Objectives

- Network Flow
  - Max flow
  - Min cut
  - Application: Bipartite Matching

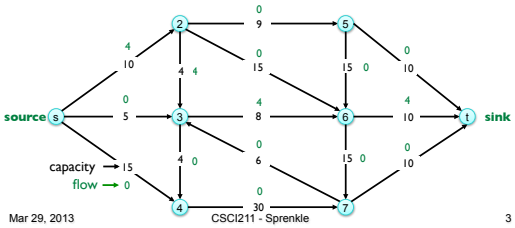
Review

- What are the characteristics of the network flow graph we're dealing with?
- What was the problem we were trying to solve?
- Describe our algorithm to solve the problem

Review: Flows: Definitions

Flow can't exceed capacity

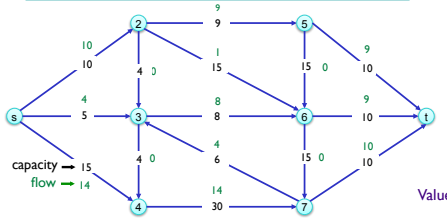
- An **s-t flow** is a function that satisfies
  - ***Capacity* condition**: For each e ∈ E: 0 ≤ f(e) ≤ c(e)
  - ***Conservation* condition**: For each v ∈ V – {s, t}: $\sum_{e \text{ into } y} f(e) = \sum_{e \text{ out of } y} f(e)$ ← *Flow in == Flow out*



Maximum Flow Problem

- Make network most efficient
  - Use most of available capacity

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



Augmenting Path Algorithm

c=capacity

```
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

```
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
```

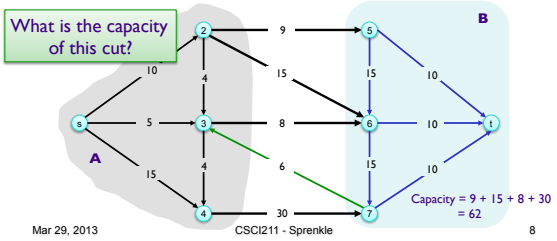
```
Augment(f, c, P)
   b = bottleneck(P) # edge on P with least capacity
   foreach e ∈ P
      if (e ∈ E) f(e) = f(e) + b  # forward edge, ↑ flow
      else       f(eR) = f(e) - b  # forward edge, ↓ flow
   return f
```

Why does alg work? What is happening at each iteration?
What is the running time? Need more analysis …

MINIMUM CUTS

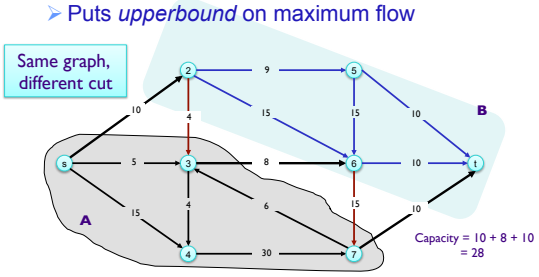
Cuts

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



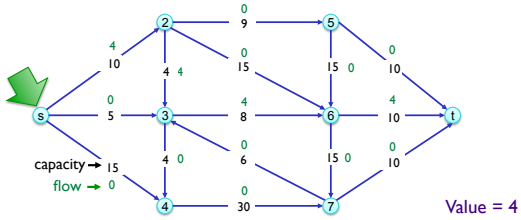
Minimum Cut Problem

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



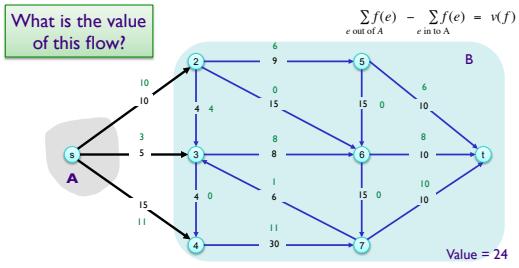
Recall

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



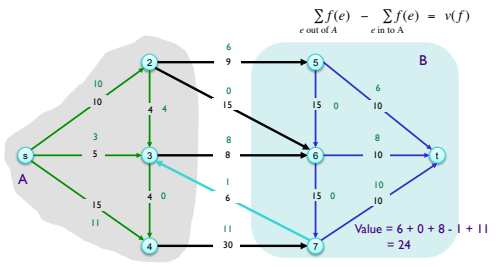
Flow Value Lemma

- Let *f* be *any* flow, and let (A, B) be *any s-t* cut. Then, the **value of the flow** is = $f^{out}(A) - f^{in}(A)$.



Flow Value Lemma

- Let *f* be *any* flow, and let (A, B) be *any s-t* cut. Then, the **value of the flow** is = $f^{out}(A) - f^{in}(A)$.

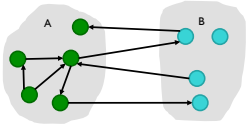


Flow Value Lemma (FVL)

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

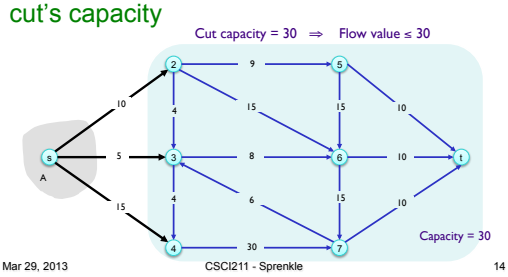
$$
\begin{aligned}
v(f) &= \sum_{e \text{ out of } s} f(e) && \text{By definition} \\
&= \sum_{e \text{ out of } s} f(e) + \sum_{v \in A \neq s} \Big(\sum_{e \text{ out of } v} f(e) - \sum_{e \text{ into } v} f(e) \Big) && \text{by flow conservation, all terms except v = s are 0} \\
&= \sum_{e \text{ out of } s} f(e) - \sum_{e \text{ into } s} f(e) + \sum_{v \in A \neq s} \Big(\sum_{e \text{ out of } v} f(e) - \sum_{e \text{ into } v} f(e) \Big) \\
&= \sum_{v \in A} \Big(\sum_{e \text{ out of } v} f(e) - \sum_{e \text{ into } v} f(e) \Big) \\
&= \sum_{e \text{ out of } A} f(e) - \sum_{e \text{ in to } A} f(e)
\end{aligned}
$$

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



Weak Duality

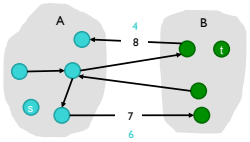
- Let *f* be any flow and let (A, B) be any *s-t* cut.
- ➜Then the value of the flow is *at most* the cut's capacity



Weak Duality

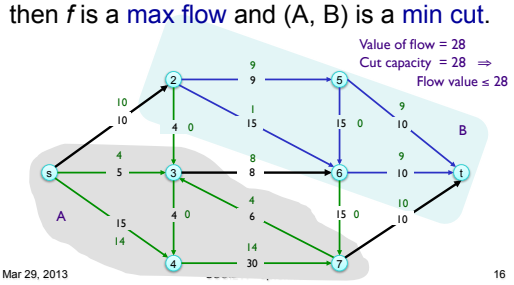
- Let *f* be any flow.
  Then, for any *s-t* cut (A, B) v(f) ≤ cap(A, B).
- Pf.

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



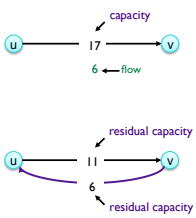
Certificate of Optimality

- 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.



Recall: 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

Recall: 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
```

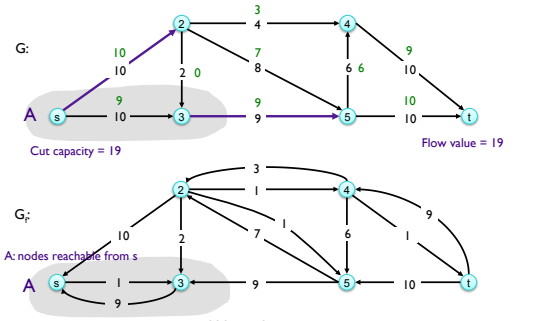
```
Augment(f, c, P)
   b = bottleneck(P) # edge on P with least capacity
   foreach e ∈ P
      if (e ∈ E) f(e) = f(e) + b  # forward edge, ↑ flow
      else       f(eR) = f(e) - b  # forward edge, ↓ flow
   return f
```

Intuition Behind Correctness of F-F Algorithm

- Let A be set of vertices *reachable* from *s* in residual graph at end of F-F alg execution
- By definition of A, s ∈ A
- By definition of the F-F algorithm's resulting flow, t ∉ A

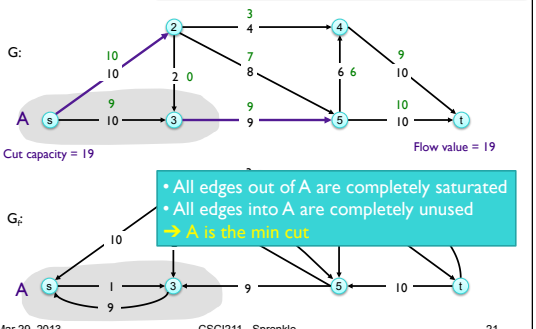
Ford-Fulkers

- What do we know about the flow out of A?
- What do we know about the flow into A?



Ford-Fulkers

- What do we know about the flow out of A?
- What do we know about the flow into A?



- All edges out of A are completely saturated
- All edges into A are completely unused
- → A is the min cut

Max-Flow Min-Cut Theorem

Augmenting path theorem.
Flow *f* is a max flow iff there are no augmenting paths.

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

- Proof strategy. Prove both simultaneously by showing 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.

See formal proof in book

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

```
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)

```
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

Running Time

- Assumption. All capacities are integers between 1 and F.
- Invariant. Every flow value f(e) and every residual capacity's $c_f(e)$ remains an integer throughout algorithm.

- Theorem. Algorithm terminates in at most $v(f^*) \leq nF$ iterations.
- Pf. Each augmentation increases value by at least 1.
- Corollary. If F = 1, Ford-Fulkerson runs in O(mn) time.

- Integrality theorem. If all capacities are integers, then there exists a max flow *f* for which every flow value f(e) is an integer.
- Pf. Since algorithm terminates, theorem follows from invariant.

Discussion: Max Flow Problem

- What is the form of the solution to the max flow problem?
- Is there only one solution to a given max flow problem?

Power of Max Flow Problem

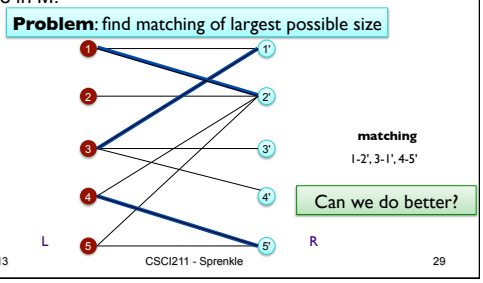
Some problems with non-trivial combinatorial searches can be formulated as **max flow** or **min cut** in a directed graph

BIPARTITE MATCHING

Bipartite Matching

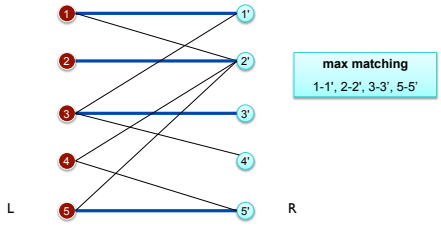
- Input: undirected, bipartite graph G = (L ∪ R, E) (V)
  - Edges: one end in L, one end in R
- Matching M ⊆ E such that each node appears in at most 1 edge in M.

Problem: find matching of largest possible size



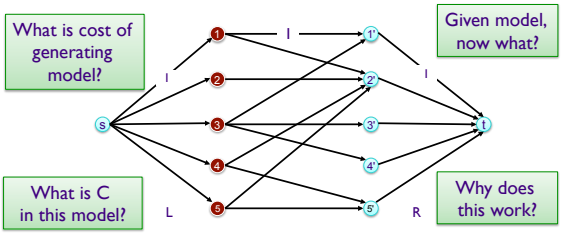
Bipartite Matching

- Input: undirected, bipartite graph G = (L ∪ R, E) (V)
  - Edges: one end in L, one end in R
- Matching M ⊆ E such that each node appears in at most 1 edge in M.



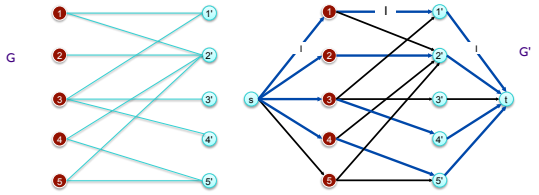
Max Flow Formulation

1. Create digraph G' = (L ∪ R ∪ {s, t}, E')
2. Direct all edges from L to R, and assign unit capacity
3. Add source s, and unit capacity edges from s to each node in L
4. Add sink t, and unit capacity edges from each node in R to t



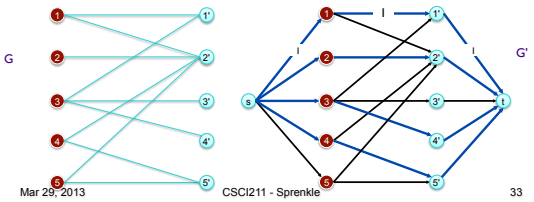
Bipartite Matching: Proof of Correctness

- Theorem. Max cardinality matching in G = value of max flow in G'.
- Proof: Need to show in both directions



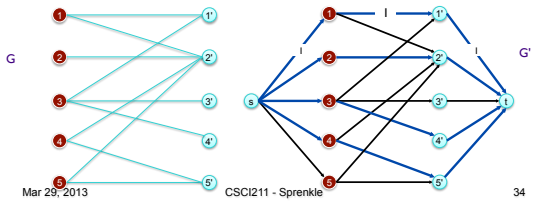
Bipartite Matching: Proof of Correctness

- Theorem. Max cardinality matching in G = value of max flow in G'.
- Pf. →
  - Given max matching M of cardinality *k*.
  - Consider flow f that sends 1 unit along each of *k* paths.
  - *f* is a flow and has cardinality *k*. ▪



Bipartite Matching: Proof of Correctness

- Theorem. Max cardinality matching in G = value of max flow in G'.
- Pf. ←
  - Let f be a max flow in G' of value k.
  - Integrality theorem ⇒ k is integral and can assume f is 0-1.
  - Consider M = set of edges from L to R with f(e) = 1.
    - each node in L and R participates in at most one edge in M
    - |M| = k: consider cut (L ∪ s, R ∪ t) ▪



Network Flow Solutions

1. Model problem as a flow network
   - Describe what nodes, edges, and capacity represent
   - Describe what flow represents and how that maps to your solution
   - Run Ford-Fulkerson algorithm
2. Prove that the solution found is correct/ feasible/optimal
3. Prove that you find all solutions
4. Analyze running time
   - Creating model
   - FF algorithm

This Week

- Problem Set 9 due Friday
- Wiki Reading
  - 7.1-7.2, 7.5, 7.7