Monday, Feb. 6, 2023
Lecture # 9
MSSC 6000

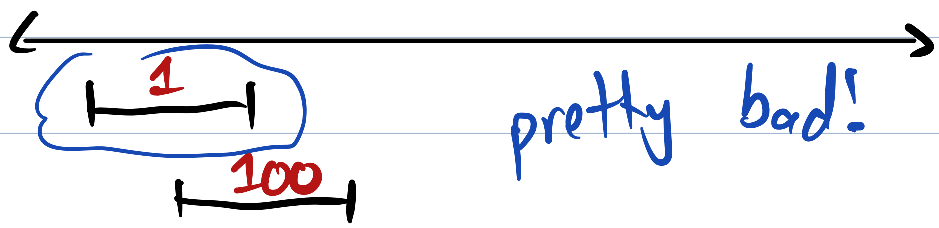
Announcements

* HW 1 due tonight, 11:59pm → turn in on D2L
* Office Hours 1pm-2pm in Cu 307.

Problem #3: Weighted Interval Scheduling

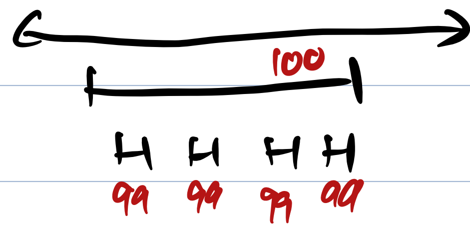
This is like regular interval scheduling, except each request r_i comes with a value v_i and your goal is to maximize the total value of requests satisfied.

How does our previous greedy algo do?



Possible Greedy Algos:

* best = highest value



* best = shortest meeting

* best = highest value density
  ↳ $\frac{\text{value}}{\text{duration}}$

> There is an algorithm to find optimal solutions using a technique called "dynamic programming." Run time with n requests is $\approx n^2$.

Problem #4 - Knapsack Problem

You have n items. They each have a value v_i and a weight w_i. You have a knapsack that can carry a total weight of C. (capacity) What combination of items has a total weight $\leq C$ and the highest value?

Ex:

items	weight	value
1	8	13
2	3	7
3	5	10
4	5	10
5	2	1
6	2	1
7	2	1

Capacity = 10

Some possibilities:

* Items 1 and 5
  weight: 8+2=10
  value: 13+1=14

* Items 2,4,7
  weight = 3+5+2=10
  value = 7+10+1 = 18

* Items 3,4
  weight: 5+5=10
  value = 10+10 = 20
  optimal!

Greedy possibilities:

* value density = $\frac{\text{value}}{\text{weight}}$
* minimal weight
* maximum value

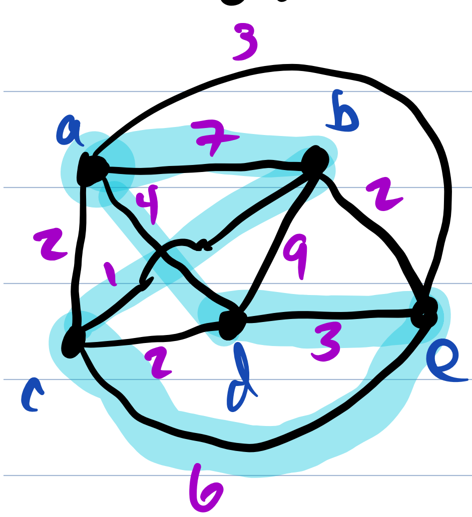
None of these are optimal but they do okay.

Dynamic programming can solve it quickly.

Problem #5 - Traveling Salesman Problem (TSP)

There are n cities that a salesman needs to visit, then return home. What is the shortest route that visits each city exactly once and returns back to the start?

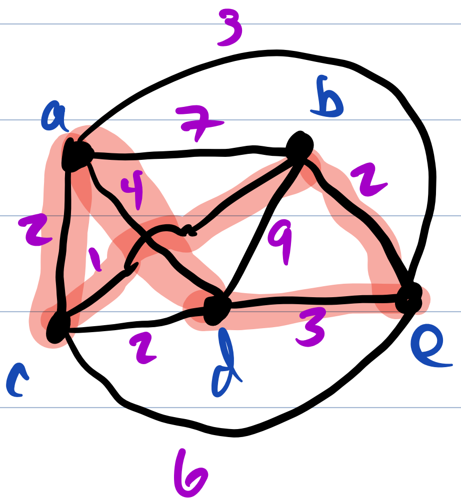
More formally: Consider a weighted graph G. Which ordering of the vertices gives you the smallest sum of the edge weights when you traverse the vertices in that order?



One solution:

a → d → e → c → b → a

4 + 3 + 6 + 1 + 7 = 21



$a \to c \to b \to e \to d \to a$

$2 + 1 + 2 + 3 + 2 = 10$

⑤

Greedy algorithm:

* pick any start vertex v_1
* pick v_2 to be the closest vertex to v_1
* pick v_3 to be the closest unvisited vertex to v_2

⋮

* at the end, return home to v_1

Notes:
- might fail if it's not possible to go from any city to any other city
- does okay, but usually picks some dumb edges
- brute force (try every possibility) is very slow.

$n! = n \cdot (n-1) \cdot (n-2) \cdot (n-3) \cdot \dots \cdot 3 \cdot 2 \cdot 1$

↳ $(n-1)!$

- dynamic programming version takes $\approx n^2 \cdot 2^n$ calculations

We'll learn lots of techniques ("metaheuristics") to get <u>very good</u> solutions quickly.