CS 9

Week 3 Problems

Andrew Benson
Ian Tullis

Personal life [edit]

He was an accomplished linguist speaking several foreign languages: German, Latin, French, Italian, Romanian.[8] He learned the violin and performed in Vienna.

> It is related of him that he was challenged by thirteen officers of his garrison, a thing not unlikely to happen considering how differently he thought from everyone else. He fought them all in succession—making it his only condition that he should be allowed to play on his violin for an interval between meeting each opponent. He disarmed or wounded all his antagonists. It can be easily imagined that a temperament such as his was not one congenial to his military superiors. He was retired in 1833.[9]

3 Problems

- Problem 3-1 will have a video walkthrough on Canvas (coming later tonight)
  - watching these counts as out-of-class extra work for grading purposes!
- Problem 3-2 **(note: hard)** will be discussed in-class
- Problem 3-3 **(note: really hard)** will have a written explanation at the end of the slides

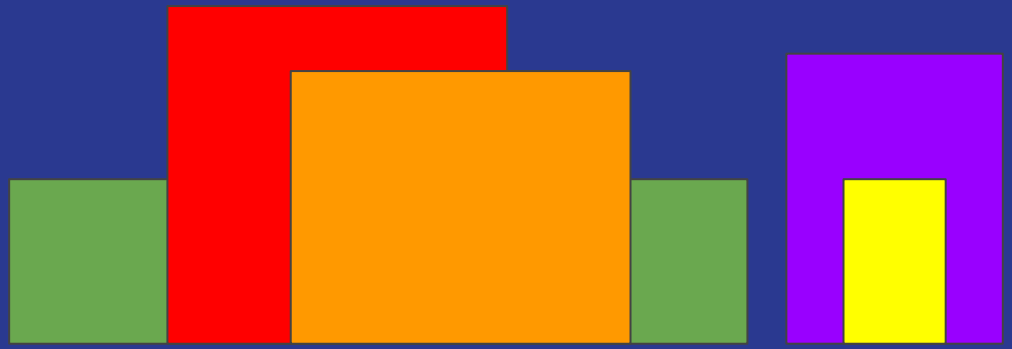
⭐ Problem 3-1: Dual Parties

- It is early 19th century Europe, so people are always dueling.
- You are in charge of hosting two separate parties, one for the King and one for the Queen (they are having a spat). There are N friends of the Crown (2 <= N <= 1000), and each person must be invited to exactly one of the two parties.
- Some of the people are mortal enemies and will duel if they are at the same party. Then the evening will be ruined and His or Her Majesty will be displeased.
- You are given a list of pairs of (mutual) enemies Ai, Bi.
- If it is possible to invite the people in such a way that no two enemies are at the same party, give the lists of attendees at the two parties. Otherwise print "IMPOSSIBLE!"

ex. N = 5, pairs 1 3, 4 5, 2 3: one answer is [3, 4], [1, 2, 5]
N = 5, pairs 1 2, 3 4, 1 5, 2 3, 4 5: IMPOSSIBLE!

⭐⭐⭐ Problem 3-2: Skyline

- Leetcode #218, if you want to solve it live!
- Given a list of rectangular buildings and where (horizontally) they start and stop, find the skyline (top border of buildings)



⭐⭐⭐ Problem 3-2: Skyline

- Leetcode #218, if you want to solve it live!
- Given a list of rectangular buildings and where (horizontally) they start and stop, find the skyline (top border of buildings)

skyline!

⭐⭐⭐ Problem 3-2: Skyline

- **Input:** [3, 9, 6], [5, 11, 5], [15, 17, 3], [0, 13, 3], [14, 19, 5]

- **Output:** [0, 3], [3, 6], [9, 5], [11, 3], [13, 0], [14, 5], [19, 0]

each building is [start x coordinate, stop x coordinate, height]

[x coordinate where something changes, new height], sorted by x coordinate

⭐⭐⭐ Problem 3-2: Skyline

Quick note: I had this as an interview question (a million years ago), and then (years later) actually needed the same algorithm for a work task. (not a contest problem!)

skyline!

⭐⭐⭐ Problem 3-3: Party Foul

- Consider the same setup as in the Dual Parties problem, but now it is OK to ruin *either* the King's party or the Queen's party (but not both). (One angry royal, you can deal with. Two, and your head is not long for your body.)
- ⭐ Find a brute-force way to determine whether it is possible for the unruined party to have at least 2 people, or at least 3 people.
- ⭐⭐⭐⭐ Do you have any ideas for doing either of those more efficiently, or can you argue that there is no such way?
- ⭐⭐⭐⭐⭐⭐⭐⭐⭐⭐⭐⭐⭐⭐⭐⭐⭐⭐⭐⭐⭐⭐⭐⭐⭐⭐⭐⭐⭐⭐⭐⭐⭐⭐⭐ What if your goal is to make the size of the unruined

Our friend the priority queue

- Got Ian his Google job, probably!
- **When you need to be able to find the smallest thing**
  - or the k smallest
  - or largest instead of smallest…

  while also inserting and deleting.
- Operations:
  - Insert in constant or O(log *n*) time
  - Find minimum in constant time
  - Delete minimum in O(log *n*) time
- Various implementations; you should know one (heap?)

Example

- Top Kit Kat varieties
- You want to maintain a database
- Things that can happen:
  - New bar just dropped! (and you rate it)
  - Old bar discontinued :(
  - People ask you for the top 10 bars on the market

Options

- Maintaining a sorted list by rating:
  - **New bar appears:** add it to the list in the right place (O(log *n*) to find it with binary search)
    - array: lots of extra work to shift stuff around
    - linked list: binary searching is a pain
  - **Someone asks for top 10:** Go through start of list, deleting any outdated bars

- Maintaining a PQ:
  - **New bar appears:** add it to the PQ (O(log *n*))
  - **Someone asks for top 10:** Keep popping from PQ, deleting any outdated bars (O(log *n*))

The Skyline Problem: one solution

- Scan left to right, keeping track of current height
- We're interested in "events":
  - A new building starts
  - An old building ends
- So turn the original building list into an event list, sort it

- Use a priority queue to keep track of the tallest building we're currently aware of
  - Keys: (a building's height, where it ends)
- Special "building" for the ground (height 0, ends at infinity)

A new building starts:

- If it's taller than our current height:
  - Add a new skyline segment up to now (and remember when next segment starts)
  - Update current height
- Also, regardless:
  - Add this building's height and ending location to our priority queue

An old building stops:

- Keep popping the top of the priority queue and removing anything that has ended, until we reach a building that still exists
- Set current height to height of that building
- If our current height changed:
  - Add a new skyline segment up to now

Annoying edge cases

- Multiple starts/stops at the same time
  - One option: merge these in the event list
  - Another: when adding a new skyline segment, if we already have one starting where we are, overwrite it
  - Also make sure code handles buildings starting and stopping in the same place

Time complexity

- Sort the n buildings by start coordinate: $O(n \log n)$
- Processing each of the n buildings using the priority queue takes $\log n$ time
- Overall: $O(n \log n)$

```python
import heapq
class Solution:
    def getSkyline(self, buildings: List[List[int]]) -> List[List[int]]:
        events = []
        for l, r, h in buildings:
            events.append((l, -h, "in", r))  # -h since heaps default to smallest first
            events.append((r, -h, "out", None))
        events.sort()
        pq = [(0, events[-1][0] + 1)]  # fake building as ground
        heapq.heapify(pq)
        curr_height = 0
        skyline = []
        next_start = -1 # before any building
        for x, h, kind, out_time in events:
            if kind == "in":
                heapq.heappush(pq, (h, out_time))
                if h < curr_height:
                    if skyline and skyline[-1][0] == next_start:
                        skyline.pop(-1)
                    skyline.append([next_start, -curr_height])
                    curr_height = h
                    next_start = x
            else: # kind == "out"
                while pq and pq[0][1] <= x:
                    heapq.heappop(pq)
                h = pq[0][0]
                if h != curr_height:
                    if skyline and skyline[-1][0] == next_start:
                        skyline.pop(-1)
                    skyline.append([next_start, -curr_height])
                    curr_height = h
                    next_start = x
        skyline.pop(0)
        skyline.append((x, 0)) # handle last segment
        return skyline
```

Ian's ugly (but accepted) solution

Party Foul: 2 people

- We can put 2 people in the no-dueling party only if there is some pair of people who are not enemies. We can find such a pair (if it exists) by e.g. making a list of which people each person hates, then seeing if anyone's list is not complete. This is $O(n^2)$.

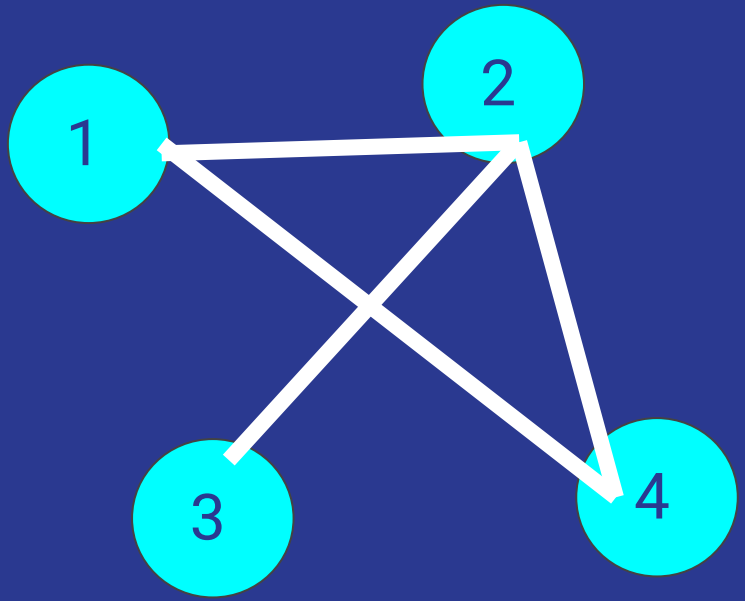
- I think we can't do better than this. Consider the scenario in which there is just one pair of people who do not hate each other. We would need to read the entire dataset to narrow down to that pair… and the dataset would have $[(n * n\text{-}1) / 2] - 1$ pairs, which is $O(n^2)$ just to read.

Party Foul: 3 people

- We can put 3 people in the no-dueling party only if there is some "triangle" of people who are not enemies. Naively, we can find such a triangle (if it exists) by e.g. making a list of which people each person hates, then checking each of the $O(n^3)$ possible triangles of people.

- It is hard to figure out how to do better than this! Probably not something to just come up with during an interview, but here is one wild solution…

Party Foul: 3 people



note: edges mean non-enemies here

take the cube of the matrix representing the graph's edges

if there's a triangle, you get nonzero elements on the diagonal

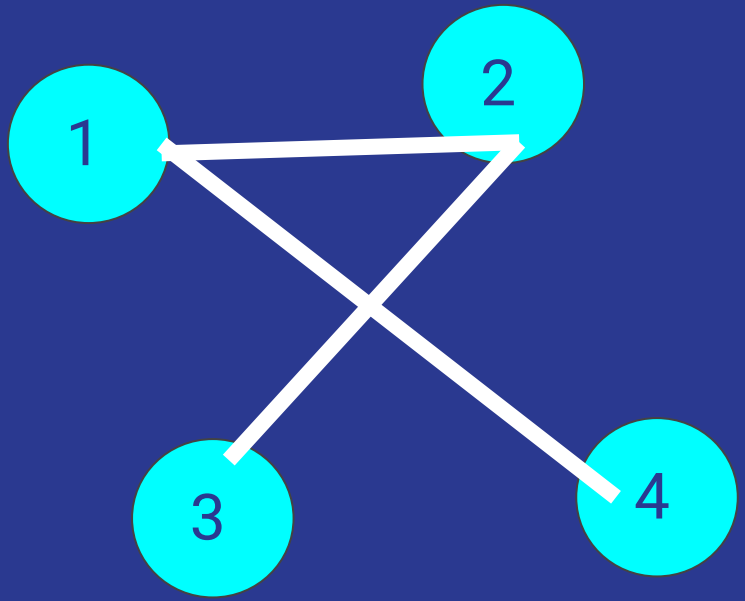
Input

$$\begin{pmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \end{pmatrix}^3$$

Result

$$\begin{pmatrix} 2 & 4 & 1 & 3 \\ 4 & 2 & 3 & 4 \\ 1 & 3 & 0 & 1 \\ 3 & 4 & 1 & 2 \end{pmatrix}$$

Party Foul: 3 people



note: edges mean non-enemies here

if there's no triangle, you get only zero elements on the diagonal

Input

$$\begin{pmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix}^3$$

Result

$$\begin{pmatrix} 0 & 3 & 0 & 2 \\ 3 & 0 & 2 & 0 \\ 0 & 2 & 0 & 1 \\ 2 & 0 & 1 & 0 \end{pmatrix}$$

Party Foul: 3 people

- So, cubing a matrix solves the problem!
- This takes $O(n^3)$ time using naive matrix multiplication, but there are weird ways to multiply matrices faster, e.g. Strassen's. You can go all the way down to $O(n^{2.3})$ or so!
- Therefore we have an $O(n^{2.3})$ solution.
  - note: with ridiculous constant factors

Party Foul: as many people as possible

- Extending the previous naive idea (check all possible subsets of people to see if they don't all hate each other, then take the biggest one that works), we get something terrible like $O(2^n n^2)$. (There are 2^n subsets, and checking each one for enemies takes $O(n^2)$ time.)

- This is a famously intractable problem, the "clique problem". You would not be asked to solve it during an interview, but it is good to be able to say "hey, that's the clique problem! It is known to be intractable to solve!"