Scientific Computing

Wed, Jan 21

Announcements

* Homework 1 assigned. See the "Dropbox" section on D2L.
  - → Due 11:59pm on Fri, Jan 30
  - → Read instructions carefully.
  - → Start now!

* Friday: No in-person lecture. I'll send a video for you to watch instead.

  (unless classes are canceled from cold)

Office Hours:

Mon, 9:30-10:30

Fri, 2:00-3:00

Cudahy 307

Topic 2 - Jupyter vs .py files

* Jupyter is good for playing around and for presenting, but not for final products.
* It lets you run code out of order.

[demo]

* I strongly recommend <u>against</u> doing your work in Jupyter and then copy+pasting into a .py file. Always leads to problems.

* You can write your code in VS Code and run it in a terminal right inside VS Code. This is a "unix" terminal, same as a Mac, Linux, and the "Git for Windows" terminal.

↑ Makes this not necessary anymore.

[demo: Find the smallest positive integer that when you square it, the digits start with 2026]

Topic 3 - Greedy Algorithms

A lot of topics we'll cover in this class fall into the category of problem solving paradigms — a catalogue of ways to approach new problems.

"heuristics"

What is a problem? — super vague
You might have input data and/or constraints.
The question might be:
- Is it possible to satisfy all the constraints?

Ex: Every year, the NFL has to come up with a season schedule. There are many constraints! 17 games

- 32 teams in 2 conferences of 16 teams each
- Each conference split into 4 divisions of 4 teams each

Each team plays:

- 3 division rivals, twice each (H+A)
- each team in another division in the same conference (2H+2A)
- five teams in the other conference
- two more teams in their own conf.

plus:
- stadium constraints
- TV constraints
- holiday games
- one week off between Week 6 and 14
- many more...

Question: Can this be done?

(Obviously, yes)

The NFL says it takes 1000s of computers to generate ~1000 valid schedules, and then humans pick the best one.

Instead of "Is it possible?" the question could be "What's the optimal solution?" (lowest cost, highest profit, etc)

Ex: If Amazon has 100 packages to deliver to different houses in Milwaukee, and 5 delivery vans, which route
- uses the least gas
- travels the fewest miles
- takes the least time

etc.

Greedy Algorithms

Vague definition: A greedy algorithm is a way of solving a problem that builds up a solution bit by bit, always picking the next bit that is the best, even that leads to a suboptimal full solution.

"neurotic"

They are:
- normally lightning fast
- much better than random solutions
- sometimes pretty bad, sometimes pretty good, sometimes provably optimal, depending on the problem.

Ex: Giving change - How does a cashier give change? Suppose you owe $3.27 and pay with $20. They start giving you bills and coins from largest to smallest.

You get $16.73

"Cashier's Algorithm" - As you give change, just give denomination possible at each step.

$100	$50	$20	$10	$5	$1	$0.25	$0.10	$0.05	$0.01
X	X	X	1	1	1	2	2	0	3
			6.73	1.73	$0.73	$0.23	$0.03		$0

1+1+1+2+2+3 = 10 coins/bills

Is this the combination with the least # of bills/coins that adds to $16.73?

This is a greedy algorithm!

At every step, the cashier gives you the largest possible bill.

Is the cashier's algorithm optimal? yes

fewest # of things

(what are some other versions of "optimal"?)

Theorem: For the US currency denominations listed, the cashier's algorithm is optimal.

Proof: To simplify things, let's just assume the denominations 1, 5, 10, 25 cents. Suppose we are making x cents.

Lemma: The optimal solution will have ≤ 4 P.

Proof: If it had ≥ 5 Ps, we could replace with a nickel.

Lemma: The optimal solution will have ≤ 1 N.
(same proof)

Lemma: The optimal solution will have $\#N + \#D \leq 2$.

Proof: 2N : bad

1N + 2D : bad $\rightsquigarrow$ = 1Q

0N + 3D : bad $\rightsquigarrow$ = 1N + 1Q.

Main proof by induction:

Base case: 0¢ with 0 coins. Optimal ✓

Now assume we're making x ¢.

Case 1: $x < 5$: x pennies, only option

Case 2: $5 \le x < 10$: must have 1 N (or else >4 P)

1N + (sol for x-5)

optimal by ind.

Case 3: $10 \le x < 25$: must have 1D (or else >4 P or >1 N)

1D + (sol for x-10)

optimal by ind

Case 4: $x \geq 25$: must have 1Q (or else ≥ 4 P
or ≥ 1 N
or 1N+2D
or 3D)

Why? The best you can do without a quarter is 4P+1N+1D or 4P+2D.

So 1Q + (sol for $x-25$)

optimal by ind.

■

To include dollar denominations, more cases are needed.

Q: Is the cashier's algorithm optimal for any set of denominations?

Ex: US Postage Denominations

1, 2, 3, 5, 10, 20, 35, 36, 55, 65, 75, 95, 100, 120, 200, 500, 795, 1000, 2635

Ex: US Postage Denominations

1, 2, 3, 5, 10, 20, 35, 36, 55, 65, 75, 95, 100, 120, 200, 500, 795, 1000, 2635

To make 72¢: greedy solution = 65 + 5 + 2

optimal solution = 36 + 36

Greedy ≠ Optimal!

Throughout this course we're going to learn about a catalog of problems that model all kinds of real-world problems you might face.

Problem #1: Interval Scheduling (Algorithm Design, by Kleinberg + Tardos)

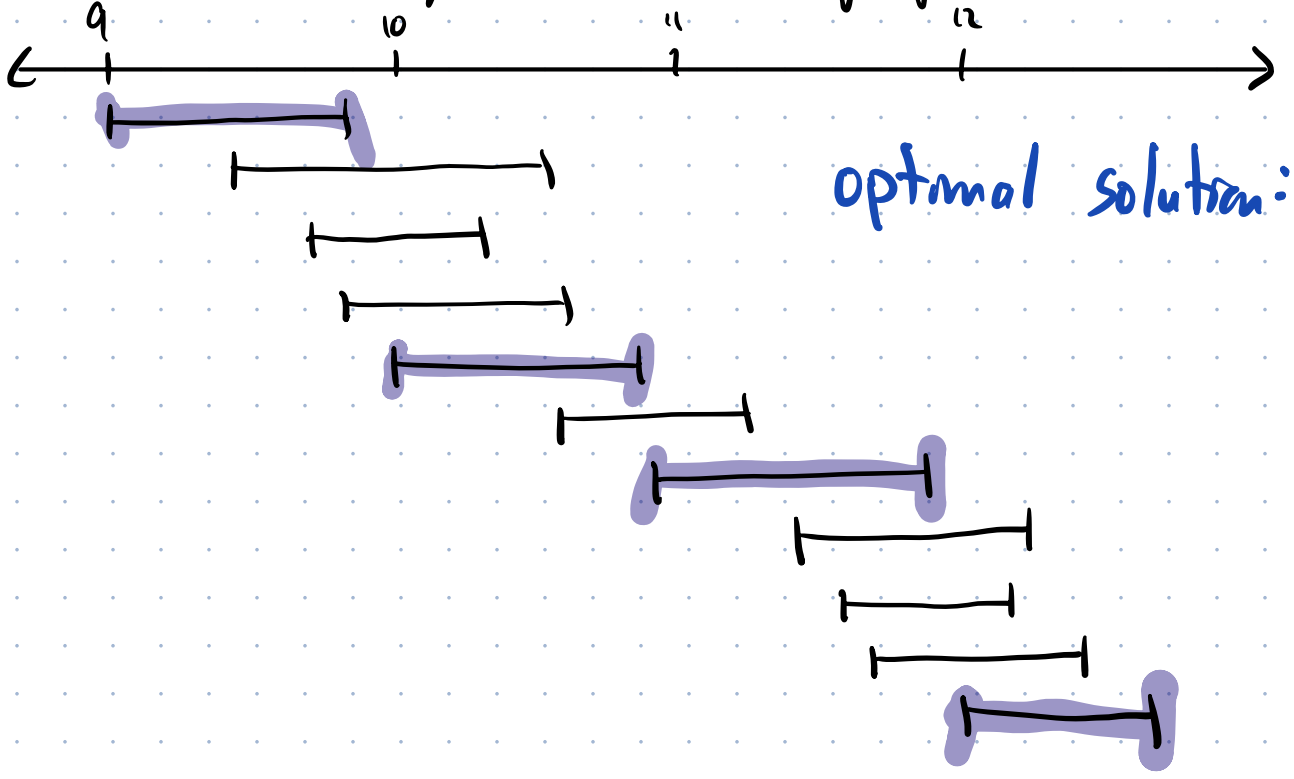
Suppose you are in charge of a conference room that a lot of people want to use to hold meetings. A bunch of people tell you the times they want to book the room for, and your goal is to accomodate as many groups as possible.

Ex:

Reservations:

9am – 9:50am
9:30am – 10:30am
9:45am – 10:15am
9:50am – 10:30am
10:00am – 10:50am
10:30am – 11:15am
11:00am – 11:50am
11:30am – 12:15pm
11:35am – 12:10pm
11:40am – 12:20pm
12:00pm – 12:30pm

What is the largest # of meetings you can book?



optimal solution: 4 meetings

Formal setup:

- n requests
- each request has a start time s_i and a finish time f_i (real numbers) with $s_i < f_i$.

Goal: find a maximal size subset of nonoverlapping requests

Two requests (s_i, f_i) and (s_j, f_j) overlap if:

Assume $s_i < s_j$ overlap if $f_i > s_j$

Formal setup:

- n requests
- each request has a start time s_i and a finish time f_i (real numbers) with $s_i < f_i$.

Goal: find a maximal size subset of nonoverlapping requests

Two requests (s_i, f_i) and (s_j, f_j) overlap if:

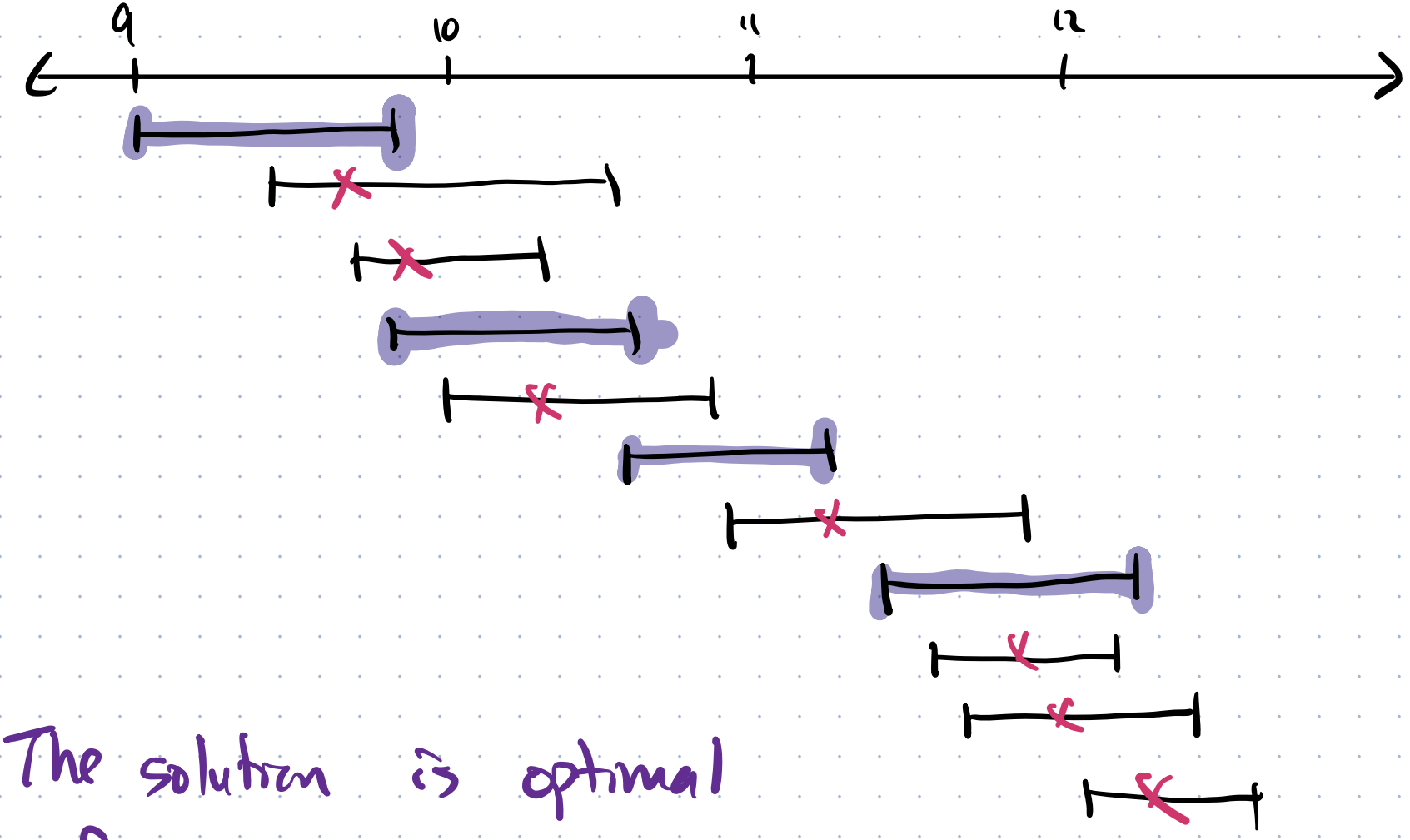
Alternate version:

$s_j < f_i$ and $s_i < f_j$

Let's think about possible greedy approaches.

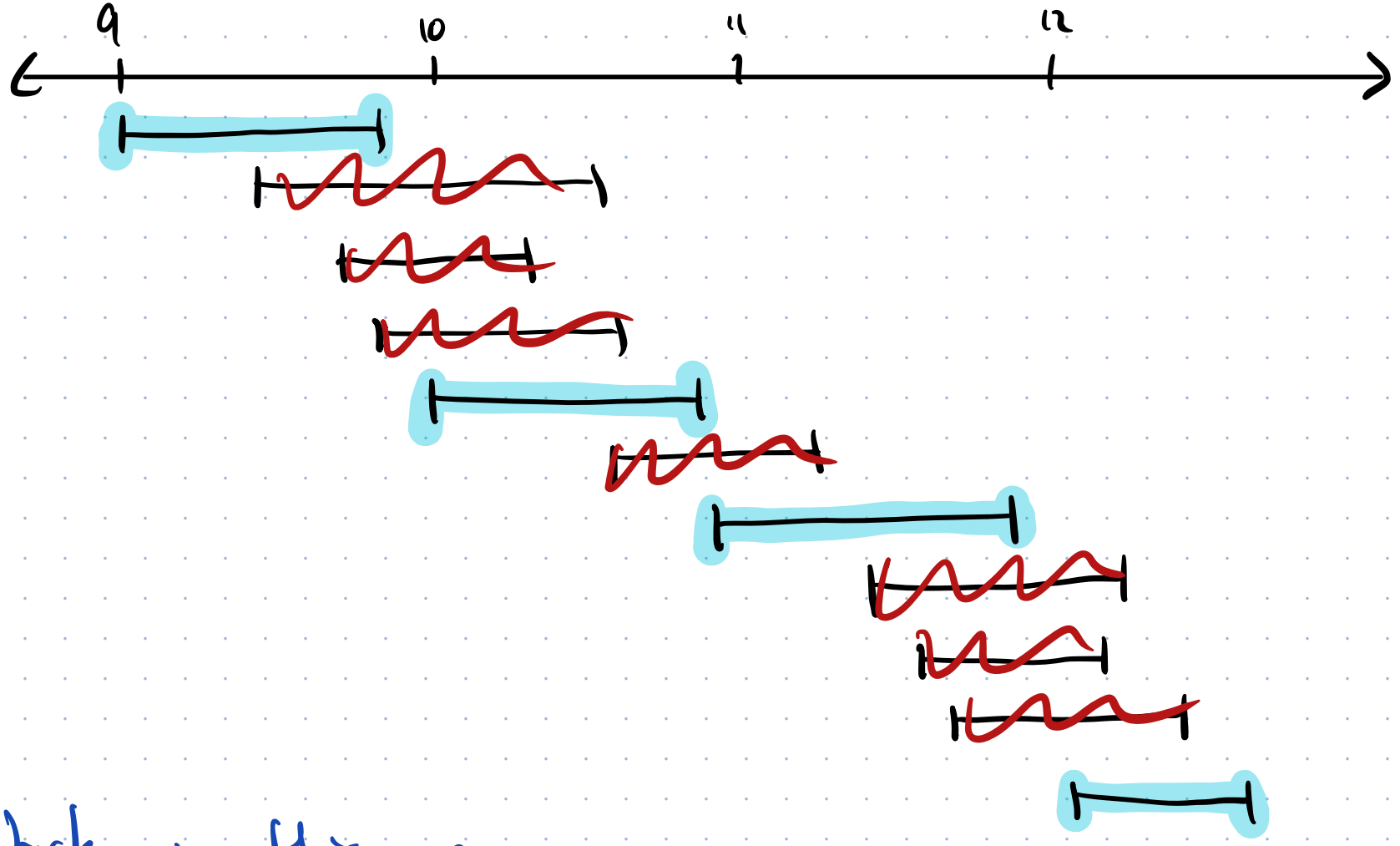
General idea:
* decide on a rule for which meeting is "best"
* pick it, then eliminate conflicts
* repeat

Idea #1: best = earliest start time



The solution is optimal for this example

Idea #1: best = earliest start time



Works in this case.

Can we break it?

Idea #1: best = earliest start time

Can we break it?