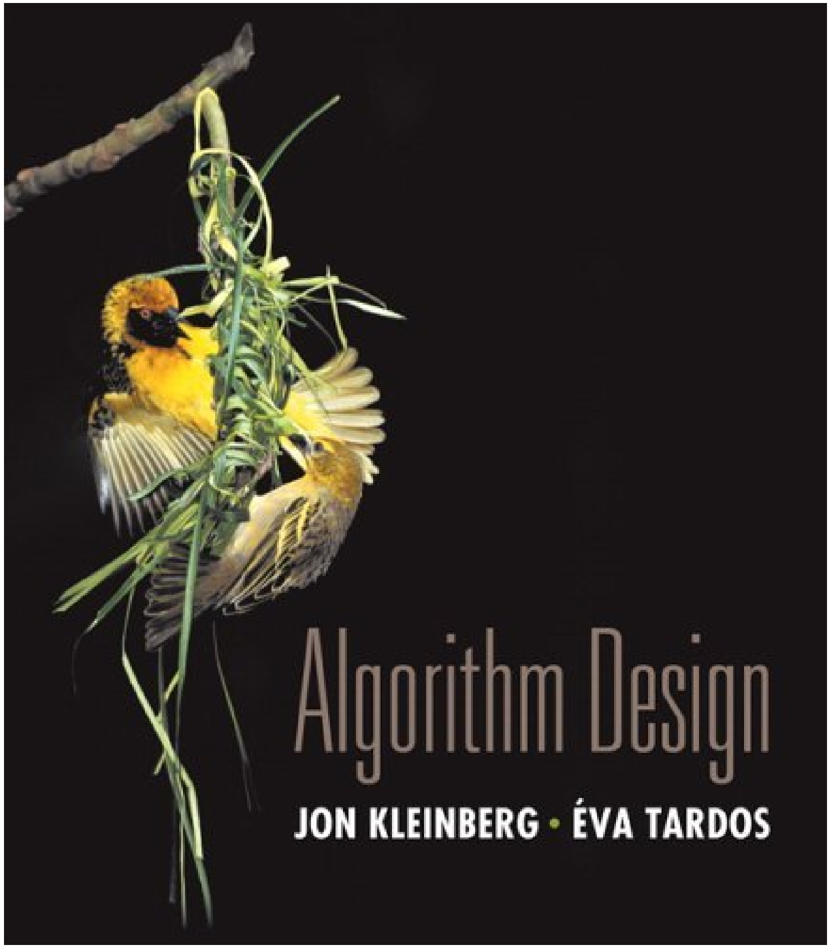
CS 381 – FALL 2019

Week 11.3, Friday, Nov 1

Midterm 2: Grading in progress
Homework 6: Planned Released on Monday, November 4th



Network Flow

Slides by Kevin Wayne.

Max Flow Recap

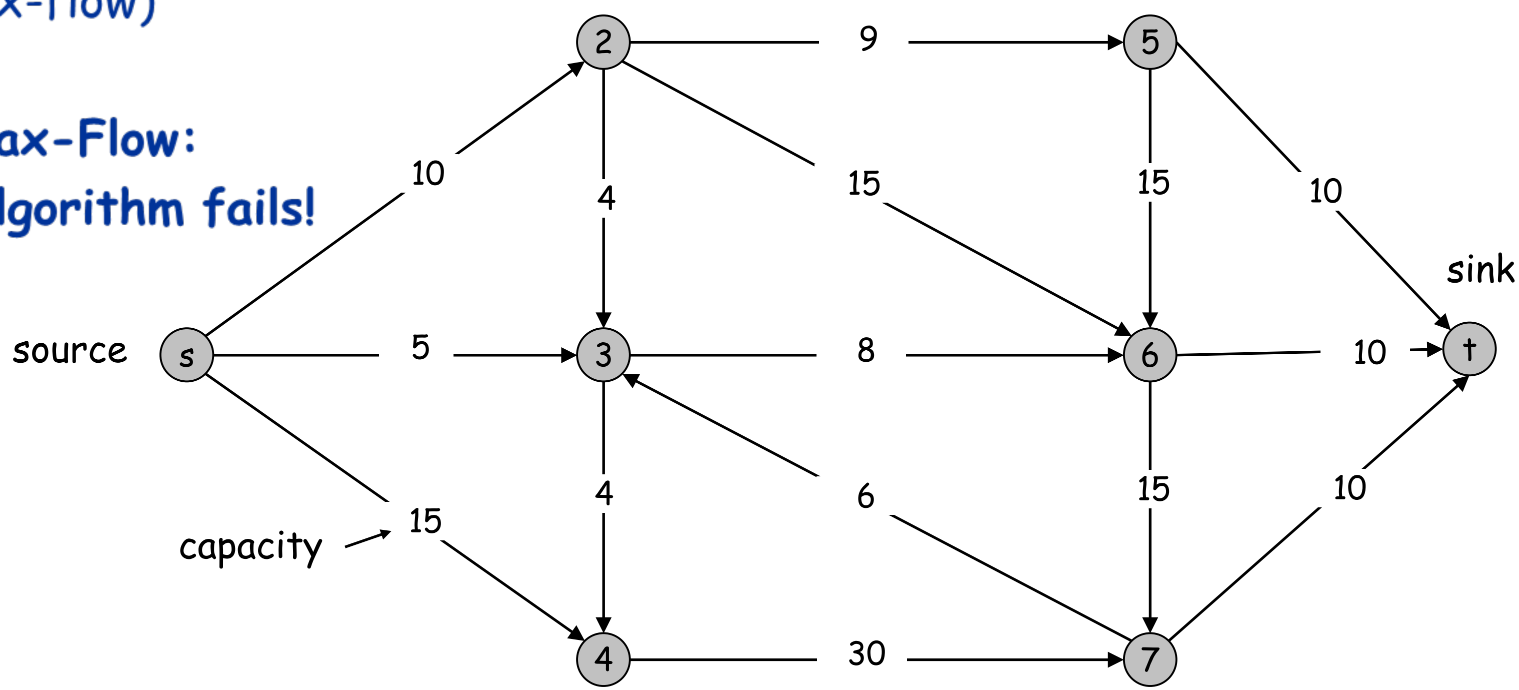
Max-Flow Problem, Min Cut Problem

- **Definition of a s-t flow f(e) and a s-t cut (A,B)**
- **Value of a flow f**
- **Capacity of a s-t cut (A,B)**

Weak Duality Lemma: For any flow f and s-t cut A,B we have $v(f) \leq cap(A,B)$ (i.e., capacity of minimum cut is upper bound on max-flow)

Finding a Max-Flow:

- **Greedy algorithm fails!**

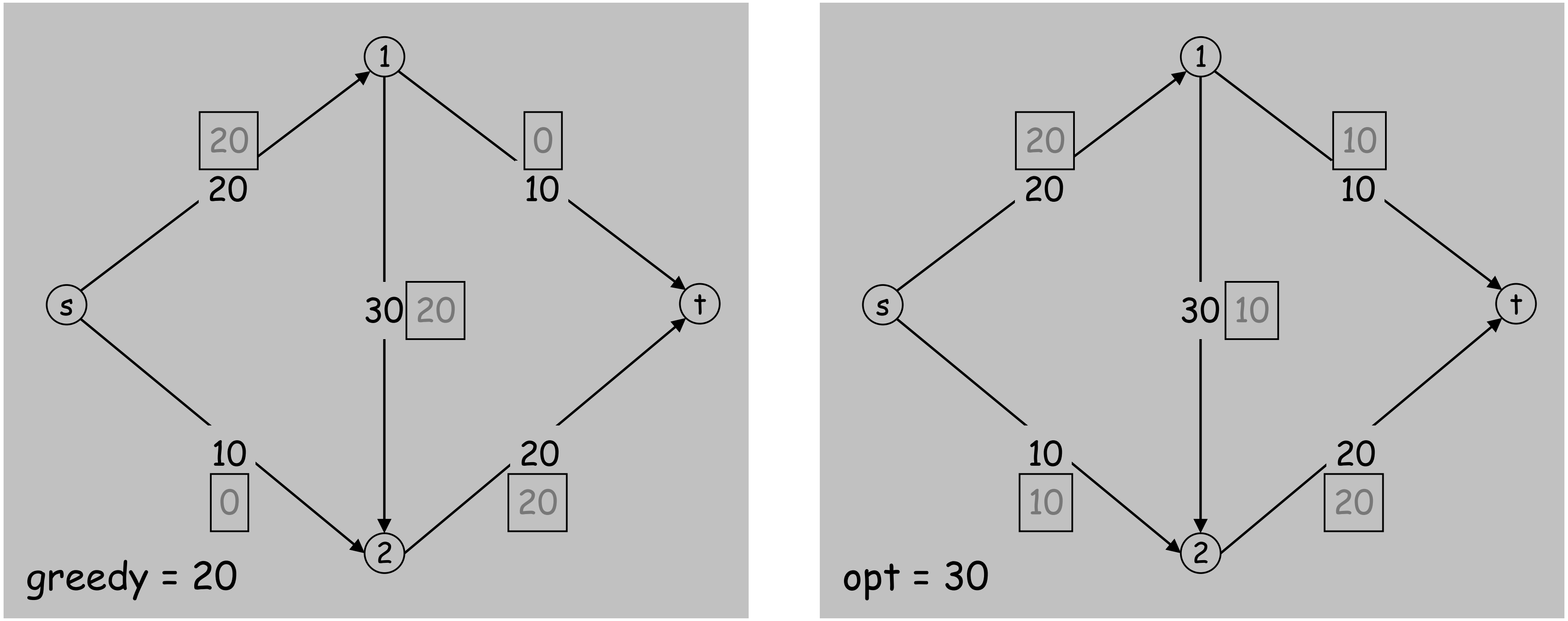


Towards a Max Flow Algorithm

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.

↖ locally optimality $\not\Rightarrow$ global optimality



Clicker Question: Greedy Max Flow Algorithm

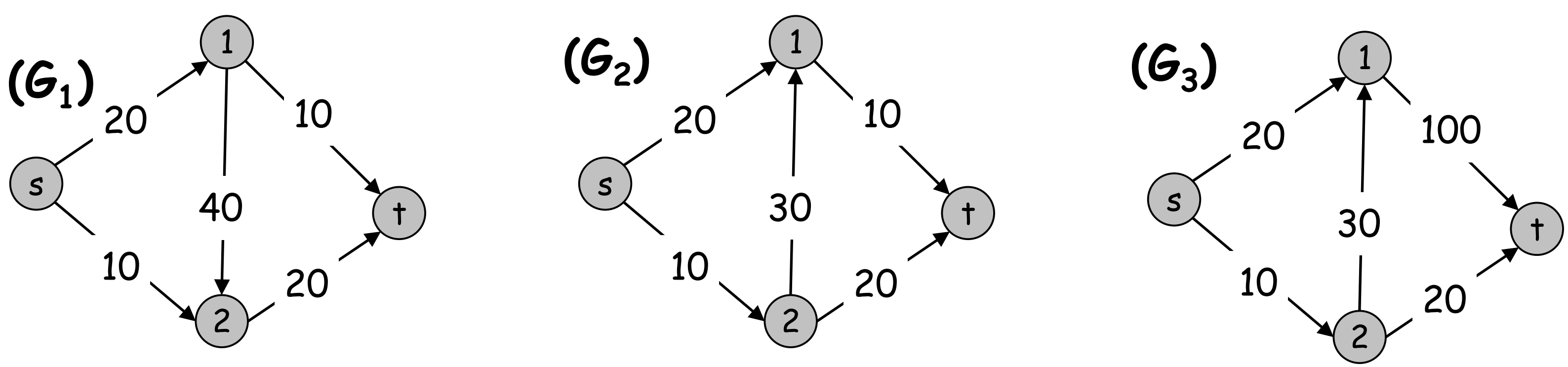
Greedy algorithm.

- 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.

For which of the following graphs is the greedy algorithm guaranteed to find the maximum flow?

A. Graph G_1 only   B. Graph G_2 only   C. Graph G_3 only

D. Graphs G_3 and G_2 E. None of them



Clicker Question: Greedy Max Flow Algorithm

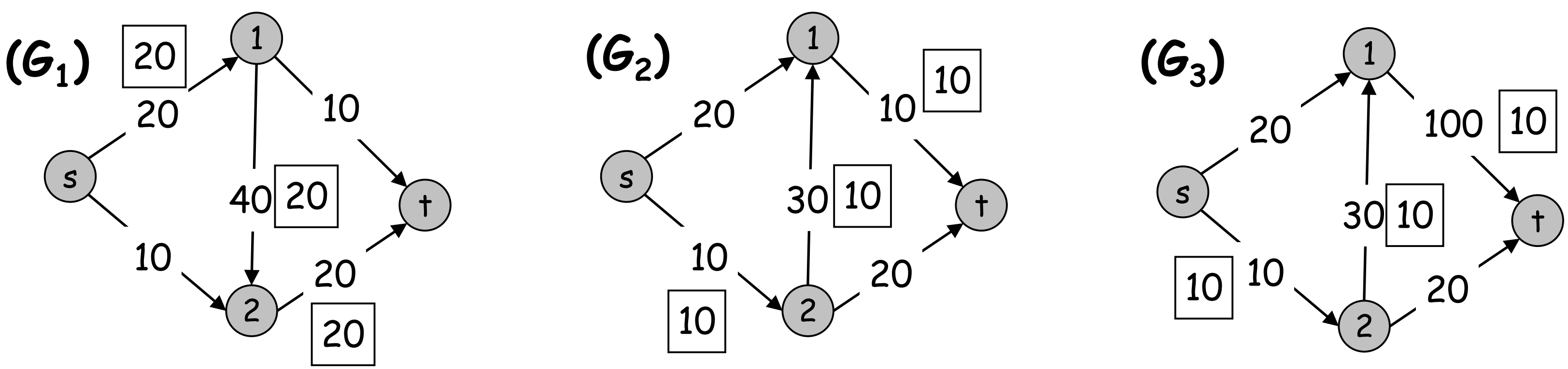
Greedy algorithm.

- 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.

For which of the following graphs is the greedy algorithm guaranteed to find the maximum flow?

A. Graph G_1 only B. Graph G_2 only C. Graph G_3 only

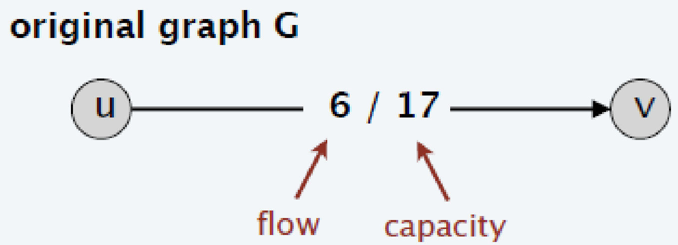
D. Graphs G_3 and G_2 E. None of them



Residual graph

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

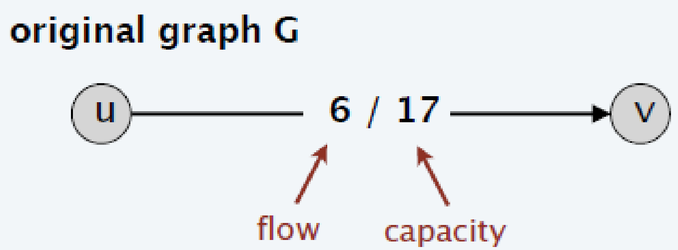
- Flow $f(e)$.
- Capacity $c(e)$.



Residual graph

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

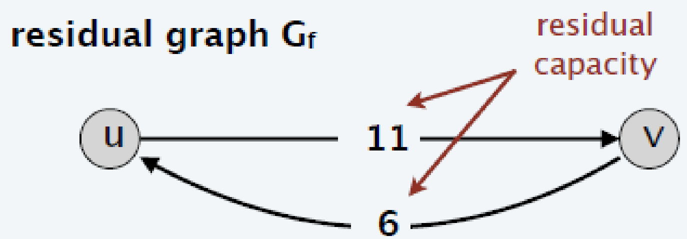
- Flow $f(e)$.
- Capacity $c(e)$.



Residual edge.

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

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

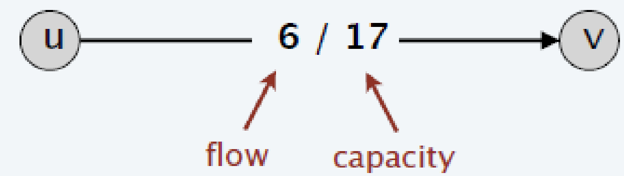


Residual graph

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

- Flow $f(e)$.
- Capacity $c(e)$.

original graph G

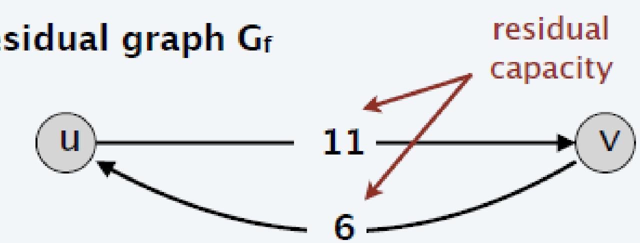


Residual edge.

- "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

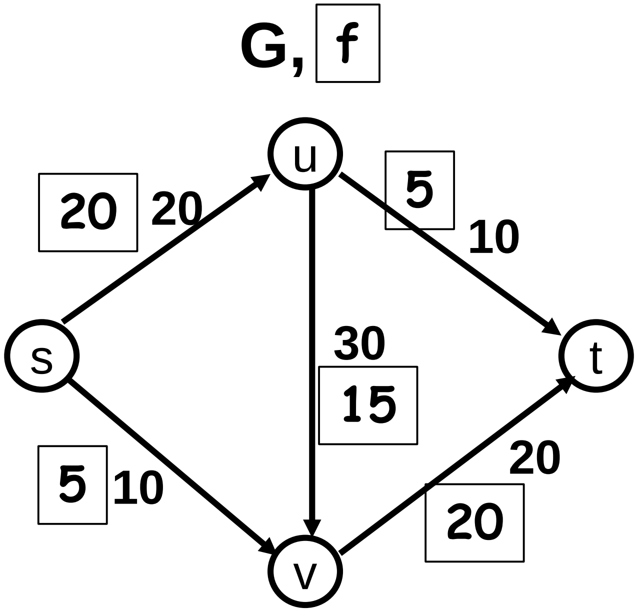


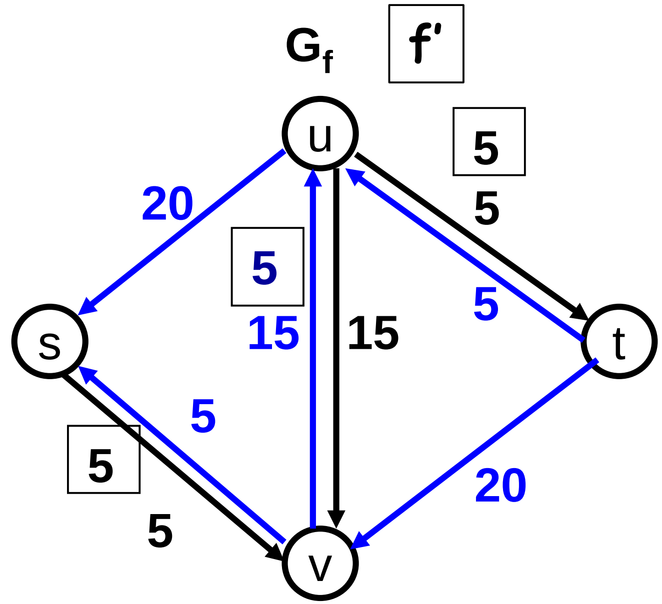
Residual graph: $G_f = (V, E_f)$.

- Residual edges with positive residual capacity.
- $E_f = \{e : f(e) < c(e)\} \cup \{e^R : f(e) > 0\}$.
- Key property: f' is a flow in G_f iff $f + f'$ is a flow in G.

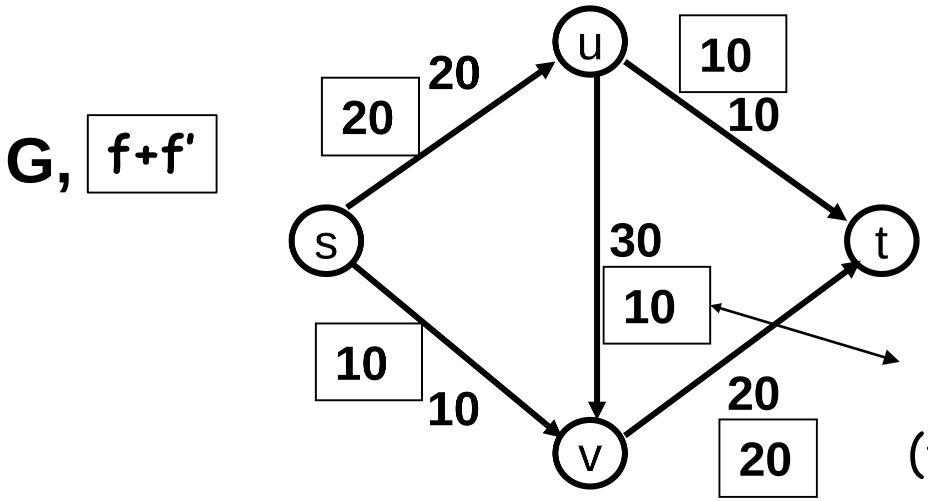
where flow on a reverse edge negates flow on a forward edge

Example





Augmenting path
s → v → u→ t
with bottleneck
capacity 5
Results in a flow of
30



10=f(u,v) – f'(v,u) = 15-5
(flow negates on reverse edge

Augmenting path

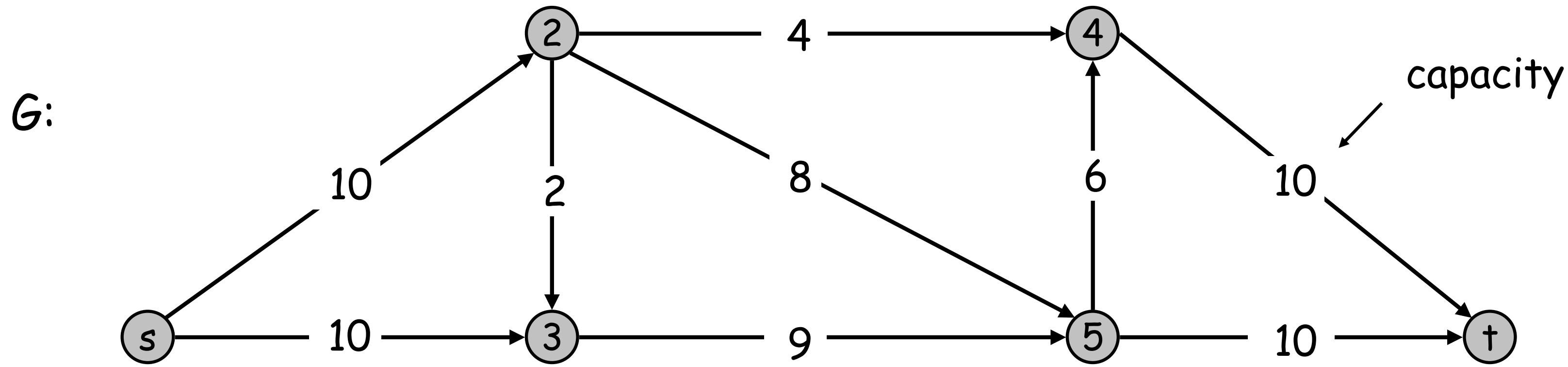
Def. An augmenting path is a simple $s \rightsquigarrow t$ path P in the residual graph G_f.

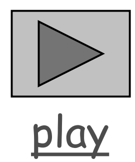
Def. The bottleneck capacity of an augmenting P is the minimum residual capacity of any edge in P.

Key property. Let f be a flow and let P be an augmenting path in G_f. Then f' is a flow and $val(f') = val(f) + bottleneck(G_f, P)$.

```
AUGMENT (f, c, P)
---------------------------------------------
b ← bottleneck capacity of path P.
FOREACH edge e ∈ P
    IF (e ∈ E) f(e) ← f(e) + b.
    ELSE       f(e^R) ← f(e^R) − b.
RETURN f.
---------------------------------------------
```

Ford-Fulkerson Algorithm





Initialize: f(e)=0 //empty flow
While there remains an augmenting path P // s-t path in residual graph G_f
Augment(f,c,P) // Increases v(f)
Update G_f

Augmenting Path Algorithm

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

forward edge

reverse edge

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

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

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. [Elias-Feinstein-Shannon 1956, Ford-Fulkerson 1956]
The value of the max flow is equal to the value of the min cut.

Pf. We prove both simultaneously by showing 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 path.

Proof of Max-Flow Min-Cut Theorem

(iii) $\Rightarrow$ (i)

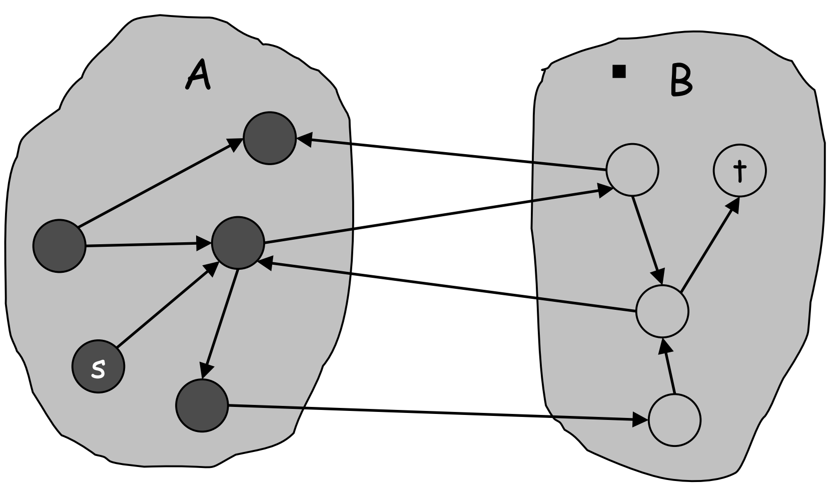
(No augmenting paths relative to f → cap(A,B)=v(g) for some cut A,B)

- 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, $t \notin A$.

Must be 0 since there is no Edge from A to B in residual graph

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

Must be c(e) since there is no edge from A to B in residual graph



original network

Proof of Max-Flow Min-Cut Theorem

(iii) $\Rightarrow$ (i)

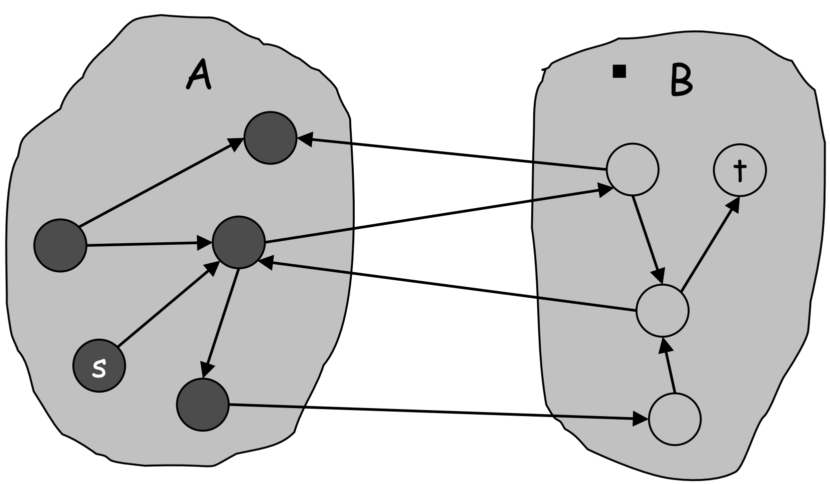
- Let f be a flow with no augmenting paths.
- Let A be set of vertices reachable from s in residual graph.
- By definition of A, $s \in A$.
- By definition of f, $t \notin A$.

Must be 0 since there is no Edge from A to B in residual graph

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

$$= \sum_{\text{e out of A}} c(e)$$

$$= \text{cap}(A, B)$$



original network

Running time

Assumption. Capacities are integers between 1 and C.

Integrality invariant. Throughout the algorithm, the flow values $f(e)$ and the residual capacities $c_f(e)$ are integers.

Theorem. The algorithm terminates in at most $val(f^*) \leq nC$ iterations.
Pf. Each augmentation increases the value by at least 1. ▪

Pseudo-polynomial

Corollary. The running time of Ford-Fulkerson is $O(mnC)$.
Corollary. If $C = 1$, the running time of Ford-Fulkerson is $O(mn)$.

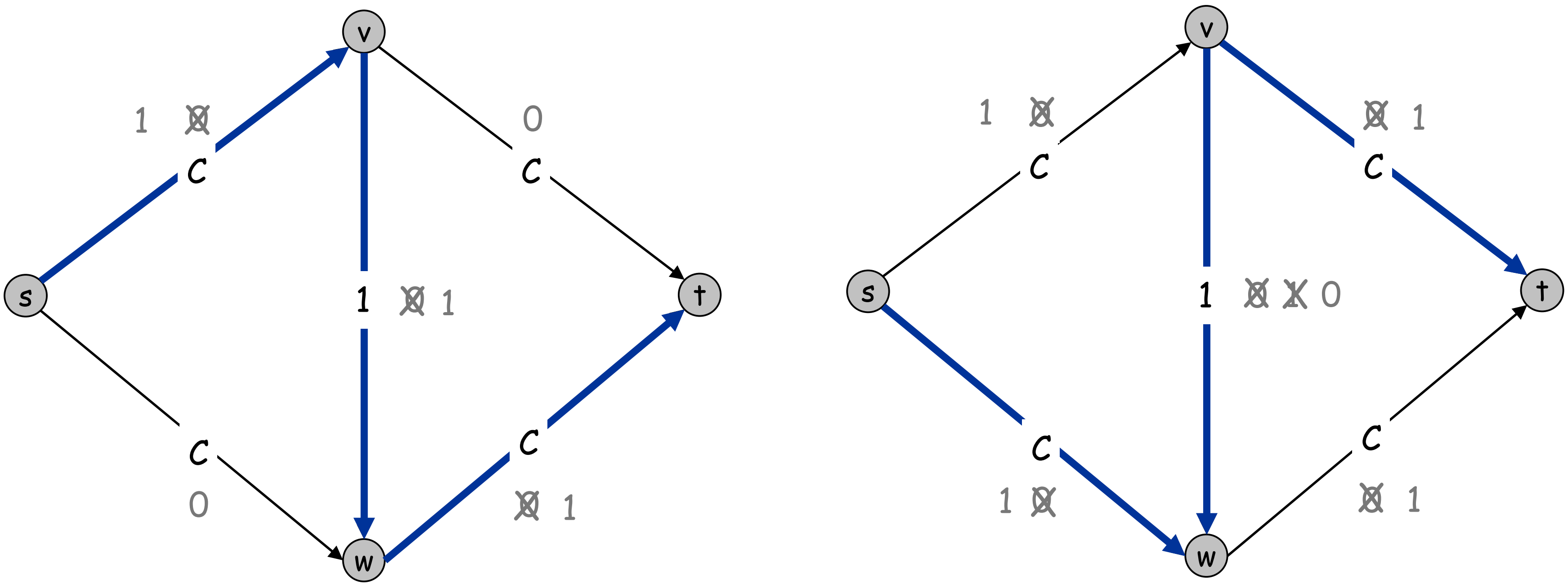
Integrality theorem. Then exists a max-flow f^* for which every flow value $f^*(e)$ is an integer.
Pf. Since algorithm terminates, theorem follows from invariant. ▪

Ford-Fulkerson: Exponential Number of Augmentations

Q. Is generic Ford-Fulkerson algorithm polynomial in input size?

m, n, and log C

A. No. If max capacity is C, then algorithm can take C iterations.



Bad case for Ford-Fulkerson

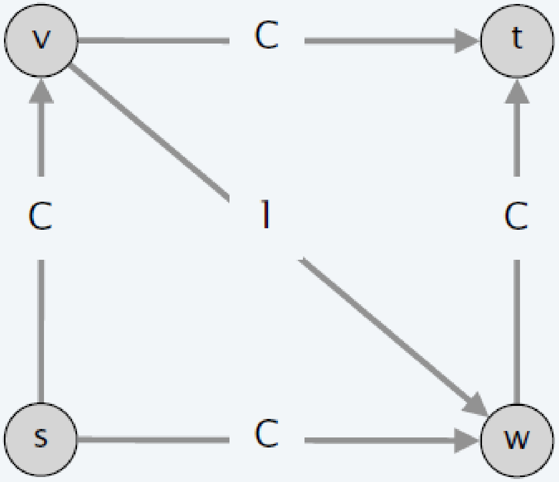
Q. Is generic Ford-Fulkerson algorithm poly-time in input size?

m, n, and log C

A. No. If max capacity is C, then algorithm can take $\geq C$ iterations.

- $s \to v \to w \to t$
- $s \to w \to v \to t$
- $s \to v \to w \to t$
- $s \to w \to v \to t$
- …
- $s \to v \to w \to t$
- $s \to w \to v \to t$

each augmenting path
sends only 1 unit of flow
(# augmenting paths = 2C)



7.3 Choosing Good Augmenting Paths

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 e.g., BFS in residual graph.

Interested in knowing more about MaxFlow?

2014 CACM Review paper by Goldberg and Tarjan posted on Piazza

http://cacm.acm.org/magazines/2014/8/177011-efficient-maximum-flow-algorithms/abstract