A process cannot be understood by stopping it. Understanding must move with the flow of the process, must join it and flow with it.

— The First Law of Mentat, in Frank Herbert's *Dune* (1965)

Contrary to expectation, flow usually happens not during relaxing moments of leisure and entertainment, but rather when we are actively involved in a difficult enterprise, in a task that stretches our mental and physical abilities.... Flow is hard to achieve without effort. Flow is not "wasting time."

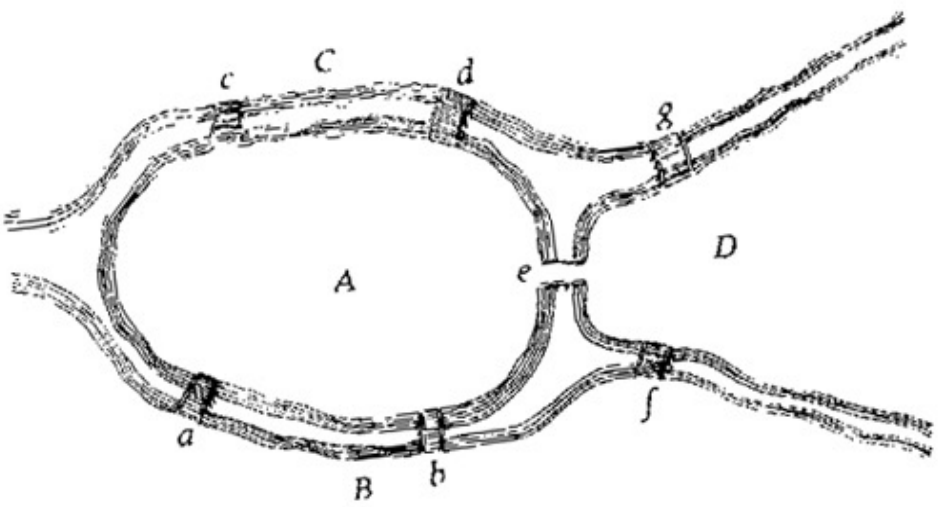
— Mihaly Csíkszentmihályi, *Flow: The Psychology of Optimal Experience* (1990)

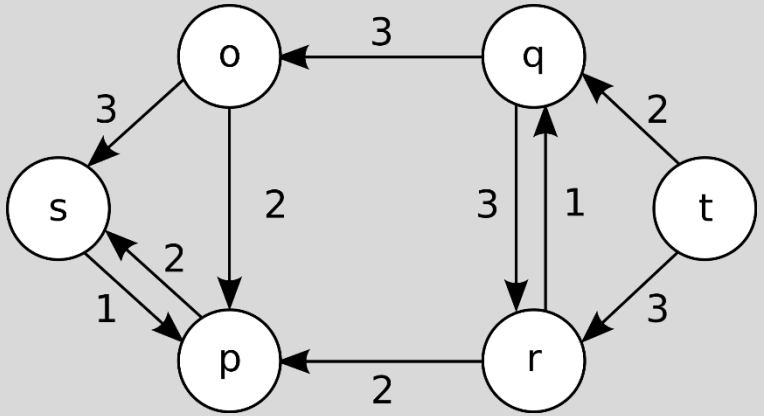
There's a difference between knowing the path and walking the path.

— Morpheus [Laurence Fishburne], *The Matrix* (1999)

From Jeff Erickson's
http://algorithms.wtf

Lecture 6 –
Network Flows (Theory)





THE UNIVERSITY OF SYDNEY

General techniques in this course

- Greedy algorithms [Lecture 2]
- Divide & Conquer algorithms [Lecture 3]
- Dynamic programming algorithms [Lectures 4 and 5]
- Network flow algorithms [today and Lecture 7-8]
  - Theory [today]
  - Applications [Lectures 7-8]

Algorithmic Paradigms

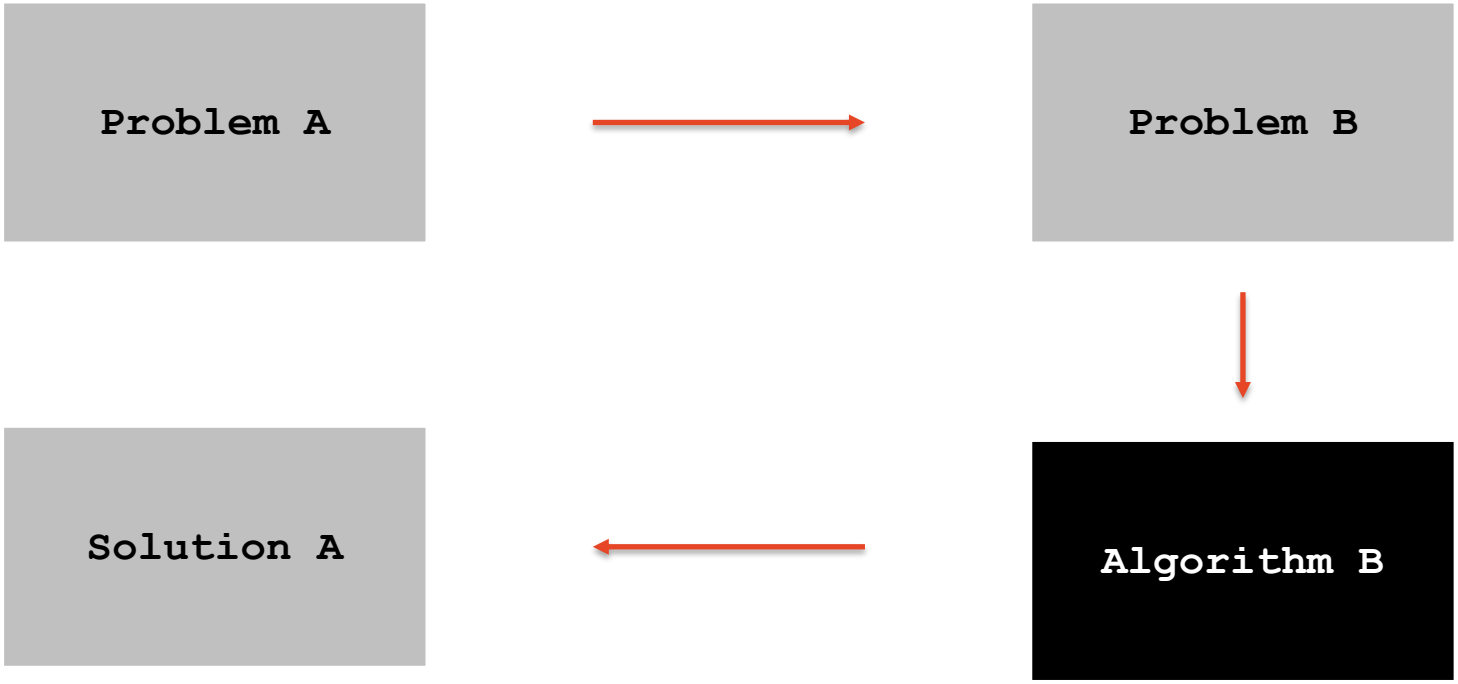
- **Greedy.** Build up a solution incrementally, myopically optimizing some local criterion.

- **Divide-and-conquer.** Break up a problem into two sub-problems, solve each sub-problem *independently*, and combine solution to sub-problems to form solution to original problem.

- **Dynamic programming.** Break up a problem into a series of *overlapping* sub-problems, and build up solutions to larger and larger sub-problems.

- **Network flows.**

Reduction: Powerful Idea in Computer Science

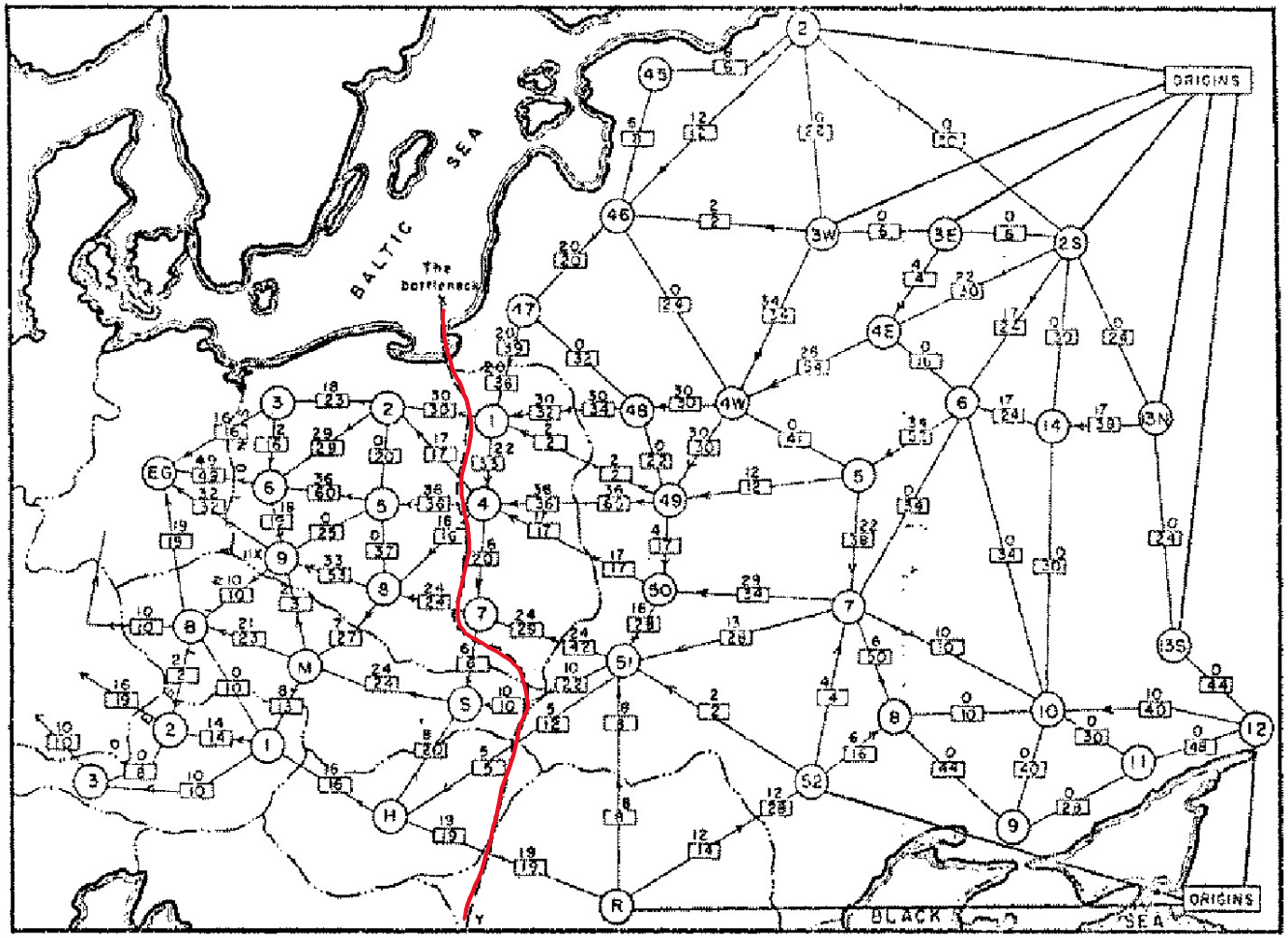


Problem B is a smaller instance of Problem A: Divide-and-Conquer, Dynamic programming OR

Problem B is easier than Problem A: Network Flows, NP-hardness

Soviets : Max flow
NATO : Min cut

Soviet Rail Network, 1955



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

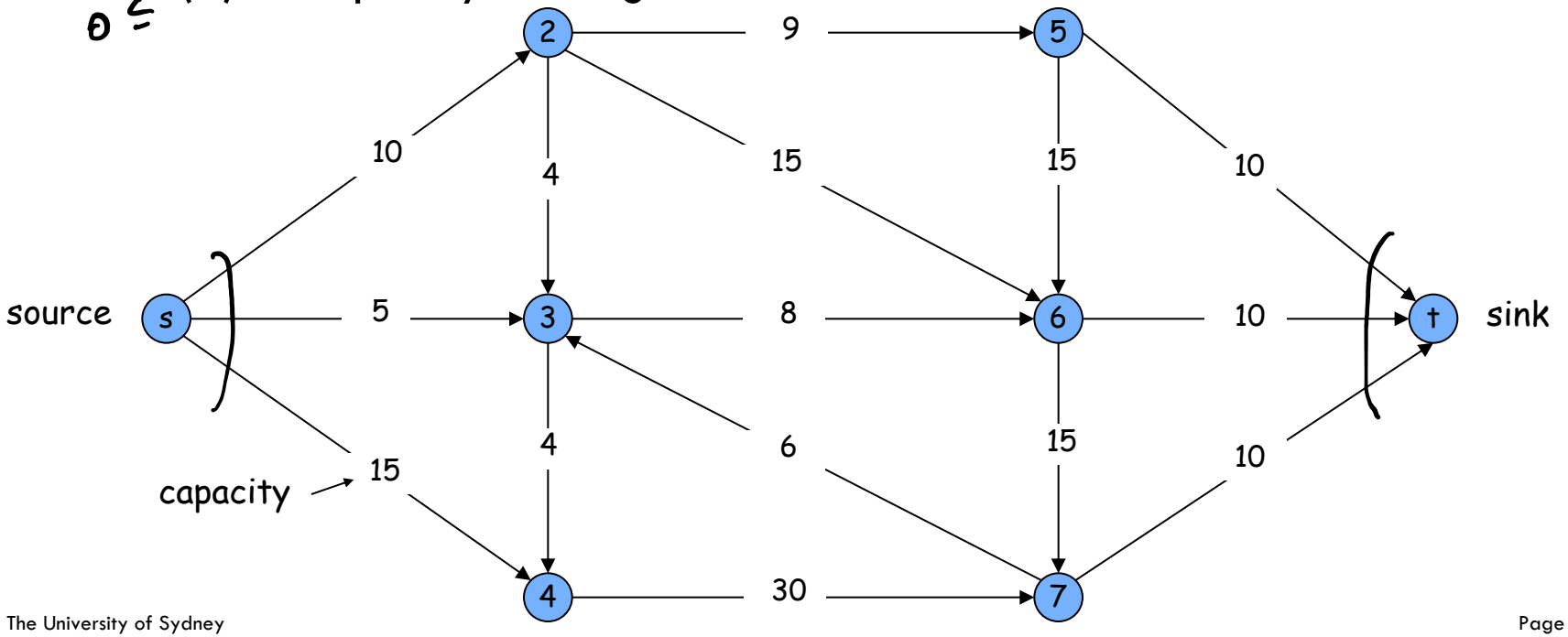
Maximum Flow and Minimum Cut

- Max flow and min cut.
  - Two very rich algorithmic problems.
  - Cornerstone problems in combinatorial optimization.
  - Mathematical duality / certificate of optimality
  - Special case of Linear Programming

- 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.



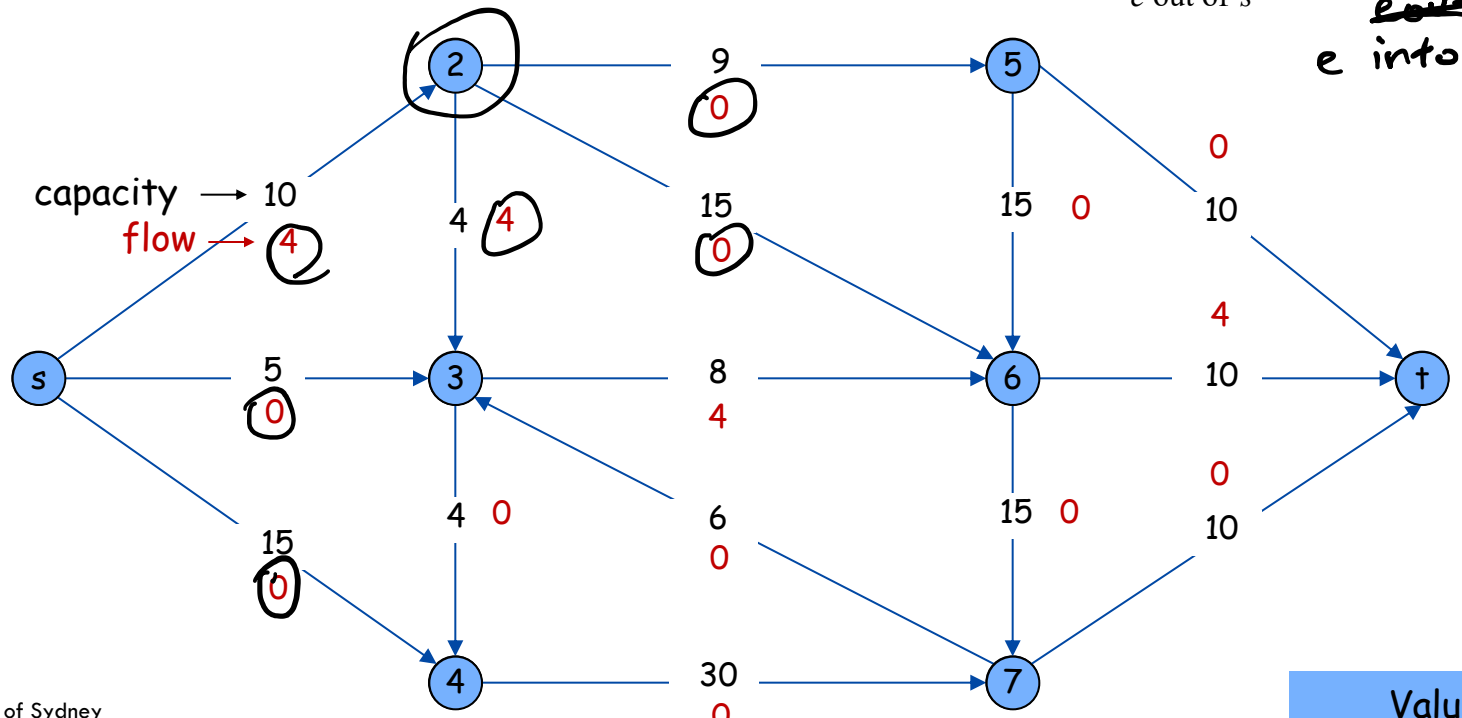
Flows

- **Definition:** An s-t flow is a function that satisfies:
  - For each e ∈ E: $0 \leq f(e) \leq c(e)$ (capacity)
  - We say e is saturated if f(e) = c(e)
  - For each v ∈ V – {s, t}: $\sum_{e \text{ in to } v} f(e) = \sum_{e \text{ out of } v} f(e)$ (conservation)

outgoing flow

incoming flow

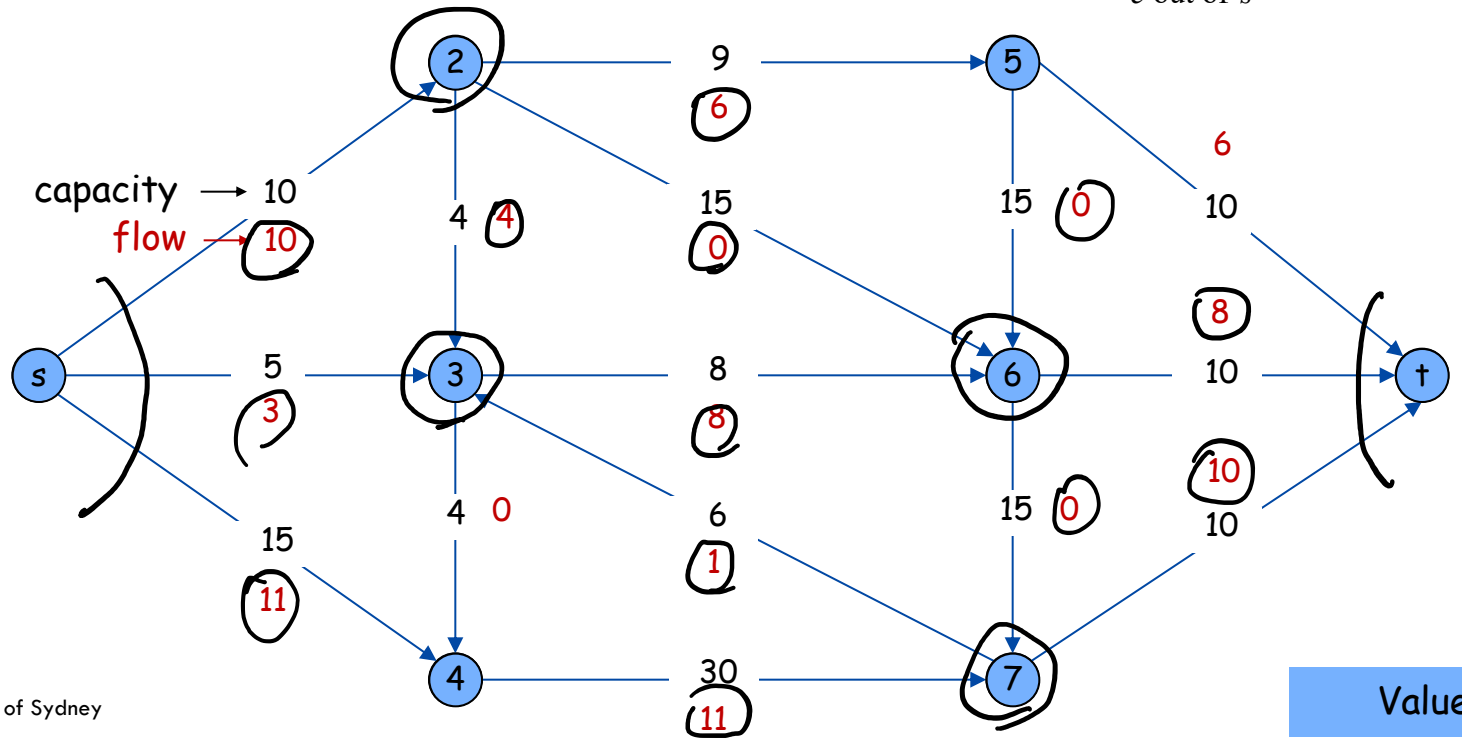
- **Definition:** The value of a flow f is: $v(f) = \sum_{e \text{ out of } s} f(e) = \sum f(e)$ e into t



Value = 4

Flows

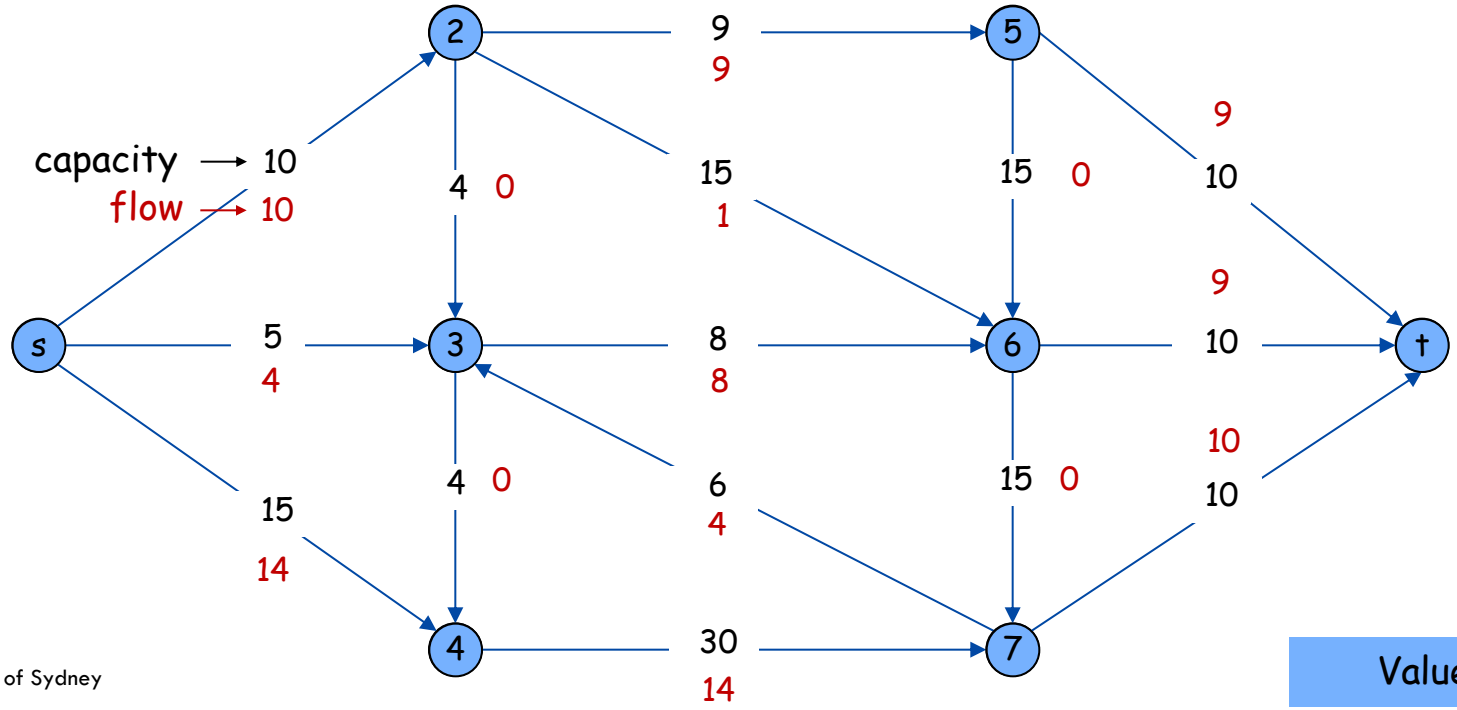
- Definition: An s-t flow is a function that satisfies:
  - For each e ∈ E: $0 \le f(e) \le c(e)$ (capacity)
  - We say e is saturated if f(e) = c(e)
  - For each v ∈ V – {s, t}: $\sum_{e \text{ in to } v} f(e) = \sum_{e \text{ out of } v} f(e)$ (conservation)
- Definition: The value of a flow f is: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



Value = 24

Maximum Flow Problem

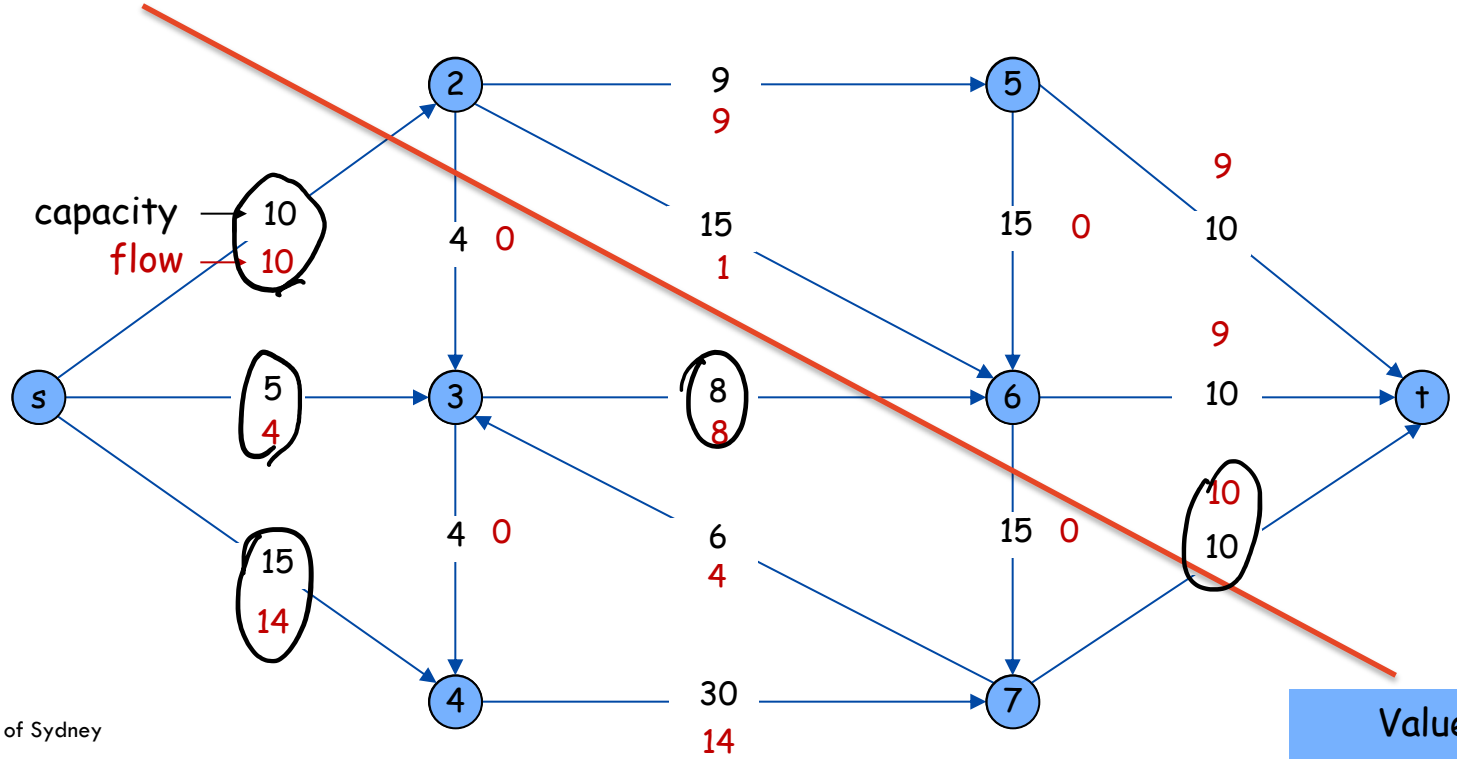
- Max flow problem. Find s-t flow of maximum value.
- Question: How to characterize optimal solution?
- DP and D&C uses a recurrence equation. Not known if max flows admit such a recurrence.



Characterizing Max Flow

Simple conditions implying f is a max flow:

- f saturates every edge out of s OR
- f saturates every edge into t

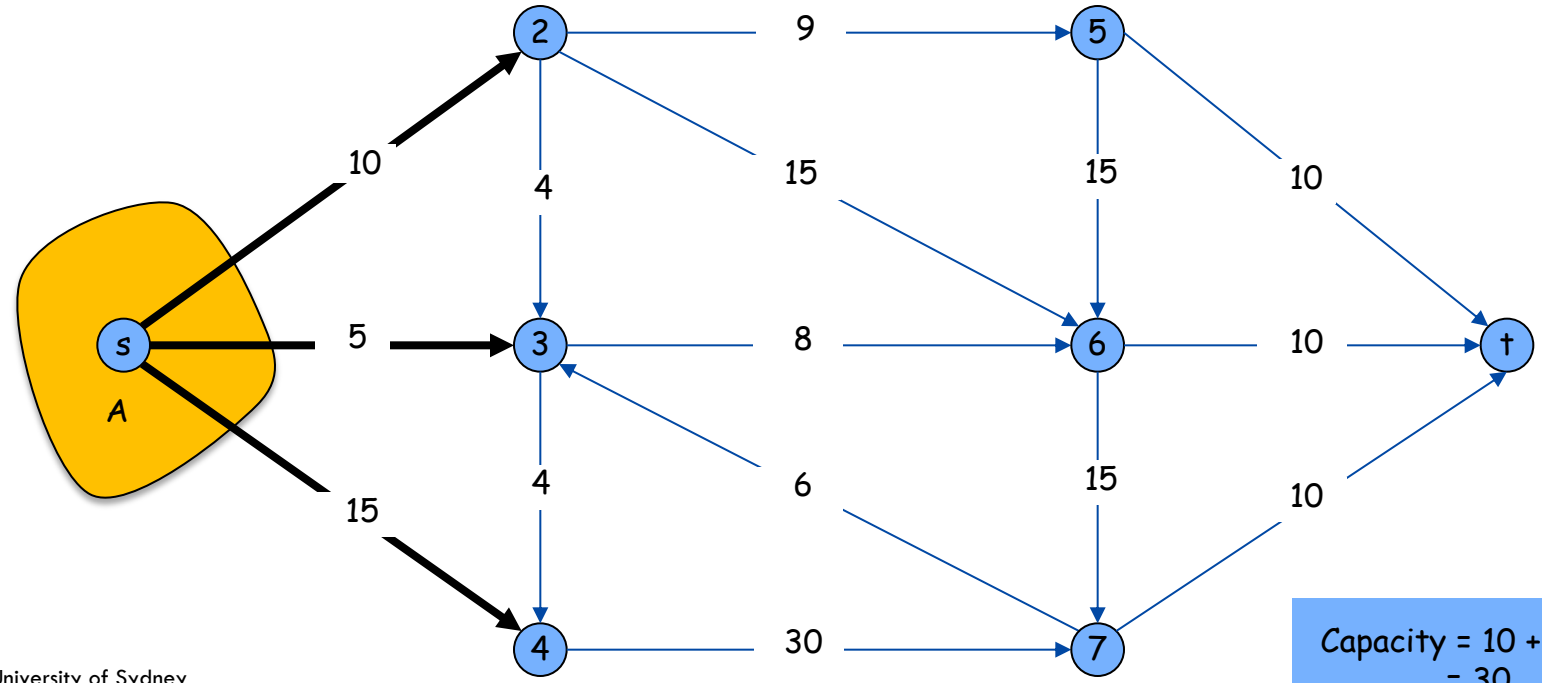


Cuts

Definitions:

source-side → sink-side

- 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: $cap(A, B) = \sum_{e \text{ out of } A} c(e)$
- Only count capacity of outgoing edges!



Capacity = 10 + 5 + 15
= 30

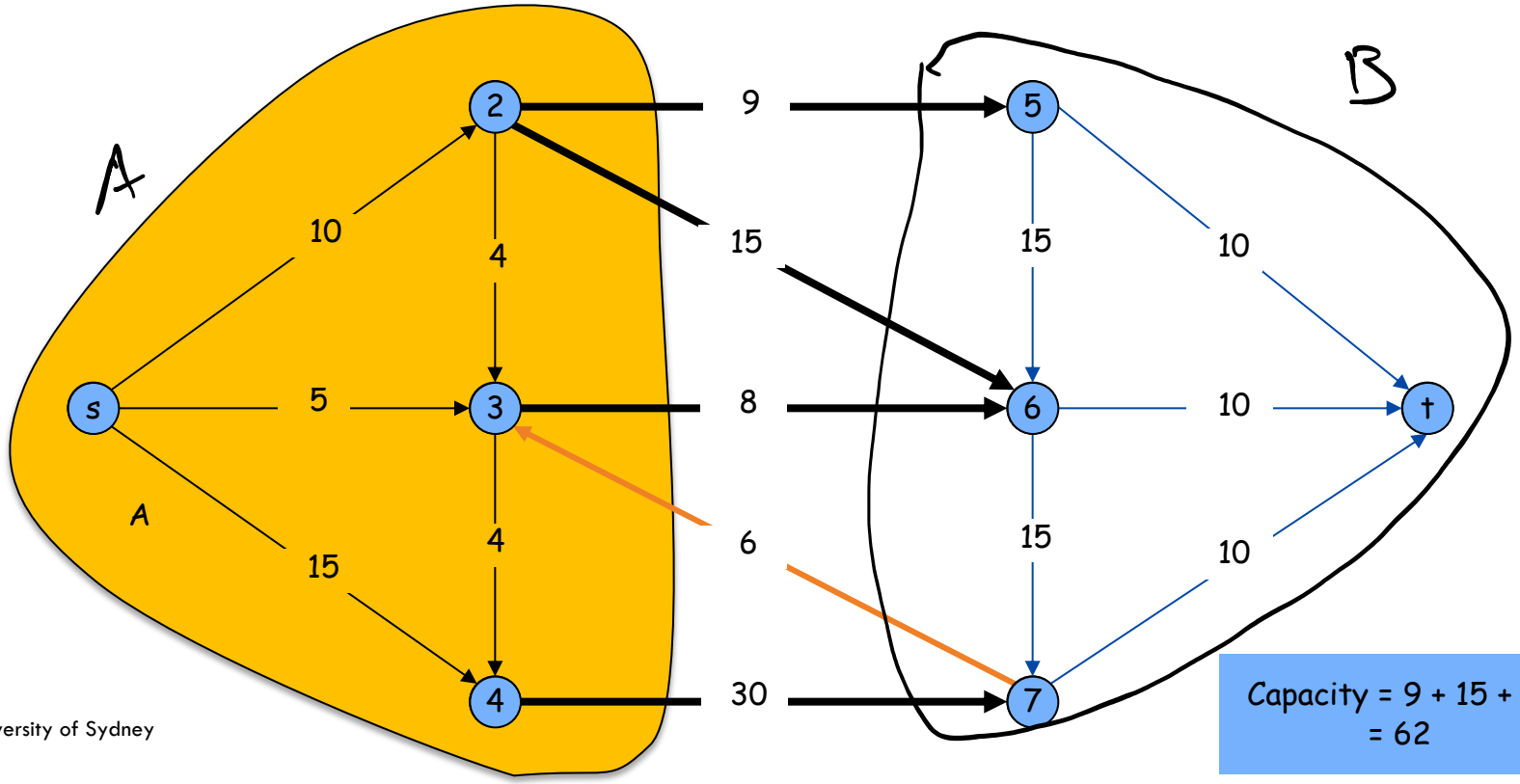
Cuts

$|B| = |V| - |A|$

$|A| \geq 1 \quad , |B| \geq 1$

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: $cap(A, B) = \sum_{e \text{ out of } A} c(e)$
- Only count capacity of outgoing edges!



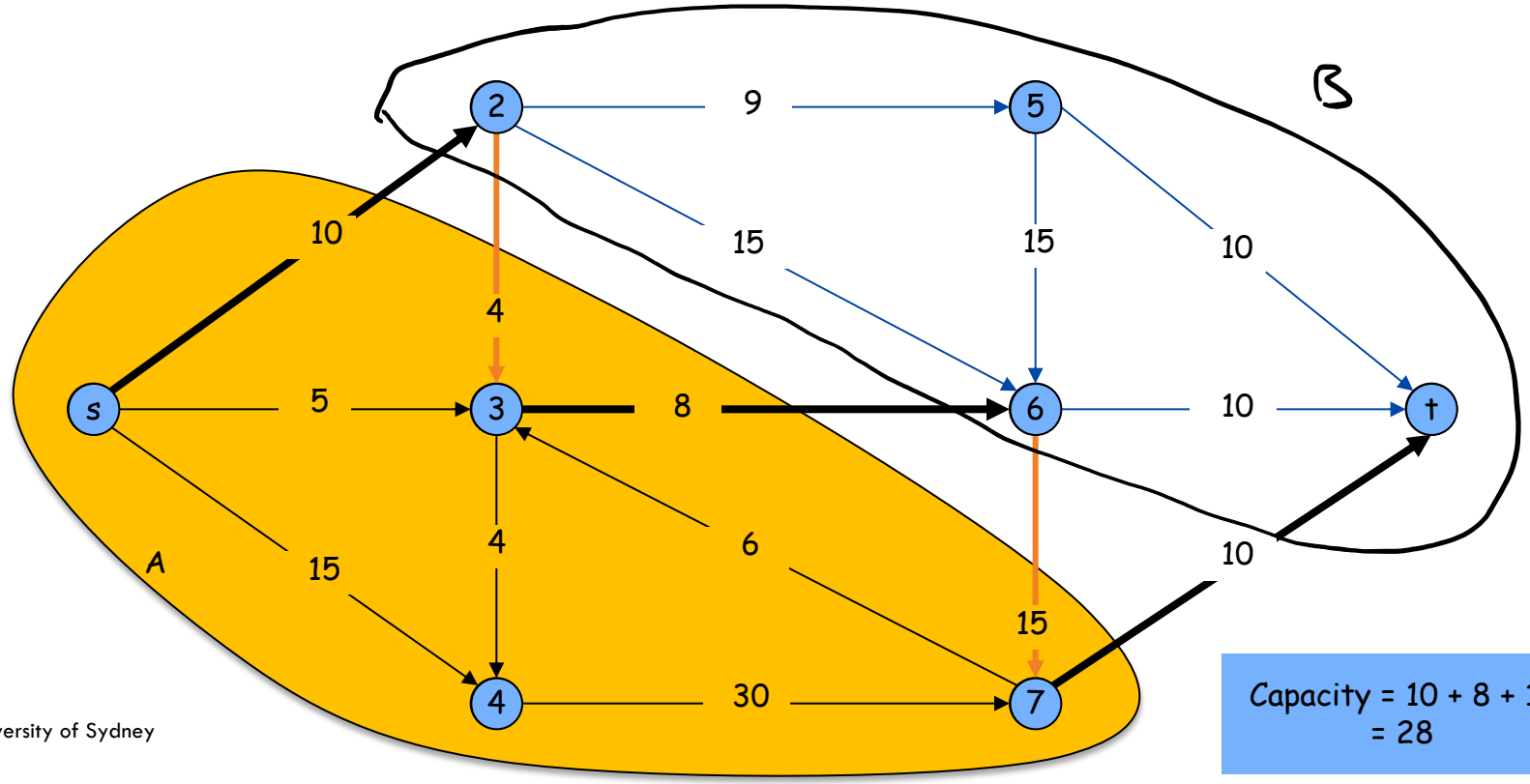
Capacity = 9 + 15 + 8 + 30
= 62

Minimum Cut Problem

Min s-t cut problem:

Find an s-t cut of minimum capacity.

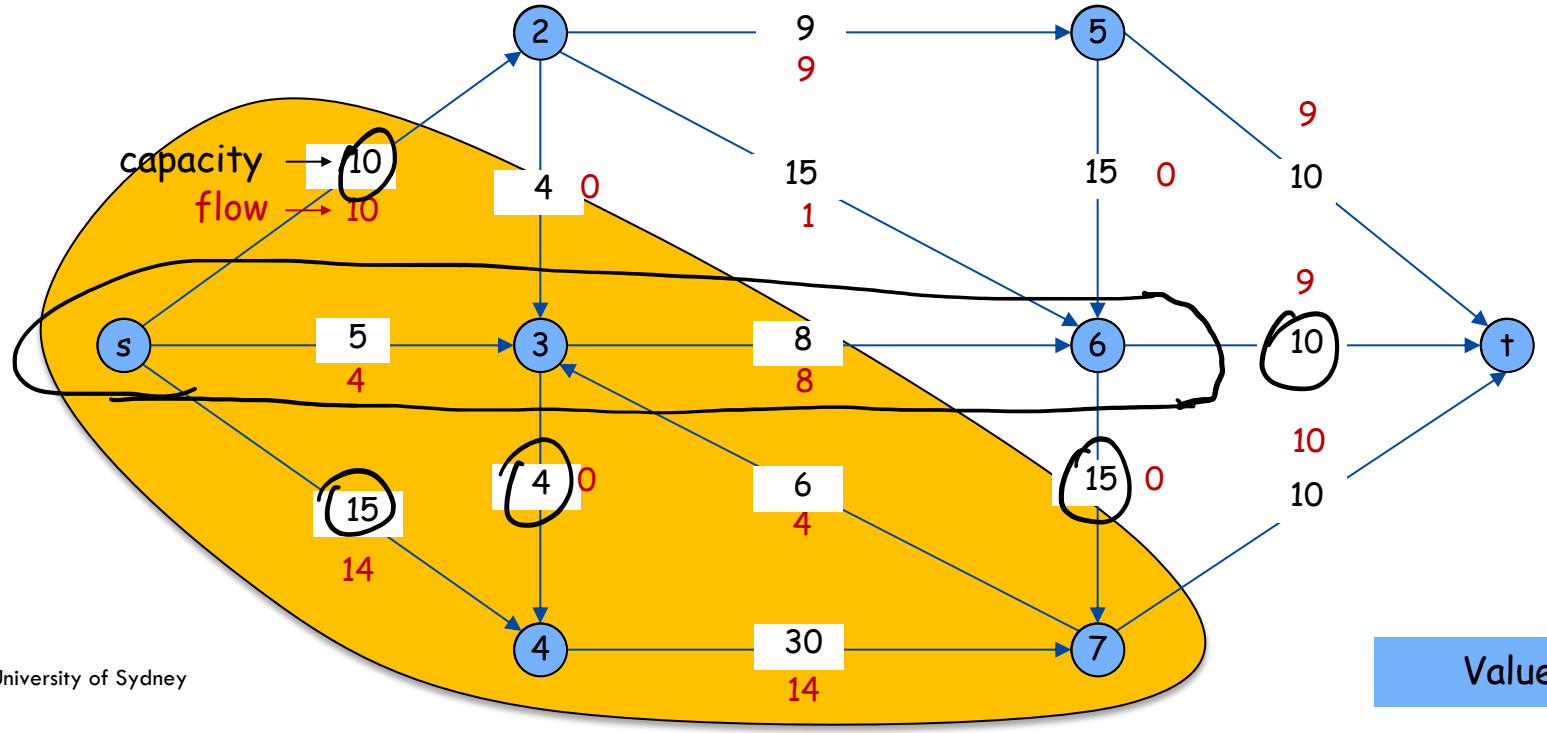
- Question: How to characterize optimal solution?
- Not known if min cuts admit a DP-style recurrence.



Capacity = 10 + 8 + 10
= 28

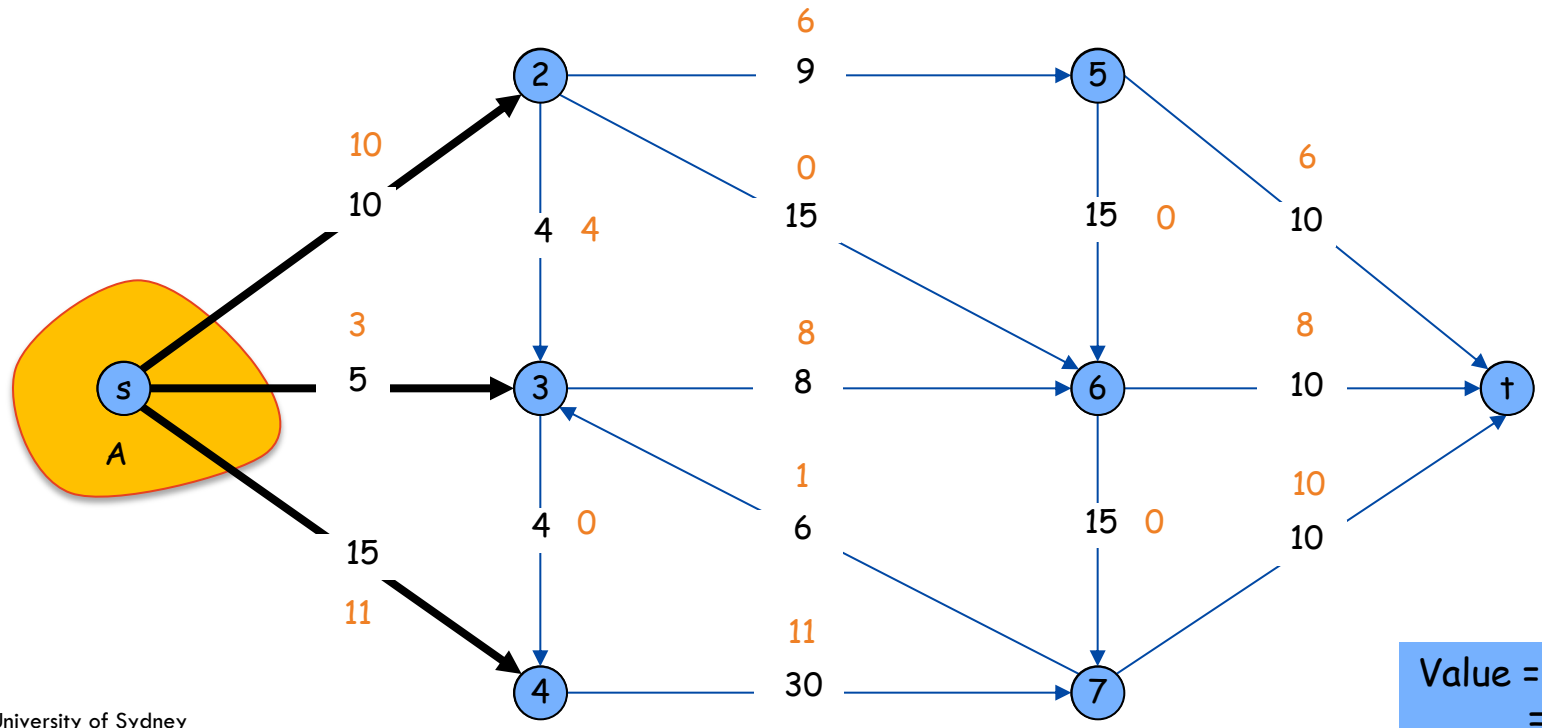
Max-Flow = Min-Cut

1. Max-Flow ≤ Min-Cut
2. Algorithm for Max-Flow finds a flow f and a cut (A,B) such that v(f) = cap(A,B) ⇒ f is max-flow, (A, B) is min-cut



Notation

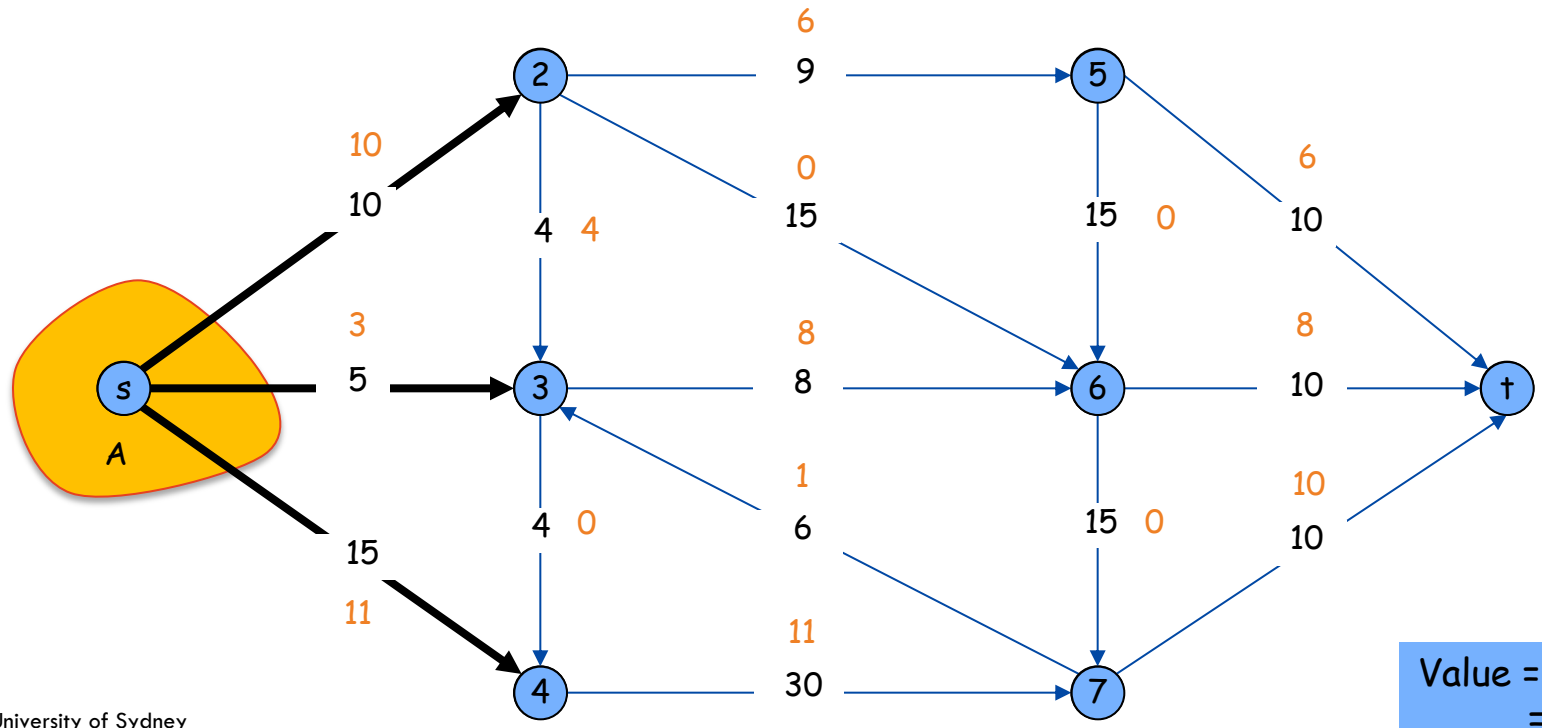
- Given a vertex u, define $f^{out}(u)$ = total flow on edges leaving u
- Given a vertex subset S, define $f^{out}(S)$ = total flow on edges leaving S
- Similarly, define $f^{in}(u)$ and $f^{in}(S)$ on edges entering u and S, resp.
- Can rewrite $v(f) = f^{out}(s)$ and $f^{in}(u) = f^{out}(u)$ for u not s,t



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.

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

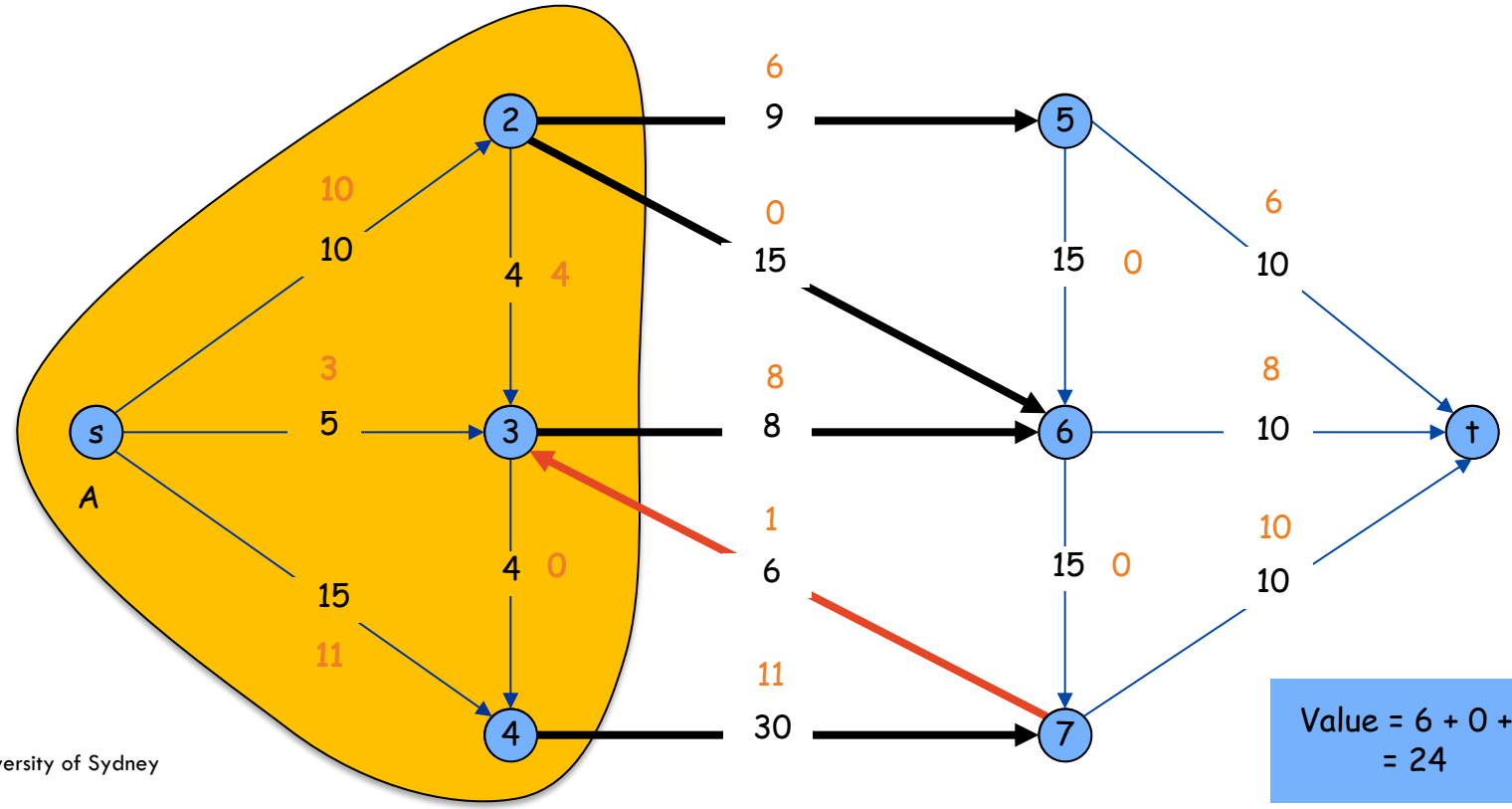


Value = 10+3+11
= 24

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.

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

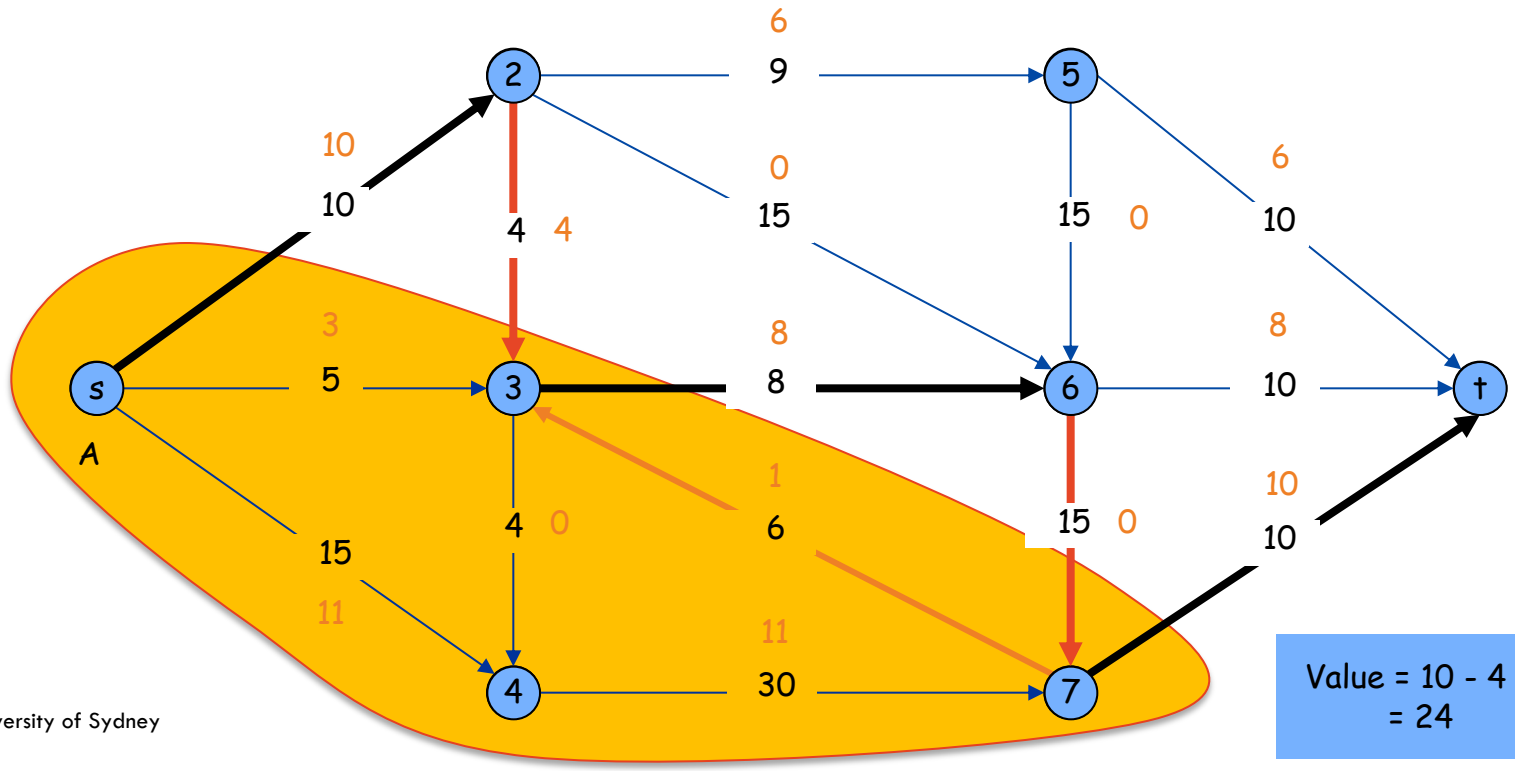


Value = 6 + 0 + 8 - 1 + 11
= 24

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.

$$v(f) = f^{out}(A) - f^{in}(A) = 24$$



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

Proof 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.

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

Note: True by definition for A = {s}

Proof 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.

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

by def

Proof:

$$v(f) = f^{out}(s) = f^{out}(s) - \overbrace{f^{in}(s)}^{0}$$

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

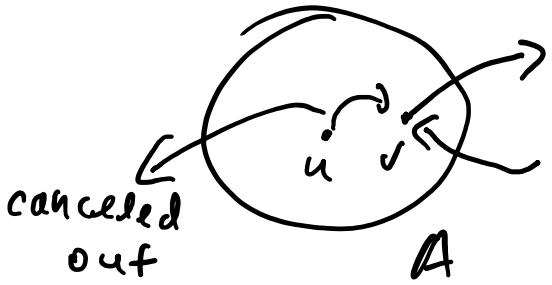
by flow conservation, all terms except v = s are 0, i.e. $f^{out}(v) - f^{in}(v) = 0$

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

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

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

by def

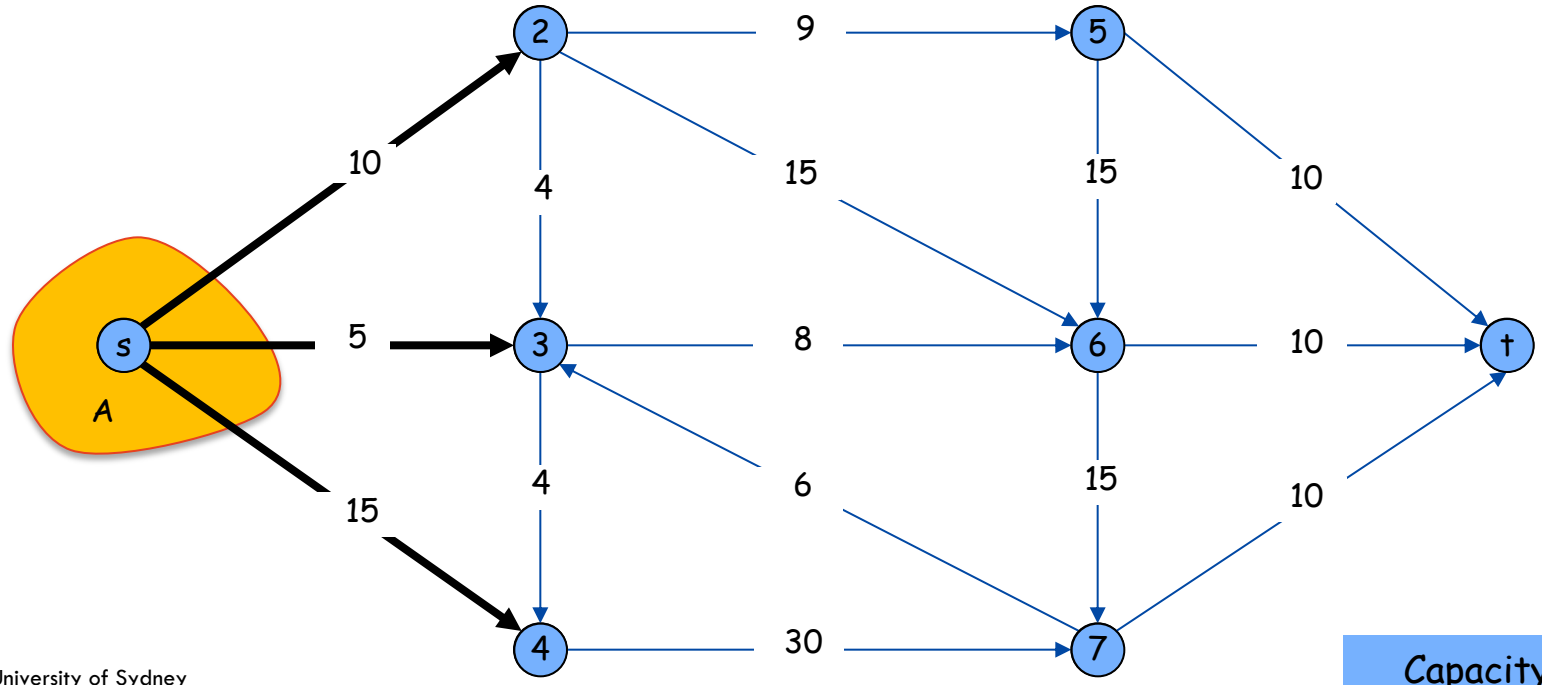


Flows and Cuts

- 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.

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

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

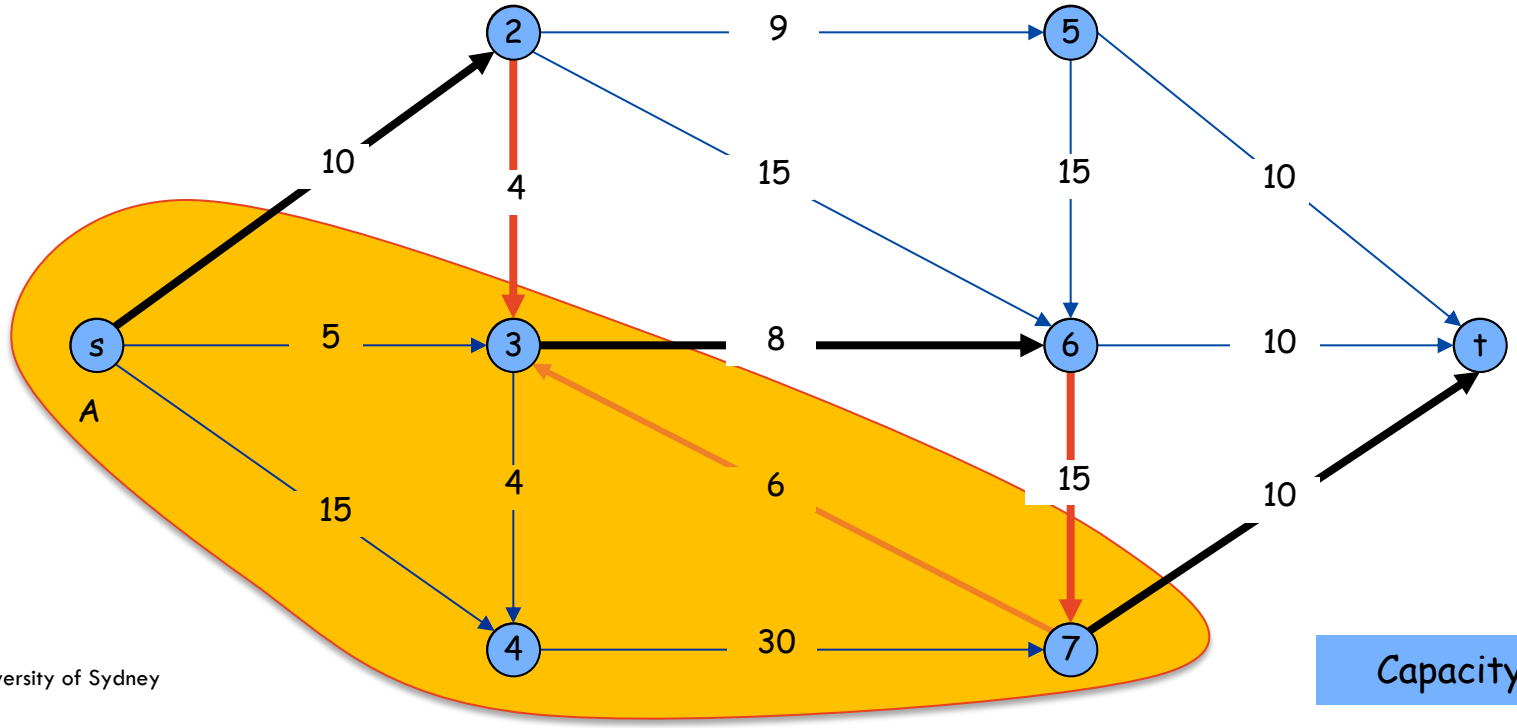


Flows and Cuts

- 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, i.e.

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

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



Flows and Cuts

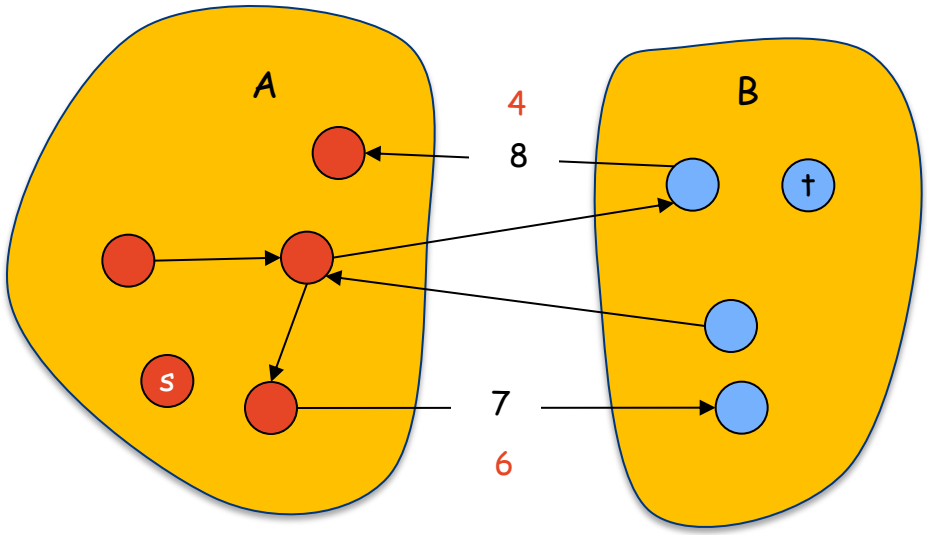
Weak duality. Let f be any flow, and let (A, B) be any s-t cut. Then the value of the flow is at most the capacity of the cut, i.e.,

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

Proof:

flow value lemma

$$\begin{aligned} v(f) &= f^{out}(A) - f^{in}(A) \\ &\leq f^{out}(A) \qquad f^{in}(A) \geq 0 \\ &= \sum_{e \text{ out of } A} f(e) \qquad \text{by def} \\ &\leq \sum_{e \text{ out of } A} c(e) \qquad f(e) \leq c(e) \\ &= c(A,B) \end{aligned}$$



Certificate of Optimality

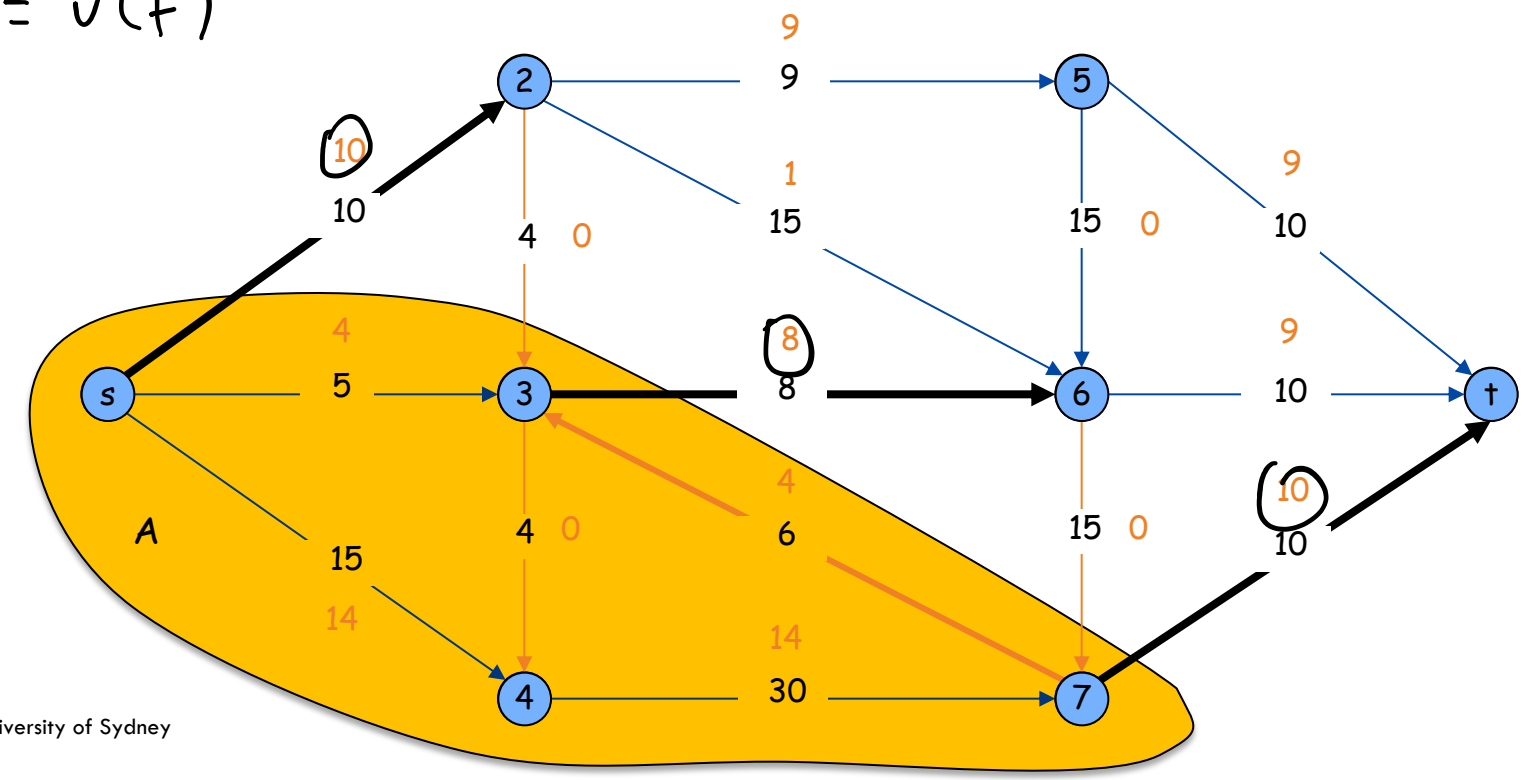
Corollary: Let f be any flow, and let (A, B) be any cut.
If v(f) = cap(A, B) then f is a max flow and (A, B) is a min cut.

∀ flow f′ ≠ f

v(f′) ≤ cap(A, B)

= v(f)

Value of flow = 28
Cut capacity = 28 ⇒ Flow value ≤ 28



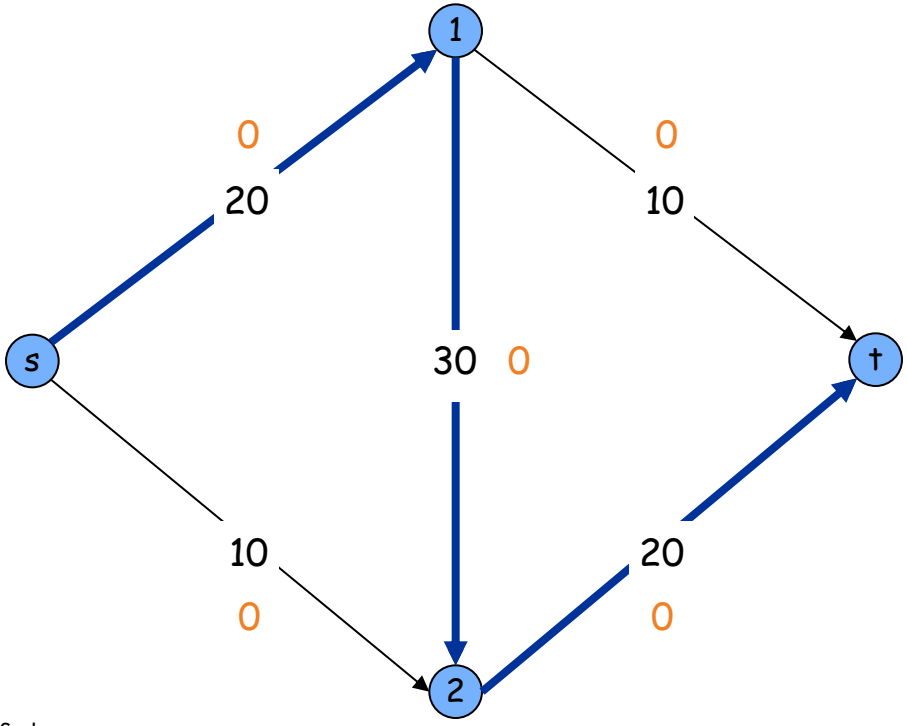
Summary (so far)

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 ∈ E.
- Find an s-t path P where each edge has f(e) < c(e).
- Augment flow as much flow as possible along path P.
- Repeat until you get stuck.

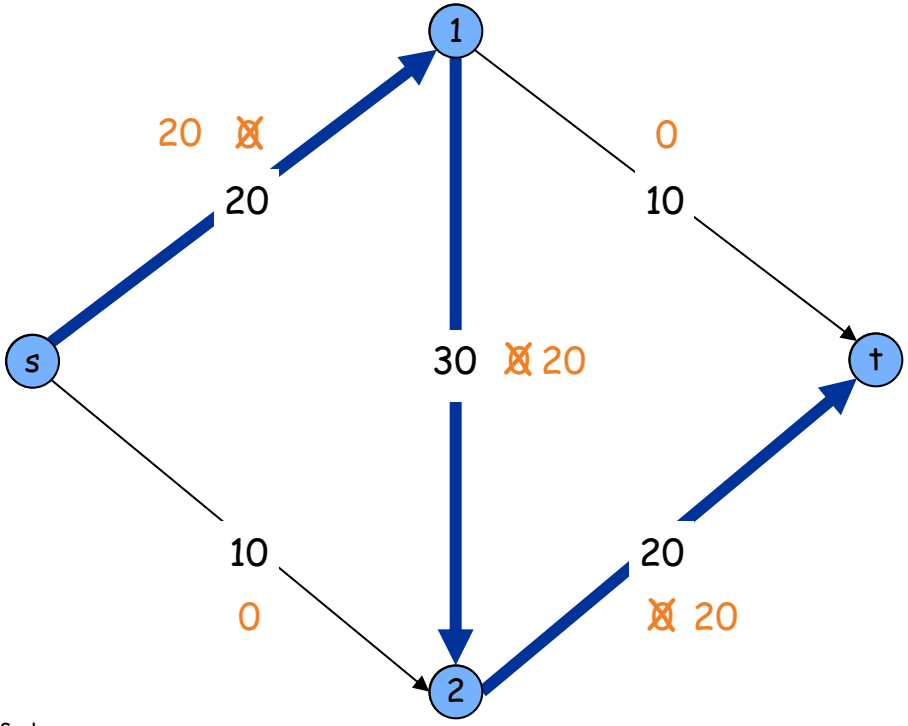


Flow value = 0

Towards a Max Flow Algorithm

Greedy algorithm.

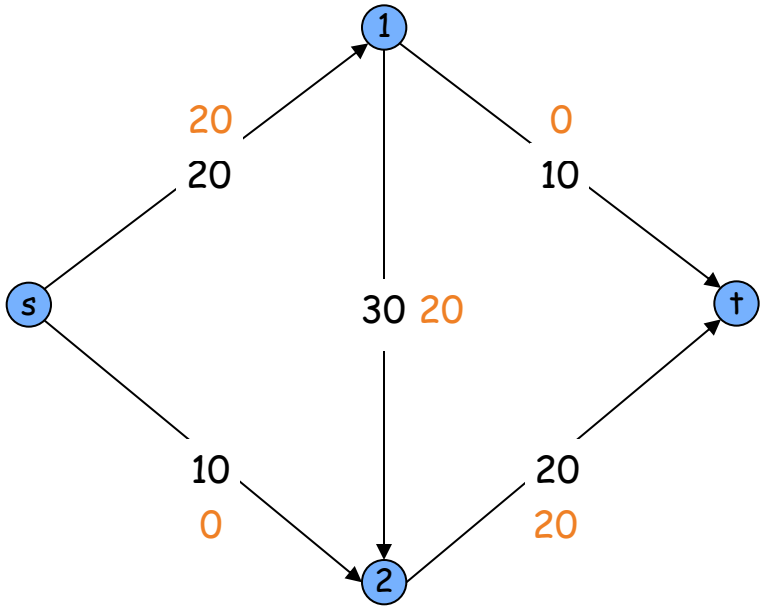
- Start with f(e) = 0 for all edge e ∈ E.
- Find an s-t path P where each edge has f(e) < c(e).
- Augment as much flow as possible along path P.
- Repeat until you get stuck.



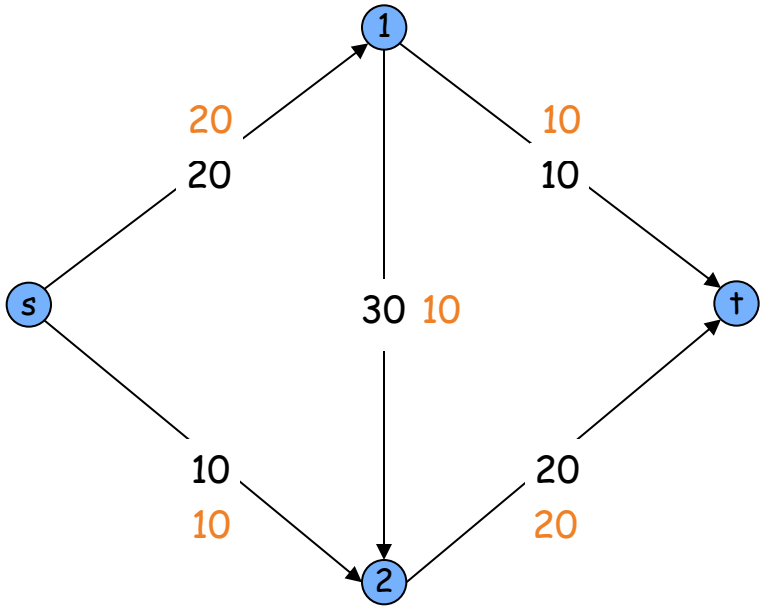
Flow value = 20

Towards a Max Flow Algorithm

Augmenting greedy flow to get optimal flow



greedy = 20

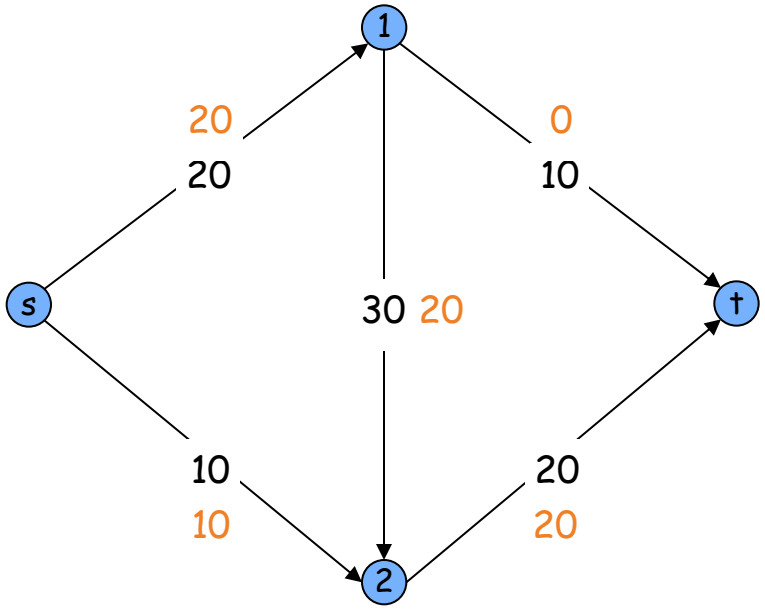


opt = 30

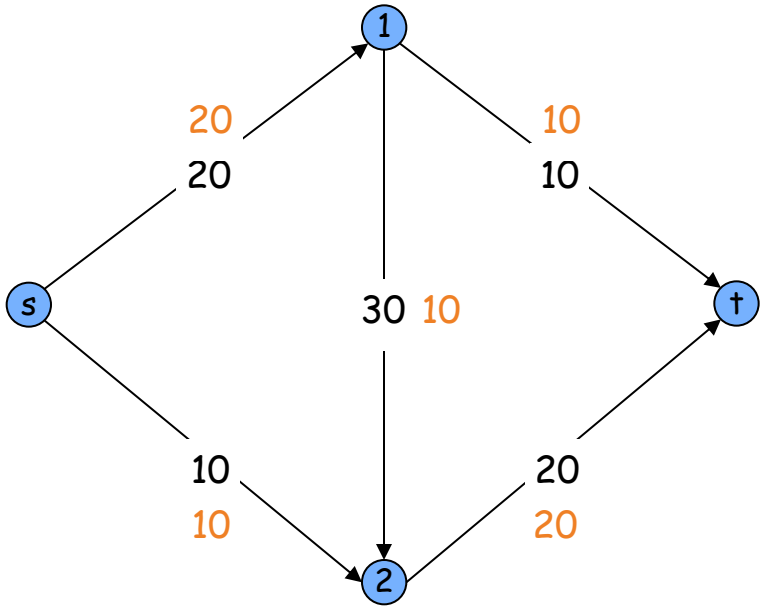
Towards a Max Flow Algorithm

Augmenting greedy flow to get optimal flow

- Send 10 units on (s,2) edge



greedy = 20

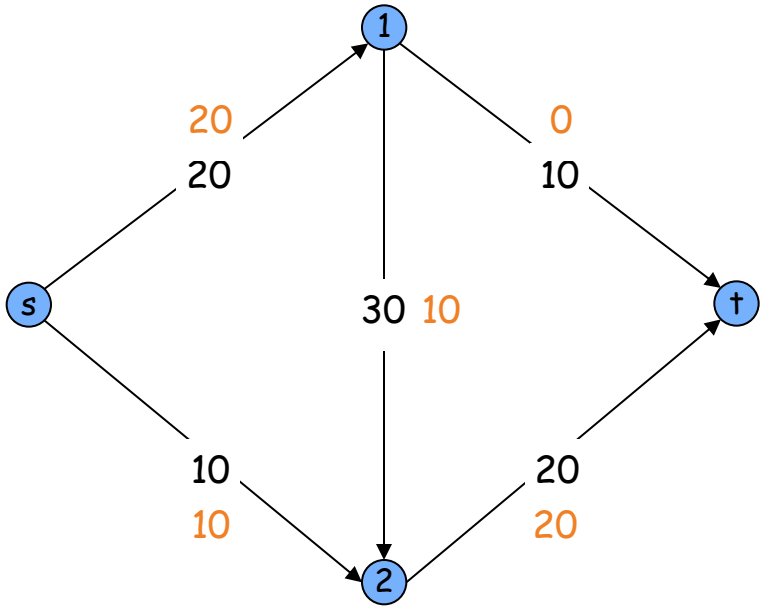


opt = 30

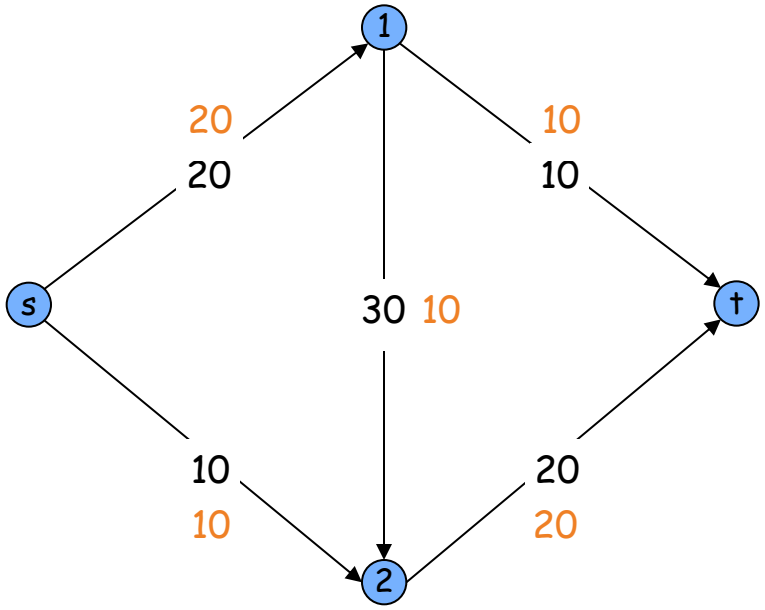
Towards a Max Flow Algorithm

Augmenting greedy flow to get optimal flow

- Send 10 units on (s,2) edge
- "Undo" 10 units on (1,2) edge to preserve conservation at vertex 2



greedy = 20

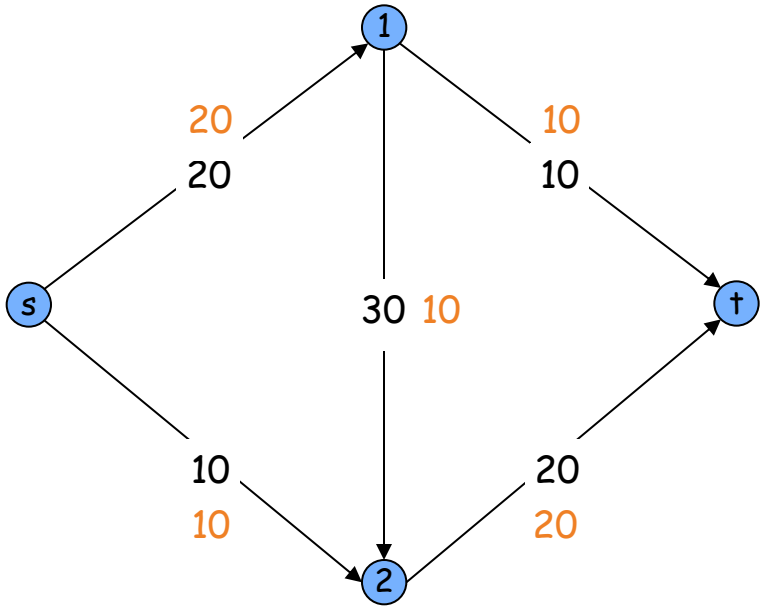


opt = 30

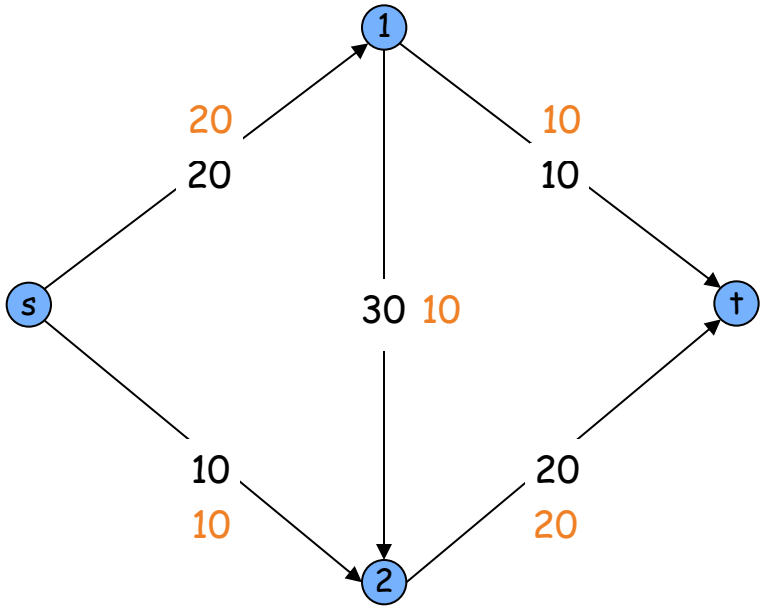
Towards a Max Flow Algorithm

Augmenting greedy flow to get optimal flow

- Send 10 units on (s,2) edge
- "Undo" 10 units on (1,2) edge to preserve conservation at vertex 2
- Send 10 units on (1,t) edge to preserve conservation at vertex 1



greedy = 20



opt = 30

Towards a Max Flow Algorithm

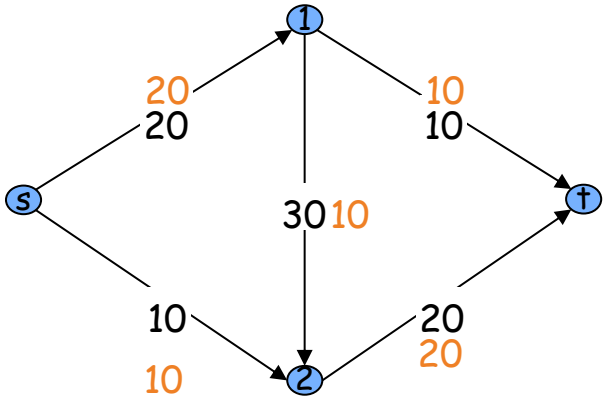
Augmenting greedy flow to get optimal flow

- Send 10 units on (s,2) edge
- "Undo" 10 units on (1,2) edge to preserve conservation at vertex 2
- Send 10 units on (1,t) edge to preserve conservation at vertex 1

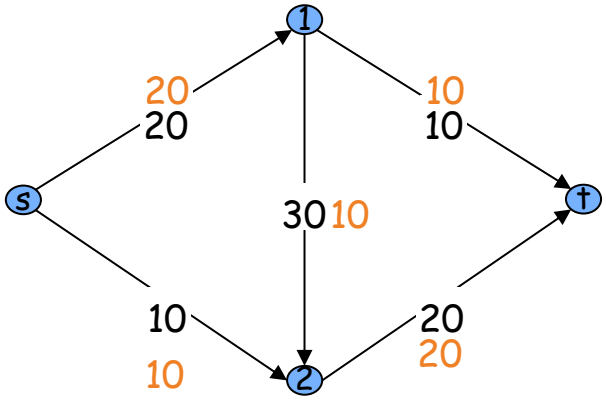
Intuitively, want a "path" from s to t to "push" additional flow

On each edge e

- Either increase the flow if there is leftover capacity
- Or decrease the flow and divert it elsewhere
- Need to ensure new flow satisfies conservation, capacity and non-negative



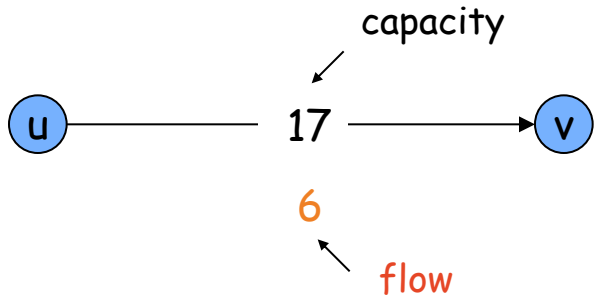
greedy = 20



opt = 30

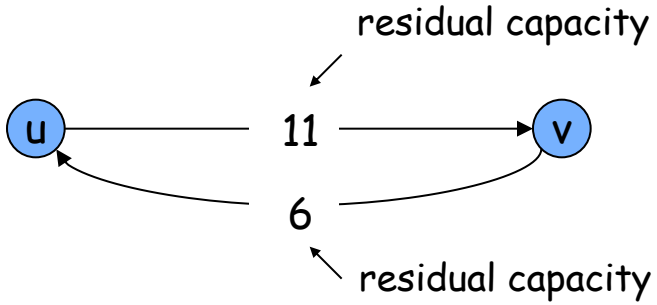
Residual Graph $G_f = (V, E_f)$ for Flow f

- Original edge: $e = (u, v) \in E$.
  - Flow f(e), capacity c(e).



- Residual edge.
  - $e = (u, v)$ and $e^R = (v, u)$.
  - If e in E, then e is a "forward" edge, else "backward" edge
  - Residual capacity of forward edge e represents spare capacity of e
  - Residual capacity of backward edge e^R represents current flow on e that we can reverse
  - 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.
  - $E_f = \{e : f(e) < c(e)\} \cup \{e^R : f(e) > 0\}$.

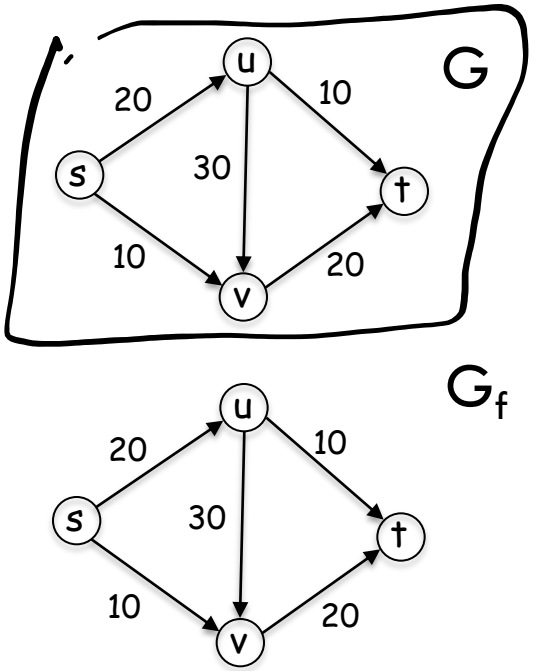
- Idea: can push extra flow along a s-t path in G_f

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.



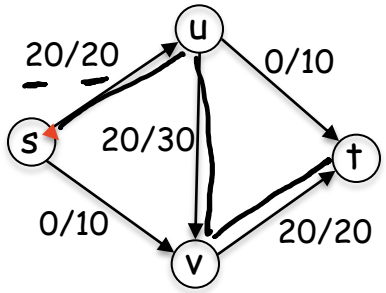
Augmenting Path Algorithm

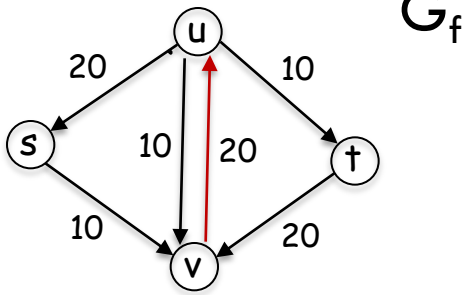
Notations:

x/y means f(e) = x, c(e) = y

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

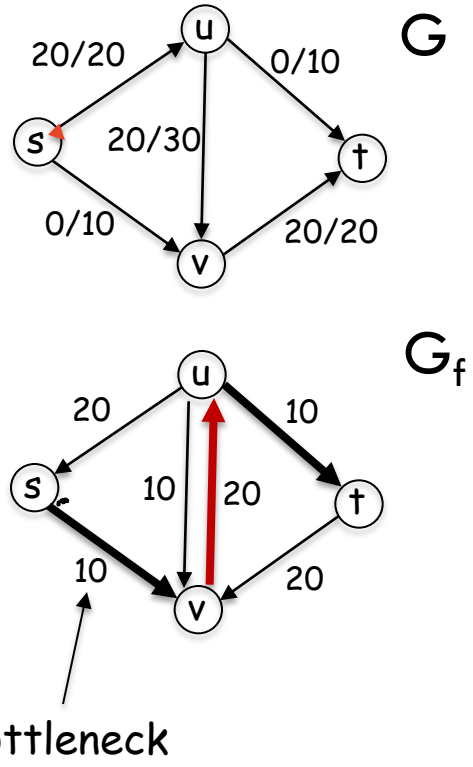
Notations:

x/y means f(e) = x, c(e) = y

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

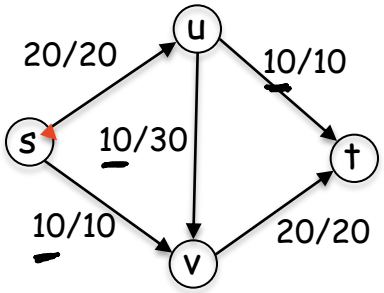
Notations:

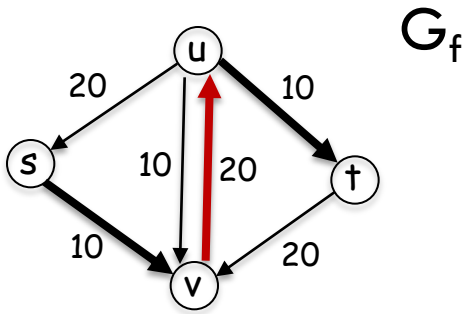
x/y means f(e) = x, c(e) = y

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 with v(f') = b + v(f)

Augmenting Path Algorithm

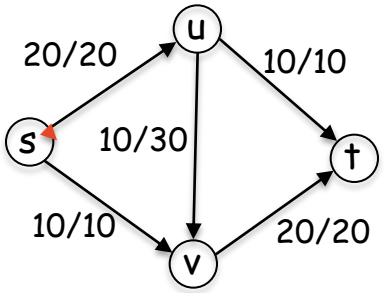
Notations:

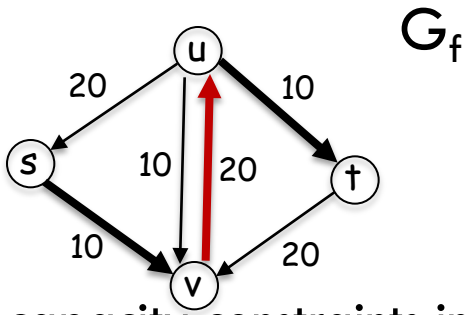
x/y means f(e) = x, c(e) = y

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
}
```





Capacity on residual edges ensure new flow satisfies capacity constraints in G and is non-negative

Flow conservation also satisfied due to augmenting on a s-t path

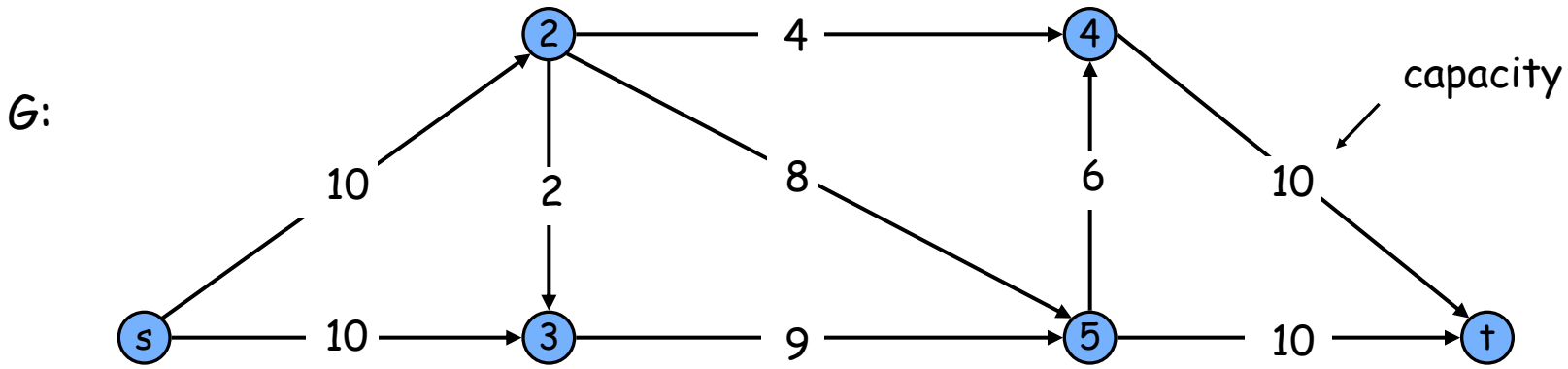
(See textbooks for full proof)

Augmenting Path Algorithm

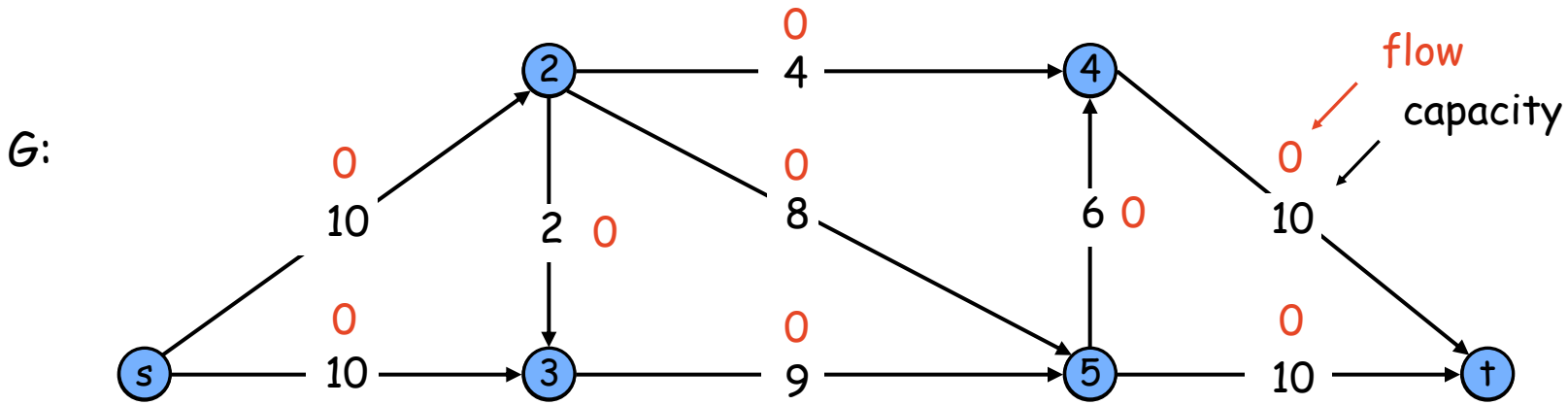
```
Ford-Fulkerson(G,s,t) {
    foreach e ∈ E
        f(e) ← 0
    G_f ← residual graph

    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

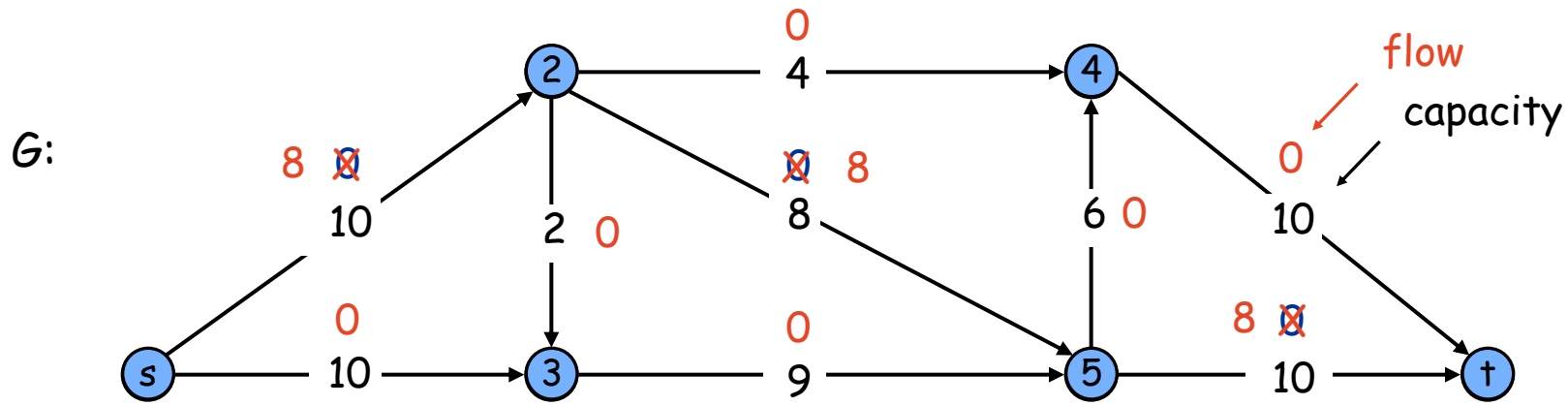


Ford-Fulkerson Algorithm

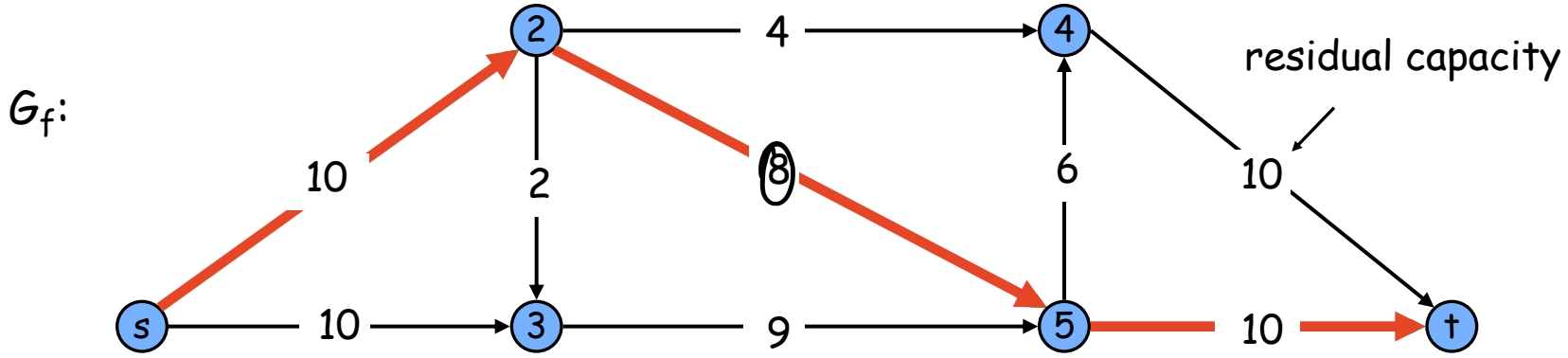


Flow value = 0

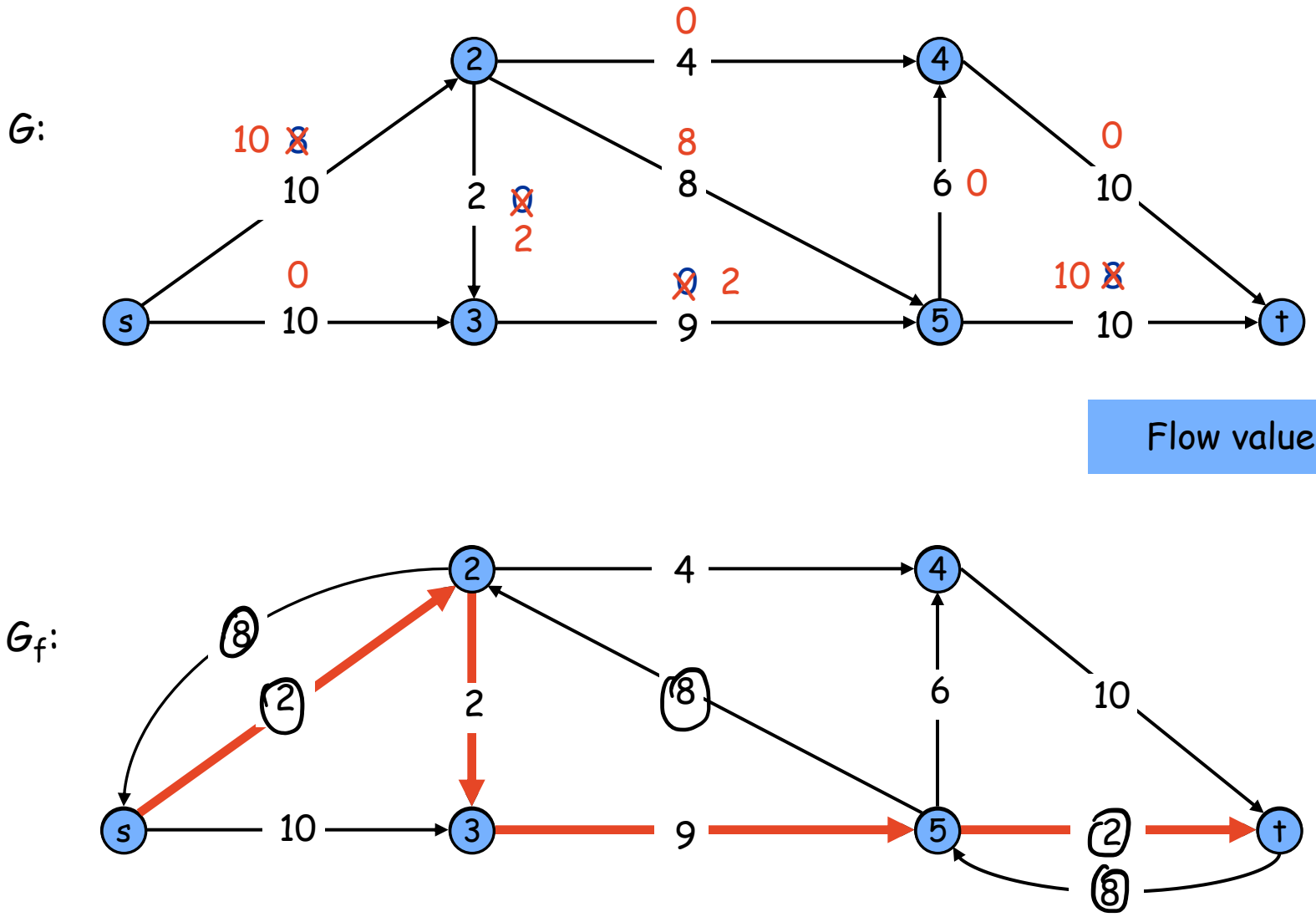
Ford-Fulkerson Algorithm



Flow value = 0



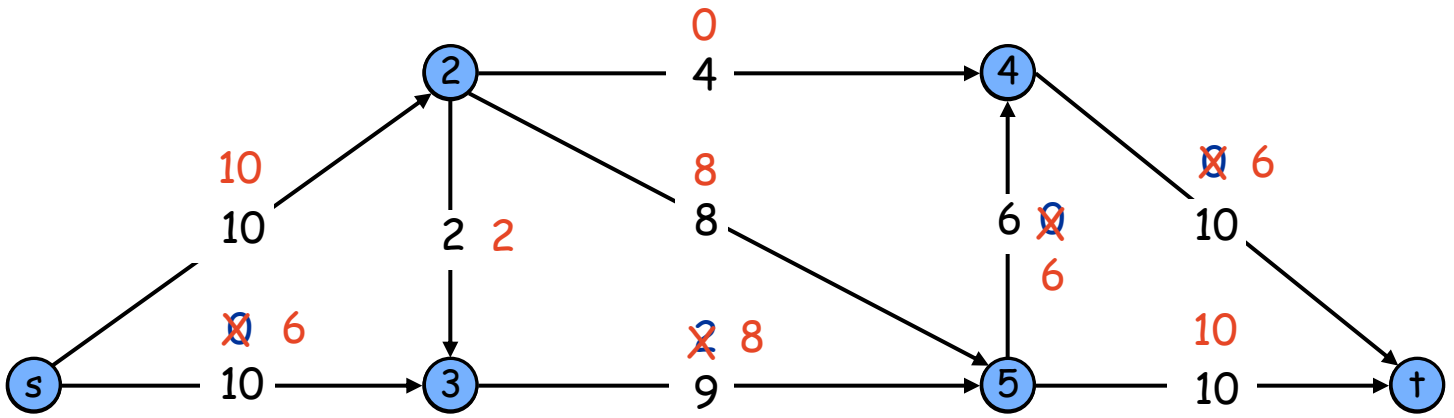
Ford-Fulkerson Algorithm



Flow value = 8

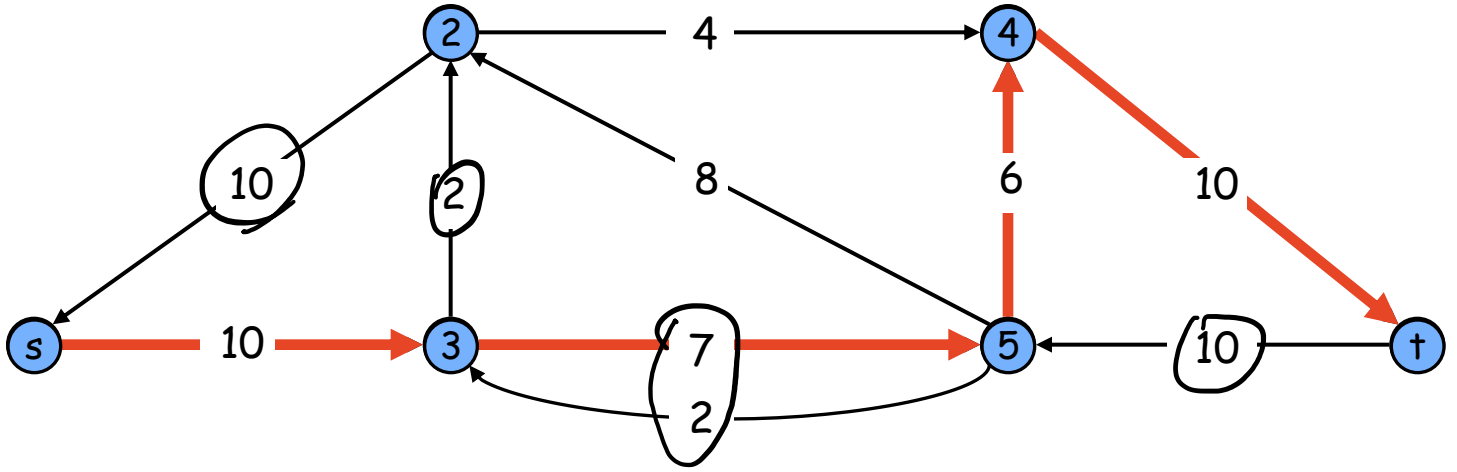
Ford-Fulkerson Algorithm

G:

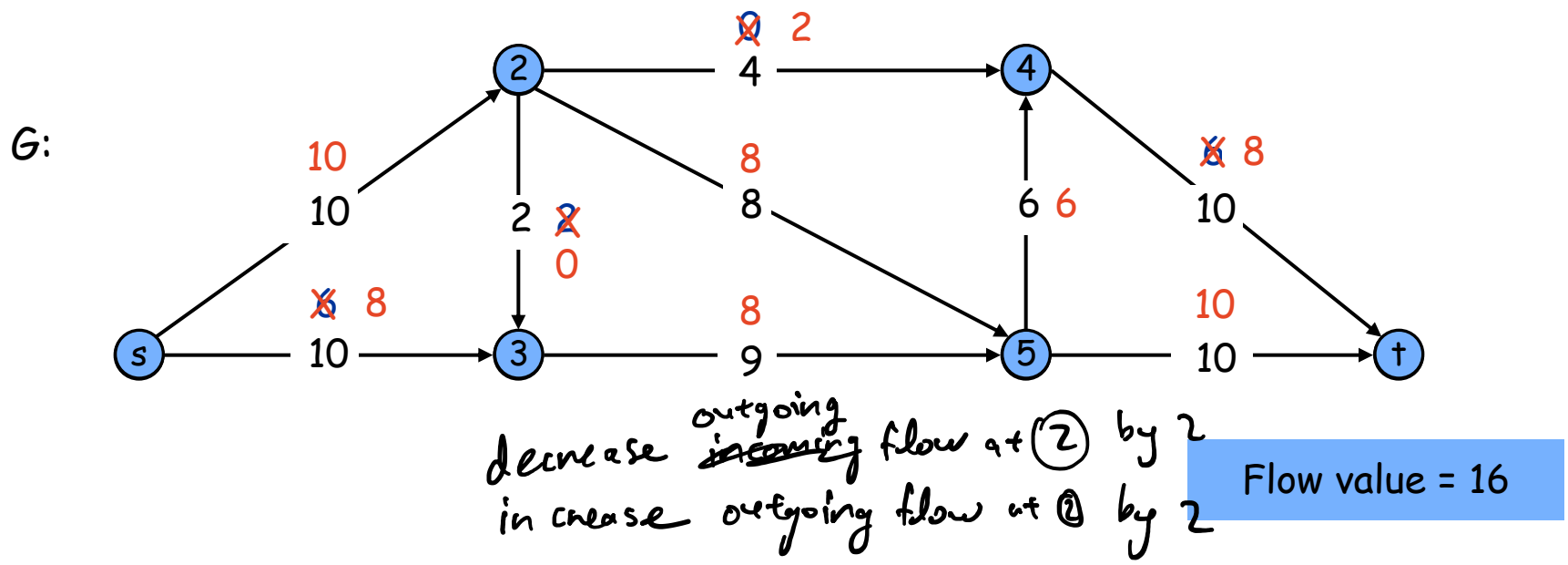


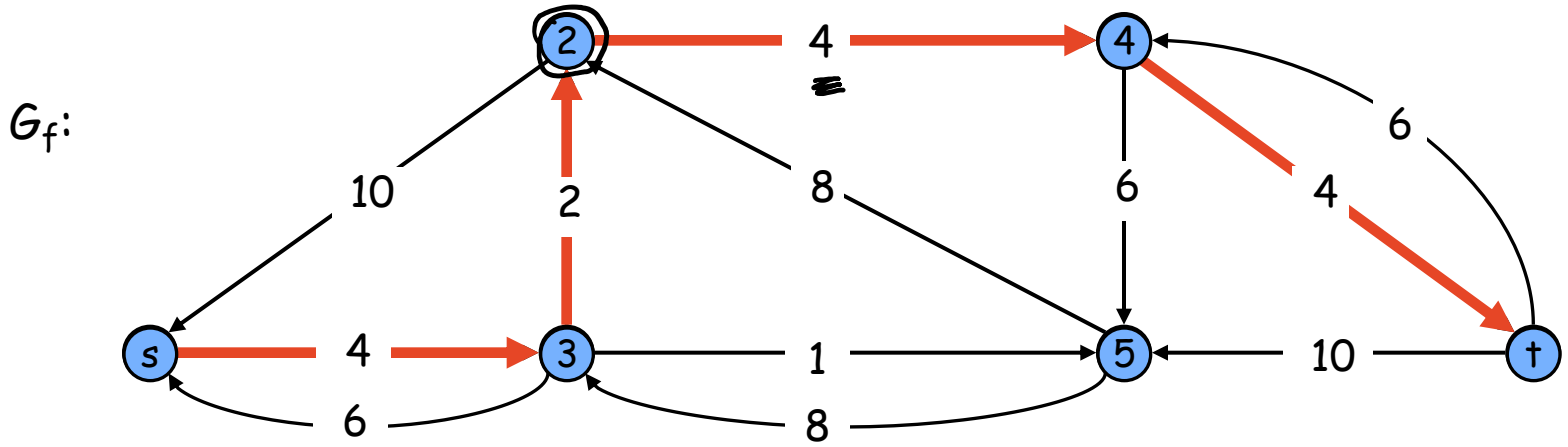
Flow value = 10

G_f:

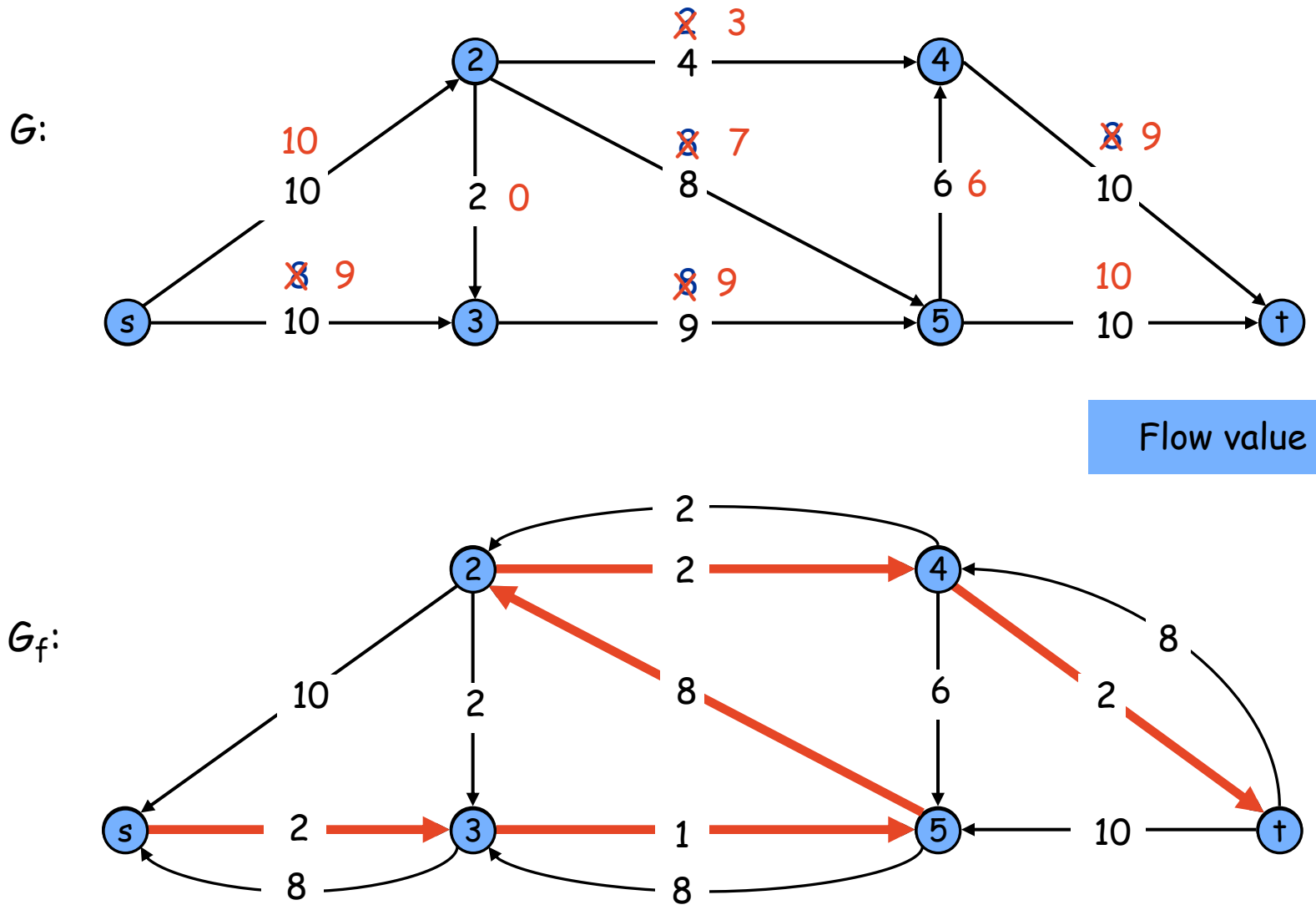


Ford-Fulkerson Algorithm



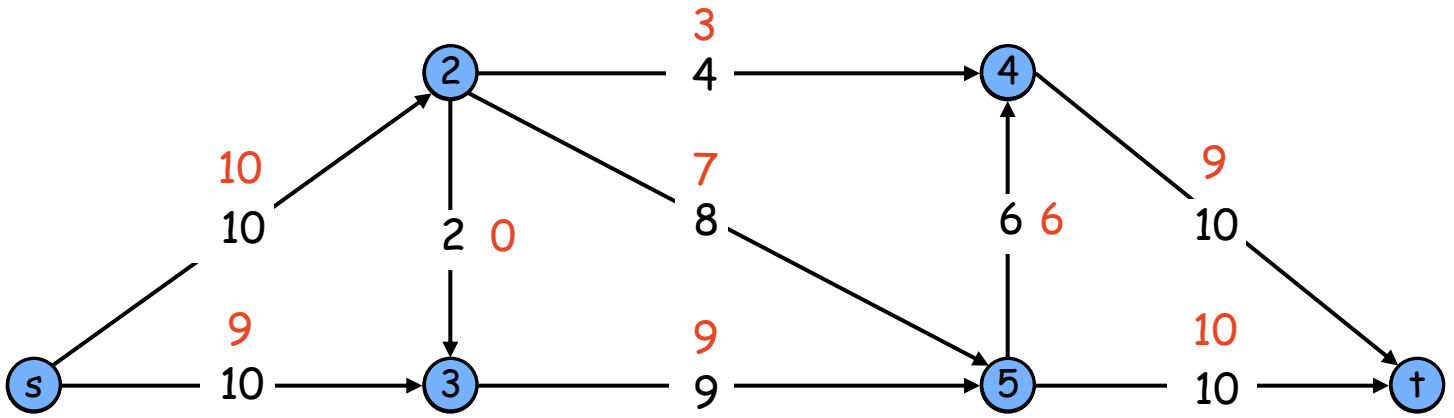


Ford-Fulkerson Algorithm



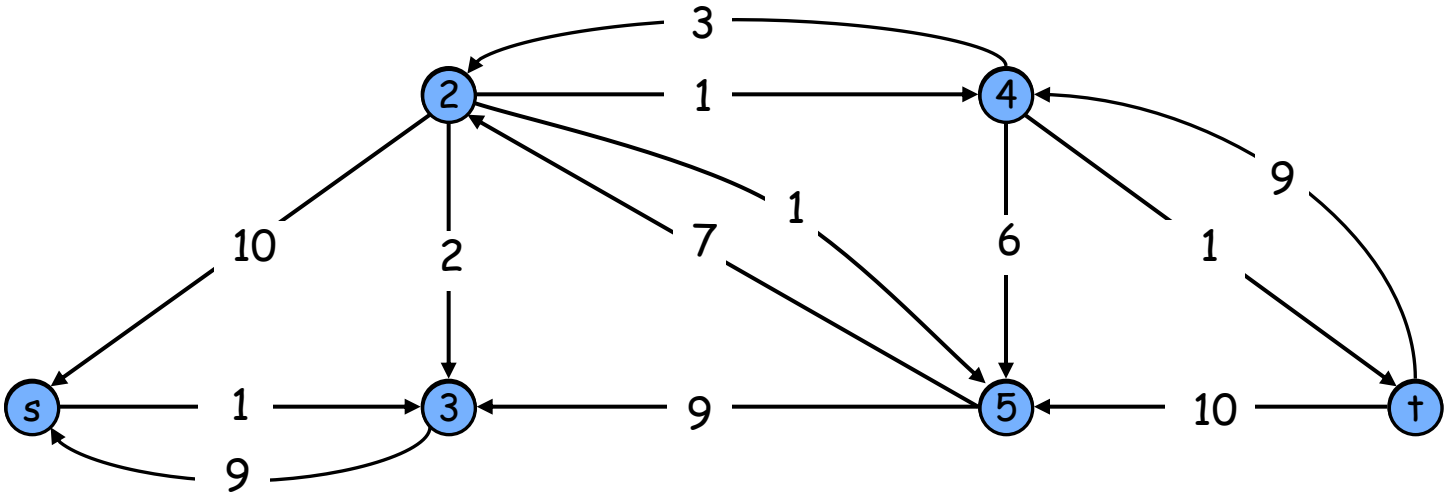
Ford-Fulkerson Algorithm

G:

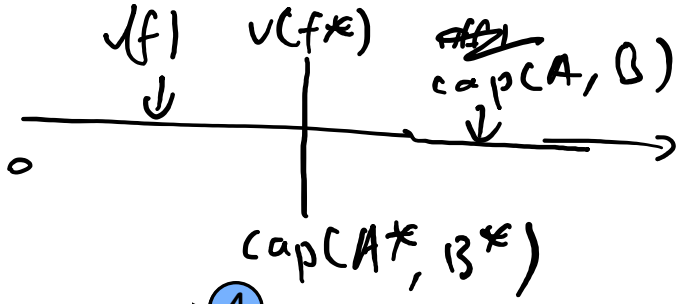


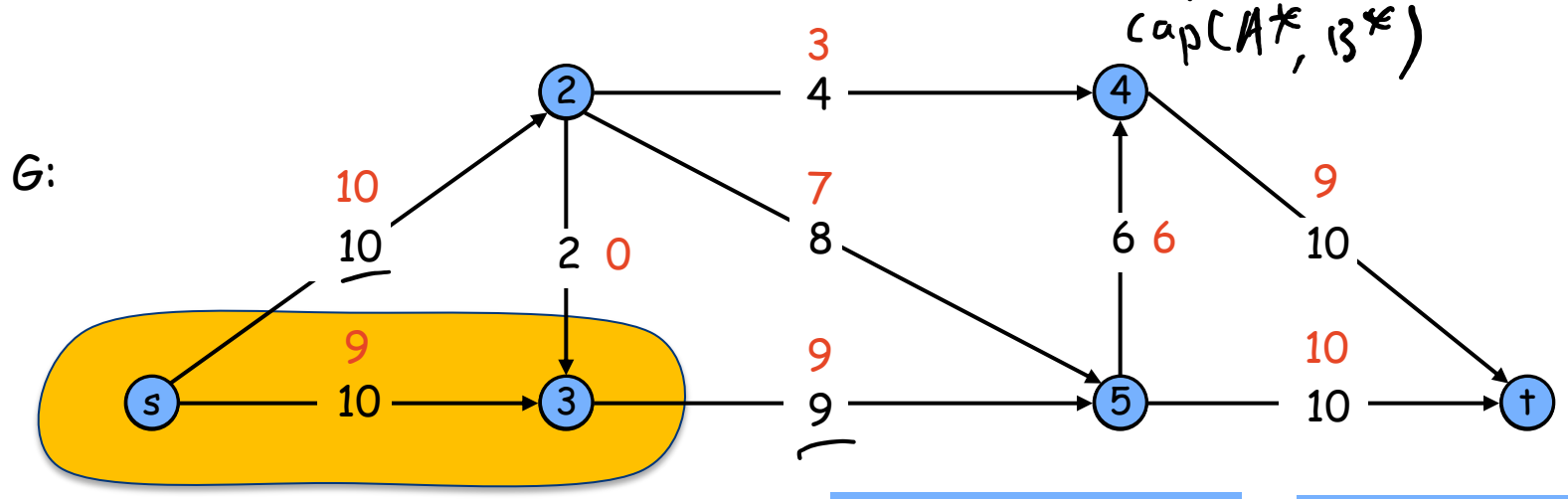
Flow value = 19

G_f:

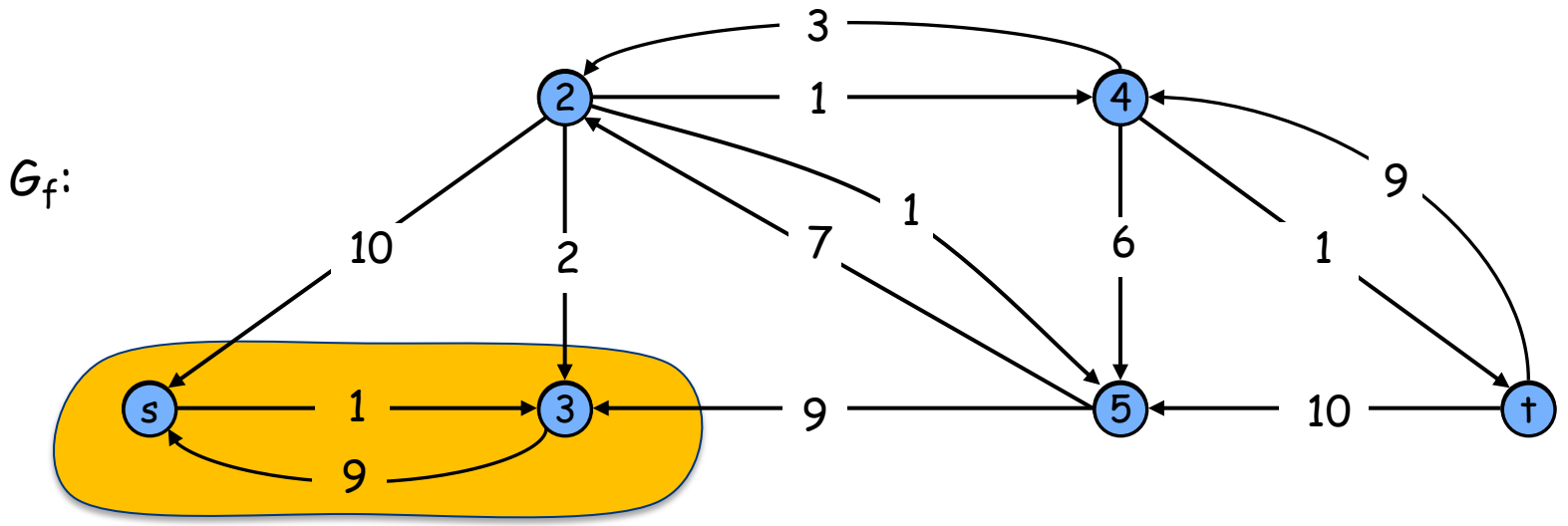


Ford-Fulkerson Algorithm





max flow ≤ 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
}
```

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. Let f be a flow. Then 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.

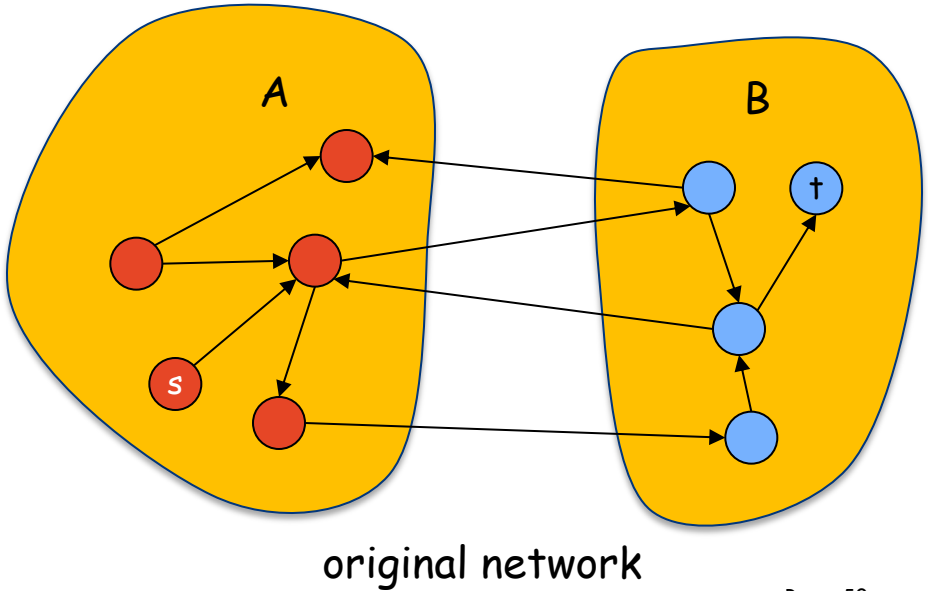
- (i) $\Rightarrow$ (ii) This was the corollary to the weak duality lemma.

- (ii) $\Rightarrow$ (iii) If there exists an augmenting path P, then we can improve f

Proof of Max-Flow Min-Cut Theorem

- (iii) $\Rightarrow$ (i) (No augmenting path $\Rightarrow$ there exists (A,B) s.t 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.

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



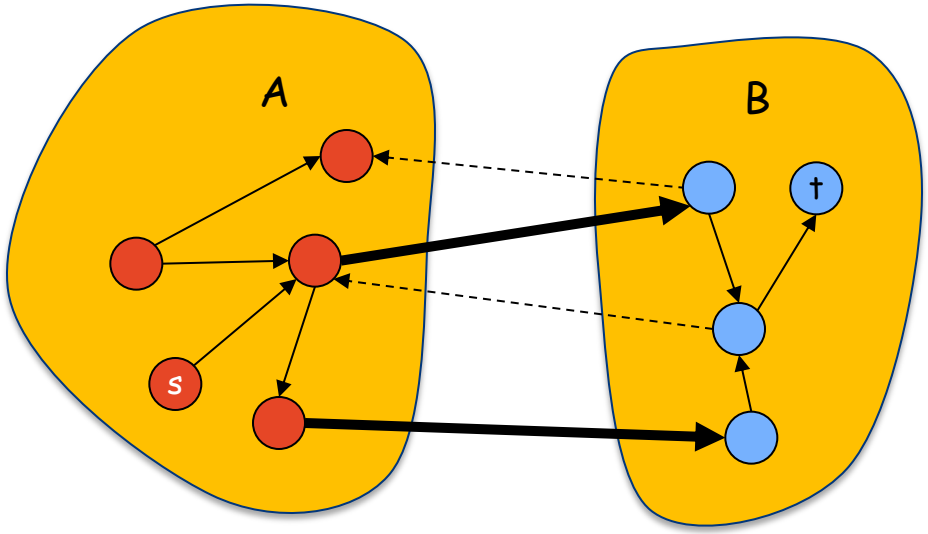
original network

Proof of Max-Flow Min-Cut Theorem

- (iii) $\Rightarrow$ (i) (No augmenting path $\Rightarrow$ there exists (A,B) s.t 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.

No augmenting path from A to B => every edge leaving A saturated, every edge entering A is empty

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



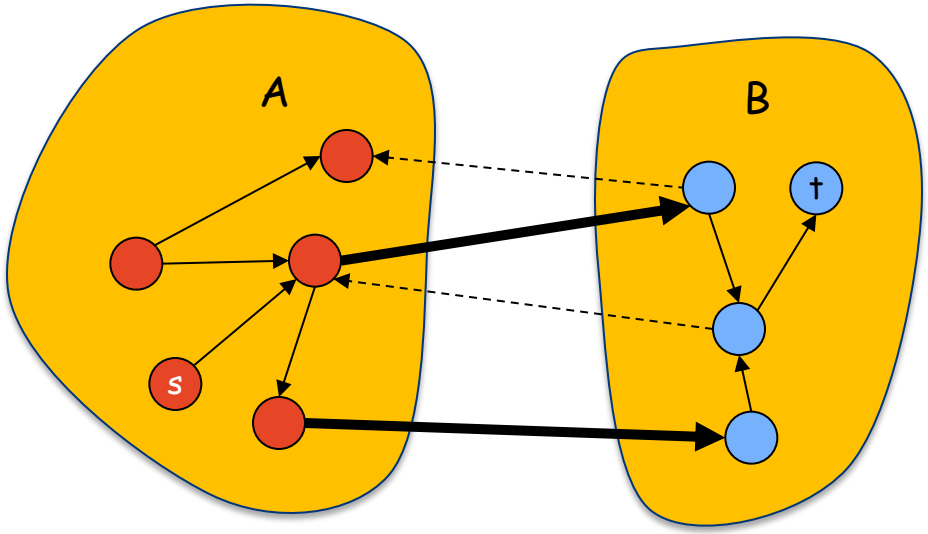
original network

Proof of Max-Flow Min-Cut Theorem

- (iii) $\Rightarrow$ (i) (No augmenting path $\Rightarrow$ there exists (A,B) s.t 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.

No augmenting path from A to B => every edge leaving A saturated, every edge entering A is empty

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



original network

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. Let f be a flow. Then 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.

Note: This implies we can check if a given flow f is max flow in time $O(n + m)$!

Ford-Fulkerson: Running Time

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.

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

Theorem. The algorithm terminates in at most $v(f_{max}) \leq F$ iterations, where F = value of max flow.

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

Ford-Fulkerson: Running Time

Corollary:

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

Proof: F 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(n) time.

7.3 Choosing Good Augmenting Paths

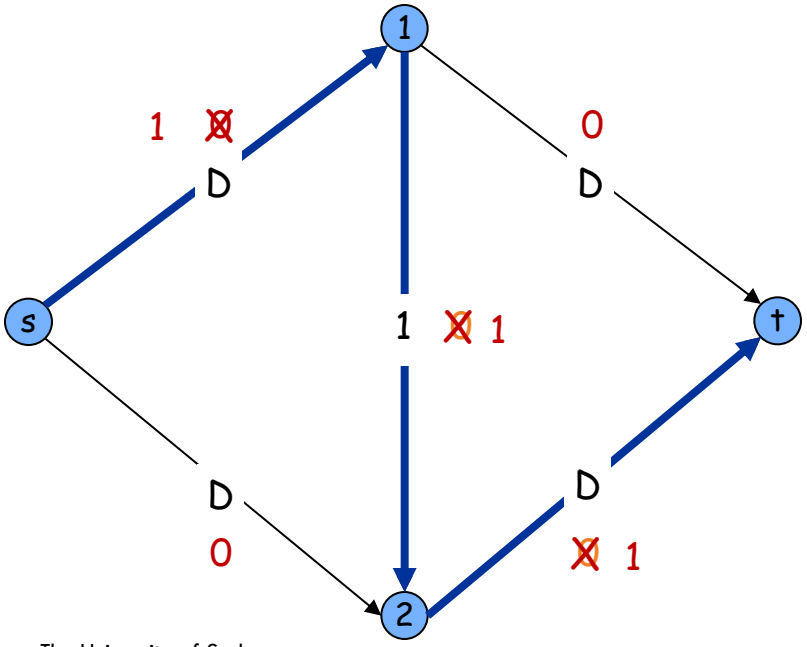
Is O(F(m+n)) a good time bound?

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

Ford-Fulkerson: Exponential Number of Augmentations

Question: Is generic Ford-Fulkerson algorithm polynomial in input size? ← m, n, and log (total capacity)

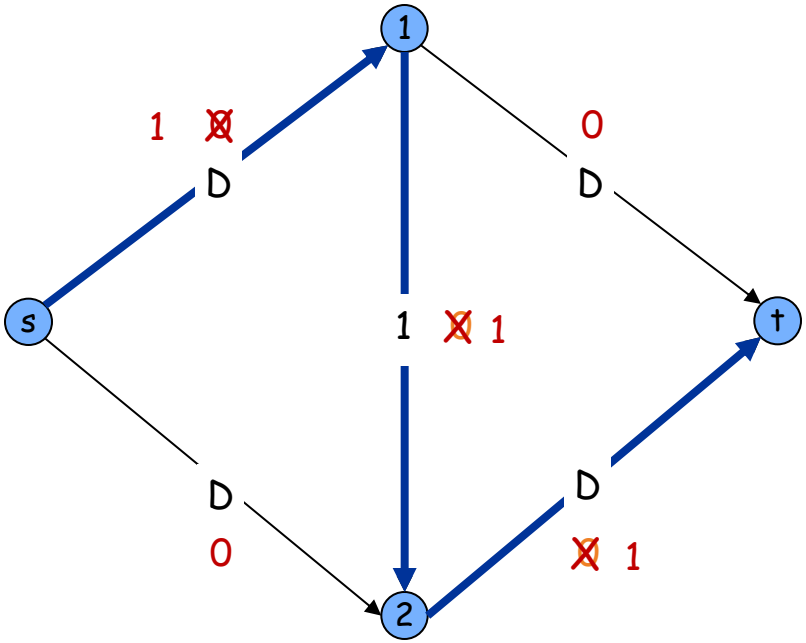
Answer: No. If max capacity is D, then algorithm can take D iterations.

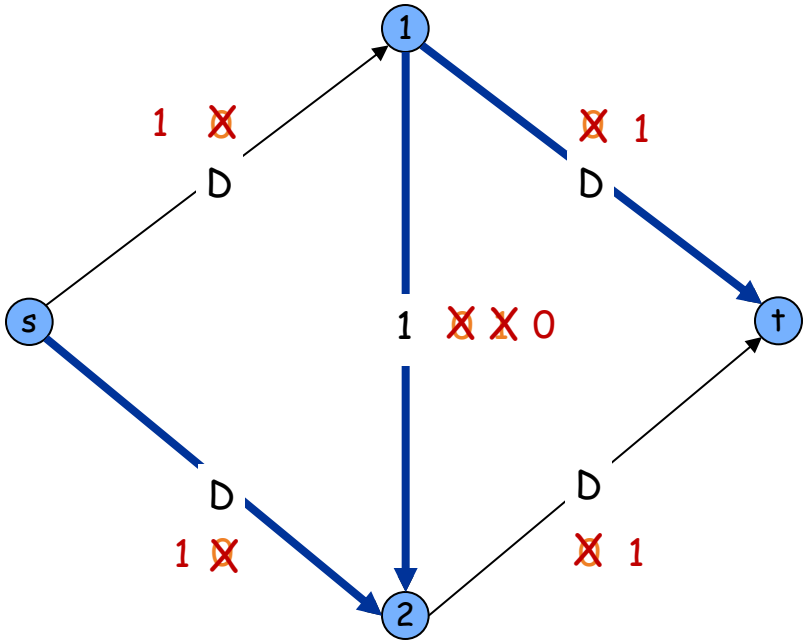


Ford-Fulkerson: Exponential Number of Augmentations

Question: Is generic Ford-Fulkerson algorithm polynomial in input size? ← m, n, and log (total capacity)

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 via capacity scaling (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 bottleneck capacity at least F/m.

Edmonds-Karp #1

Pick the augmenting path with largest capacity

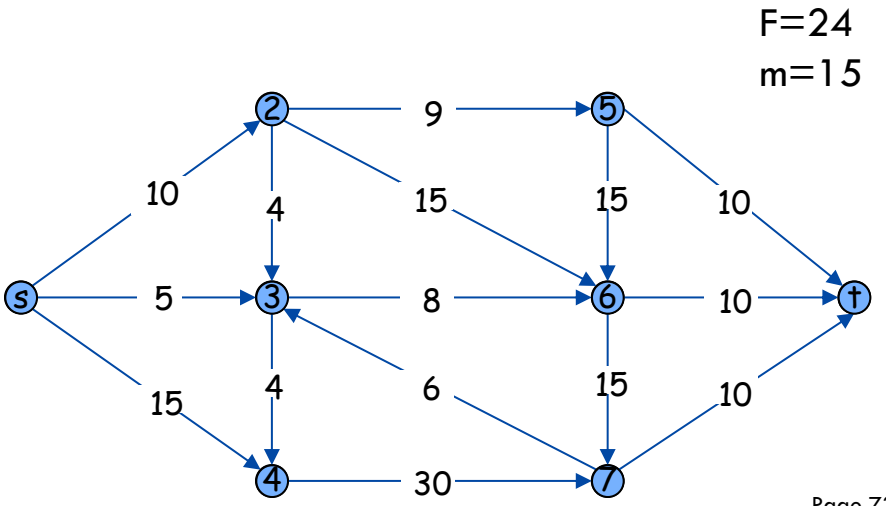
[maximum bottleneck path]

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

Proof:

Delete all edges of capacity less than F/m.

Is the graph still connected?



Edmonds-Karp #1

Pick the augmenting path with largest capacity

[maximum bottleneck path]

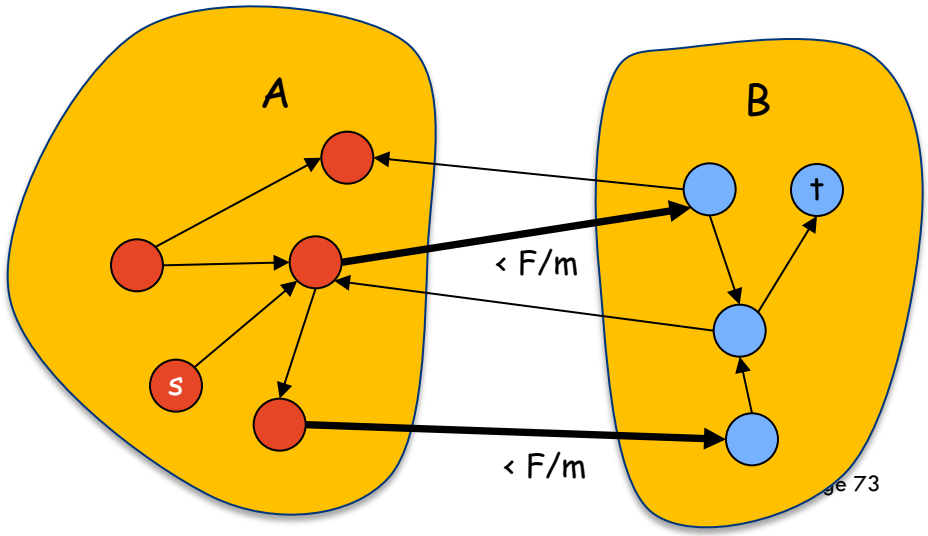
Claim: If maximum flow in G is F, there must exists a path from s to t with bottleneck capacity at least F/m.

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.



Edmonds-Karp #1

Pick the augmenting path with largest capacity

[maximum bottleneck path]

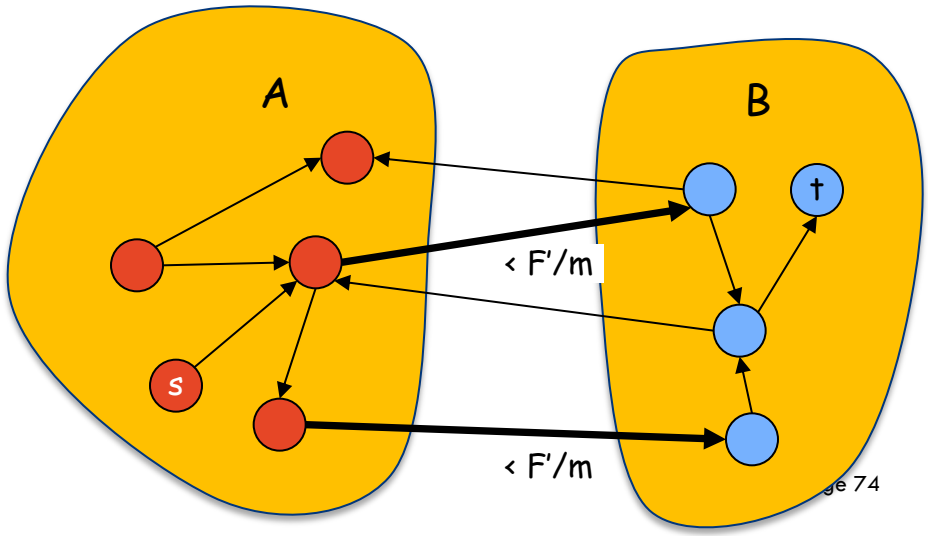
Claim: If maximum flow in residual graph G_f is F', there must exists a path from s to t with bottleneck capacity at least F'/m.

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'.



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?, i.e. what is the smallest x such that $F \cdot (1-1/m)^x < 1$?

Useful inequality: For any a, $(1-1/a)^a < 1/e$

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

Applications (Next lecture)

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

Summary

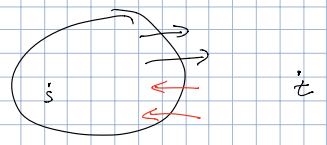
1. Max flow problem
2. Min cut problem
3. Ford-Fulkerson:
    1. Residual graph
    2. correctness
    3. complexity
4. Max-Flow Min-Cut theorem
5. Edmonds-Karp

Appendix: Proof of Flow Value Lemma by Induction

Flow value lemma

Given flow f, cut (A, B)

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



$$v(f) = f^{out}(\{s\}) - \underbrace{f^{in}(\{s\})}_{=0}$$

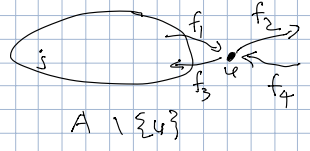
$$v(f) \le f^{out}(A)$$

Pf by induction on $|A|$

Base case: $A = \{s\}$

Inductive case: $|A| > 1$

Let $u \in A \setminus \{s\}$



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

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

Equiv. WTS $\underbrace{f^{out}(A) - f^{out}(A \setminus \{u\})}_{\text{Increase in } f^{out}} = \underbrace{f^{in}(A) - f^{in}(A \setminus \{u\})}_{\text{Inc in } f^{in}}$

$$f_2 - f_1 = f_4 - f_3$$

Flow conservation $\Rightarrow$ $\underbrace{f_1 + f_4}_{f^{in}(u)} = \underbrace{f_2 + f_3}_{f^{out}(u)}$