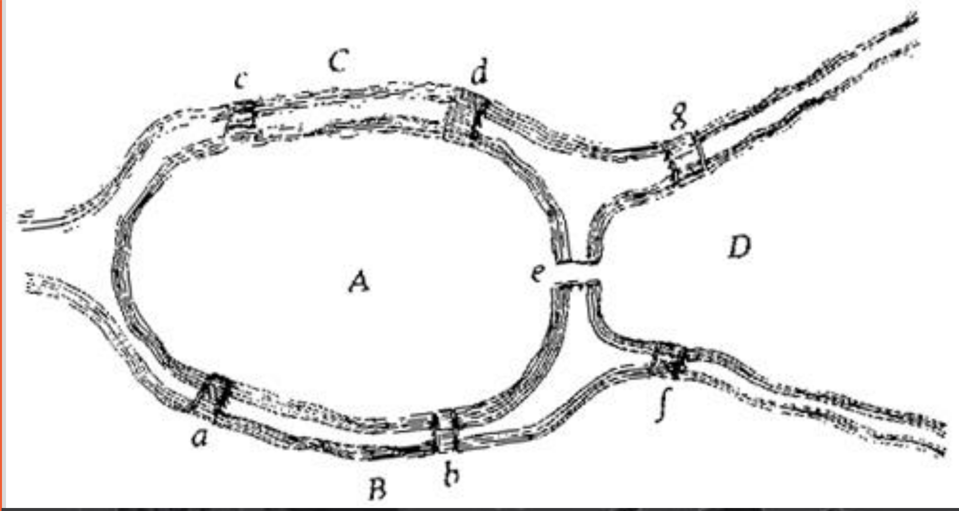
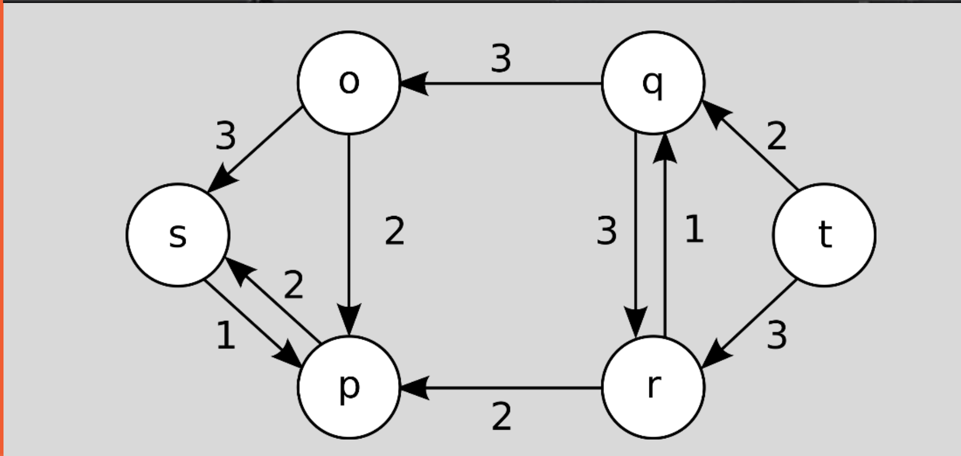
Lecture 8 – Flow networks I





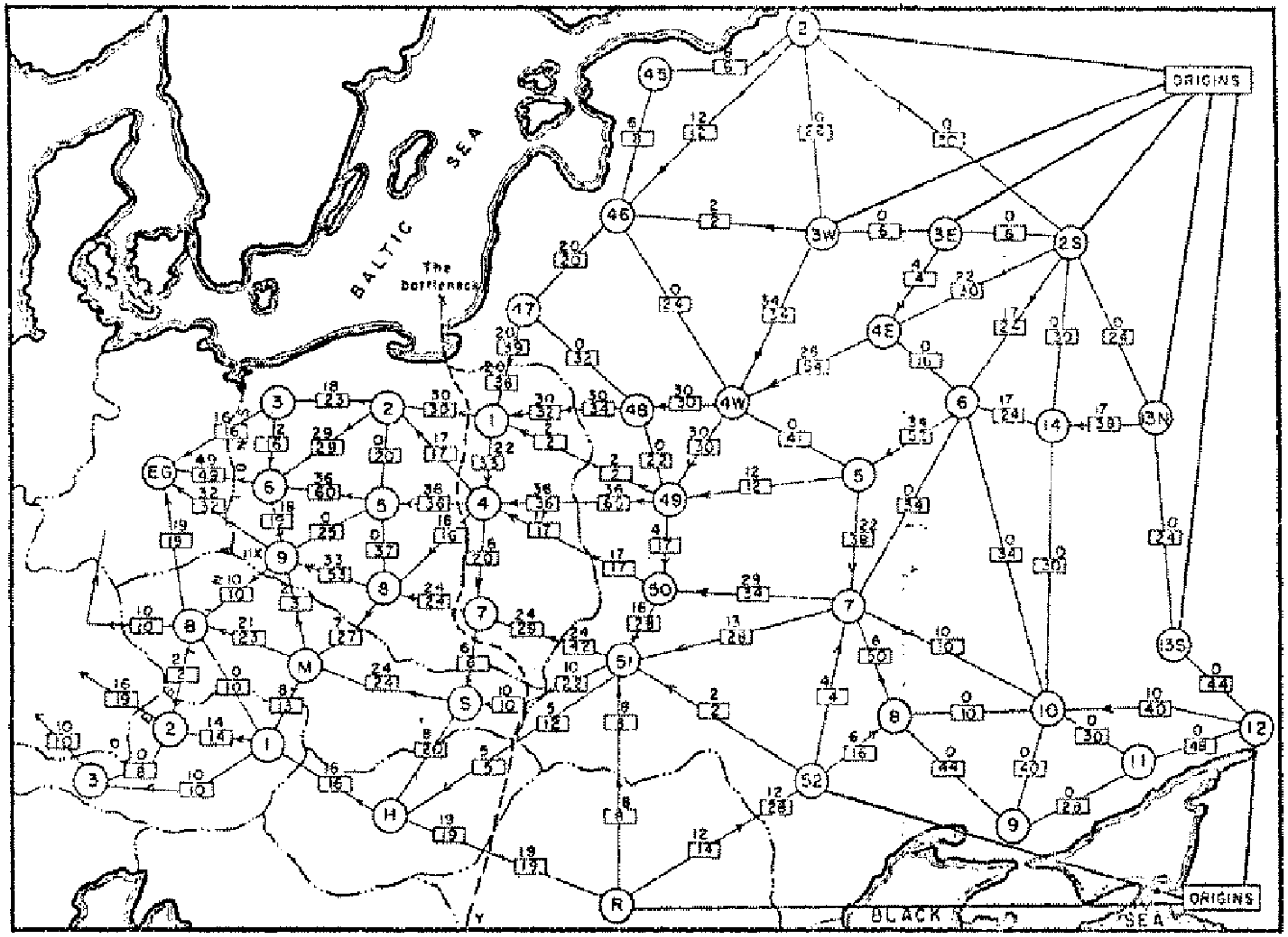
THE UNIVERSITY OF SYDNEY

General techniques in this course

- Greedy algorithms [Lecture 3]
- Divide & Conquer algorithms [Lecture 4]
- Sweepline algorithms [Lecture 5]
- Dynamic programming algorithms [Lectures 6 and 7]
- Network flow algorithms [today and 9 Oct]
  - Theory [today]
  - Applications [9 Oct]

- NP and NP-completeness
- Coping with hardness

Soviet Rail Network, 1955



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

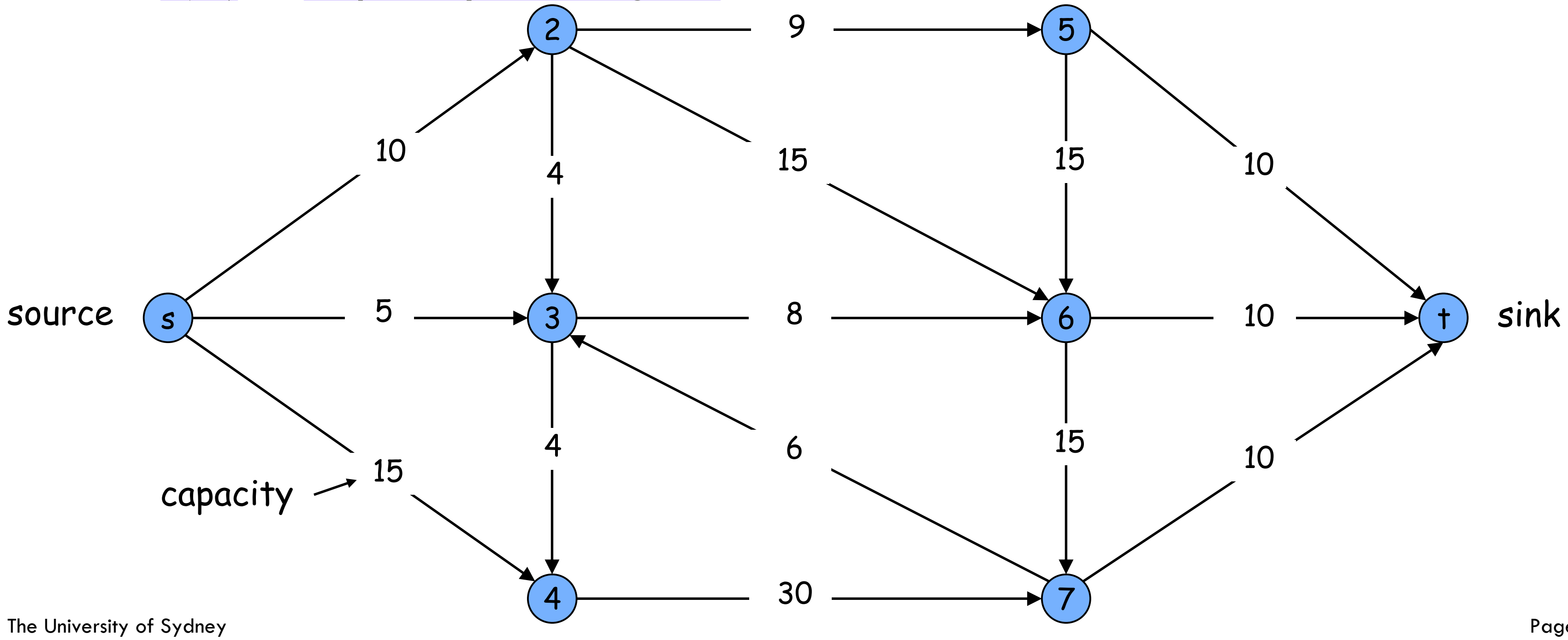
Maximum Flow and Minimum Cut

- Max flow and min cut.
  - Two very rich algorithmic problems.
  - Cornerstone problems in combinatorial optimization.
  - Mathematical duality.

- Nontrivial applications / reductions.
  - Data mining.
  - Open-pit mining.
  - Project selection.
  - Airline scheduling.
  - Bipartite matching.
  - Baseball elimination.
  - Image segmentation.
  - Network connectivity.
  - Network reliability.
  - Distributed computing.
  - Egalitarian stable matching.
  - Security of statistical data.
  - Network intrusion detection.
  - Multi-camera scene reconstruction.
  - Many many more . . .

Flow network

- Abstraction for material flowing through the edges.
- G = (V, E): a directed graph with no parallel edges.
- Two distinguished nodes: s = source, t = sink.
- The source has no incoming edges and the sink has no outgoing edges.
- c(e) = capacity of edge e.



Flows

– Definition: An s-t flow is a function that satisfies:

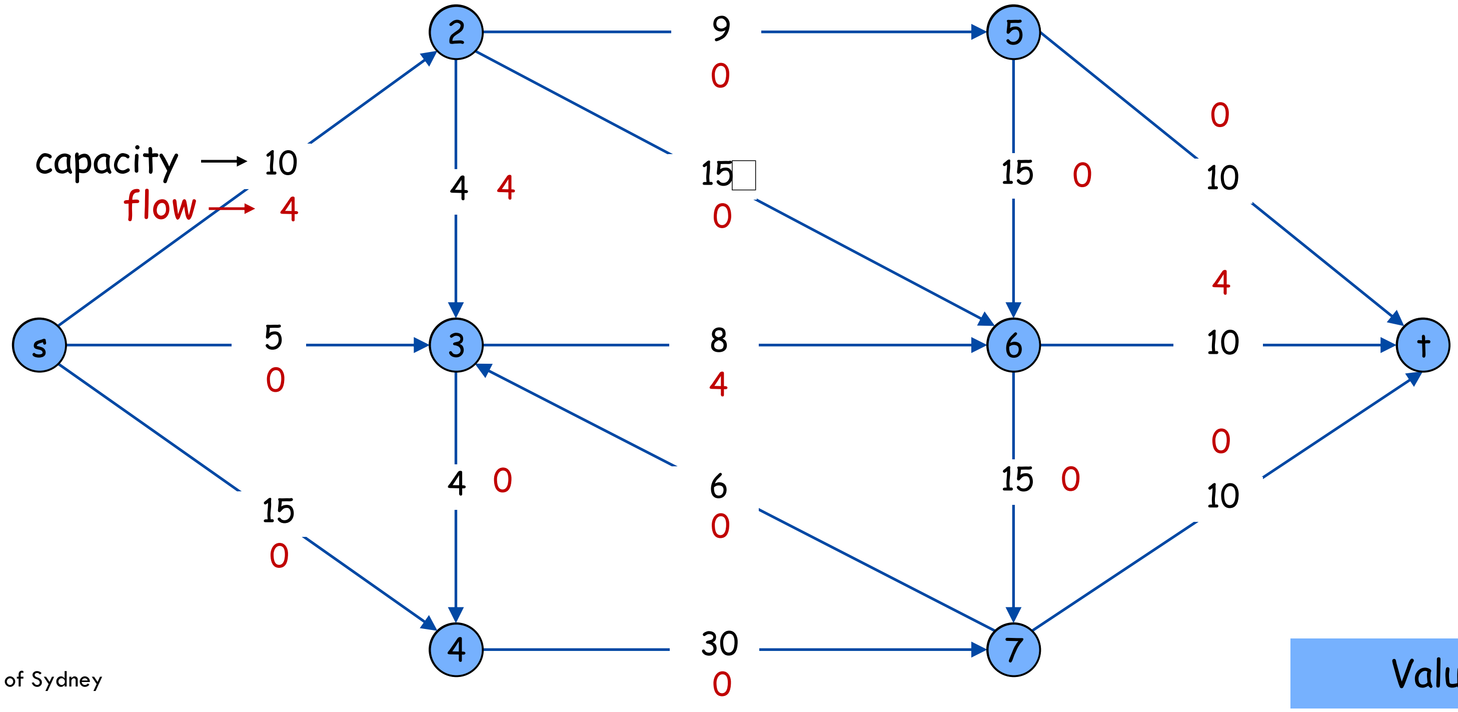
  – For each e ∈ E: $0 \le f(e) \le c(e)$ (capacity)

  – For each v ∈ V – {s, t}: $\sum_{e \text{ in to } v} f(e) = \sum_{e \text{ out of } v} f(e)$ (conservation)

Capacity不能越界

进出相等

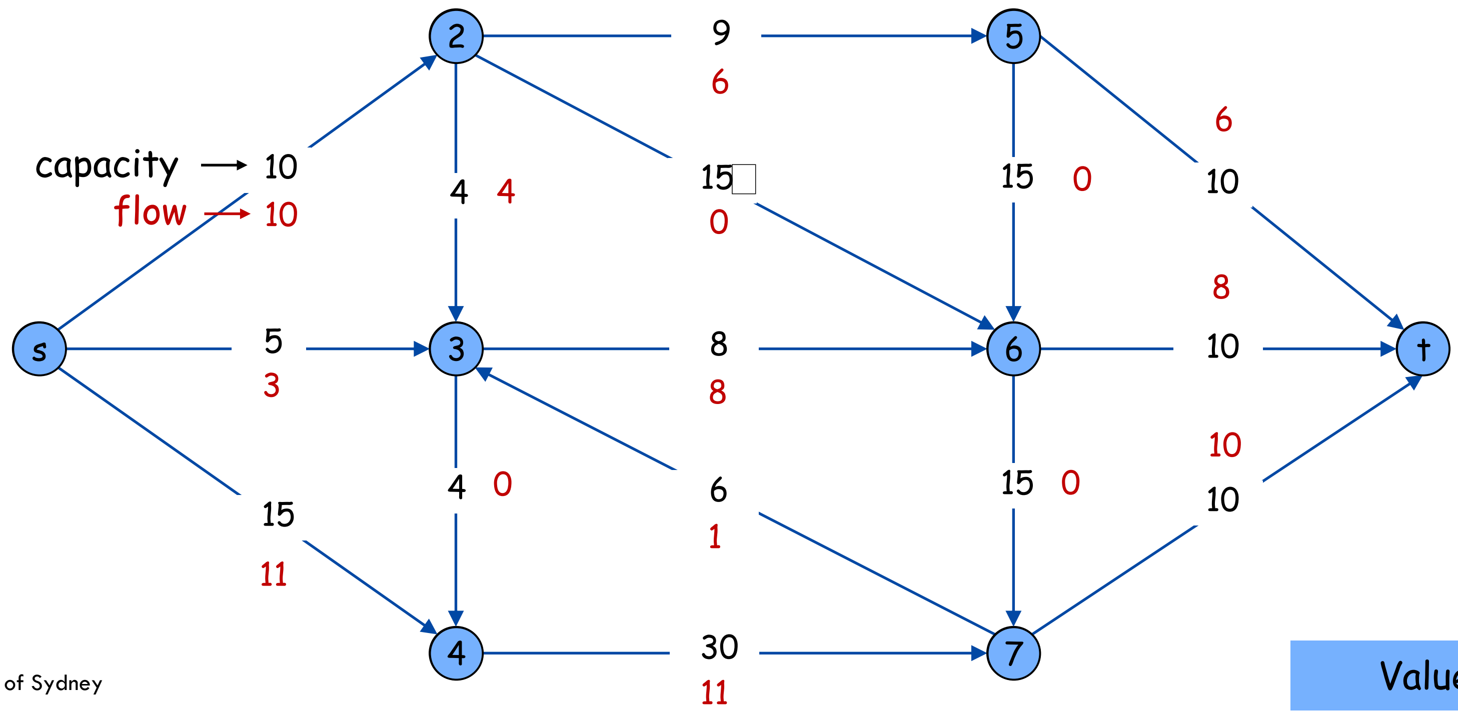
– Definition: The value of a flow f is: $v(f) = \sum_{e \text{ out of } s} f(e)$.



Value = 4

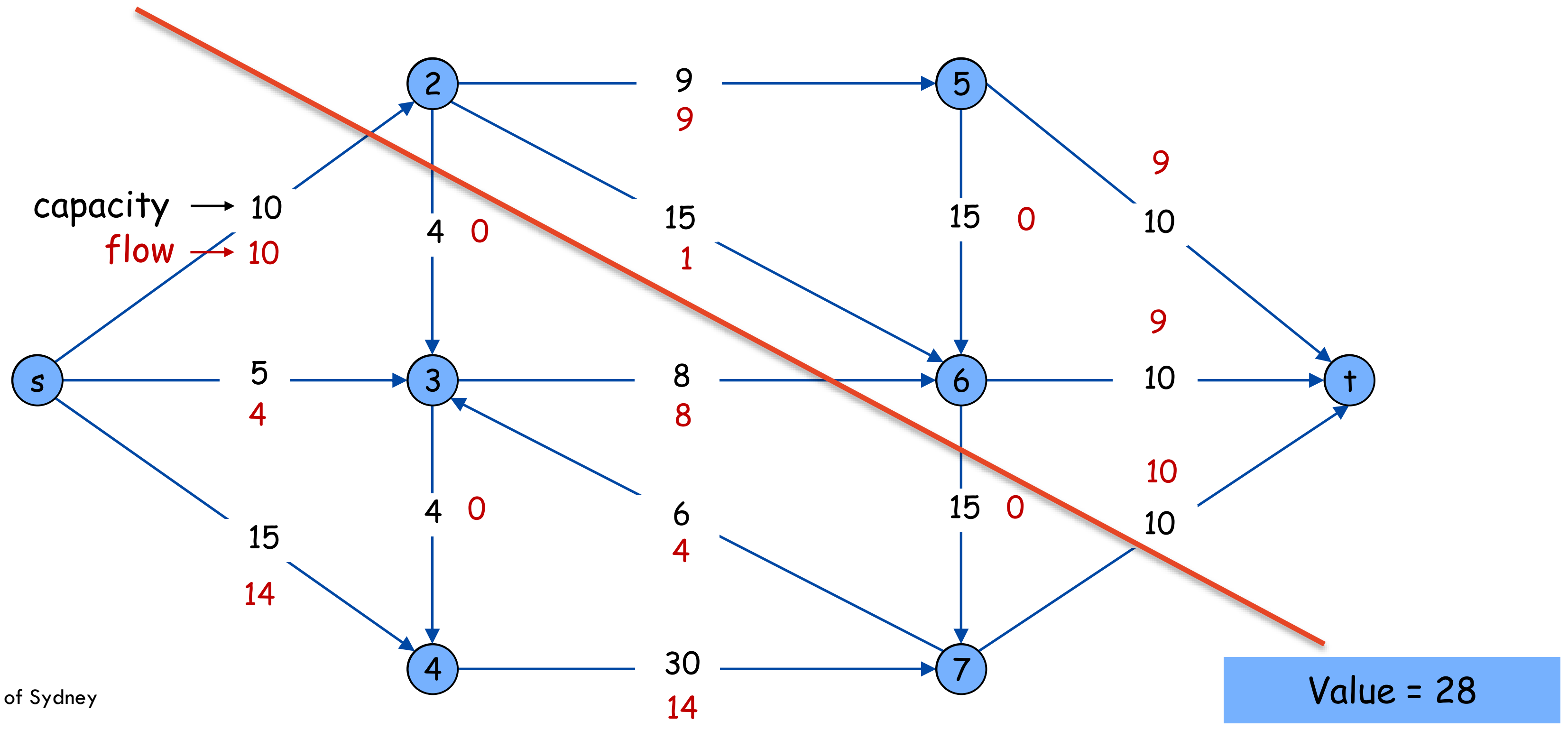
Flows

- Definition: An s-t flow is a function that satisfies:
  - For each e ∈ E: $0 \le f(e) \le c(e)$ (capacity)
  - For each v ∈ V – {s, t}: $\sum_{e \text{ in to } v} f(e) = \sum_{e \text{ out of } v} f(e)$ (conservation)
- Definition: The value of a flow f is: $v(f) = \sum_{e \text{ out of } s} f(e)$.



Maximum Flow Problem

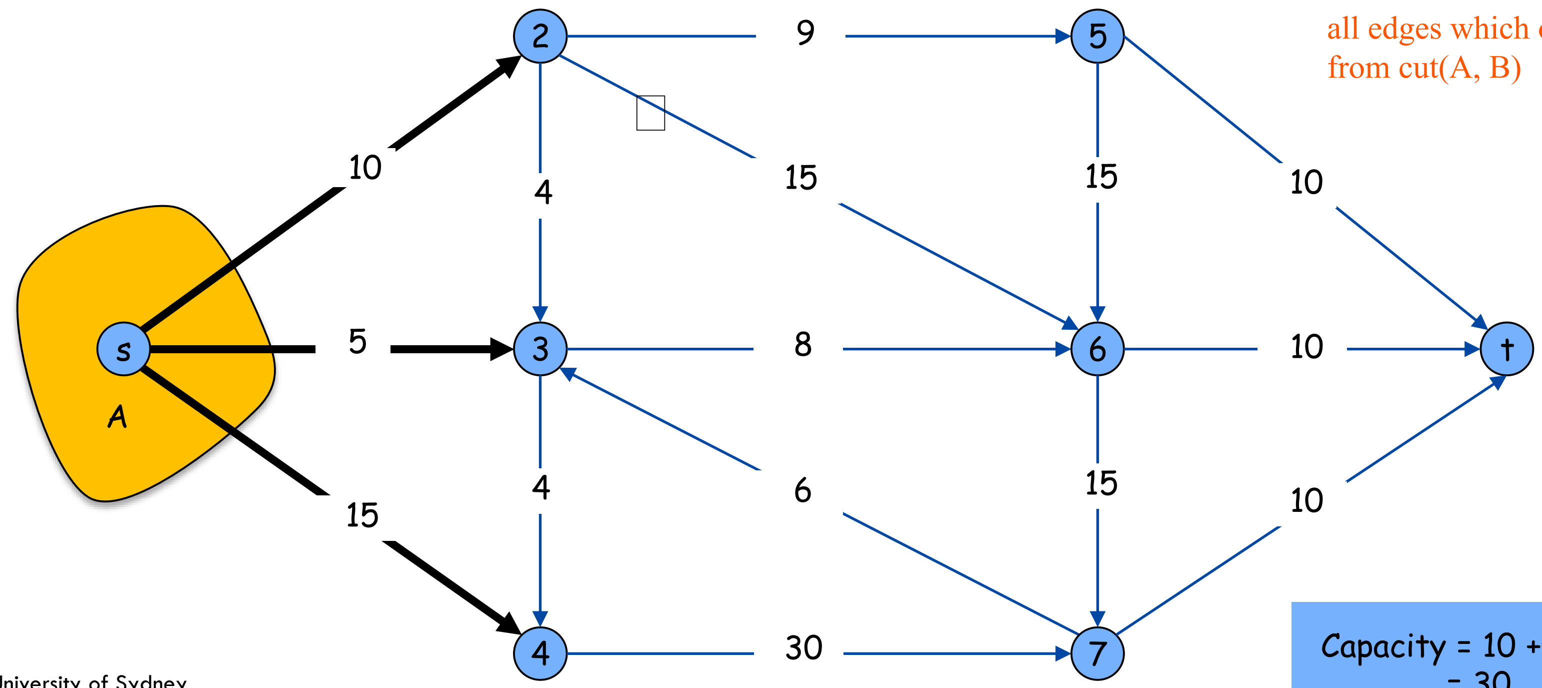
– Max flow problem. Find s-t flow of maximum value.



Cuts

Definitions:

- An s-t cut is a partition (A, B) of V with $s \in A$ and $t \in B$.
- The capacity of a cut (A, B) is: $cap(A, B) = \sum_{e \text{ out of } A} c(e)$



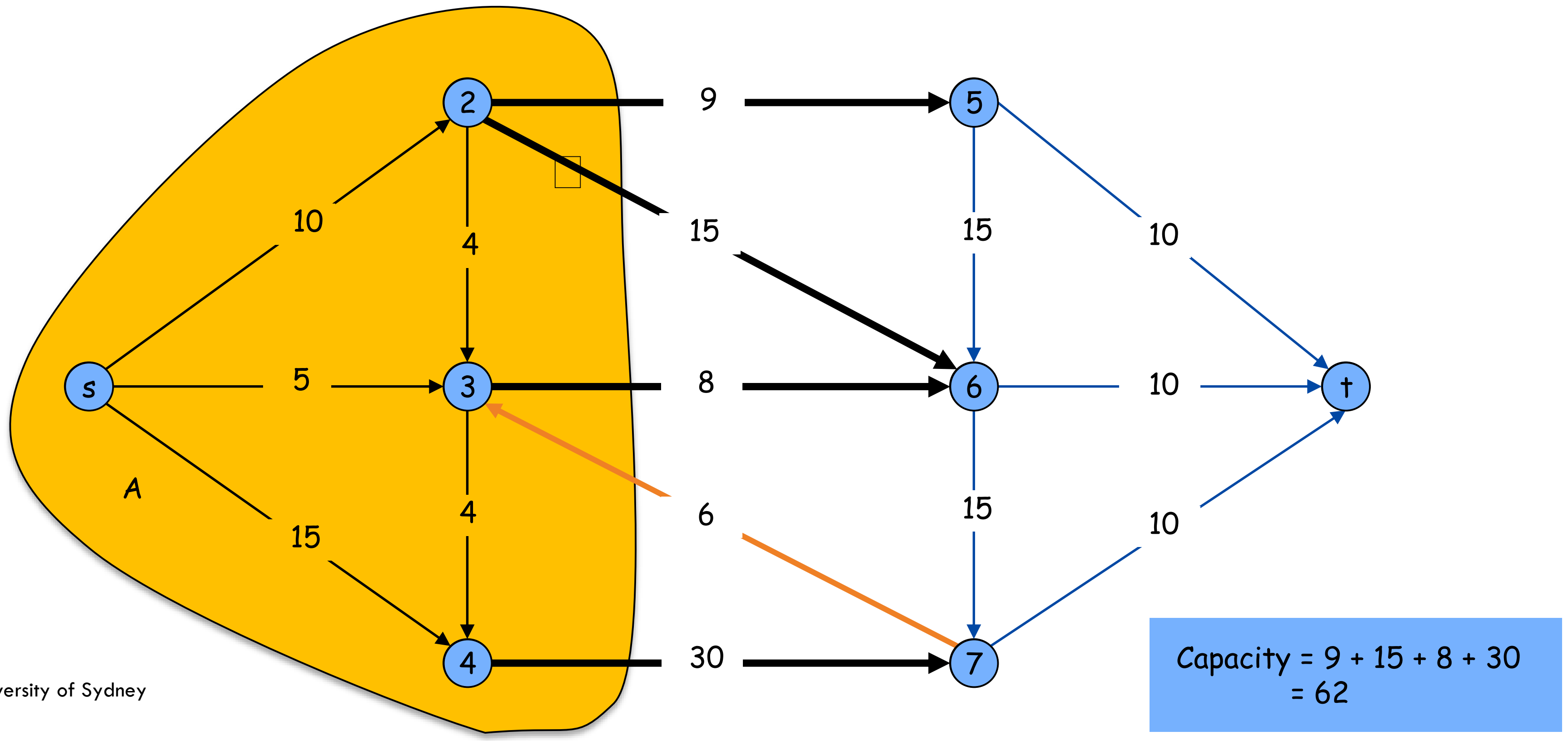
The sum of the capacities of all edges which outgoing from cut(A, B)

Capacity = 10 + 5 + 15
= 30

Cuts

Definitions:

- An s-t cut is a partition (A, B) of V with s ∈ A and t ∈ B.
- The capacity of a cut (A, B) is: $cap(A, B) = \sum_{e \text{ out of } A} c(e)$

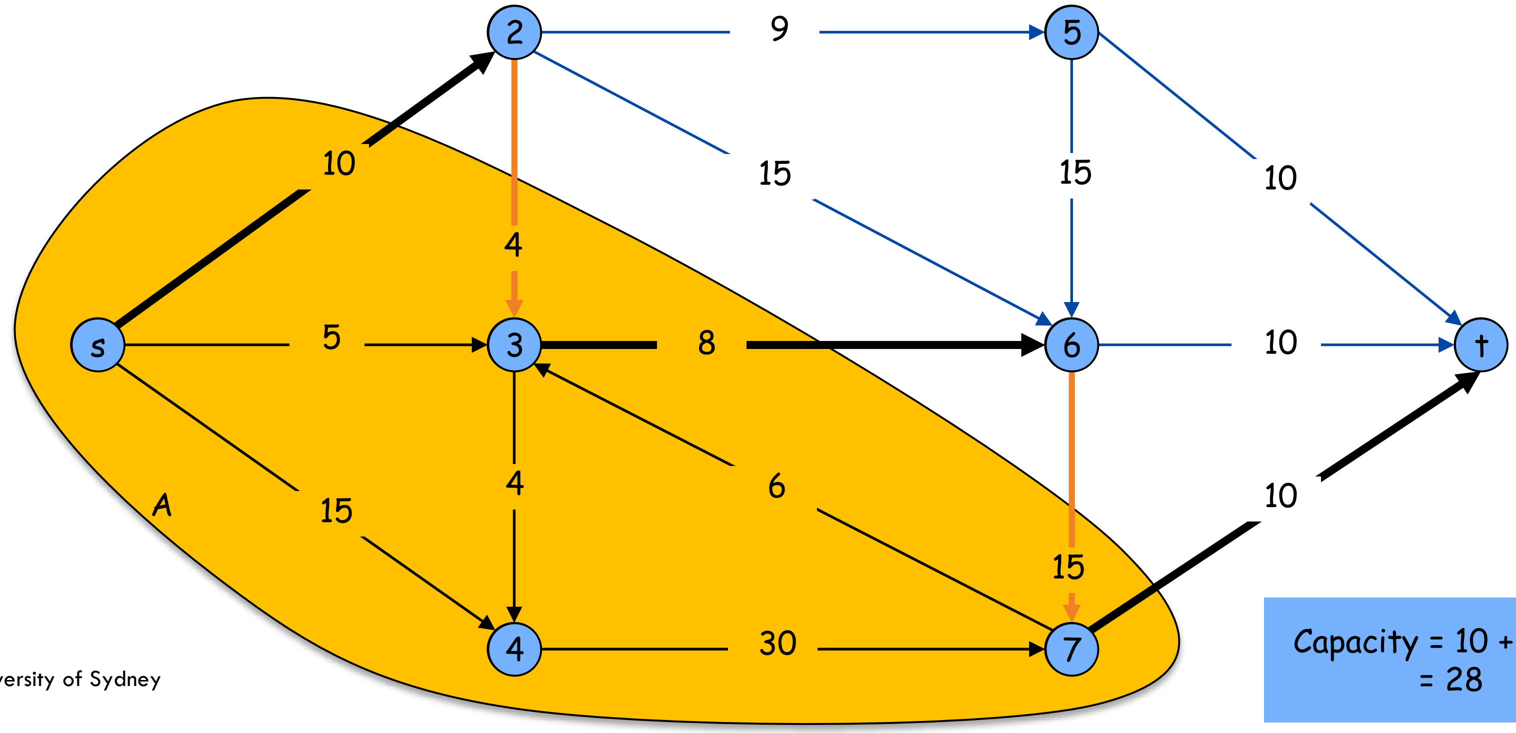


Capacity = 9 + 15 + 8 + 30
= 62

Minimum Cut Problem

Min s-t cut problem:

Find an s-t cut of minimum capacity.

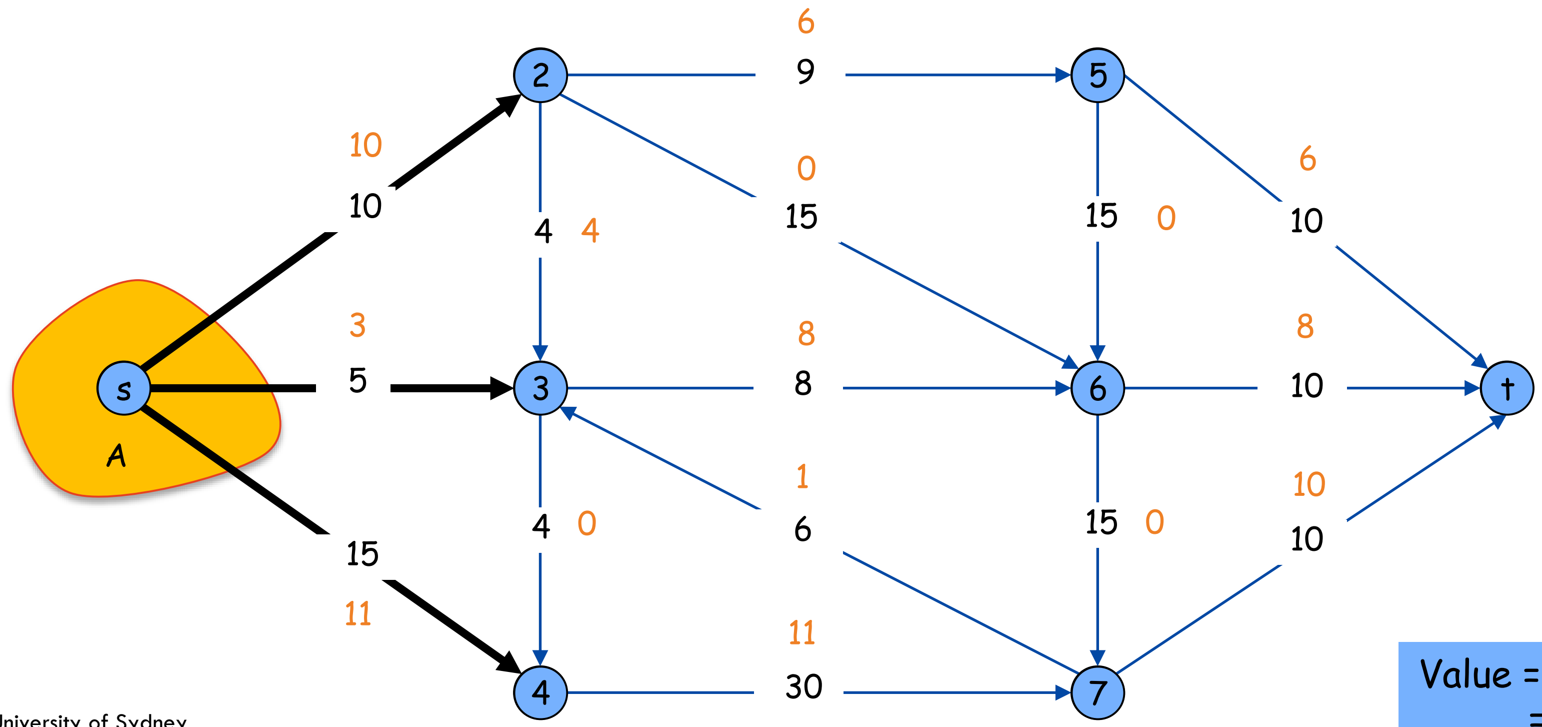


Capacity = 10 + 8 + 10
= 28

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.

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

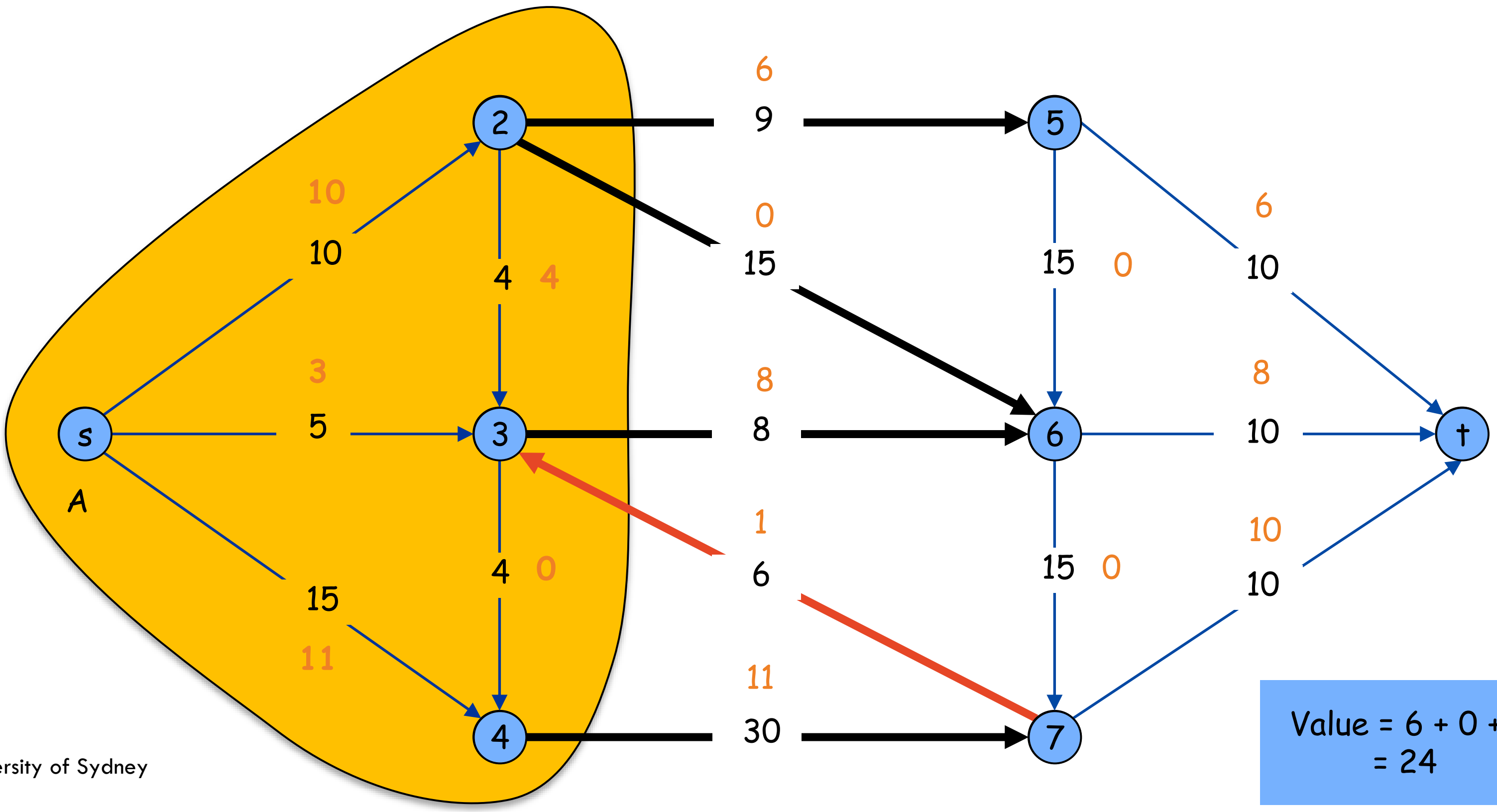


Value = 10+3+11 = 24

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.

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

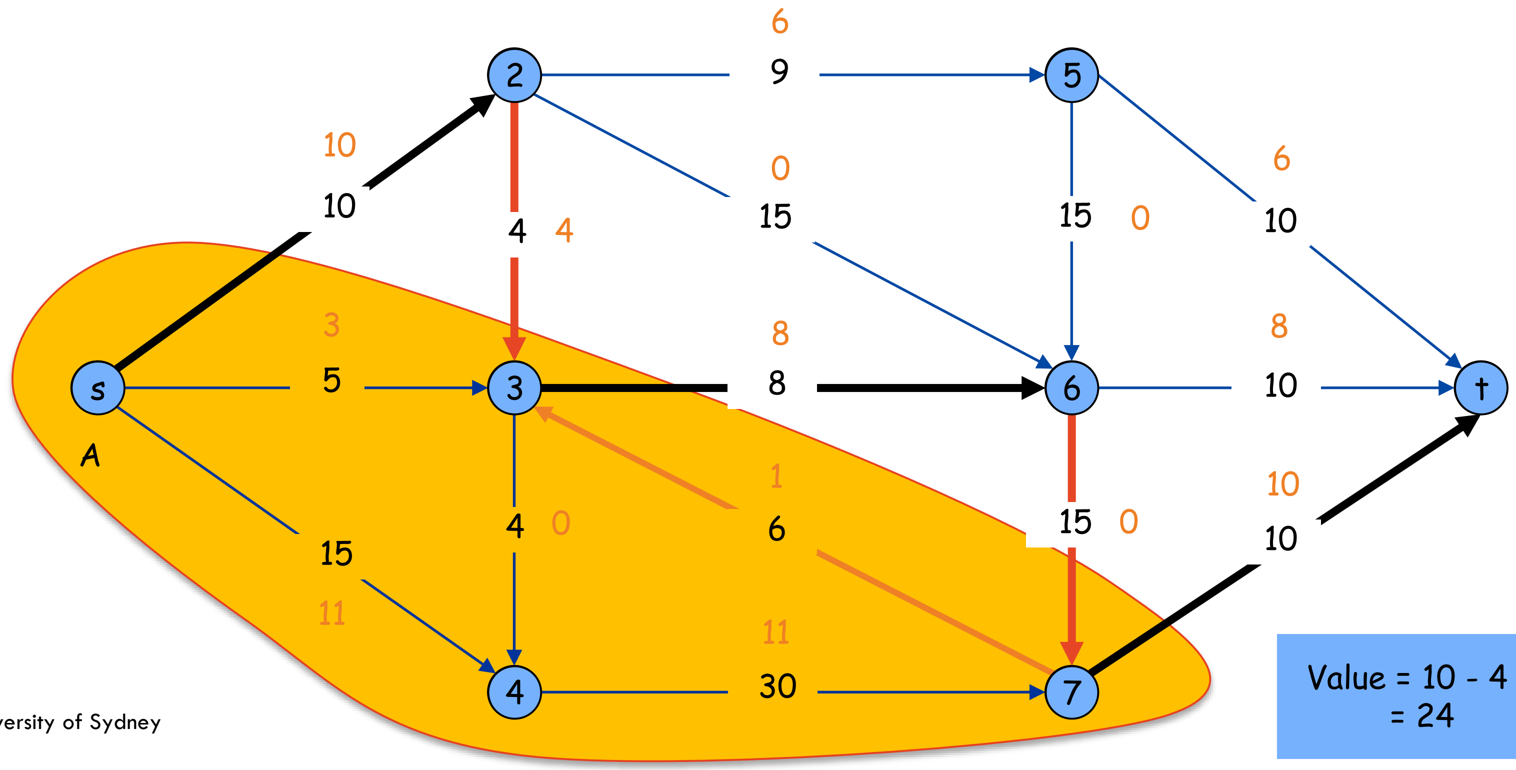


Value = 6 + 0 + 8 - 1 + 11
= 24

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.

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



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

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

Proof:

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

by flow conservation, all terms except v = s are 0, i.e. $f^{out}(v) - f^{in}(v) = 0$ $\longrightarrow$

$$= \sum_{v \in A} (f^{out}(v) - f^{in}(v))$$

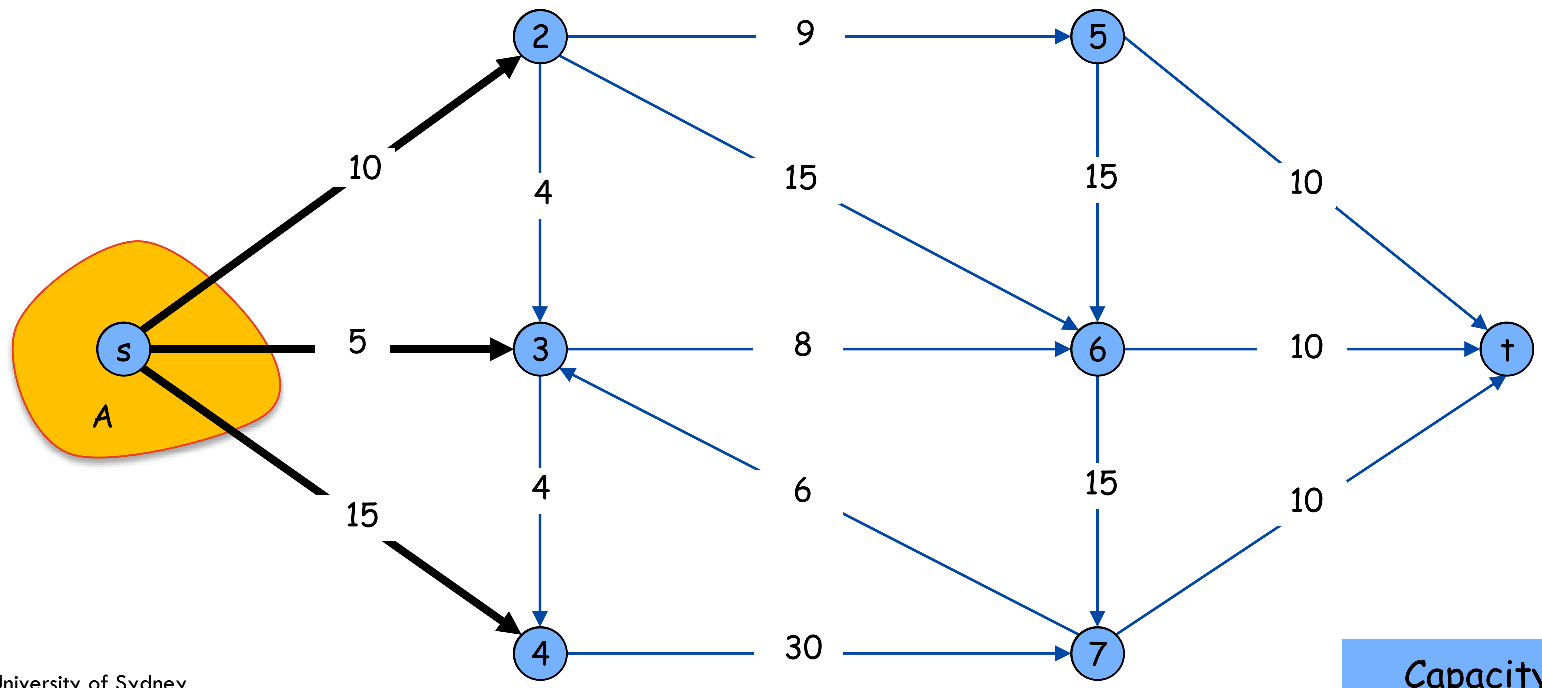
$$= \sum_{e \text{ out of } A} f(e) - \sum_{e \text{ into } A} f(e)$$

$$= f^{out}(A) - f^{in}(A)$$

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 ⇒ Flow value ≤ 30

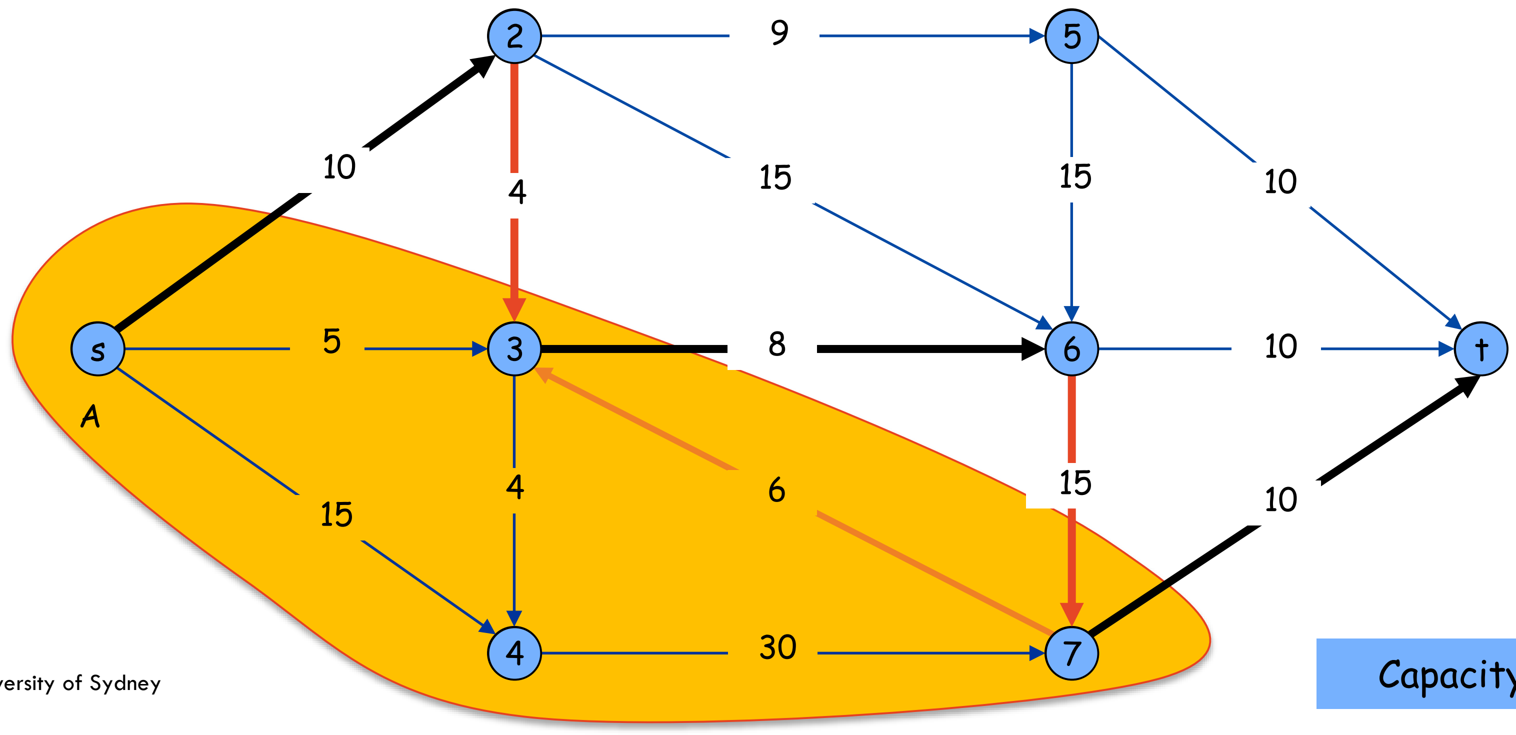


Capacity = 30

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 = 28 $\Rightarrow$ Flow value $\leq$ 28



Capacity = 28

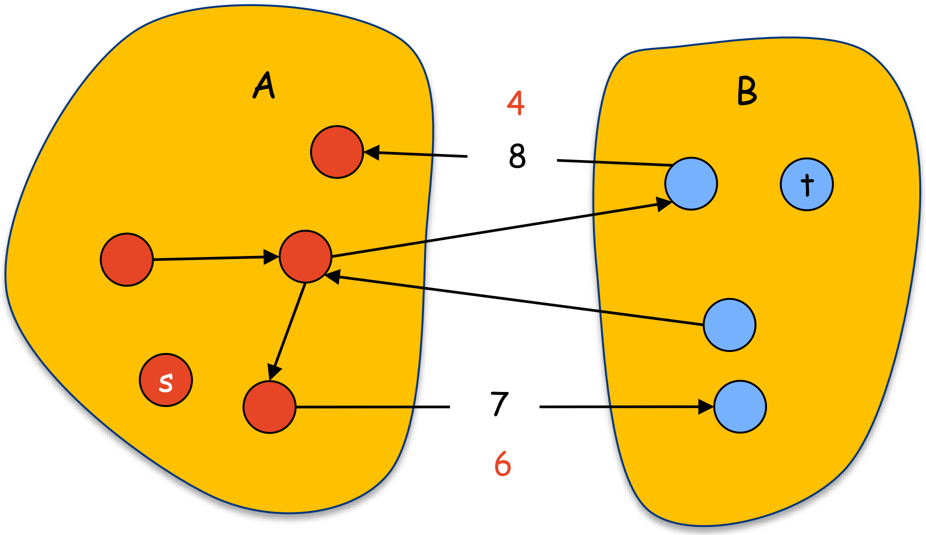
Flows and Cuts

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, i.e.,

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

Proof:

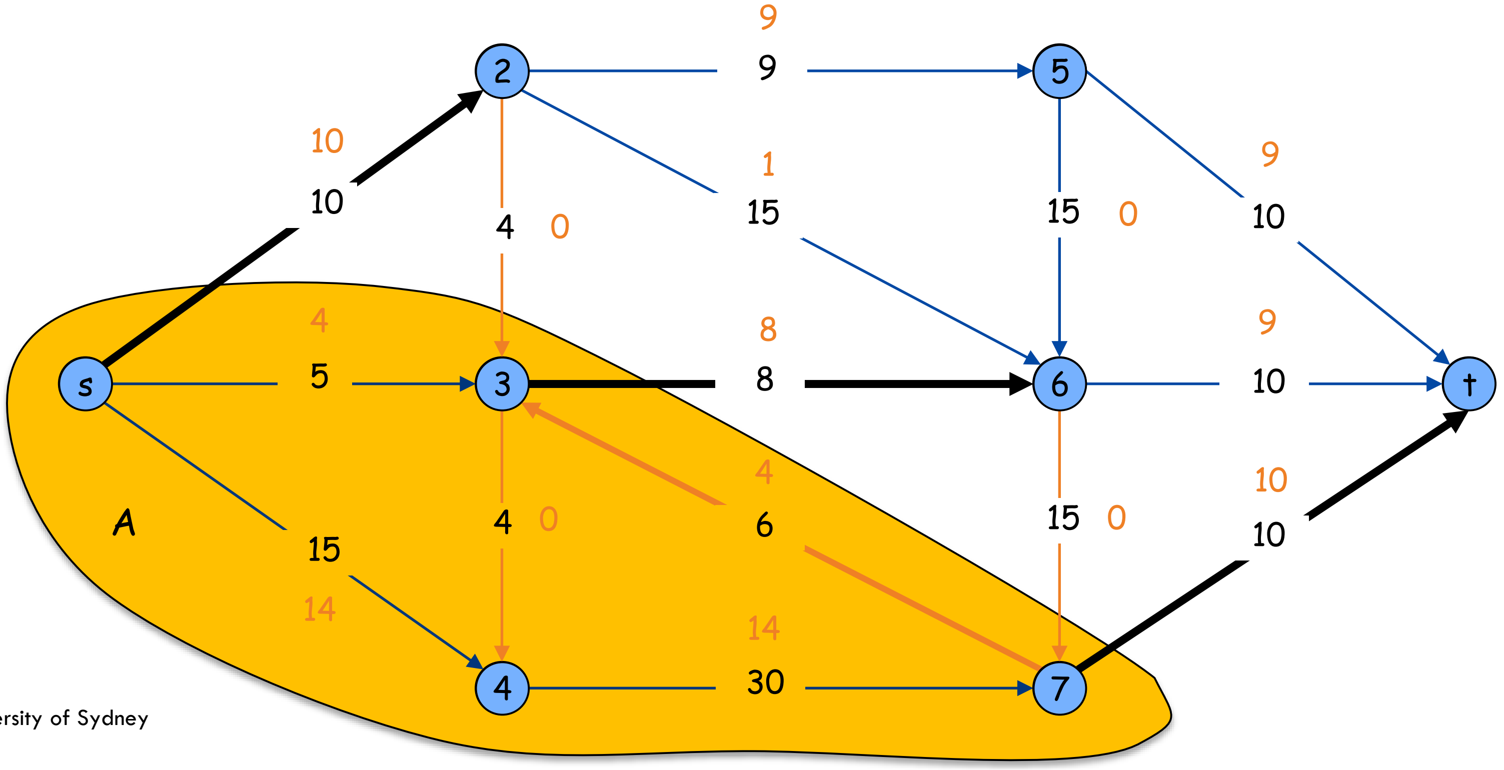
$$\begin{aligned} v(f) &= f^{out}(A) - f^{in}(A) \\ &\leq f^{out}(A) \\ &= \sum_{e \text{ out of } A} f(e) \\ &\leq \sum_{e \text{ out of } A} c(e) \\ &= c(A,B) \end{aligned}$$



Certificate of Optimality

Corollary: 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



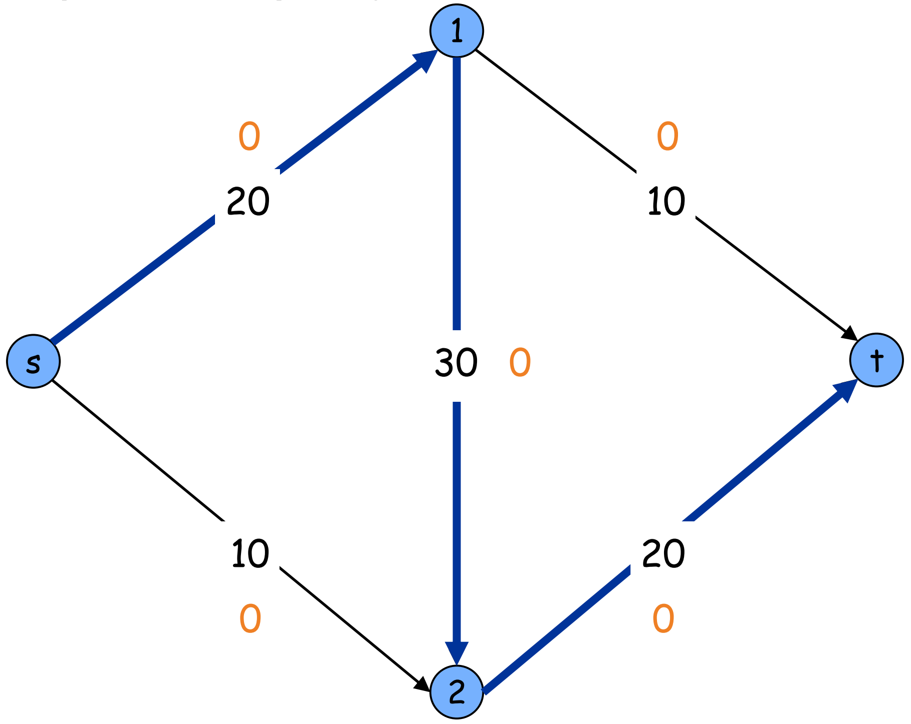
Summary (so far)

1. Max flow problem
2. Min cut problem
3. Theorem: Max flow $\leq$ Min cut

Towards a Max Flow Algorithm

Greedy algorithm.

- 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.

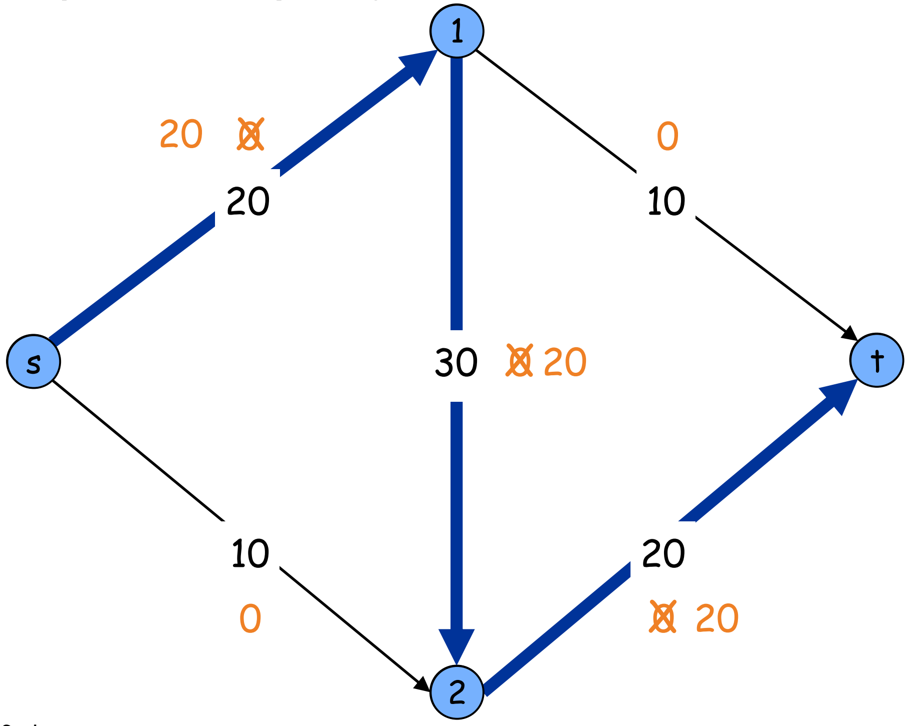


Flow value = 0

Towards a Max Flow Algorithm

Greedy algorithm.

- 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.

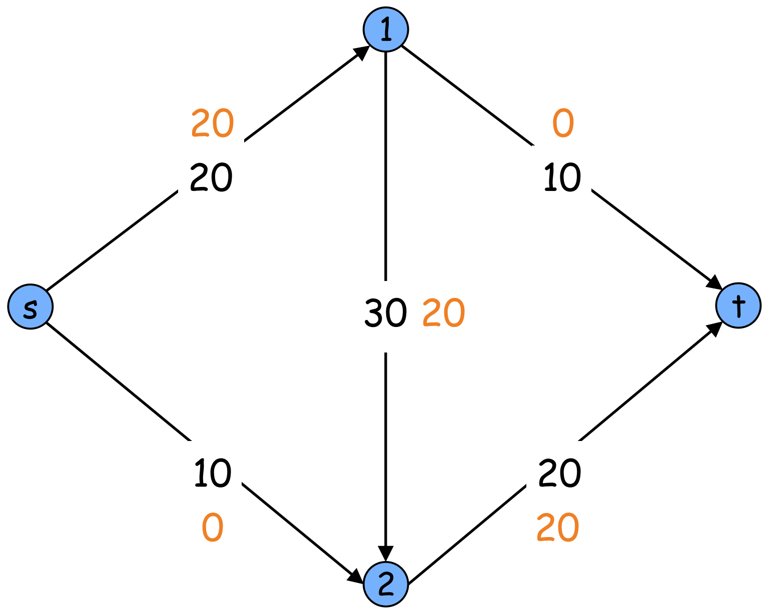


Flow value = 20

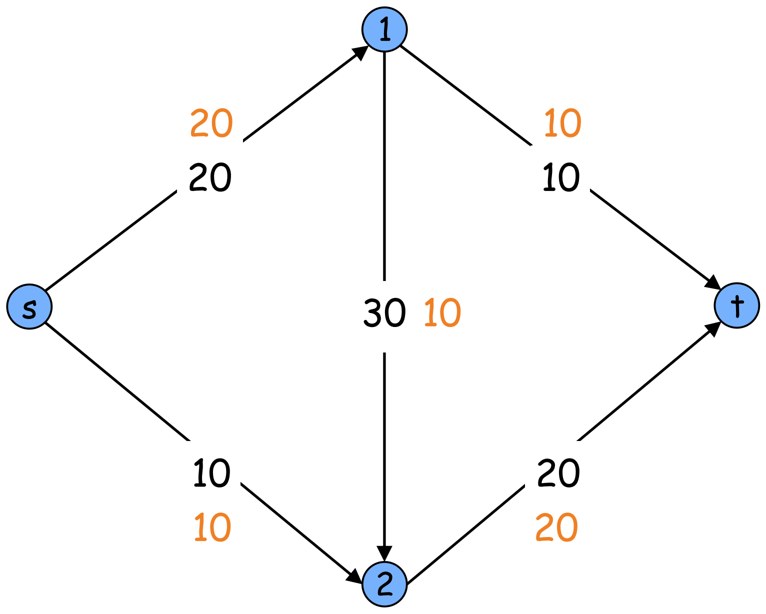
Towards a Max Flow Algorithm

Greedy algorithm.

- 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.



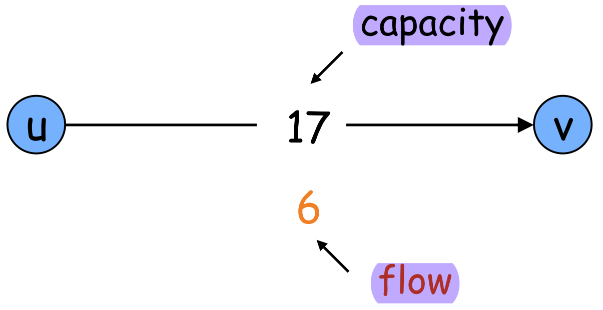
greedy = 20



opt = 30

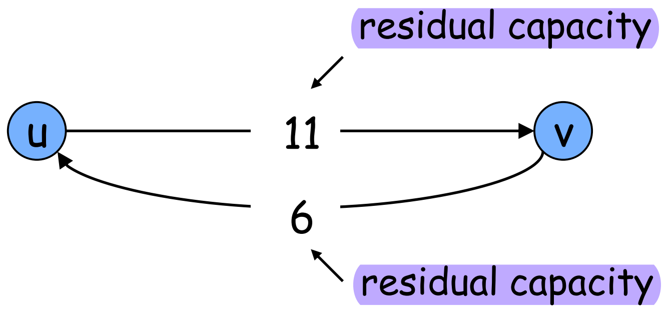
Build a Residual Graph $G_f = (V, E_f)$

- 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:

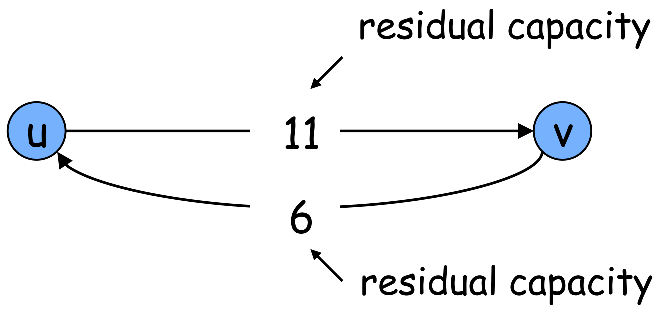
$$c_f(e) = \begin{cases} c(e) - f(e) & \text{if } e \in E \\ f(e) & \text{if } e^R \in E \end{cases}$$



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

Build a Residual Graph $G_f = (V, E_f)$

- The residual capacity of an edge in G_f tells us how much flow we can send, given the current flow.

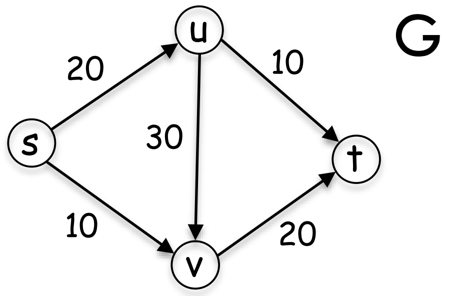


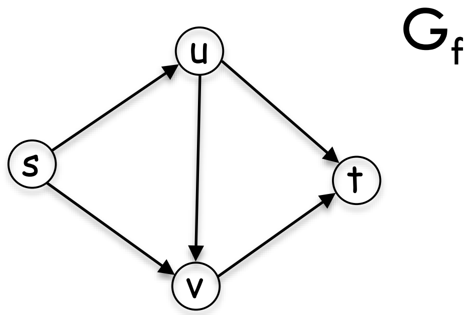
Augmenting Path Algorithm

Notations:

P = a simple s-t path in G_f

bottleneck(P,f) = minimum residual capacity of any edge on P with respect to the current flow f.



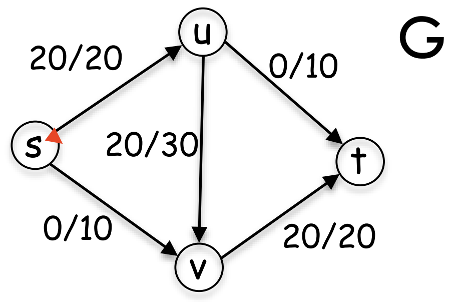


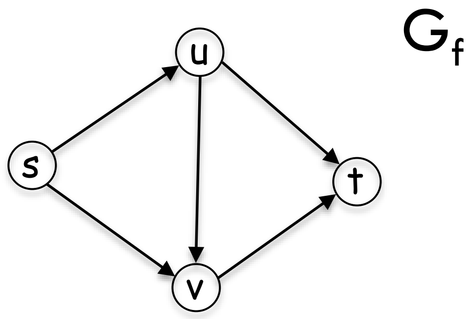
Augmenting Path Algorithm

Notations:

P = a simple s-t path in G_f

bottleneck(P,f) = minimum residual capacity of any edge on P with respect to the current flow f.



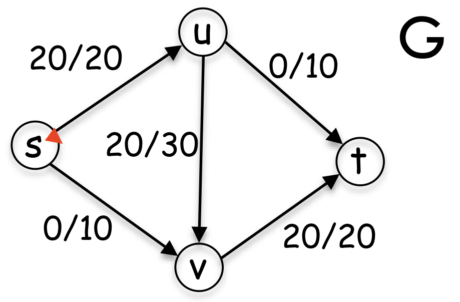


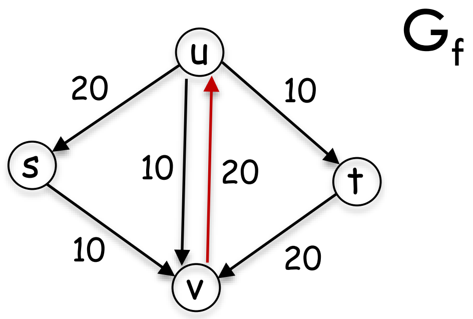
Augmenting Path Algorithm

Notations:

P = a simple s-t path in G_f

bottleneck(P,f) = minimum residual capacity of any edge on P with respect to the current flow f.





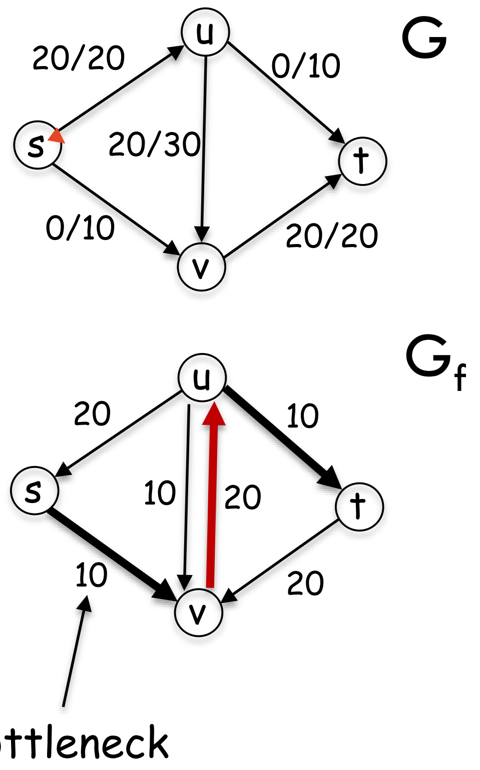
Augmenting Path Algorithm

Notations:

P = a simple s-t path in G_f

bottleneck(P,f) = minimum residual capacity of any edge on P with respect to the current flow f.

```
Augment(f,P) {
   b ← bottleneck(P,f)
   foreach e =(u,v) ∈ P {
      if e is a forward edge then
         increase f(e) in G by b
      else (e is a backward edge)
         decrease f(e) in G by b
   }
   return f
}
```



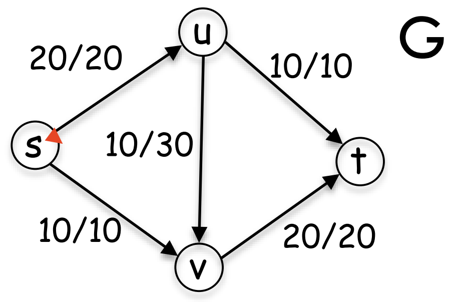
Augmenting Path Algorithm

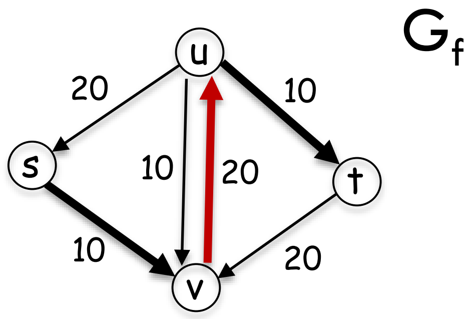
Notations:

P = a simple s-t path in G_f

bottleneck(P,f) = minimum residual capacity of any edge on P with respect to the current flow f.

```
Augment(f,P) {
   b ← bottleneck(P,f)
   foreach e =(u,v) ∈ P {
      if e is a forward edge then
         increase f(e) in G by b
      else (e is a backward edge)
         decrease f(e) in G by b
   }
   return f
}
```





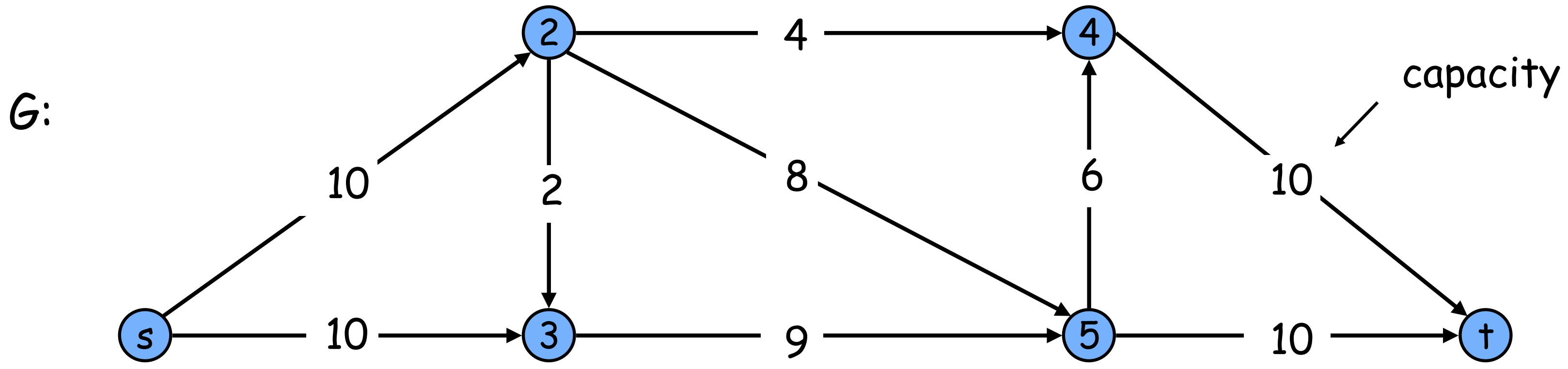
Augment(f,P) gives a new flow f' in G

Augmenting Path Algorithm

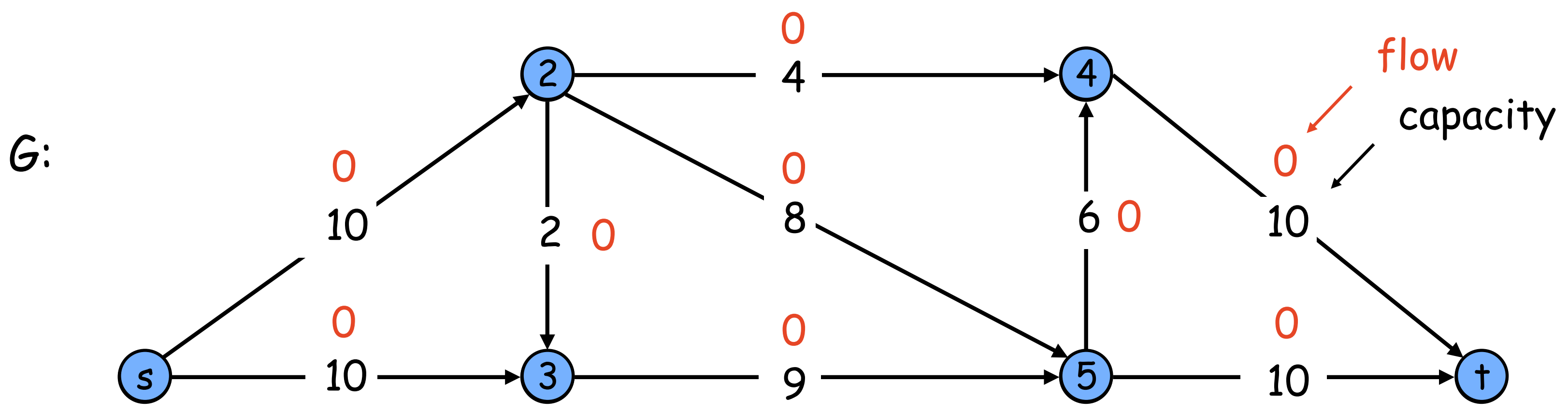
```
Ford-Fulkerson(G,s,t) {
   foreach e ∈ E
        f(e) ← 0
   G_f ← residual graph

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

```
Augment(f,P) {
   b ← bottleneck(P,f)
   foreach e =(u,v) ∈ P {
      if e is a forward edge then
         increase f(e) in G by b
      else (e is a backward edge)
         decrease f(e) in G by b
   }
   return f
}
```

Ford-Fulkerson Algorithm



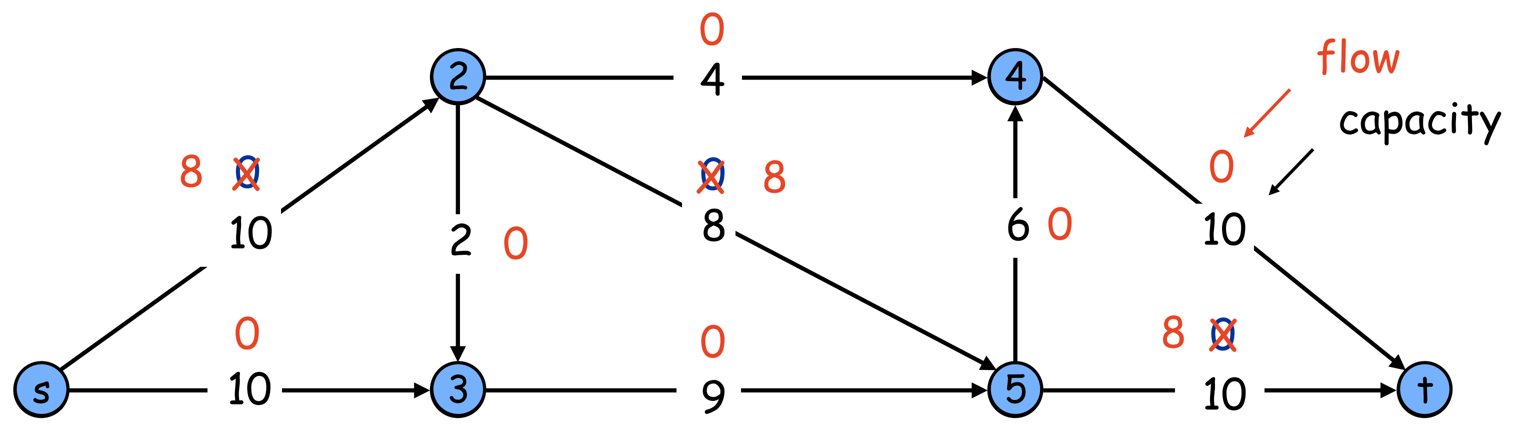
Ford-Fulkerson Algorithm



Flow value = 0

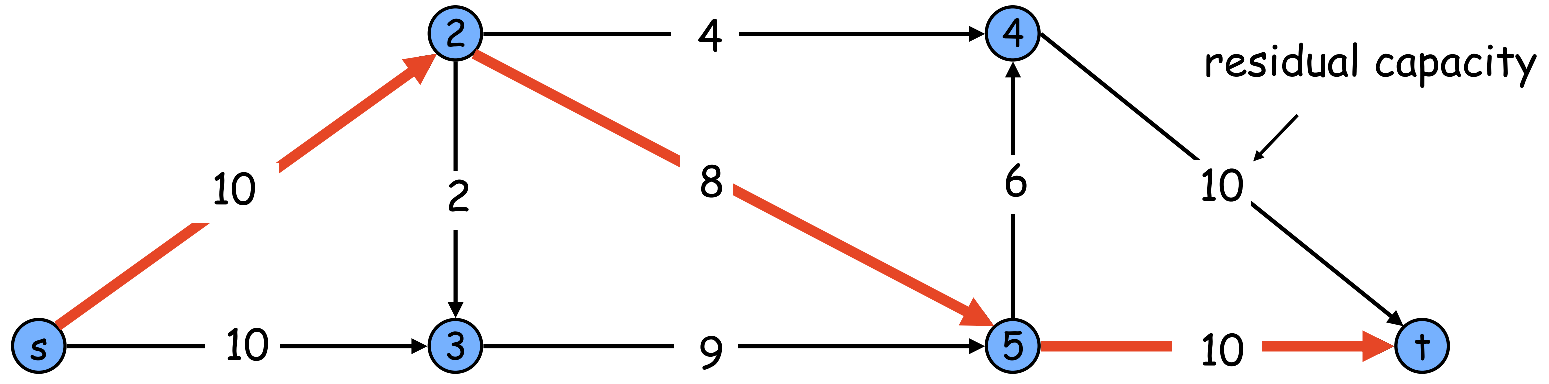
Ford-Fulkerson Algorithm

G:

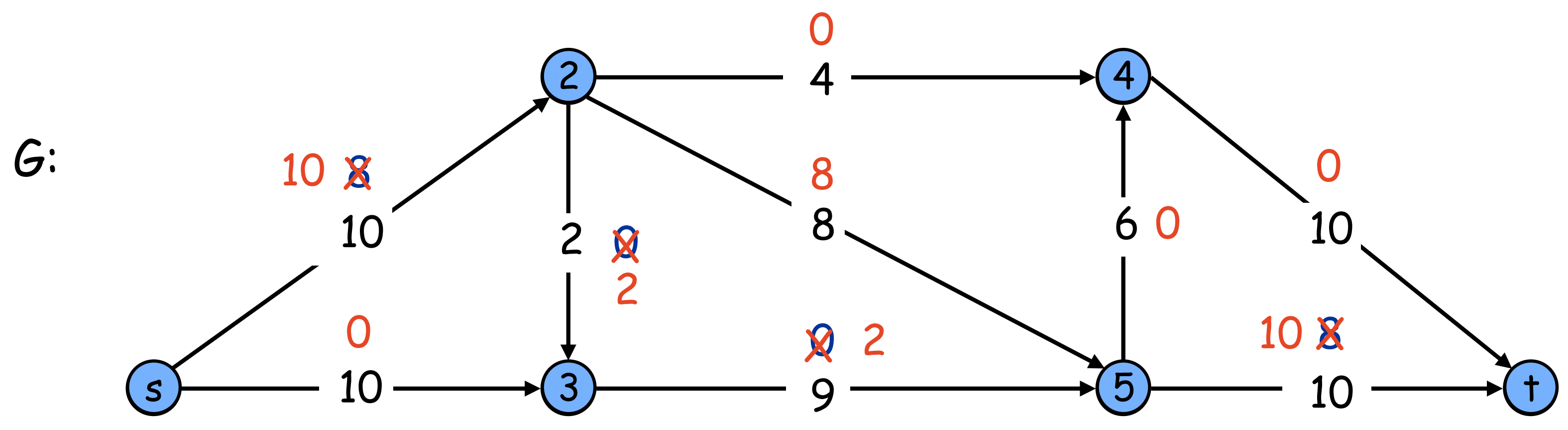


Flow value = 0

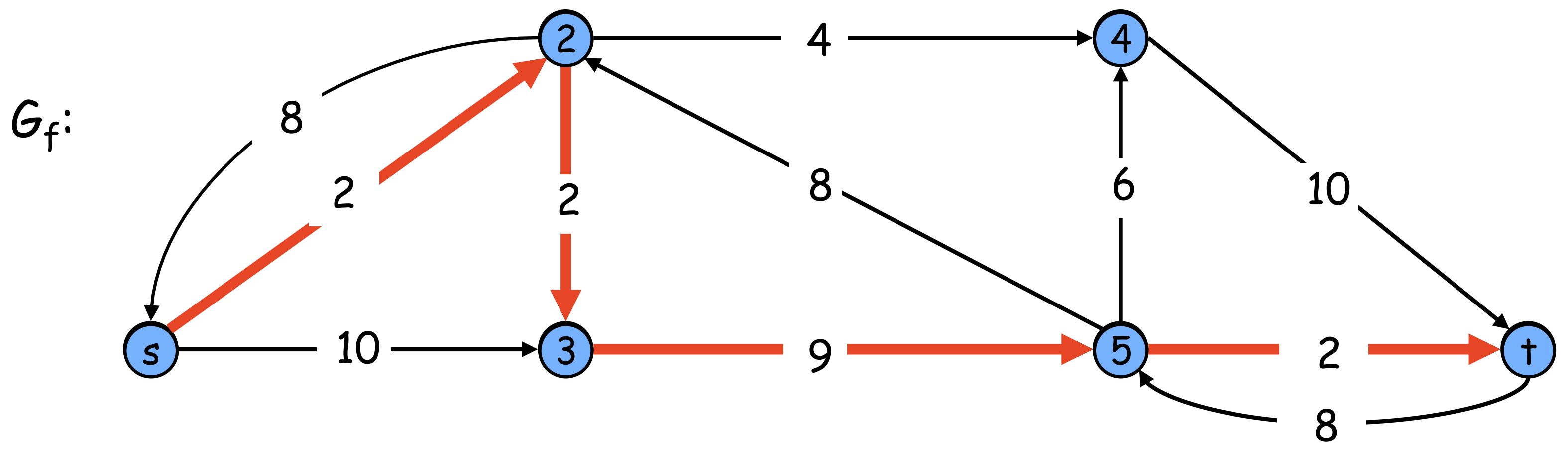
Gf:



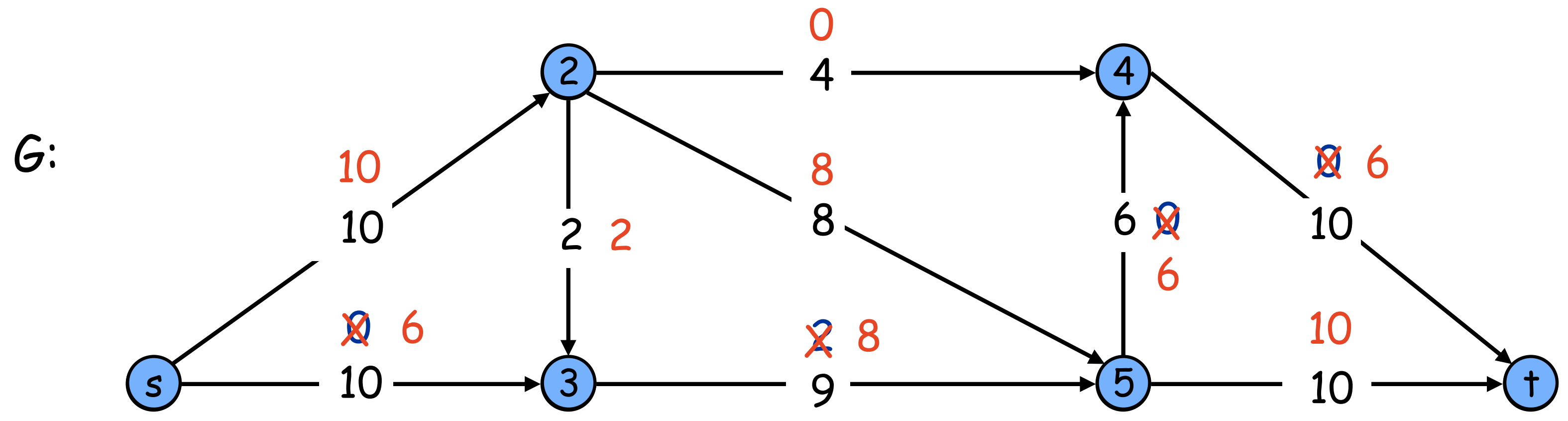
Ford-Fulkerson Algorithm



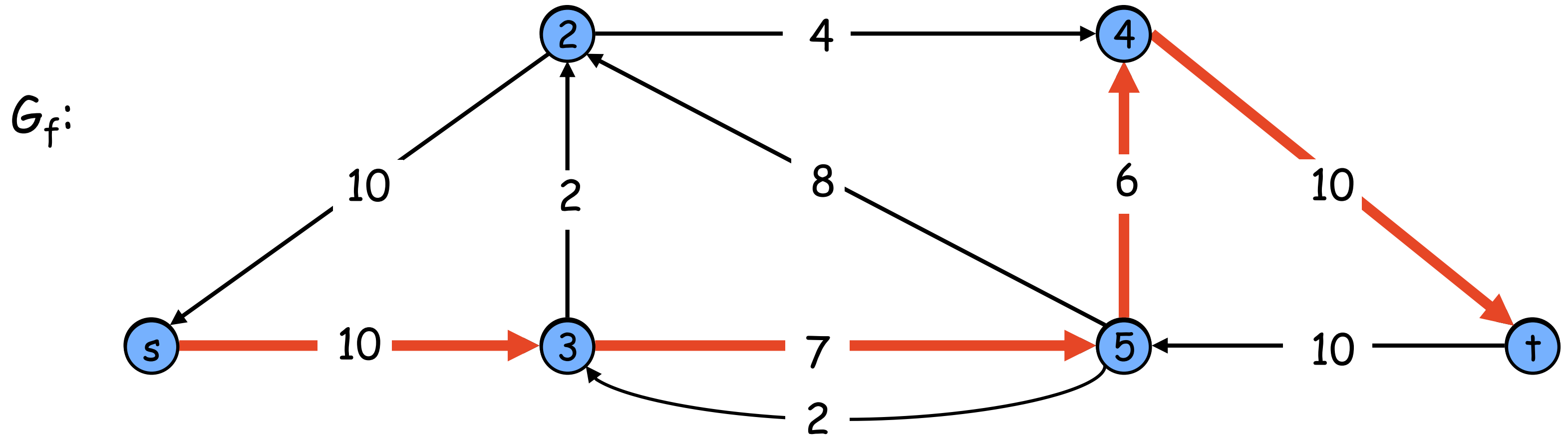
Flow value = 8



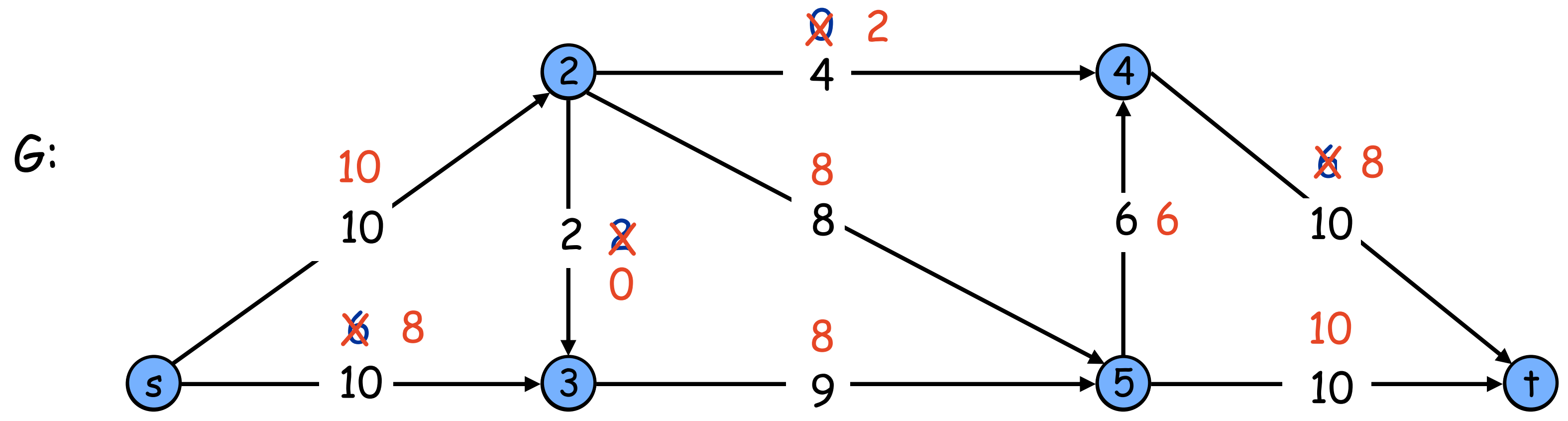
Ford-Fulkerson Algorithm



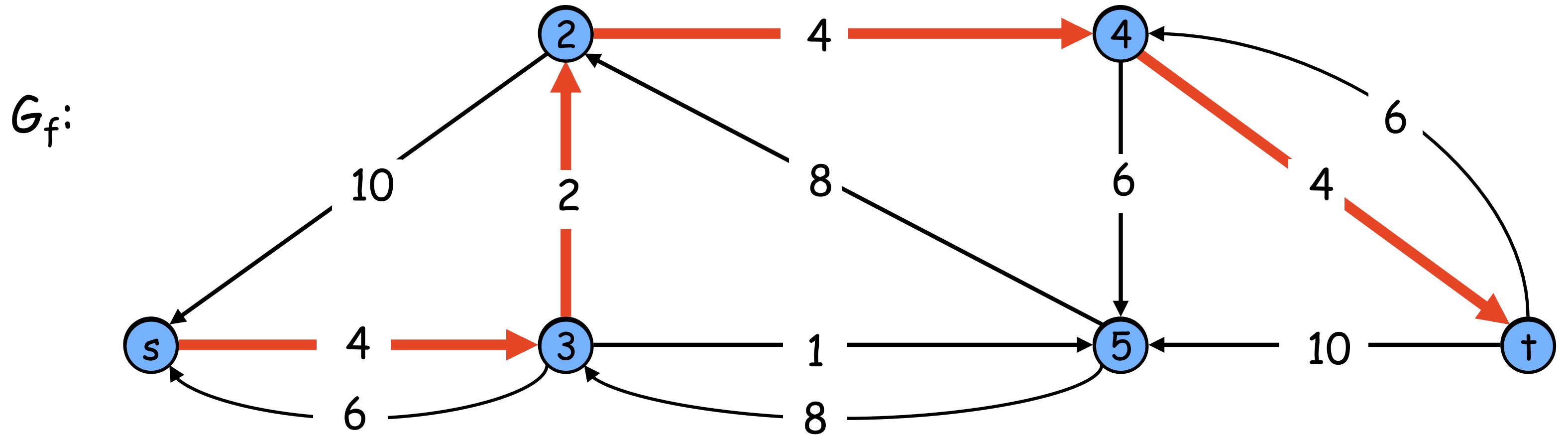
Flow value = 10



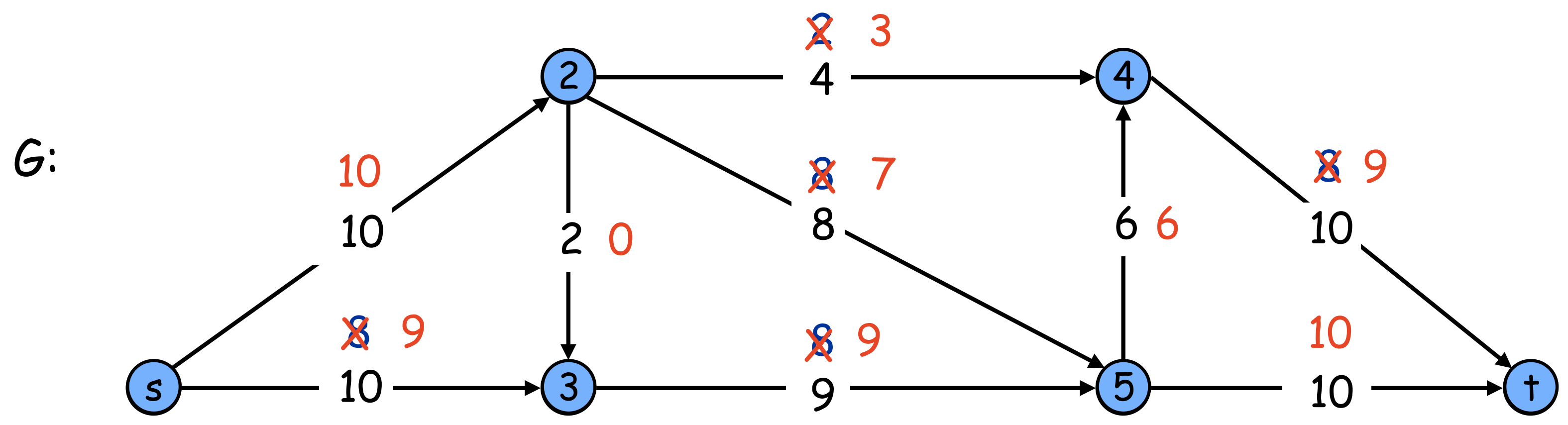
Ford-Fulkerson Algorithm



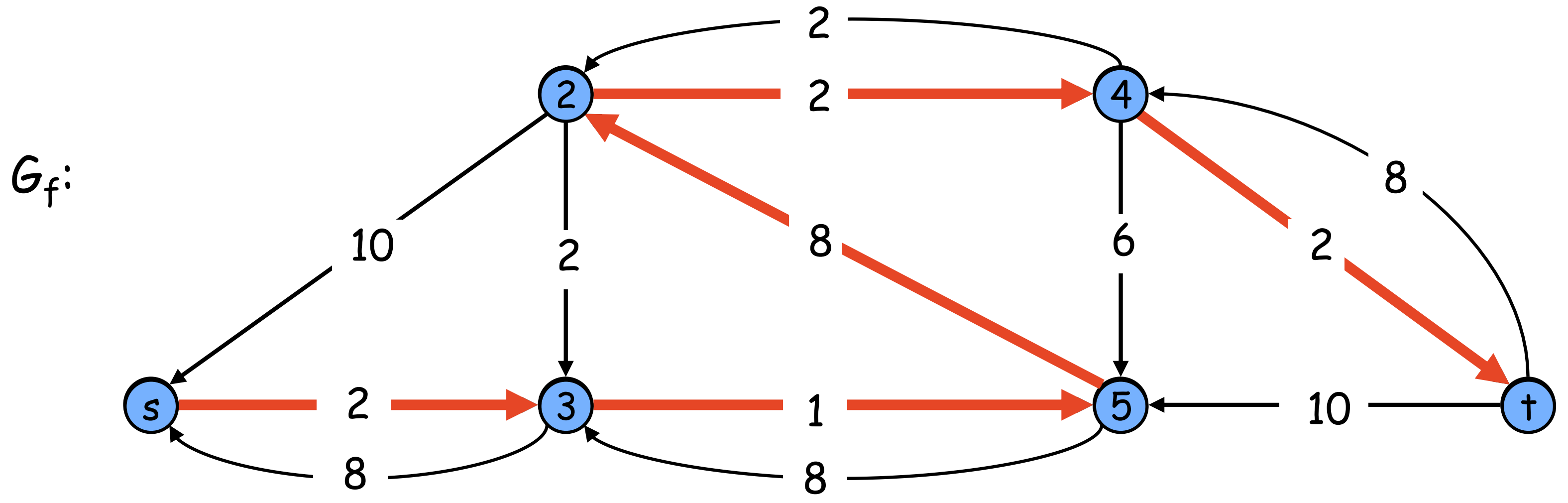
Flow value = 16



Ford-Fulkerson Algorithm

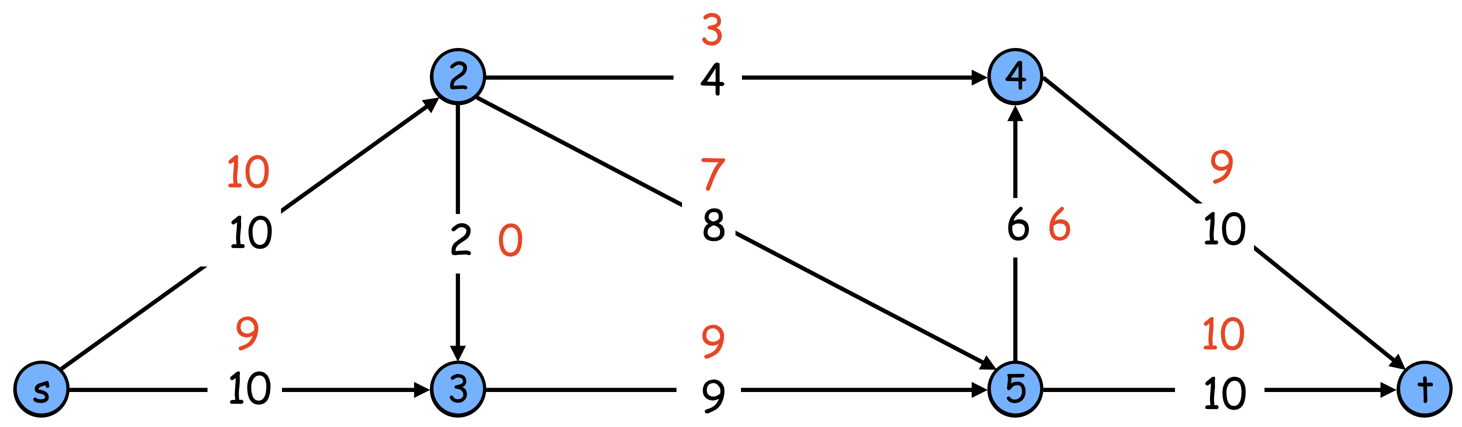


Flow value = 18



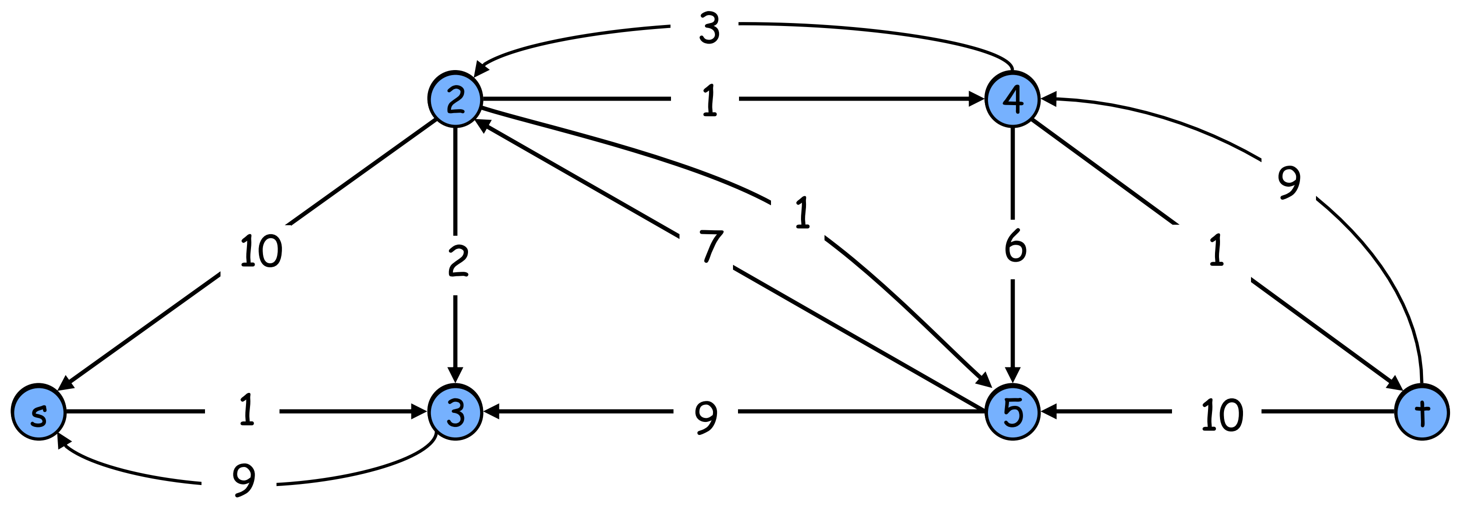
Ford-Fulkerson Algorithm

G:

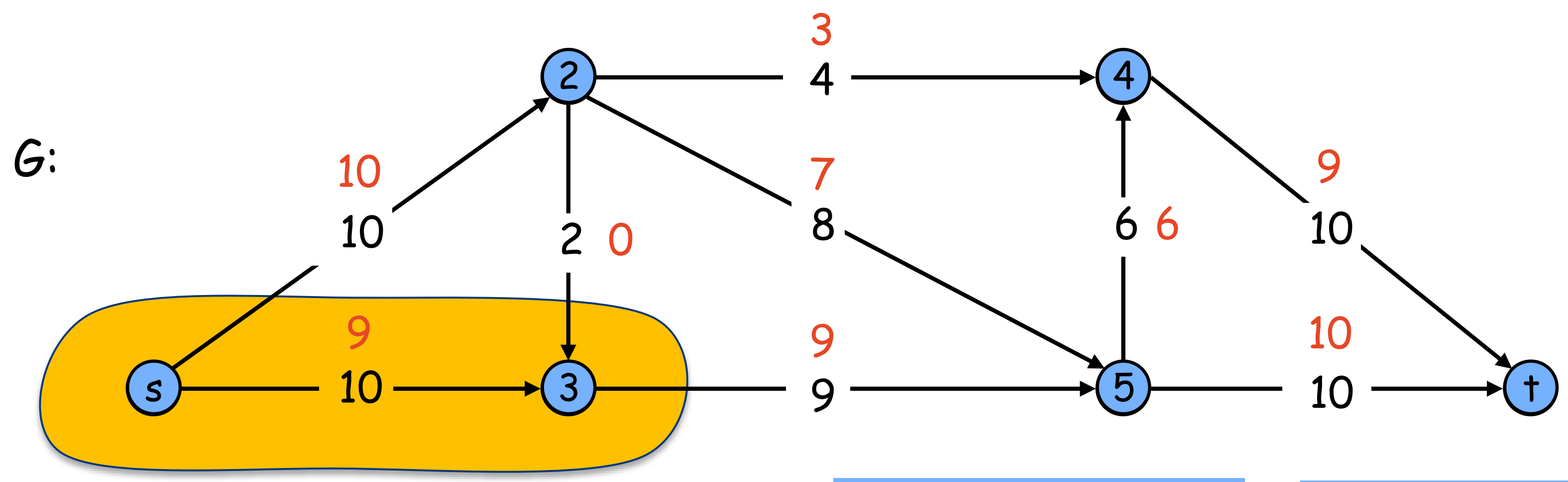


Flow value = 19

Gf:

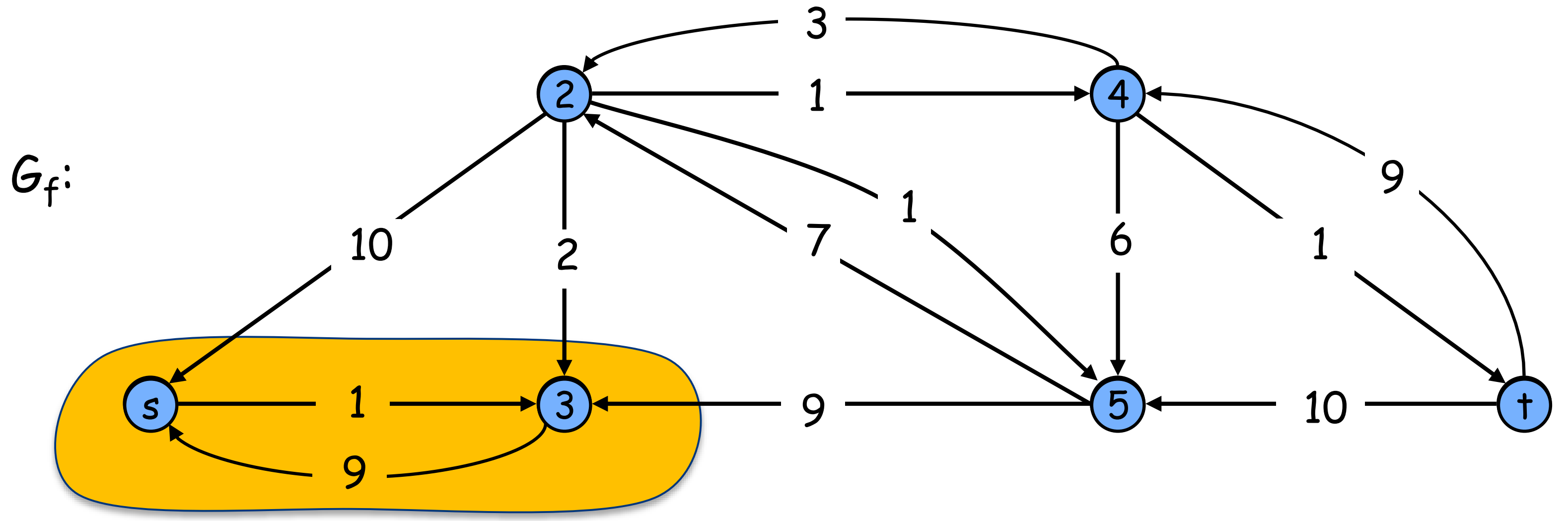


Ford-Fulkerson Algorithm



Cut capacity = 19

Flow value = 19



Augmenting Path Algorithm

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

   while (there exists augmenting path P in $G_f$){
      f ← Augment(f,P)
      update $G_f$
   }
   return f
}
```

```
Augment(f,P) {
   b ← bottleneck(P,f)
   foreach e =(u,v) ∈ P {
      if e is a forward edge then
         increase f(e) in G by b
      else (e is a backward edge)
         decrease f(e) in G by b
   }
   return f
}
```

Ford-Fulkerson: Analysis

Assumption. All initial capacities are integers.

Lemma. At every intermediate stage of the Ford-Fulkerson algorithm the flow values and the residual graph capacities in G_f are integers.

Proof: (proof by induction)

Base case: Initially the statement is correct.

Induction hyp.: True after j iterations.

Induction step: Since all the residual capacities in G_f are integers the bottleneck-value must be an integer. Thus the flow will have integer values and hence also the capacities in the new residual graph.

Integrality theorem. If all capacities are integers, then there exists a max flow f for which every flow value f(e) is an integer.

Max-Flow Min-Cut Theorem

Augmenting path theorem: Flow f is a max flow if and only if there are no augmenting paths.

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

Proof strategy: We prove both simultaneously.

(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 the weak duality lemma.

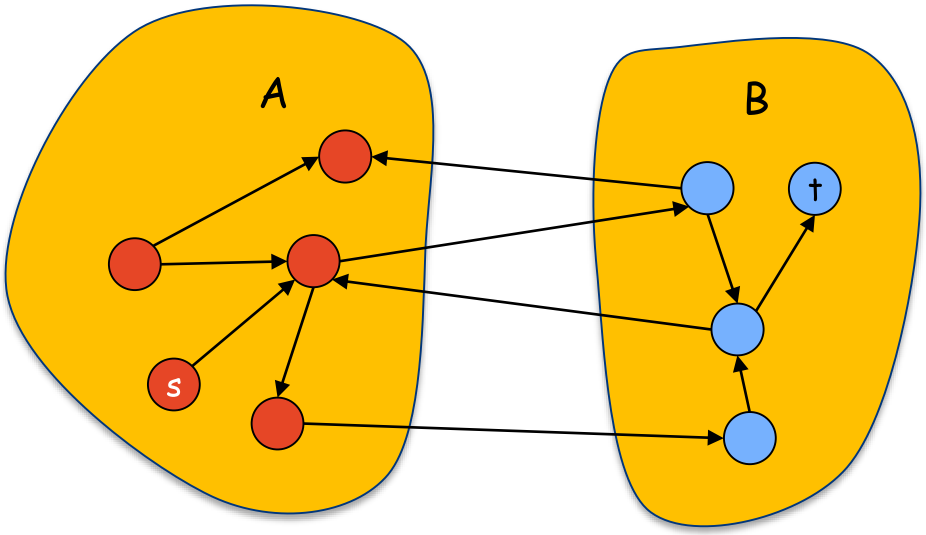
– (ii) $\Rightarrow$ (iii) We show contrapositive.
  – Let f be a flow. If there exists an augmenting path, then we can improve f by sending flow along a path P and augment the flow in G.

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.

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

Since there is no augmenting path from A to B



original network

Ford-Fulkerson: Running Time

Observation:

Let f be a flow in G, and let P be a simple s-t path in G_f.

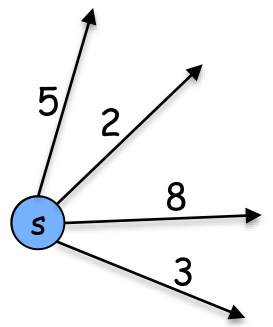
$$v(f') = v(f) + \text{bottleneck}(f,P)$$

and since bottleneck(f,P)>0

$$v(f') > v(f).$$

$\Rightarrow$ The flow value strictly increases in an augmentation

Ford-Fulkerson: Running Time



Notation: $C = \sum_{e \text{ out of } s} c(e)$

Observation: C is an upper bound on the maximum flow.

Theorem. The algorithm terminates in at most $v(f_{max}) \leq C$ iterations.

Proof: Each augmentation increase flow value by at least 1.

Ford-Fulkerson: Running Time

Corollary:

Ford-Fulkerson runs in O((m+n)C) time, if all capacities are integers.

Proof: C iterations.

Path in G_f can be found in O(m+n) time using BFS.

Augment(P,f) takes O(n) time.

Updating G_f takes O(m+n) time.

7.3 Choosing Good Augmenting Paths

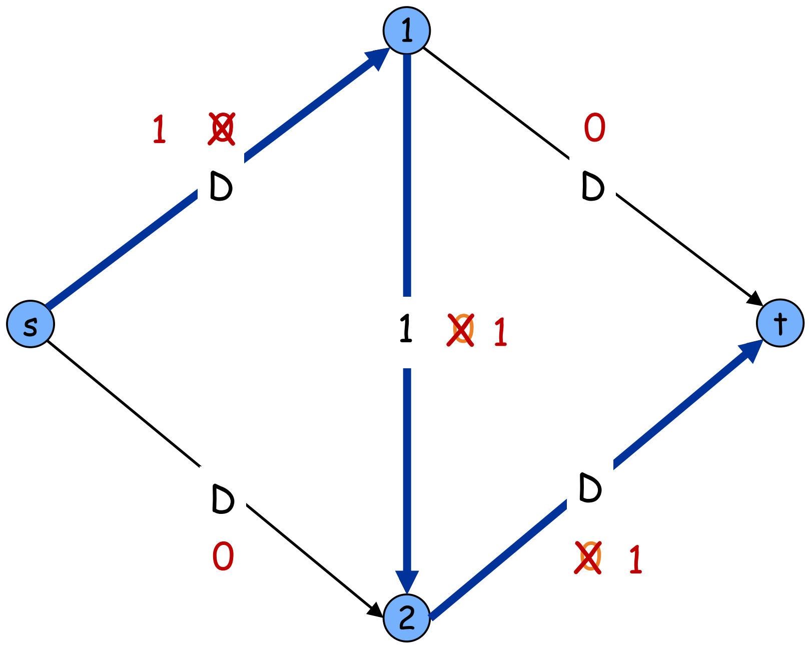
Is O(C(m+n)) a good time bound?

- Yes, if C is small.
- If C is large, can the number of iterations be as bad as C?

Ford-Fulkerson: Exponential Number of Augmentations

Question: Is generic Ford-Fulkerson algorithm polynomial in input size? ← m, n, and log C

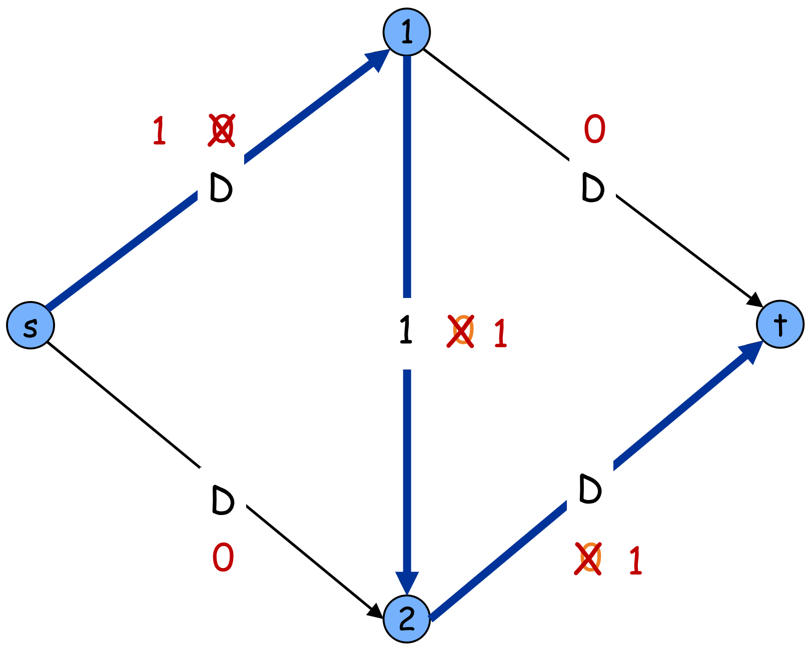
Answer: No. If max capacity is D, then algorithm can take D iterations.

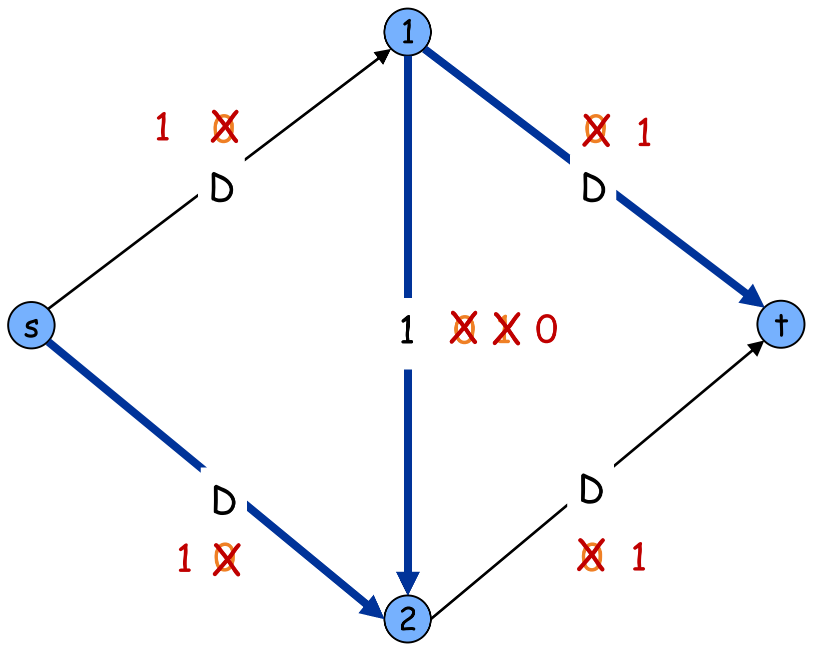


Ford-Fulkerson: Exponential Number of Augmentations

Question: Is generic Ford-Fulkerson algorithm polynomial in input size? ← m, n, and log C

Answer: No. If max capacity is D, then algorithm can take D iterations.





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.

Choosing Good Augmenting Paths

- Ford Fulkerson

  Choose any augmenting path (C iterations)

- Edmonds Karp #1 (m log C iterations)

  Choose max flow path

- Improved Ford Fulkerson (log C iterations)

  Choose max flow path

- Edmonds Karp #2 (O(nm) iterations)

  Choose minimum link path [Edmonds-Karp 1972, Dinitz 1970]

Edmonds-Karp #1

Pick the augmenting path with largest capacity

[maximum bottleneck path]

Edmonds-Karp #1

Pick the augmenting path with largest capacity

[maximum bottleneck path]

Claim: If maximum flow in G is F, there must exists a path from s to t with capacity at least F/m.

Edmonds-Karp #1

Pick the augmenting path with largest capacity

[maximum bottleneck path]

Claim: If maximum flow in G is F, there must exists a path from s to t with capacity at least F/m.

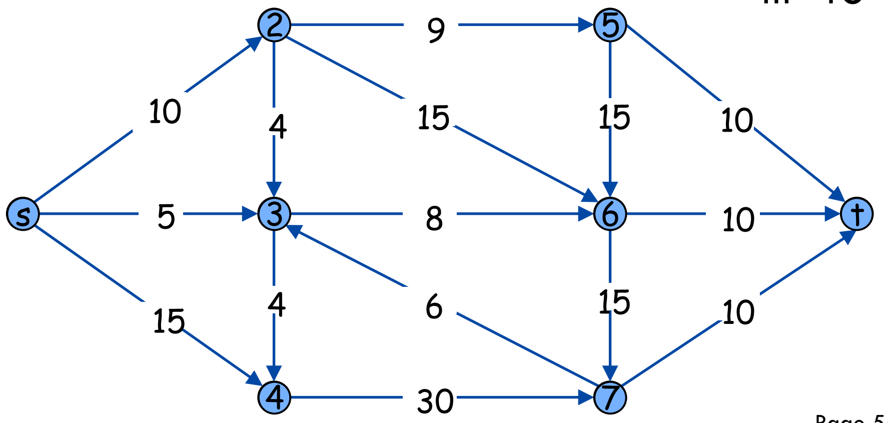
Proof:

Delete all edges of capacity less than F/m.

Is the graph still connected?

F=24

m=15



Edmonds-Karp #1

Pick the augmenting path with largest capacity

[maximum bottleneck path]

Claim: If maximum flow in G is F, there must exists a path from s to t with capacity at least F/m.

Proof:

Delete all edges of capacity less than F/m.

Is the graph still connected?

Yes, otherwise we have a cut of value less than F.
The remaining graph must have a path from s to t and since all edges have capacity at least F/m, the path itself has capacity at least F/m.

Edmonds-Karp #1

Theorem: Edmonds-Karp #1 makes at most O(m log F) iterations.

Proof:

At least 1/m of remaining flow is added in each iteration.

$\Leftrightarrow$

Remaining flow reduced by a factor of (1-1/m) per iteration.

#iterations until remaining flow <1? $\Rightarrow$ $F \cdot (1-1/m)^x < 1$?

We know: $(1-1/m)^m < 1/e$

Set $x = m \ln F$ $\Rightarrow$ $F \cdot (1-1/m)^{m \ln F} < F \cdot (1/e)^{\ln F} = 1$

Applications

- Bipartite matching
- Perfect matching
- Disjoint paths
- Network connectivity
- Circulation problems
- Image segmentation
- Baseball elimination
- Project selection

Summary

1. Max flow problem
2. Min cut problem
3. Ford-Fulkerson:
    1. Residual graph
    2. correctness
    3. complexity
4. Max-Flow Min-Cut theorem
5. Capacity scaling