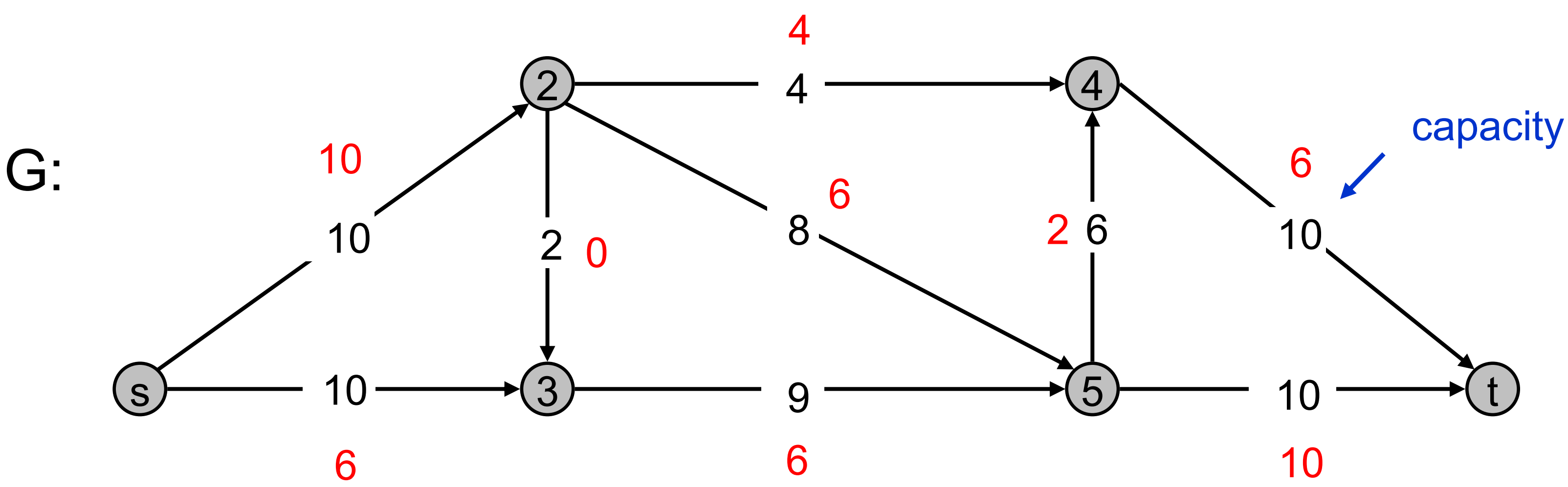
G:

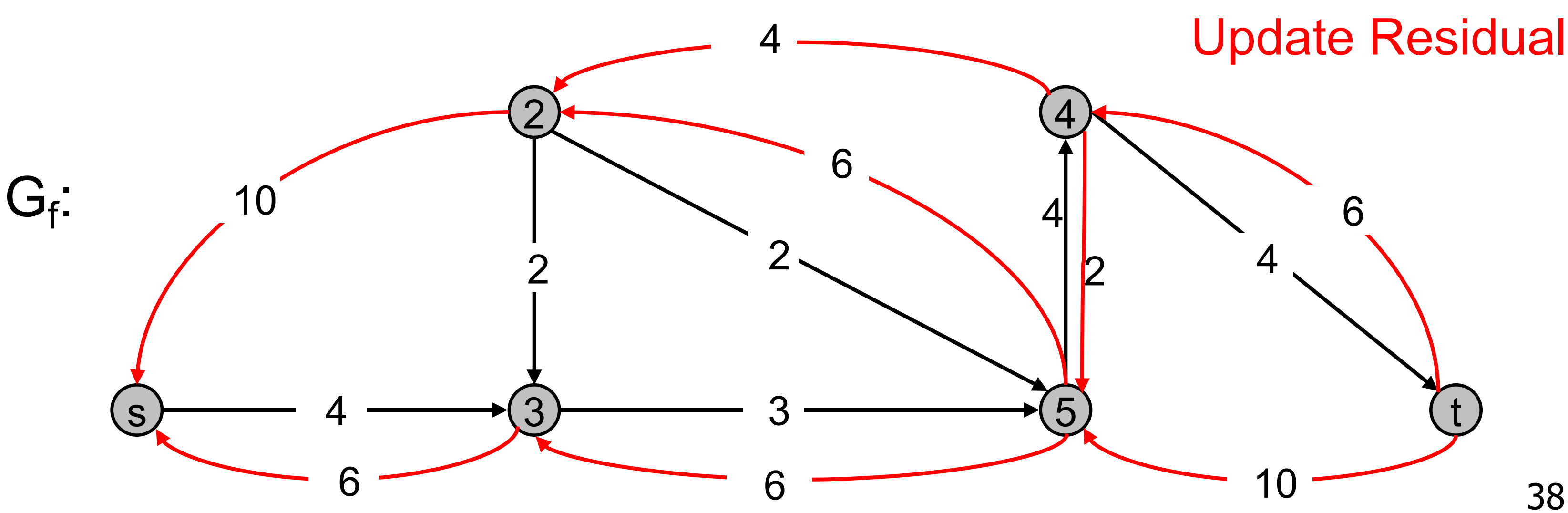


G_f:

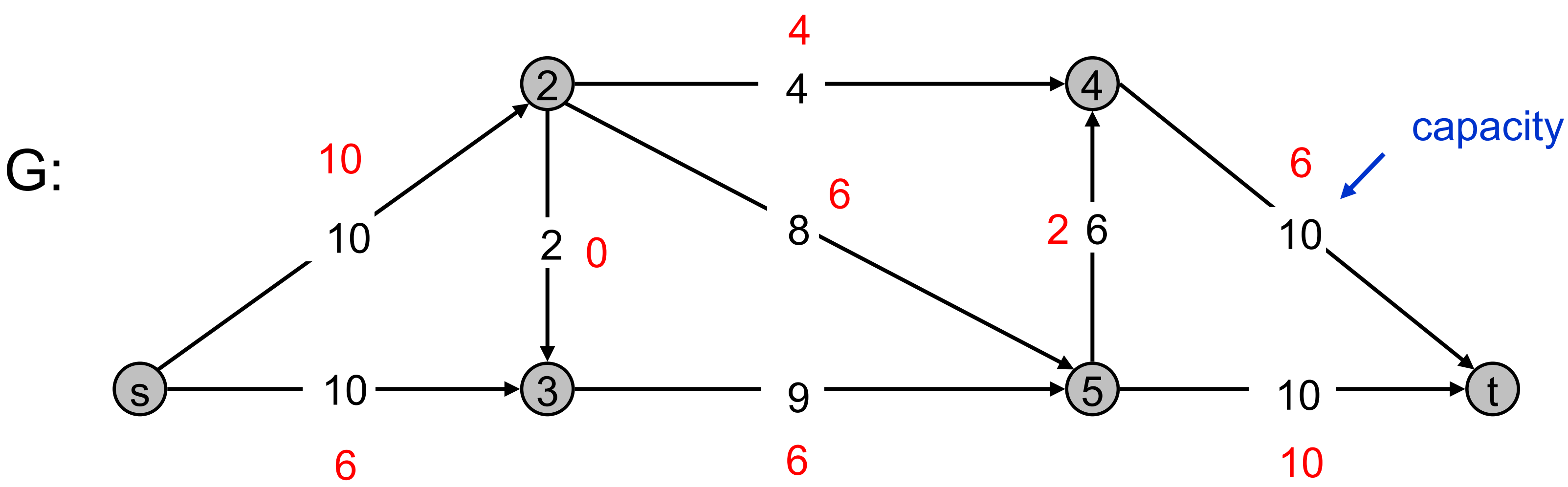


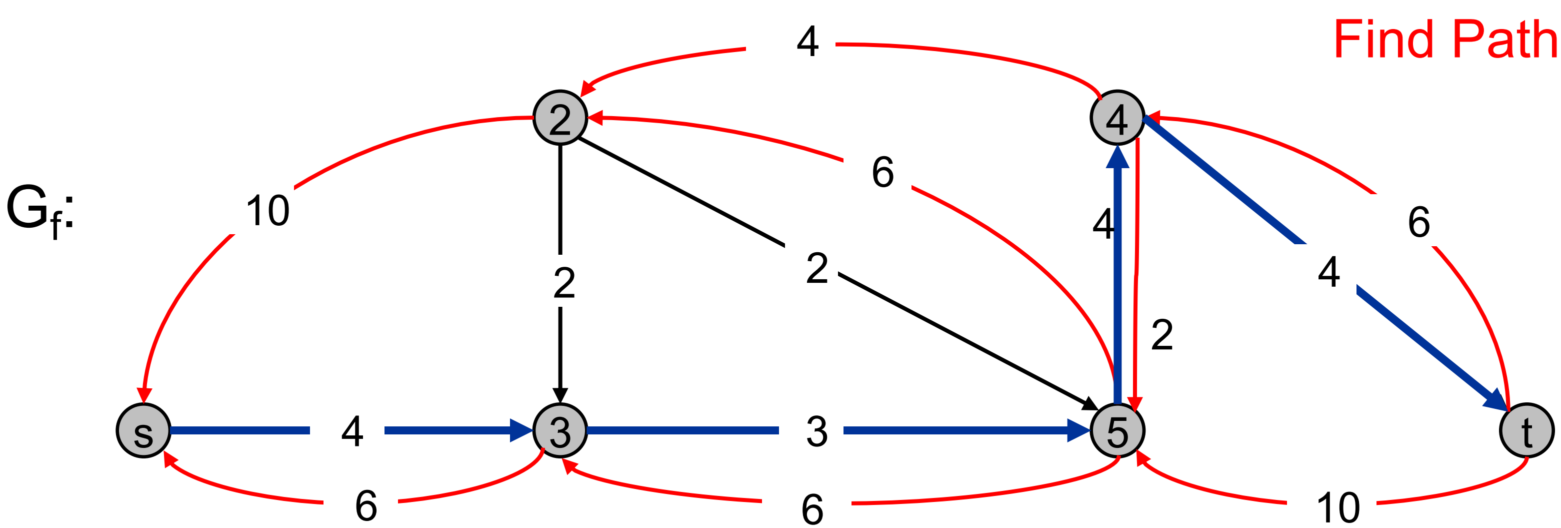
Ford-Fulkerson Alg: Greedy on G_f



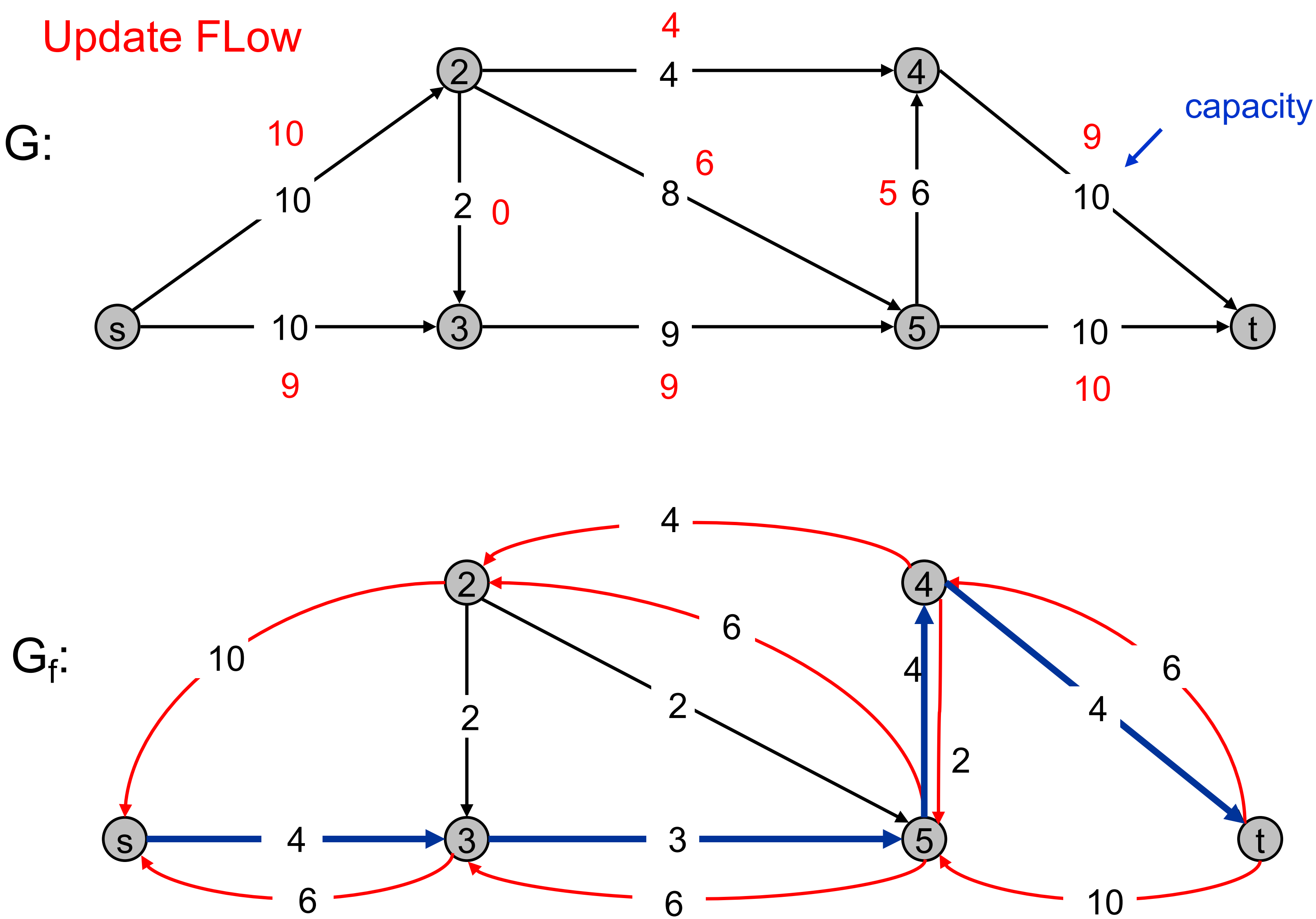


Ford-Fulkerson Alg: Greedy on G_f

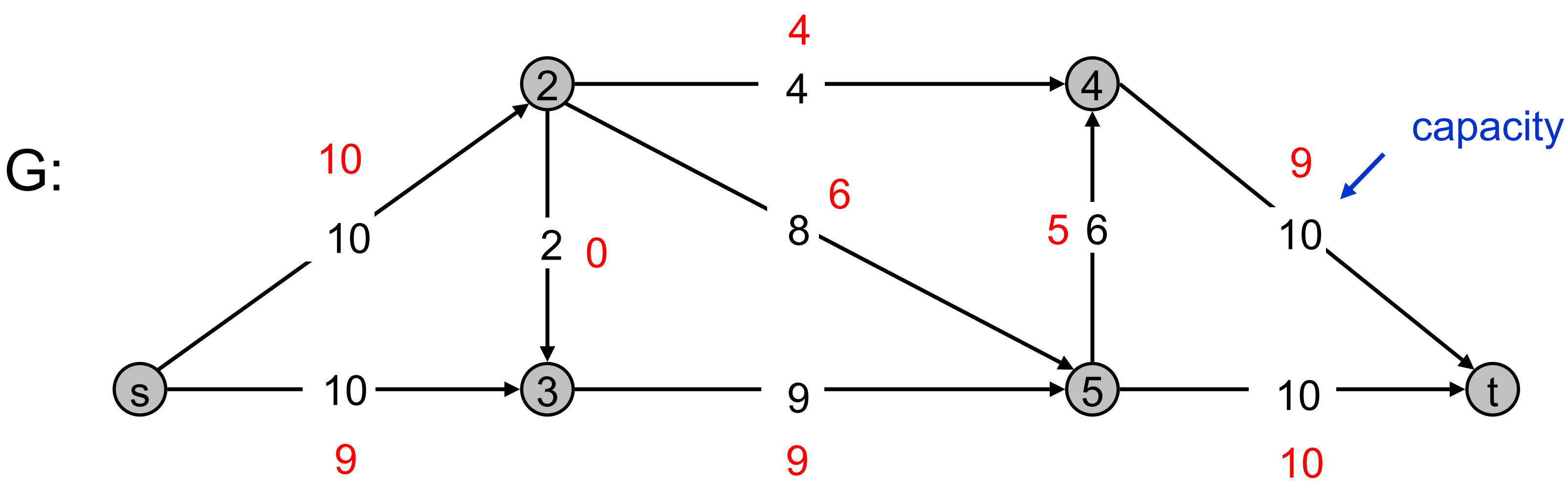


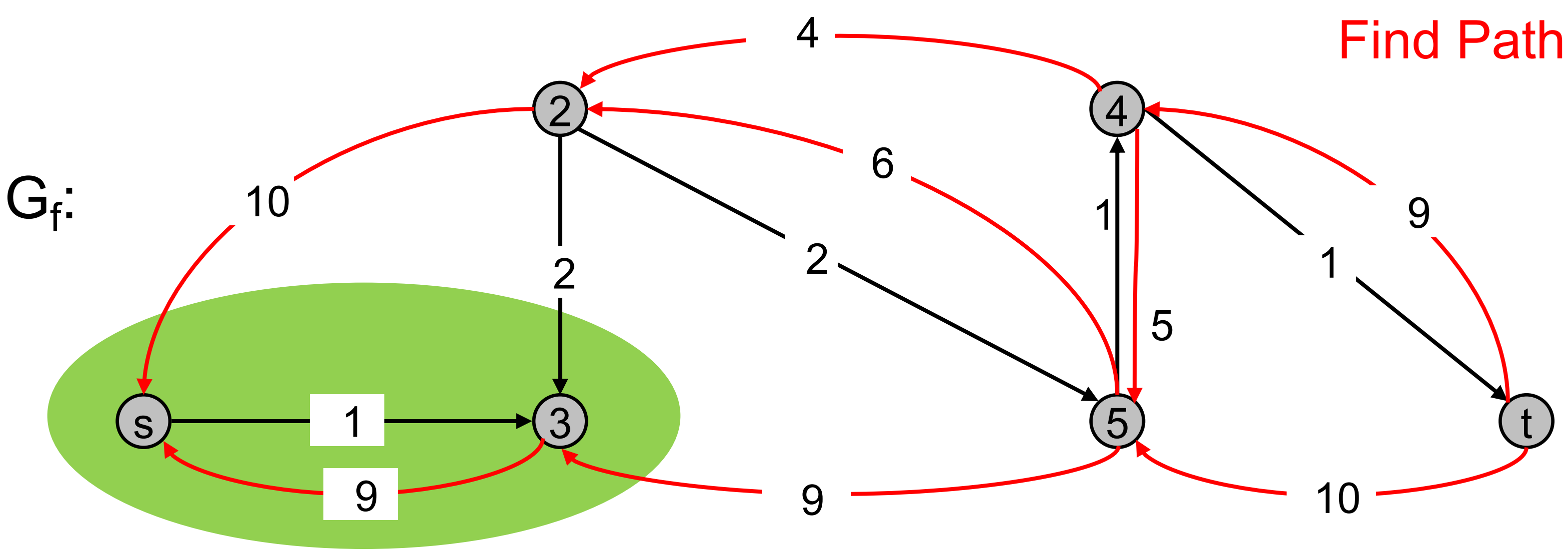


Ford-Fulkerson Alg: Greedy on G_f



Ford-Fulkerson Alg: Greedy on G_f





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(eR) ← c(eR) + b

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

Smallest capacity edge on P

Forward edge

Reverse edge

$e^R \in P$

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