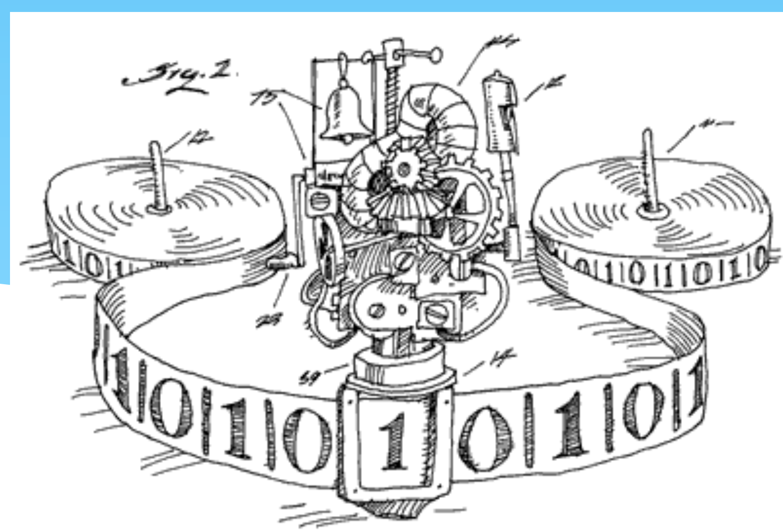
EECS 376: Foundations of Computer Science

Seth Pettie

Lecture 5



Agenda

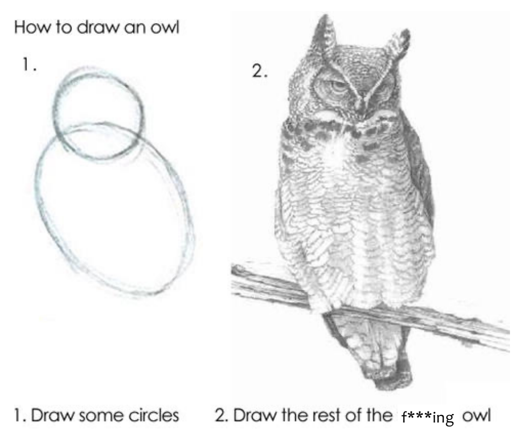
* Dynamic programming, continued: how to design a dynamic programming algorithm
* All-Pairs Shortest Path
  * The Floyd-Warshall algorithm

Principles of dynamic programming

* Dynamic Programming
  * Step 1: Find a recurrence relation
  * Step 2: Fill out a table
    * One cell for each possible set of inputs to the recurrence
    * Starting with the base cases, use the recurrence to fill the table

$$F(n) = \begin{cases} 1 & \text{if } n = 0,1 \\ F(n-1) + F(n-2) & \text{if } n \geq 2 \end{cases}$$

DP-F(n):
allocate $F[1..n]$
$F[0] \leftarrow 1, F[1] \leftarrow 1$
for $i = 2..n$
  $F[i] \leftarrow F[i-1] + F[i-2]$
return $F[n]$



Principles of dynamic programming

- Step 1, how to recurse:
  - *principle of optimality* – part of the optimal solution must be the optimal solution to a smaller part of the problem
    - Example: If 146 is an LCS of $X = 1546, Y = 7146$, then 14 must be the optimal solution to a smaller problem (the LCS of $X[1..3], Y[1..3]$)
  - Want that optimal solution to the smaller problems, plus minimal info from the original problem, to be enough to find the optimal solution

$LCS(i, j)$ = length of longest common subsequence of $X[1..i]$ and $Y[1..j]$

$$LCS(i,j) = \begin{cases} 0 & i = 0 \text{ or } j = 0 \\ 1 + LCS(i-1, j-1) & X[i] = Y[j] \\ \max\begin{Bmatrix} LCS(i-1, j), \\ LCS(i, j-1) \end{Bmatrix} & X[i] \neq Y[j] \end{cases}$$

Principles of dynamic programming

- Step 1, how to recurse:
  - *principle of optimality* – part of the optimal solution must be the optimal solution to a smaller part of the problem
    - Example: If 146 is an LCS of $X = 1546, Y = 7146$, then 14 must be the optimal solution to a smaller problem (the LCS of $X[1..3], Y[1..3]$)
  - Want that optimal solution to the smaller problems, plus minimal info from the original problem, to be enough to find the optimal solution

$LIS(i)$ = length of longest increasing subsequence of $X[1..i]$

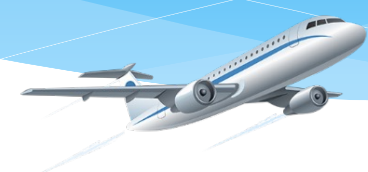
- Example: Given $LIS(i-1) = 5$, and $A[i] = 10$, what is $LIS(i)$?
  - Not clear how to do this. Have to keep track of many candidate increasing subsequences.
  - This is why we moved to $LIS_{at}(i)$

Principles of dynamic programming

* Step 1, how to recurse:
  * *principle of optimality* – part of the optimal solution must be the optimal solution to a smaller part of the problem
    * Example: If 146 is an LCS of $X = 1546, Y = 7146$, then 14 must be the optimal solution to a smaller problem (the LCS of $X[1..3], Y[1..3]$)
  * Want that optimal solution to the smaller problems, plus minimal info from the original problem, to be enough to find the optimal solution
  * Runtime: $O(\text{\# of cells} \cdot \text{time per cell})$
    * Goal is to minimize number of distinct inputs to the recurrence

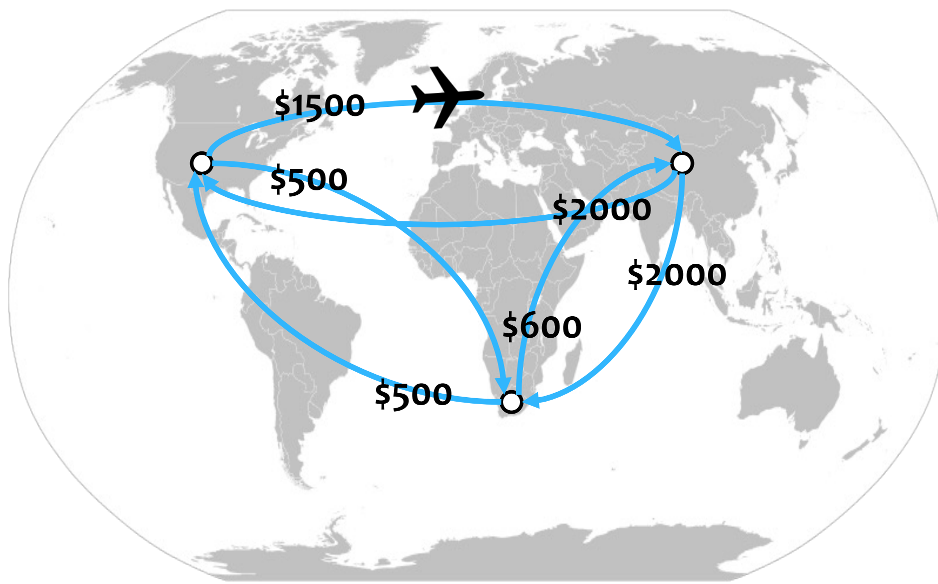
All-Pairs Shortest Paths

$c(i,j)$ might be different from $c(j,i)$

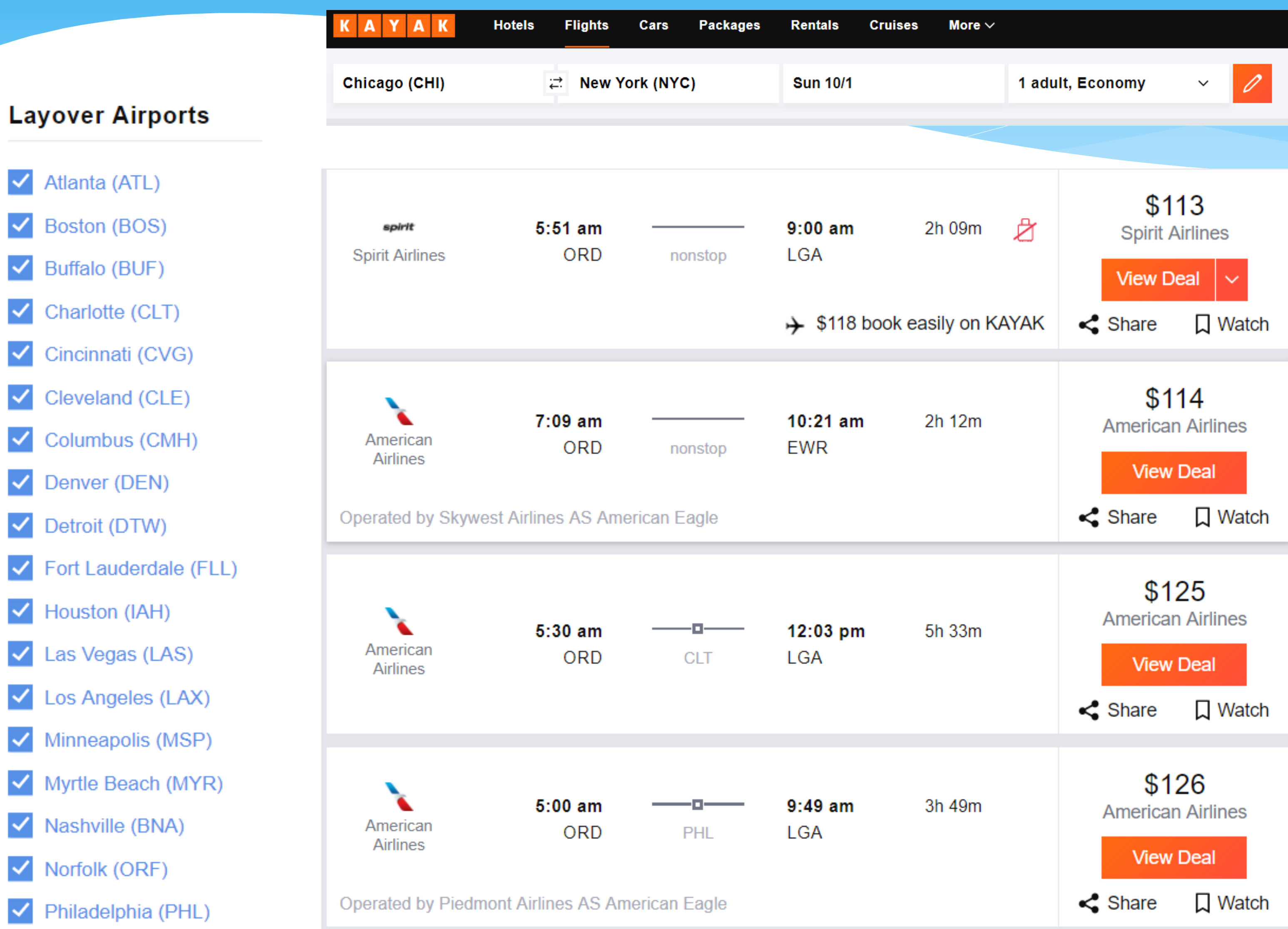


* Given n cities and (*possibly asymmetric*) costs $c(i,j)$ to directly fly from city i to j

* **Goal:** for *every* pair of cities i,j, find the *minimum* cost, $d(i,j)$, to fly from city i to city j, *when layovers are allowed*

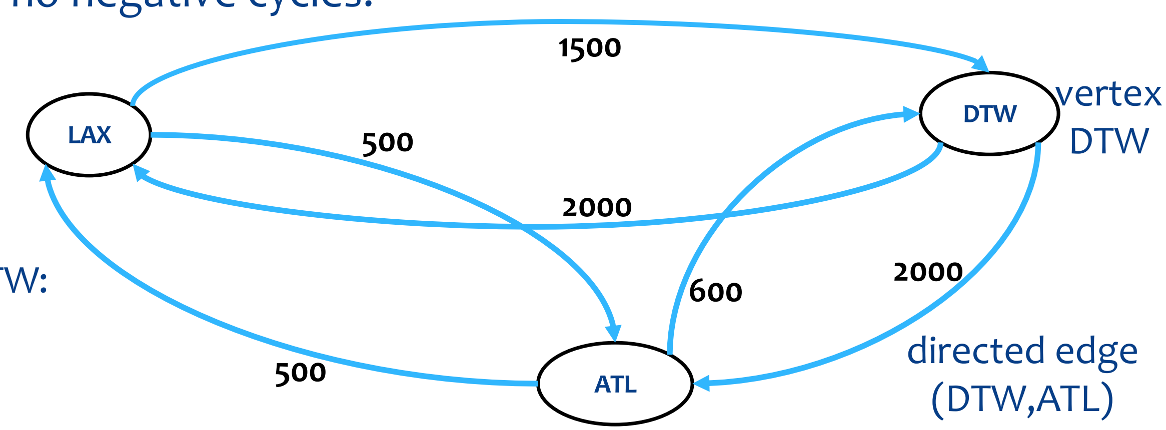


Example


Layover Airports
Atlanta (ATL)
Boston (BOS)
Buffalo (BUF)
Charlotte (CLT)
Cincinnati (CVG)
Cleveland (CLE)
Columbus (CMH)
Denver (DEN)
Detroit (DTW)
Fort Lauderdale (FLL)
Houston (IAH)
Las Vegas (LAS)
Los Angeles (LAX)
Minneapolis (MSP)
Myrtle Beach (MYR)
Nashville (BNA)
Norfolk (ORF)
Philadelphia (PHL)
KAYAK
Hotels
Flights
Cars
Packages
Rentals
Cruises
More
Chicago (CHI)
New York (NYC)
Sun 10/1
1 adult, Economy
spirit
Spirit Airlines
5:51 am
ORD
nonstop
9:00 am
LGA
2h 09m
$113
Spirit Airlines
View Deal
$118 book easily on KAYAK
Share
Watch
American Airlines
7:09 am
ORD
nonstop
10:21 am
EWR
2h 12m
$114
American Airlines
View Deal
Operated by Skywest Airlines AS American Eagle
Share
Watch
American Airlines
5:30 am
ORD
CLT
12:03 pm
LGA
5h 33m
$125
American Airlines
View Deal
Share
Watch
American Airlines
5:00 am
ORD
PHL
9:49 am
LGA
3h 49m
$126
American Airlines
View Deal
Operated by Piedmont Airlines AS American Eagle
Share
Watch

Review: Graph theory

- A ***directed graph*** G consists of some number n of (uniquely labeled) ***vertices*** and ***directed edges*** between them
- A ***weighted*** graph has its edges labeled with numbers
  - Assume weights can be positive or negative.
- A ***path*** from vertex i to vertex j is a sequence of vertices $i = v_0, v_1, \ldots, v_\ell = j$ such that each (v_{i-1}, v_i) is an edge
- The ***cost*** of a path is the sum of the weights along its edges.
  - Assume no negative cycles!



Some paths from LAX to DTW:
- LAX,DTW
- LAX,ATL,DTW
- LAX,DTW,ATL,LAX,DTW

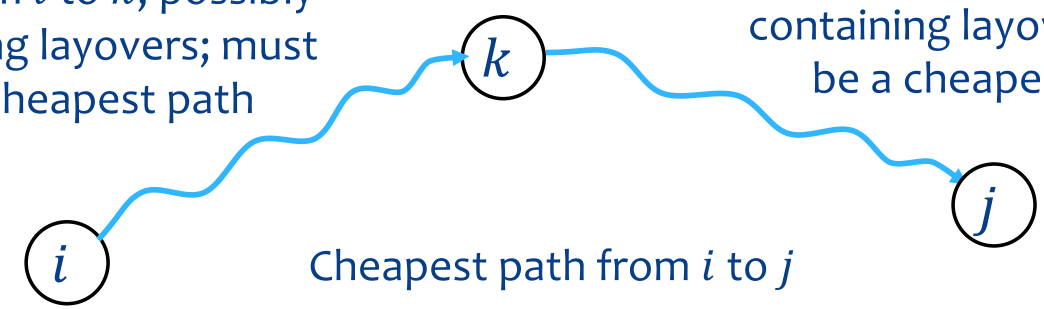
A recurrence

principle of optimality – part of the optimal solution must be the optimal solution to a smaller part of the problem

* **Idea:** Any subpath of a cheapest path is the cheapest path between its endpoints

path from i to k, possibly containing layovers; must be a cheapest path

path from k to j, possibly containing layovers; must be a cheapest path



Cheapest path from i to j

A recurrence

principle of optimality – part of the optimal solution must be the optimal solution to a smaller part of the problem

- **Idea:** Any subpath of a cheapest path is the cheapest path between its endpoints
- Let $d(i, j)$ be the minimum cost to fly from i to j, i.e., the cost of a *cheapest* path from i to j
- Recurrence: every cheapest path from i to j is either:
  - An edge from i to j (a direct flight)
  - An edge from i to some other k, and then a cheapest path from k to j
- $d(i,j) = \min\{c(i,j), \quad \min\{c(i,k) + d(k,j) \mid k \notin \{i,j\}\}\}$
- What goes wrong with this recurrence?
  - Computing $d(i,j)$ requires $d(k,j)$ which requires $d(i,j)$…

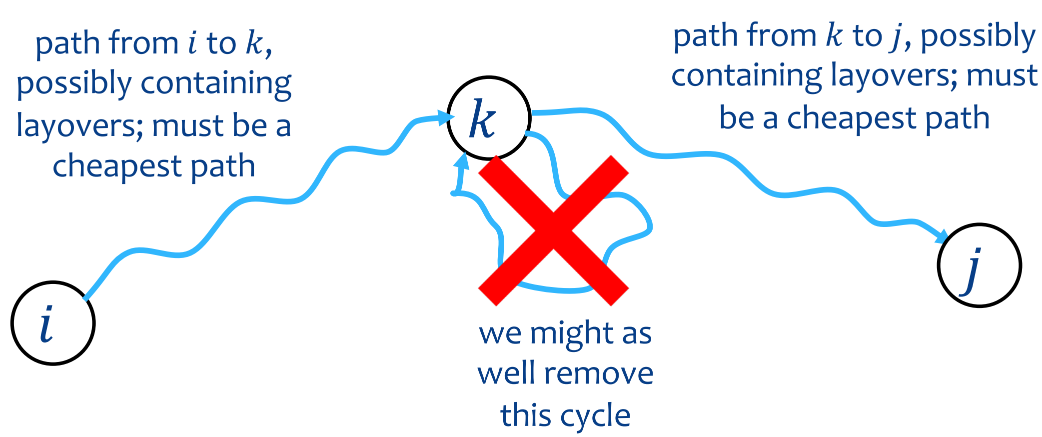
A recurrence

principle of optimality – part of the optimal solution must be the optimal solution to a ~~smaller~~ part of the problem

* **Idea:** Any subpath of a cheapest path is the cheapest path between its endpoints
* Let $d(i, j)$ be the minimum cost to fly from i to j, i.e., the cost of a *cheapest* path from i to j
* Recurrence: every cheapest path from i to j is either:
  * An edge from i to j (a direct flight)
  * An edge from i to some other k, and then a cheapest path from k to j
* $d(i,j) = \min\{c(i,j), \quad \min\{c(i,k) + d(k,j) \mid k \notin \{i,j\}\}\}$
* What goes wrong with this recurrence?
  * Computing $d(i,j)$ requires $d(k,j)$ which requires $d(i,j)$…

Finding a better recurrence

- **Idea 2:** No vertex (city) need ever be visited twice in a cheapest path
  - If there are no *negative cycles*, then there needn't be any duplicate vertices in the cheapest path



Finding a better recurrence

Goal is to minimize number of distinct inputs to the recurrence, runtime:
$O(\text{\# of cells} \cdot \text{time per cell})$

* **Idea 2:** No vertex (city) need ever be visited twice in a cheapest path
* Recurrence: every cheapest path from i to j is either:
  * An edge from i to j (a direct flight)
  * An edge from i to some other k, and then a cheapest path from k to j *that does not visit i*
* $R(G, i, j) = \min\left\{c(i,k) + R(G\backslash\{i\}, k, j) \mid k \in G\backslash\{i,j\}\right\} \cup \{c(i,j)\}$
* **Q:** How many distinct inputs are there? How long will dynamic programming take to run?
  * $O(2^n \cdot n \cdot n)$ distinct inputs
* Why were there too many inputs?

Finding a better recurrence

optimal solution to the smaller problems + minimal info from the original problem = enough to find the optimal solution

* $R(G, i, j) = \min\{c(i,k) + R(G \setminus \{i\}, k, j) \mid k \in G \setminus \{i, j\}\} \cup \{c(i,j)\}$
* "Given remaining vertices not yet chosen, which vertex k should follow i in the cheapest path from i to j?"
  * To answer this, need to know exactly which vertices have already been chosen, for which there are $O(2^n)$ possibilities
* Given remaining vertices not yet chosen, ***is vertex k in the cheapest path from i to j?***
  * $\min \begin{cases} R(G \setminus \{k\}, i, j) \\ R(G \setminus \{k\}, i, k) + R(G \setminus \{k\}, k, j) \end{cases}$
* **Q:** Do we still need to recurse over all possible *sets* of remaining vertices?

Floyd-Warshall

* **Idea 3:** Check whether vertex k is in cheapest path, and then recurse on $k-1$
* Recurrence: every cheapest path from i to j is either:
  * A cheapest path from i to j that visits only intermediate vertices $\{1, \ldots, k-1\}$, or
  * A cheapest path from i to k that visits only intermediate vertices $\{1, \ldots, k-1\}$, followed by a cheapest path from k to j that visits only intermediate vertices $\{1, \ldots, k-1\}$

Implementation

$$\mathrm{FW}(i,j,k) = \begin{cases} c(i,j) & k = 0 \\ \min\{\mathrm{FW}(i,k,k-1) + \mathrm{FW}(k,j,k-1), \mathrm{FW}(i,j,k-1)\} & k > 0 \end{cases}$$

Floyd-Warshall($C[1..n][1..n]$)**:** // $C[i][j] = c(i,j)$
allocate $D[1..n][1..n][0..n]$ // $D[i][j][k] = \mathrm{FW}(i,j,k)$
for all i, j: $D[i][j][0] \leftarrow C[i][j]$
for $k = 1..n$ and all i, j:

$$D[i][j][k] \leftarrow \min \begin{Bmatrix} D[i][k][k-1] + D[k][j][k-1], \\ D[i][j][k-1] \end{Bmatrix}$$

Runtime: $O(n^3)$

This is Floyd and Warshall's algorithm for finding ***all-pairs shortest paths*** in a *directed* graph with *weighted edges*. (Missing edges have weight ∞.)