## Objectives

- Network Flow
  - Application: Airline Scheduling
  - Choosing good augmenting paths
- Computational intractability



## Review

- What is the power of the max-flow/min-cut algorithm?
- What is our process in solving problems using network flow?



## 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

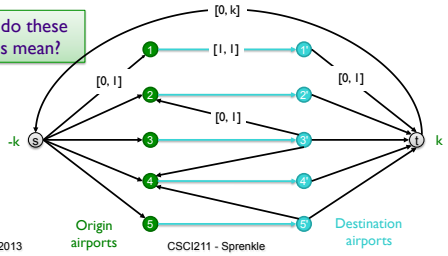
Ideas about our solution/approach?



## Airline Scheduling Algorithm

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

What do these edges mean?

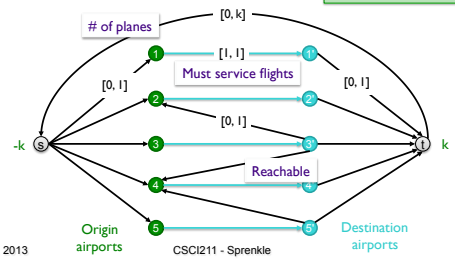





## Airline Scheduling Algorithm

- 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

- 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



## 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

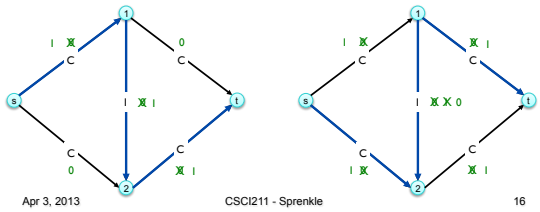


## CHOOSING GOOD AUGMENTING PATHS



## 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.





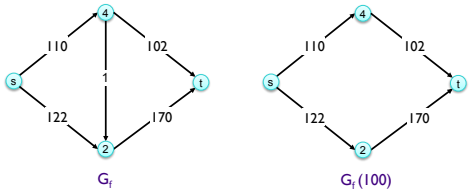

## Choosing Good Augmenting Paths

- 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
  - Fewest number of edges
  - *Sufficiently large bottleneck capacity*



## 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 $\Delta$
  - Let $G_f(\Delta)$ be the subgraph of the residual graph consisting of only edges with capacity at least $\Delta$




$G_f$ $G_f(100)$

## 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:        O(log C)
      while there exists augmenting path P in Gf(Δ) :
         f = augment(f, c, P)
         update Gf(Δ)
      Δ = Δ / 2

   return f
```



## 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 $\Delta \leq C$. $\Delta$ 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 $\Delta$-scaling phase. Then value of the maximum flow is at most v(f) + m $\Delta$.

Proof and further analysis in the book



## 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



## Algorithm Design Patterns

- Greedy
- Divide-and-conquer
- Dynamic programming
- Duality/network flow
- Reductions – Chapter 8
- Local search – Chapter 12
- Randomization – Chapter 13



## What Was Our Goal In Finding a Solution?

**Polynomial Time → Efficient**



## 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?



## Polynomial-Time Reduction

Suppose we could solve Y in polynomial time. What else could we solve in polynomial time?



## Polynomial-Time Reduction

*Suppose we could solve Y in polynomial-time. What else could we solve in polynomial time?*

- Reduction. Problem *X polynomial reduces to* problem *Y* if arbitrary instances of problem *X* can be solved using:
  - Polynomial number of standard computational steps, ***plus***
  - Polynomial number of calls to **oracle** that solves problem Y
    - Assume have a black box that can solve Y


- Notation: $X \leq_P Y$
  - "X is polynomial-time reducible to Y"
- Conclusion: If Y can be solved in polynomial time and $X \leq_P Y$, then X can be solved in polynomial time.



## Looking Ahead

- Problem Set 9 due Friday
- Course Evaluations
  - Fill out course evaluations on Sakai
  - If 60% of students fill out, 1% EC on problem sets
    - Additional 1% for every additional 12.5% who complete
    - Total problem set points: 192
  - Due Monday at midnight
- Final
  - Given out on Friday
  - Focus: Dynamic programming, network flow, computational intractability
  - Usual rules
  - Due at end of exam period: next Friday at 5 p.m.