Objectives

- Network Flow Wrap Up
  - Application: Airline Scheduling
- Algorithms Retrospective
- Computational intractability

Review

- What is a flow network?
- What is our usual goal given a flow network?
  - How do we reach that goal?
- What is the Ford-Fulkerson algorithm?
- What is the min-cut?
  - How does it relate to the max flow?
- How is network flow useful?

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
```

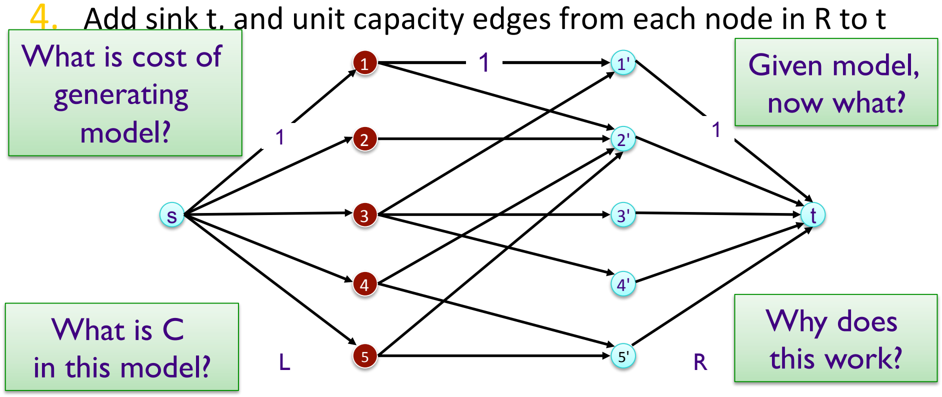
Total: O(Fm)

```
   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: $O(n) \rightarrow O(m)$, since $n \leq 2m$

Review: 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



Review

- What is our process in solving problems using network flow?

Review: 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
     - Map back to original problem
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

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: $0 \leq \ell(e) \leq f(e) \leq c(e)$ (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?

7.9 AIRLINE SCHEDULING

Airline Scheduling

- **Scheduling goal**: efficient in terms of equipment usage, crew allocation, customer satisfaction, …
- **Our simplified problem:**
  - Flight segment: origin & destination airport, departure & arrival time
  - Use a plane for two flight segments (i, j) if
    - i's destination == j's origin & enough time to perform maintenance on plane OR
    - Add a flight segment in between that gets plane to j's origin with adequate time in between

Scheduling Planes

- Maintenance time: 1 hour

Number	Origin	Departure	Destination	Arrival
1	Boston	6 a.m.	DC	7 a.m.
2	Philadelphia	7 a.m.	Pittsburgh	8 a.m.
3	DC	8 a.m.	LAX	11 a.m.
4	Philadelphia	11 a.m.	San Francisco	2 p.m.
5	San Francisco	2:15 p.m.	Seattle	3:15 p.m.
6	Las Vegas	5 p.m.	Seattle	6 p.m.

What is a valid use of one plane for > 1 segment?

Scheduling Planes

- Maintenance time: 1 hour

Number	Origin	Departure	Destination	Arrival
1	Boston	6 a.m.	DC	7 a.m.
2	Philadelphia	7 a.m.	Pittsburgh	8 a.m.
3	DC	8 a.m.	LAX	11 a.m.
4	Philadelphia	11 a.m.	San Francisco	2 p.m.
5	San Francisco	2:15 p.m.	Seattle	3:15 p.m.
6	Las Vegas	5 p.m.	Seattle	6 p.m.

What is a valid use of one plane for > 1 segment?

1 → 3 → 6

Problem Statement

- A flight j is *reachable* from flight i if it is possible to use the same plane for flight j as flight i

Goal: Determine if it's possible to serve all *m* flights using at most *k* planes

Scheduling Planes

- Maintenance time: 1 hour

Number	Origin	Departure	Destination	Arrival
1	Boston	6 a.m.	DC	7 a.m.
2	Philadelphia	7 a.m.	Pittsburgh	8 a.m.
3	DC	8 a.m.	LAX	11 a.m.
4	Philadelphia	11 a.m.	San Francisco	2 p.m.
5	San Francisco	2:15 p.m.	Seattle	3:15 p.m.
6	Las Vegas	5 p.m.	Seattle	6 p.m.

Could we schedule all flights from previous example with only 2 planes?

Scheduling Planes

- Maintenance time: 1 hour

Number	Origin	Departure	Destination	Arrival
1	Boston	6 a.m.	DC	7 a.m.
2	Philadelphia	7 a.m.	Pittsburgh	8 a.m.
3	DC	8 a.m.	LAX	11 a.m.
4	Philadelphia	11 a.m.	San Francisco	2 p.m.
5	San Francisco	2:15 p.m.	Seattle	3:15 p.m.
6	Las Vegas	5 p.m.	Seattle	6 p.m.

Yes.
Plane A: 1 → 3 → 5
Plane B: 2 → 4 → 6

Problem Statement

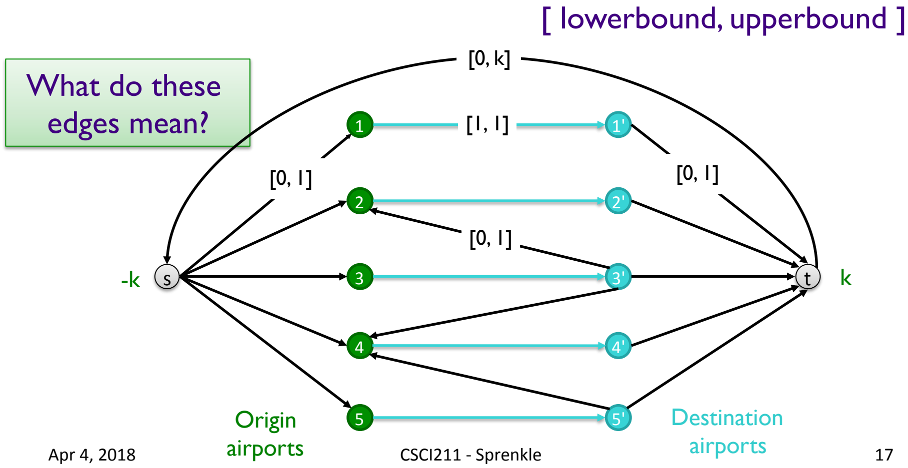
- A flight j is *reachable* from flight i if it is possible to use the same plane for flight j as flight i

Goal: Determine if it's possible to serve all *m* flights using at most *k* planes

Ideas about our solution/approach?

Airline Scheduling Algorithm

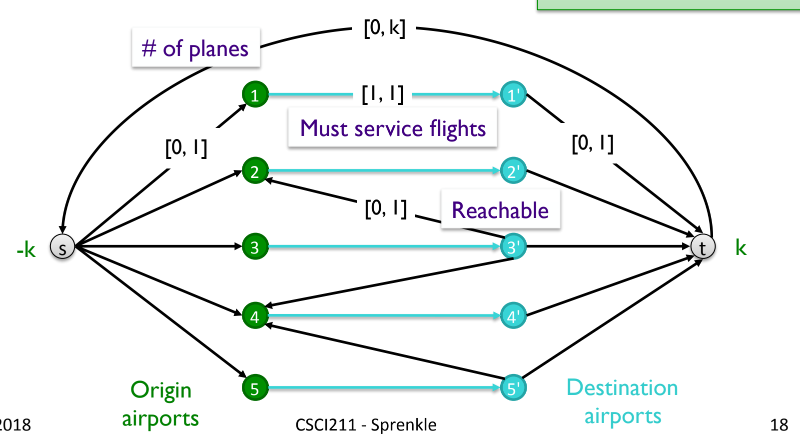
- Flow: airplanes; Nodes: airports
- Find a feasible circulation



Airline Scheduling Algorithm

- Flow: airplanes; Nodes: airports
- Find a feasible circulation

How do we know if we have a solution? How do we get the solution?



Scheduling Solution

Goal: Determine if it's possible to serve all *m* flights using at most *k* planes

- Model
  - Flow: airplanes
  - Nodes: airports
- Use FF algorithm to generate flow
  - If feasible flow → feasible circulation
- Construct schedules by following edges from *s* to origin airports
  - Represents the schedule of one plane

Analyzing Solution

Goal: Determine if it's possible to serve all *m* flights using at most *k* planes

- Cost of building the graph?
- Cost of generating flow?

Analyzing Solution

Goal: Determine if it's possible to serve all *m* flights using at most *k* planes

- Cost of building the graph: O(m^2)
  - Each flight (m)
    - Two nodes, one directed edge, with capacity [1,1]
  - Reuse plane edges (m^2) with flow [0,1]
  - Edge from source to each flight (m)
  - Edge from each flight to sink (m)

Graph: O(m^2)

- Cost of generating flow: O(k m^2)
  - FF is O(Fm), where F is flow, m is number of edges
    - For our problem, F is *k* and m is m^2

Total: O(k m^2)

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

Objectives

- Oh, the places you've been!
- Oh, the places you'll go!

> Now, everything comes down to expert knowledge of **algorithms** and **data structures**. If you don't speak fluent **O-notation**, you may have trouble getting your next job at the technology companies in the forefront.
> — Larry Freeman

Algorithm Design Patterns

- What are some approaches to solving problems?
- How do they compare in terms of difficulty?

Algorithm Design Patterns

- Greedy
- Divide-and-conquer
- Dynamic programming
- Duality/network flow

Course Objectives: Given a problem…

You'll recognize when to try an approach
- AND, when to bail out and try something different

Know the steps to solve the problem using the approach
- e.g., breaking it into subproblems, sorting possibilities in some order

Know how to ***analyze*** the run time of the solution
- e.g., solving recurrence relation

Apr 4, 2018 CSCI211 - Sprenkle 25

What Were Our Goals In Finding a Solution?

- **Correctness**
- **Polynomial Time → Efficient**

Apr 4, 2018 CSCI211 - Sprenkle 26

POLYNOMIAL-TIME REDUCTIONS

Classify Problems According to Computational Requirements

Fundamental Question:
Which problems will we be able to solve in practice?

Classify Problems According to Computational Requirements

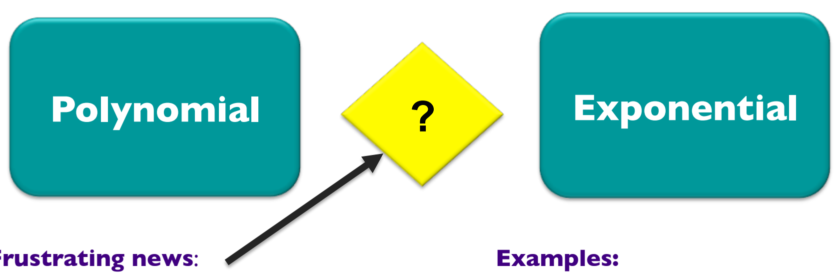
Which problems will we be able to solve in practice?

- Working definition. [Cobham 1964, Edmonds 1965, Rabin 1966] Those with polynomial-time algorithms.

Yes	Probably no
Shortest path	Longest path
Matching	3D-matching
Min cut	Max cut
2-SAT	3-SAT
Planar 4-color	Planar 3-color
Bipartite vertex cover	Vertex cover
Primality testing	Factoring

Classify Problems

Classify problems according to those that can be solved in polynomial-time and those that cannot.



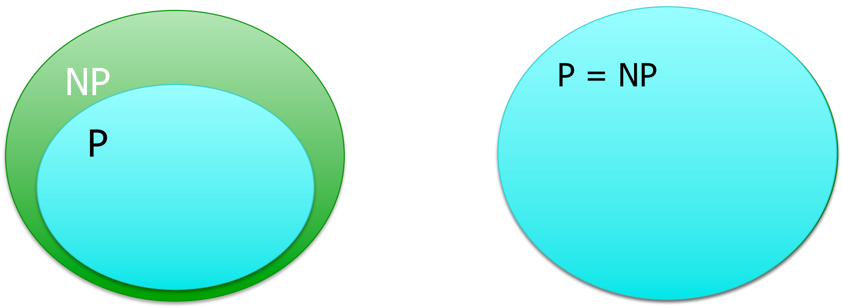
Frustrating news:
Many problems have defied classification.

Chapter 8. Show that problems are "computationally equivalent" and appear to be manifestations of one *really hard* problem.

Examples:
- Given a Turing machine, does it halt in at most *k* steps?
- Given a board position in an n-by-n generalization of chess, can black guarantee a win?

The Big Question

NP: "nondeterministic polynomial time"



P ⊆ NP

P = NP

Are there polynomial-time solutions to NP problems?

Looking Ahead

- PS 9 due Friday
- Final Exam handed out on Friday