Scientific Computing

Mon, Jan. 30

Announcements

* Homework 1 due tonight, 11:59pm.
* Office Hours today, 1:45 - 2:45 pm

* Homework 2 assigned, due Fri. Feb 13, 11:59pm.
  pdf + zip file on D2L
  start early!!
  covers Greedy Algorithms

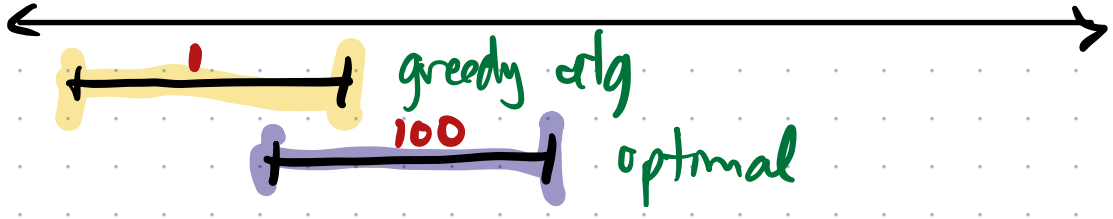
Office Hours:
Mon, 9:30 - 10:30
Fri, 2:00 - 3:00

Cudahy 307

Problem #3: Weighted Interval Scheduling

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

Our previous greedy algorithm is now pretty bad.



Possible Greedy Algos:

* best = "highest value"
* best = "shortest"
* best = "highest $\frac{\text{value}}{\text{duration}}$" (value density)

Are any of these optimal? No.

There is no known greedy algorithm that is optimal. There are 2^n subsets of a set of size n.

How long would brute force take? If there are n requests, you'd need to check all 2^n subsets of them. So, the run time would be exponential, something like $O(2^n)$ or $O(n \cdot 2^n)$.

This is "big-O" notation, and it tells you roughly how many steps an algorithm has to use.

For this particular problem, there is a technique to do it in $O(n \cdot \log(n))$ time — very fast!

"dynamic programming"

Demo!

Problem #4 - Knapsack Problem

You have n items that each have a value v_i and a weight w_i. You have a knapsack that can carry a total weight of C. What is the highest value of items you can carry in your knapsack?

Problem #4 - Knapsack Problem

You have n items that each have a value v_i and a weight w_i. You have a knapsack that can carry a total weight of C. What is the highest value of items you can carry in your knapsack?

Ex:

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

capacity 7

capacity 2

Capacity = 10

Possible Solutions:

* Items 1 and 5
  Total weight: 10 ✓
  Total value: 14
* Items 3 and 4
  Total weight = 10 ✓
  Total value = 20

optimal

Greedy possibilities:

* Always take lightest item (value = 10 in the example above)
* Always take most valuable (value = 14)
* Take the most value-dense: $\frac{\text{value}}{\text{weight}}$ (value = ~~20~~ 18)

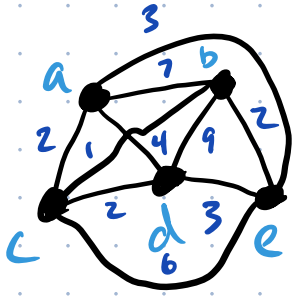
Are any of these guaranteed to be always optimal?

No

Problem #5 - Traveling Salesman Problem (TSP)

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

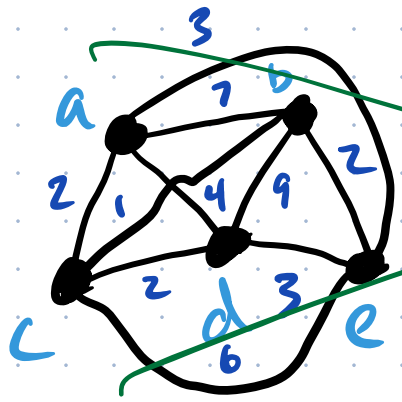
More formally: Consider a weighted graph G. Which ordering of the vertices gives you the smallest sum of edge weights?



4 3 6 1 7

a → d → e → c → b → a

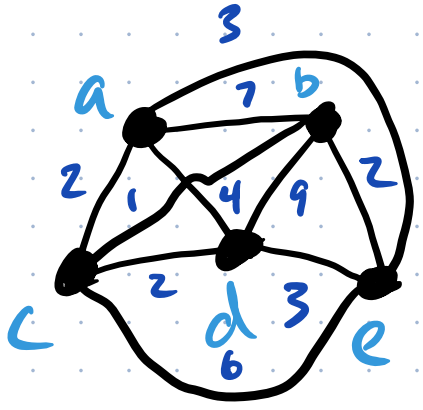
21



a → d → e → c → b → a

4 3

Same



a → c → b → e → d → a

2 1 2 3 4

(12)

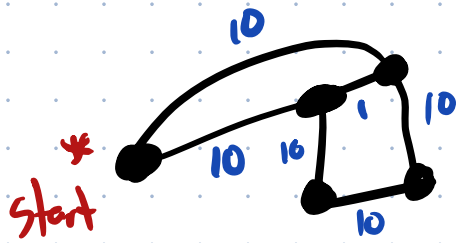
b → e → d → a → c → b

Greedy algo: * pick a start vertex v_1.

* pick v_2 to be the closest vertex to v_1
* pick v_3 to be the closest unused vertex to v_2
* repeat until last vertex is picked then go back to v_1.

Notes: - only works if it's possible to go from any city to any other city, otherwise you might get stuck

Ex:



- does okay, but usually ends up picking some dumb edges (demo in a minute)

- brute force: $O((n-1)!)$

  (recall $k! = k \cdot (k-1) \cdot (k-2) \cdot \ldots \cdot 3 \cdot 2 \cdot 1$)

  $5! = 5 \cdot 4 \cdot 3 \cdot 2 \cdot 1 = 120$

- dynamic programming: $O(n^2 \cdot 2^n)$

  still exponential!

This is just a really hard problem, but very intensely studied. We will use it as an example frequently when we learn "metaheuristic methods" later.

[Demos]