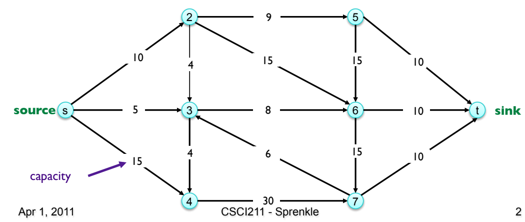
Objectives

- Network Flow
  - Max flow, Min cut
  - Choosing good augmenting paths
  - 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

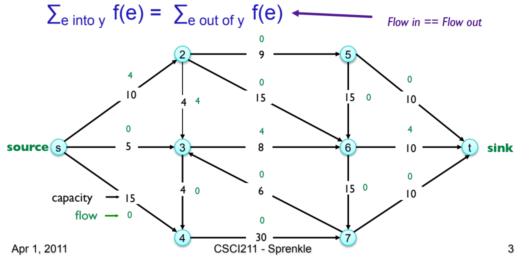


Review: Flows

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

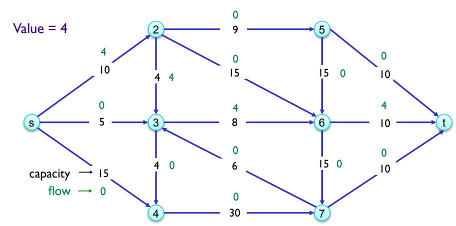
Flow can't exceed capacity

Flow in == Flow out



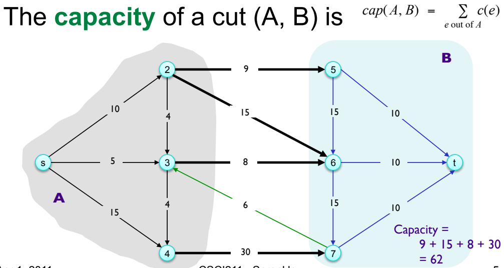
Review: Flows

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



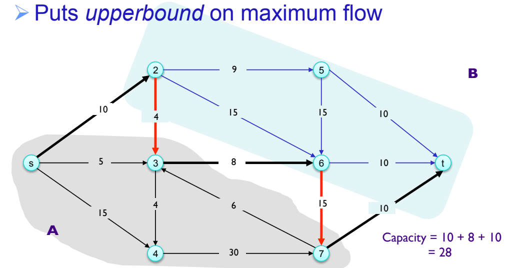
Review: Cuts

- 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

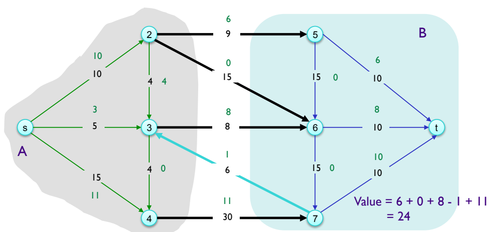
- **Goal**: Find an *s-t* cut of *minimum* capacity
  - Puts *upperbound* on maximum flow



Review: Flow Value Lemma

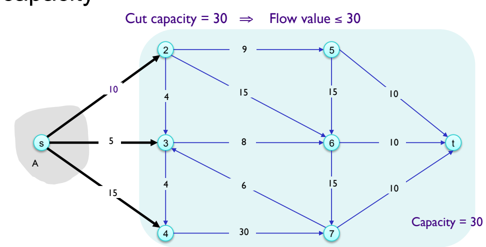
- Let f be any flow, and let (A, B) be any s-t cut. Then, the ***net flow*** sent across the cut is equal to the amount leaving s.

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



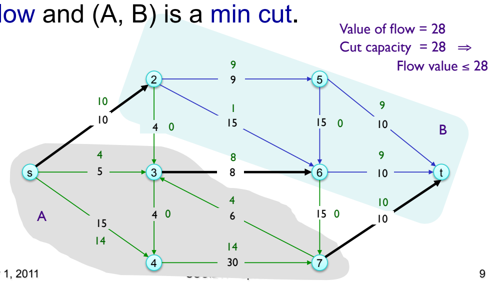
Review: Weak Duality

- 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

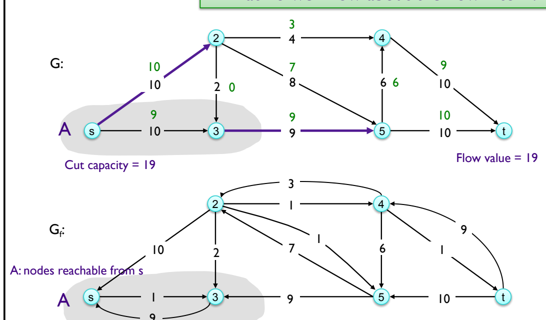
- What is the Ford-Fulkerson algorithm?
  - When does it stop?

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, $s \in A$
- By definition of the F-F algorithm's resulting flow, $t \notin A$

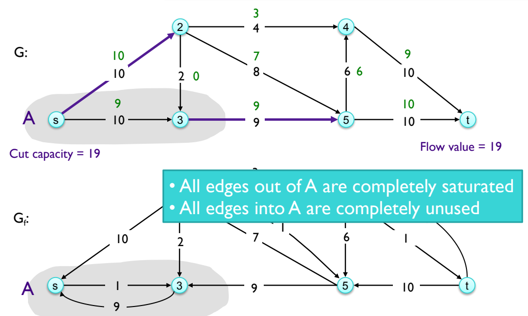
Ford-Fulkers

- 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?



- All edges out of A are completely saturated
- All edges into A are completely unused

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(C) iterations, where C = max capacity from s (and, therefore, 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(Cm)

```
     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

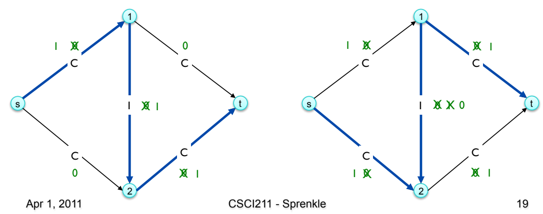
Running Time

- 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*) ≤ 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.

CHOOSING GOOD AUGMENTING PATHS

Ford-Fulkerson: Exponential Number of Augmentations

- Is generic Ford-Fulkerson algorithm polynomial in input size?
  - No. If max capacity is C, then algorithm can take C iterations.

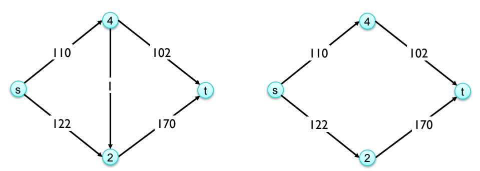


Choosing Good Augmenting Paths

- 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
- [Edmonds-Karp 1972, Dinitz 1970] Choose augmenting paths with:
  - Max bottleneck capacity
  - *Sufficiently large bottleneck capacity*
  - Fewest number of edges

Intuition for Capacity Scaling

- Choosing path with highest bottleneck capacity increases flow by max possible amount.
  - Don't worry about finding *exact* highest bottleneck path
  - Maintain scaling parameter Δ
  - Let $G_f(\Delta)$ be the subgraph of the residual graph consisting of only edges with capacity at least Δ



G_f $G_f(100)$

Capacity Scaling

```
Scaling-Max-Flow(G, s, t, c)
   foreach e ∈ E,  f(e) = 0
   Δ = greatest power of 2 less than or equal to C
   Gf = residual graph
   Gf(Δ) = Δ-residual graph

   while Δ ≥ 1:
       while there exists augmenting path P in Gf(Δ) :
           f = augment(f, c, P)
           update Gf(Δ)
       Δ = Δ / 2

   return f
```

- Why does this work?
- What is its running time?

Capacity Scaling

```
Scaling-Max-Flow(G, s, t, c)
   foreach e ∈ E,  f(e) = 0
   Δ = greatest power of 2 less than or equal to C
   Gf = residual graph
   Gf(Δ) = Δ-residual graph

   while Δ ≥ 1:
       while there exists augmenting path P in Gf(Δ) :
           f = augment(f, c, P)
           update Gf(Δ)
       Δ = Δ / 2

   return f
```

After Δ-scaling phase, pretty close to max possible flow

Capacity Scaling: Correctness

- Assumption. All edge capacities are integers between 1 and C.
- Integrality invariant. All flow and residual capacity values are integral.
- Correctness. If the algorithm terminates, then *f* is a max flow.
- Pf.
  - By integrality invariant, when $\Delta = 1 \Rightarrow G_f(\Delta) = G_f$
  - Upon termination of $\Delta = 1$ phase, there are no augmenting paths. ▪

Capacity Scaling: Running Time

- Lemma 1. The outer while loop repeats $O(\log_2 C)$ times.
- Proof. Initially $\Delta \leq C$. Δ decreases by a factor of 2 each iteration. ▪

Capacity Scaling: Running Time

What happens to the flow's value at each iteration of the loop?

- Lemma 2. Let f be the flow at the end of a Δ-scaling phase. Then value of the maximum flow is at most $v(f) + m\,\Delta$.

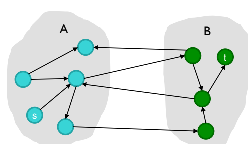
Capacity Scaling: Running Time

- Lemma 2. Let f be the flow at the end of a Δ-scaling phase. Then value of the maximum flow is at most $v(f) + m\,\Delta$.
- Proof. (similar to proof of max-flow min-cut theorem)
  - Show that at the end of a Δ-phase, there exists a cut (A, B) such that $cap(A, B) \leq v(f) + m\,\Delta$.
  - Choose A to be the set of nodes reachable from s in $G_f(\Delta)$.
  - By definition of A, $s \in A$.
  - By definition of f, $t \notin A$.

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

Bound on flow values across cut

Graph contains *m* edges



Capacity Scaling: Running Time

- Lemma 3. There are at most 2m augmentations per scaling phase.
  - Let f be the flow at the end of the previous scaling phase.
  - L2 $\Rightarrow$ $v(f^*) \leq v(f) + m\,(2\Delta)$.
  - Each augmentation in a Δ-phase increases v(f) by at least Δ. ▪
- Theorem. The scaling max-flow algorithm finds a max flow in O(m log C) *augmentations*.
  - Can be implemented to run in $O(m^2 \log C)$ time

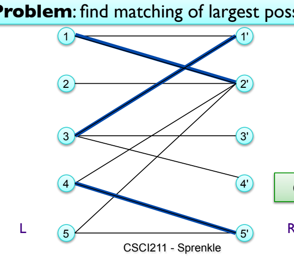
Edge's added capacity at this stage is at most 2Δ

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

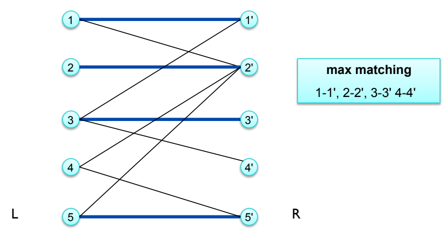


matching
1-2', 3-1', 4-5'

Can we do better?

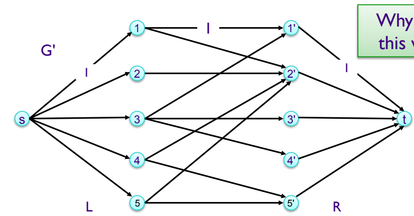
Bipartite Matching

- 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 Flow Formulation

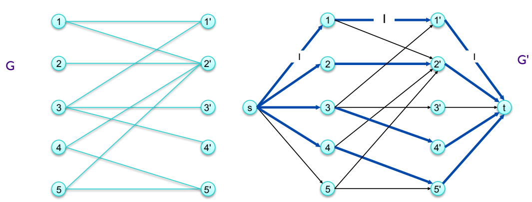
1. Create digraph $G' = (L \cup R \cup \{s, t\}, E')$
2. Direct all edges from L to R, and assign unit capacity
3. Add source s, and unit capacity edges from s to each node in L
4. Add sink t, and unit capacity edges from each node in R to t



Why does this work?

Bipartite Matching: Proof of Correctness

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



Next Week

- Wiki - Wednesday
  - Finish reading Chapter 6 (6.9)
  - Up through 7.3
- Problem Set 9 due Friday
  - Network flow problems