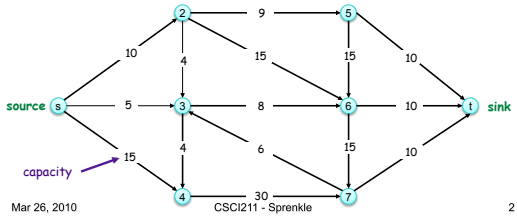
Objectives

- Network Flow
  - Max flow
  - Min cut

Flow Network

- Abstraction for material *flowing* through the edges
- G = (V, E) = directed graph, no parallel edges
- Two distinguished nodes: *s* = source, *t* = sink
- c(e) = capacity of edge e, > 0

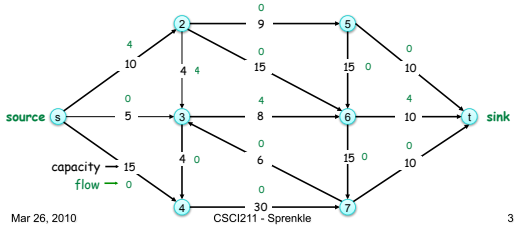


Flows

- An **s-t flow** is a function that satisfies
  - ***Capacity* condition**: For each e ∈ E: 0 ≤ f(e) ≤ c(e)
  - ***Conservation* condition**: For each v ∈ V – {s, t}: $\sum_{e \text{ into } y} f(e) = \sum_{e \text{ out of } y} f(e)$

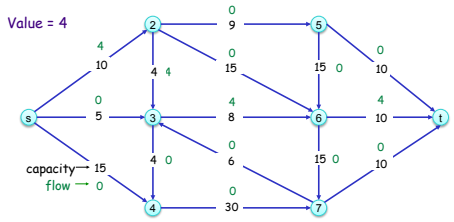
Flow can't exceed capacity

Flow in == Flow out



Flows

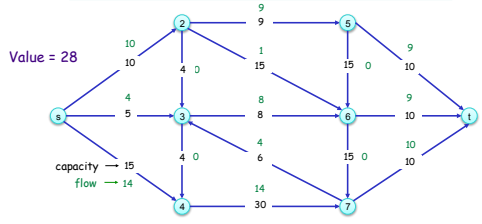
- The value of a flow f is $v(f) = \sum_{e \text{ out of } s} f(e)$



Maximum Flow Problem

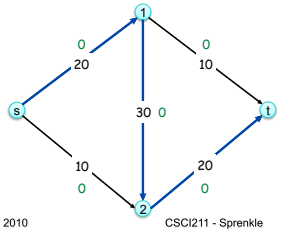
- Make network most efficient
  - Use most of available capacity

Goal: Find *s-t* flow of maximum value



Towards a Max Flow Algorithm

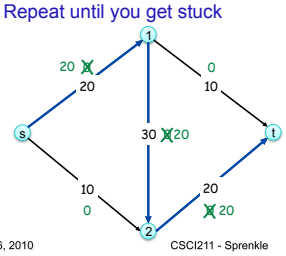
- Greedy algorithm
  - Start with f(e) = 0 for all edge e ∈ E
  - Find an *s-t* path P where each edge has f(e) < c(e)
  - Augment flow along path P
  - Repeat until you get stuck



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 flow along path P
  - Repeat until you get stuck



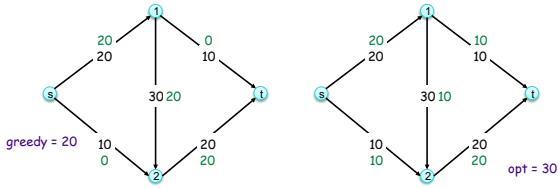
Is this optimal?

Flow value = 20

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 along path P
  - Repeat until you get stuck

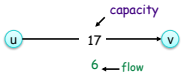
locally optimality does not ⇒ global optimality



RESIDUAL GRAPHS

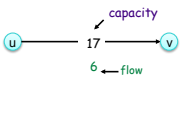
Residual Graph

- Original edge: e = (u, v) ∈ E
  - Flow f(e), capacity c(e)



Residual Graph: G_f

- Original edge: e = (u, v) ∈ E
  - Flow f(e), capacity c(e)
- Residual edge
  - e = (u, v) w/ capacity c(e) - f(e)
  - e^R = (v, u) with capacity f(e)
    - To undo flow
- Residual graph: $G_f = (V, E_f)$
  - Residual edges with *positive* residual capacity
  - $E_f = \{e : f(e) < c(e)\} \cup \{e^R : f(e) > 0\}$



Forward edges *Backward edges*

Applying Residual Graph

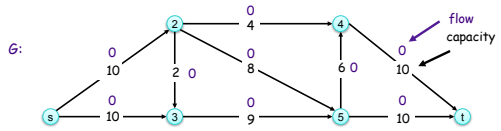
- Used to find the maximum flow
  - Use similar idea to greedy algorithm
- Residual path: simple *s-t* path in G_f
  - Also known as *augmenting path*

Augmenting Path Algorithm

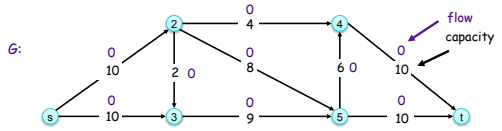
```
Ford-Fulkerson(G, s, t, c)
   foreach e ∈ E  f(e) = 0  # initially no flow
   G_f = residual graph

   while there exists augmenting path P
      f = Augment(f, c, P)     # change the flow
      update G_f               # build a new residual graph

   return f
```

```
Augment(f, c, P)
   b = bottleneck(P) # edge on P with least capacity
   foreach e ∈ P
      if (e ∈ E) f(e) = f(e) + b  # forward edge, ↑ flow
      else       f(e^R) = f(e) - b  # forward edge, ↓ flow
   return f
```


Ford-Fulkerson Algorithm

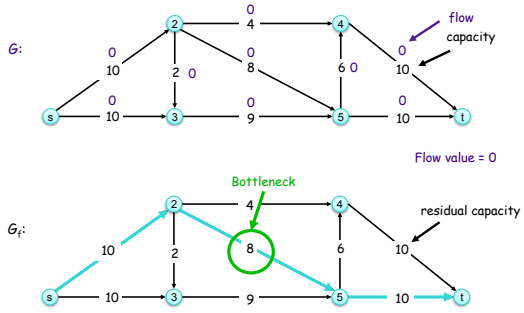


Ford-Fulkerson Algorithm

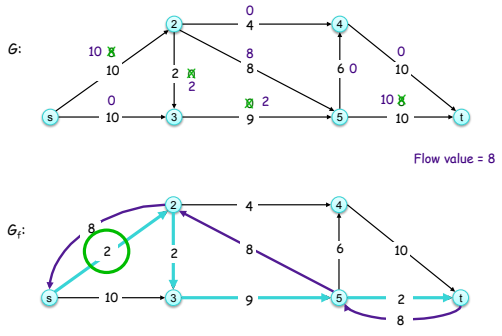


What does the residual graph look like?

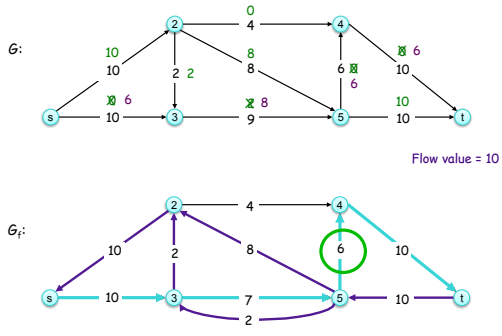
Ford-Fulkerson Algorithm



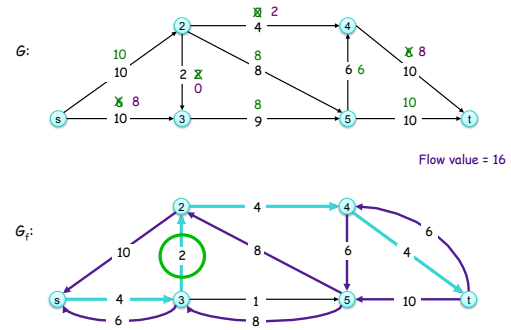
Ford-Fulkerson Algorithm



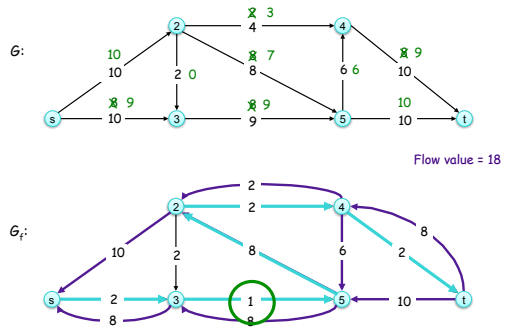
Ford-Fulkerson Algorithm



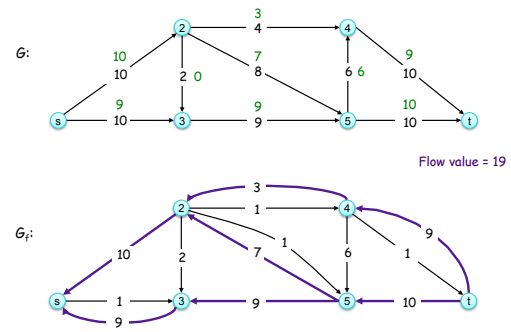
Ford-Fulkerson Algorithm



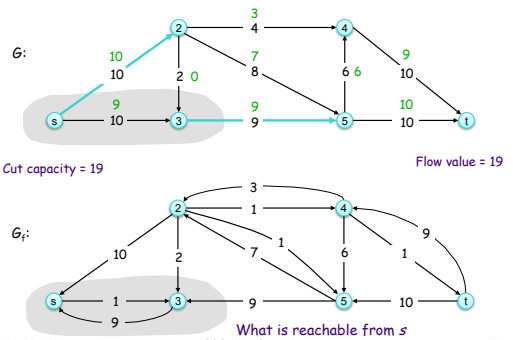
Ford-Fulkerson Algorithm



Ford-Fulkerson Algorithm



Ford-Fulkerson Algorithm



Analyzing Algorithm

- Why does this algorithm work?
- What is happening at each iteration?

Analyzing Augmenting Path Algorithm

```
Ford-Fulkerson(G, s, t, c)
   foreach e ∈ E  f(e) = 0  # initially no flow
   Gf = residual graph

   while there exists augmenting path P
      f = Augment(f, c, P)      # change the flow
      update Gf                 # build a new residual graph

   return f
```

```
Augment(f, c, P)
   b = bottleneck(P) # edge on P with least capacity
   foreach e ∈ P
      if (e ∈ E) f(e) = f(e) + b  # forward edge, ↑ flow
      else       f(eR) = f(e) - b  # forward edge, ↓ flow
   return f
```

Analyzing Augmenting Path Algorithm

```
O(m)  Ford-Fulkerson(G, s, t, c)
O(m)     foreach e ∈ E  f(e) = 0  # initially no flow
O(m)     Gf = residual graph
Find path: O(m);  Iterations: O(F) iterations, where F = max flow
         while there exists augmenting path P
O(m)        f = Augment(f, c, P)      # change the flow
O(m)        update Gf                 # build a new residual graph

         return f
```

Total: O(Fm)

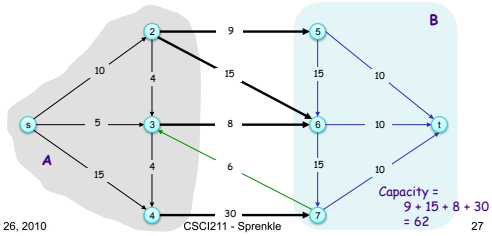
```
O(n)  Augment(f, c, P)
O(n)  b = bottleneck(P) # edge on P with least capacity
O(n)  foreach e ∈ P
O(1)     if (e ∈ E) f(e) = f(e) + b  # forward edge, ↑ flow
O(1)     else       f(eR) = f(e) - b  # forward edge, ↓ flow
      return f
```

Total: O(n) → O(m), since n ≤ 2m

MINIMUM CUTS

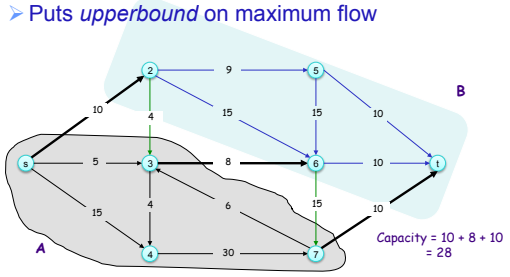
Cuts

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



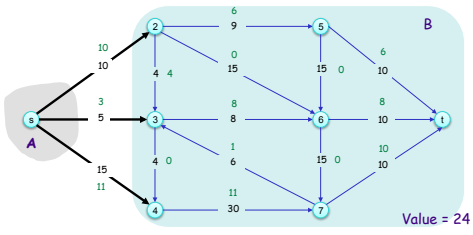
Minimum Cut Problem

- Find an *s-t* cut of *minimum* capacity
  - Puts *upperbound* on maximum flow



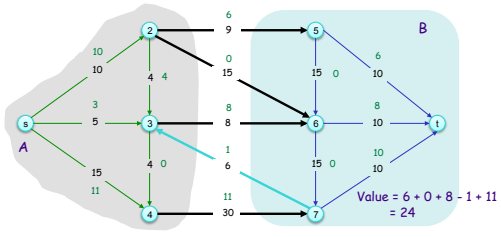
Flow Value Lemma

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



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



Flow Value Lemma

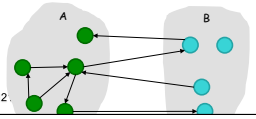
- Let *f* be any flow, and let (A, B) be any *s-t* cut.
- Then $\sum_{e \text{ out of } A} f(e) - \sum_{e \text{ in to } A} f(e) = v(f).$
- Pf. By definition $v(f) = \sum_{e \text{ out of } s} f(e)$

by flow conservation, all terms except v = s are 0 →

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

Possibilities for edge e:
- Both ends in A (0)
- Points out from A (+)
- Points in to A (-)

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



Mar 26, 2010 CSCI2 31