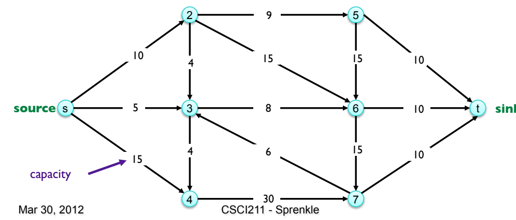
Objectives

- Network Flow
  - Wrap up Max flow, Min cut
  - Applications

Review: 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

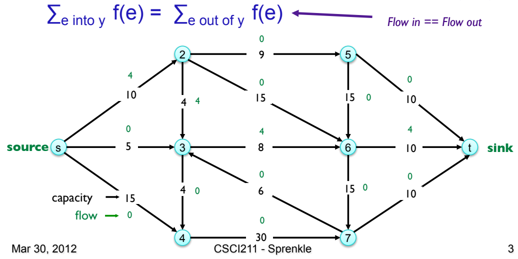


Review: Flows

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

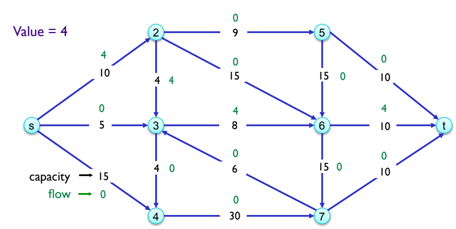
Flow can't exceed capacity

Flow in == Flow out



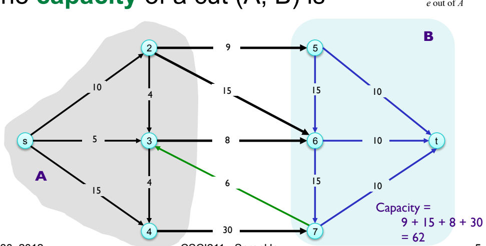
Review: Flows

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



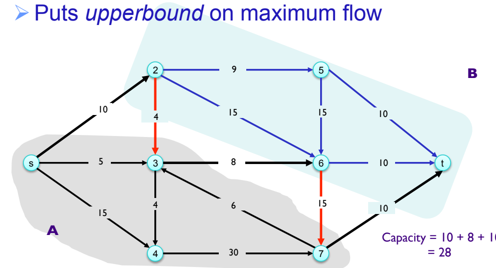
Review: Cuts

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



Review: Minimum Cut Problem

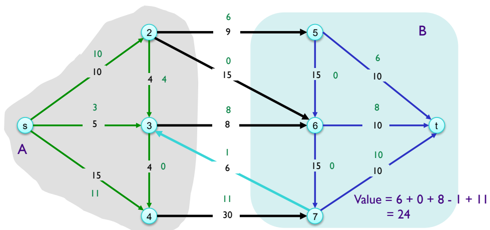
- **Goal**: Find an *s-t* cut of *minimum* capacity
  - Puts *upperbound* on maximum flow



Review: Flow Value Lemma

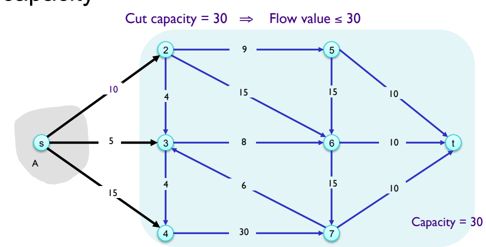
- Let *f* be any flow, and let (A, B) be any *s-t* cut. Then, the ***net flow*** sent across the cut is equal to the amount leaving *s*.

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



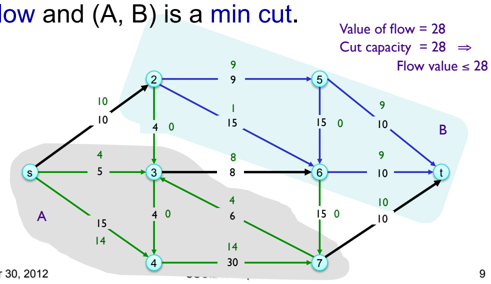
Review: Weak Duality

- 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



Review: 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.



Review

- What is the Ford-Fulkerson algorithm?
  - When does it stop?

Analyzing 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
```

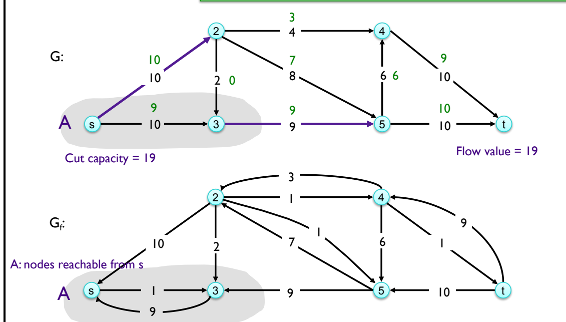
```
Augment(f, c, P)
  b = bottleneck(P) # edge on P with least capacity
  foreach e ∈ P
     if (e ∈ E) f(e) = f(e) + b  # forward edge, ↑ flow
     else       f(e^R) = f(e) - b  # forward edge, ↓ flow
  return f
```


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

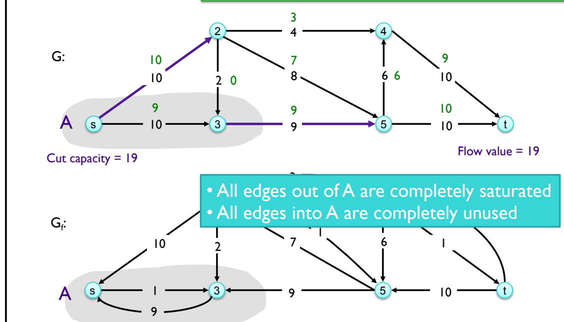
Ford-Fulkerson

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



Ford-Fulkerson

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



Max-Flow Min-Cut Theorem

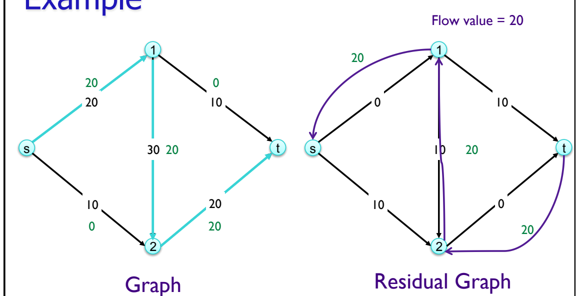
Augmenting path theorem.
Flow *f* is a max flow iff there are no augmenting paths.

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

Example



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 C.
- Invariant. Every flow value f(e) and every residual capacity's $c_f(e)$ remains an integer throughout algorithm.

- Theorem. The algorithm terminates in at most $v(f^*) \leq nC$ iterations.
- Pf. Each augmentation increases value by at least 1.
- Corollary. If C = 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.

Power of Max Flow Problem

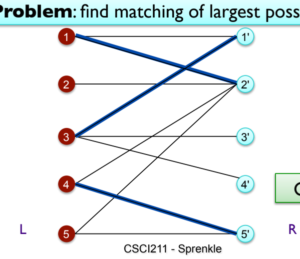
Some problems with non-trivial combinatorial searches can be formulated as **max flow** or **min cut** in a directed graph

BIPARTITE MATCHING

Bipartite Matching

- Input: undirected, bipartite graph G = $(\overbrace{L \cup R}^{V}, E)$
  - Edges: one end in L, one end in R
- Matching $M \subseteq E$ such that each node appears in at most 1 edge in M.

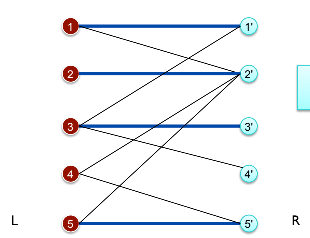
Problem: find matching of largest possible size



Can we do better?

Bipartite Matching

- Input: undirected, bipartite graph G = $(\overbrace{L \cup R}^{V}, E)$
  - Edges: one end in L, one end in R
- Matching $M \subseteq E$ such that each node appears in at most 1 edge in M.

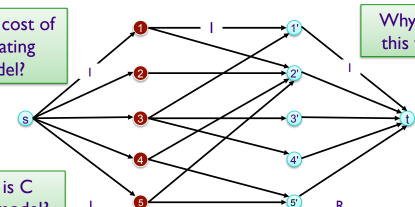


max matching
1-1', 2-2', 3-3' 4-4'

Max Flow Formulation

1. Create digraph G' = ($L \cup R \cup \{s, t\}$, 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

What is cost of generating model?

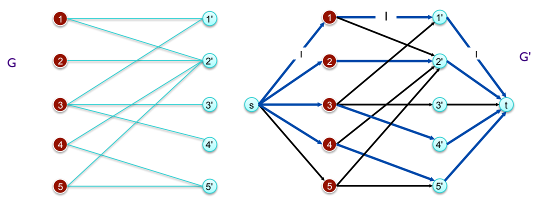


Why does this work?

What is C in this model?

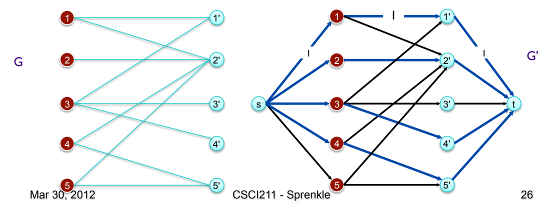
Bipartite Matching: Proof of Correctness

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



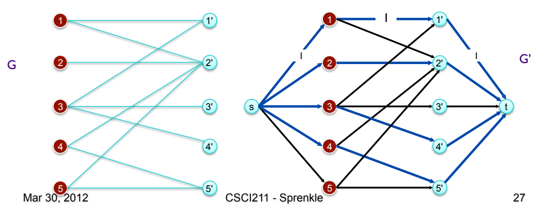
Bipartite Matching: Proof of Correctness

- 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 ($L \cup s$, $R \cup t$) ▪



Summary of Approach

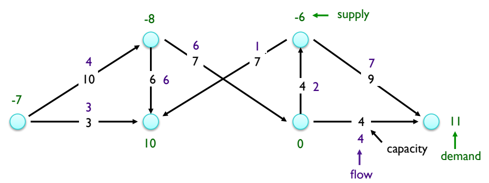
1. Model problem as a flow network
2. Run Ford-Fulkerson algorithm
3. Analyze running time
   - Creating model
   - FF algorithm

EXTENSIONS TO MAX FLOW

Circulation with Demands

- Directed graph G = (V, E)
- Edge capacities c(e), $e \in E$
- Node supply and demands d(v), $v \in V$
  - d(v) > 0 → demand
  - d(v) < 0 → supply
  - d(v) = 0 → transshipment

Example Graph: Circulation with Demands



Circulation with Demands

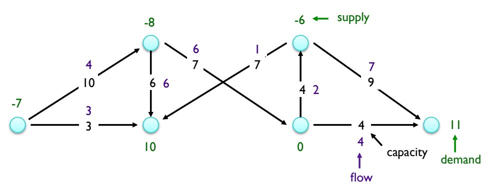
- Circulation with demands
  - Directed graph G = (V, E)
  - Edge capacities c(e), e ∈ E
  - Node supply and demands d(v), v ∈ V

demand if d(v) > 0; supply if d(v) < 0; transshipment if d(v) = 0

- Def. A ***circulation*** is a function that satisfies:
  - For each e ∈ E: $0 \le f(e) \le c(e)$ (capacity)
  - For each v ∈ V: $\sum_{e \text{ in to } v} f(e) - \sum_{e \text{ out of } v} f(e) = d(v)$ (conservation)

Circulation problem:
given (V, E, c, d), does a circulation exist?
(Can we satisfy demand with supply?)

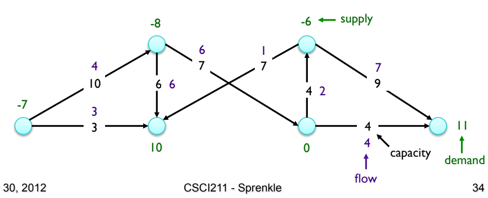
Example Graph: Circulation with Demands



Circulation with Demands

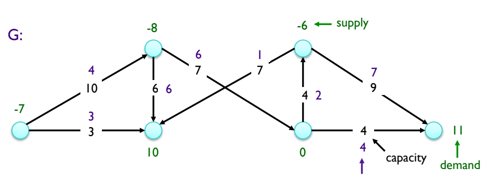
- Necessary condition:
  sum of supplies = sum of demands

$$\sum_{v : d(v) > 0} d(v) = \sum_{v : d(v) < 0} -d(v) =: D$$



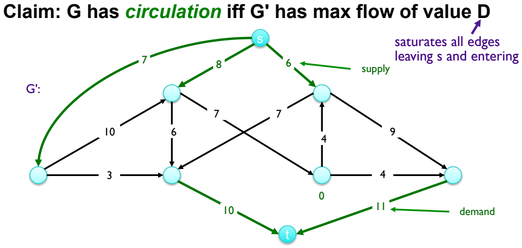
Circulation with Demands: Towards Max Flow Formulation

Ideas about how we can formulate this as a max flow problem?



Circulation with Demands: Max Flow Formulation

- Add new source s and sink t
- For each v with d(v) < 0, add edge (s, v) with capacity -d(v)
- For each v with d(v) > 0, add edge (v, t) with capacity d(v)
- **Claim: G has *circulation* iff G' has max flow of value D**



Circulation with Demands: Characterization

- Given (V, E, c, d), there does **not** exist a circulation iff there exists a node partition (A, B) such that

$$\Sigma_{v \in B}\, d_v > \text{cap}(A, B)$$

demand by nodes in B — exceeds — supply of nodes in B + max capacity of edges going from A → B

- Pf?
  - What can we use to prove this?

Circulation with Demands: Characterization

- Given (V, E, c, d), there does **not** exist a circulation iff there exists a node partition (A, B) such that

$$\Sigma_{v \in B}\, d_v > \text{cap}(A, B)$$

demand by nodes in B — exceeds — supply of nodes in B + max capacity of edges going from A → B

- Pf idea. Look at min cut in G'.

ANOTHER EXTENSION: LOWER BOUNDS

Circulation with Demands and Lower Bounds

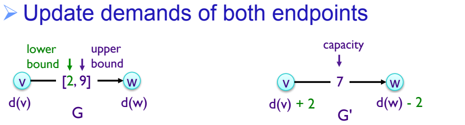
Force flow to use certain edges

- Feasible circulation
  - Directed graph G = (V, E)
  - Edge capacities c(e) and lower bounds ℓ (e), e ∈ E
  - Node supply and demands d(v), v ∈ V
- Def. A *circulation* is a function that satisfies:
  - For each e ∈ E: $0 \le \ell(e) \le f(e) \le c(e)$ (capacity)
  - For each v ∈ V: $\sum_{e \text{ in to } v} f(e) - \sum_{e \text{ out of } v} f(e) = d(v)$ (conservation)

Circulation problem with lower bounds. Given (V, E, ℓ, c, d), does a circulation exist?

Circulation with Demands and Lower Bounds

- Model lower bounds with demands
  - Send ℓ(e) units of flow along edge e
  - Update demands of both endpoints



Supply and demand decrease

Proof in book

7.8 SURVEY DESIGN

Survey Design

- Design survey asking consumers about products
- Can only survey a consumer about a product if they own it
  - Consumer can own multiple products
- Ask consumer i between c_i and c_i' questions
- Ask between p_j and p_j' consumers about product j

Goal: Design a survey that meets these specs, if possible.

How can we model this problem?

Bipartite Graph

- Nodes: customers and products
- Edge between customer and product means customer owns product
- For each customer, range of # of products asked about
- For each product, range of # of customers asked about it

What does the flow represent?

Next Week

- Wiki - Tuesday
  - Skip the rest of Chapter 6 (unless you want to)
  - Chapter 7 up through 7.2, 7.5, 7.7
- Problem Set 9 due Friday
  - Implementing pretty print
  - Network flow problems
    - As usual, check out the solved exercises at end of chapter