COP 4516 Notes for 4/9/2024

Reminders

I am gone April 13 – 20 with the programming team.

On April 16th, Arun will teach Binary Index Trees. (I would recommend you show up.)

Arun will run the contest on Friday 4/19/2024. All regular deadlines with that. I'll be back to post data by Monday, etc.

I will put up "reflection assignments" for the team when I am in Egypt.

Individual reflection by yourself (15 pts) – I ask you to list how much work (percentage wise) each person did and reflect on your contributions and what you can do personally to help the team for the team final exam

Team reflection that you write together (10 pts) – I ask the team to talk about their strengths and weaknesses over the 6 team contests and what they can do to prepare best for the final team contest.

Today's Topic is Binary Search

Tradition binary search:

Input: Array of sorted values, target value

2 4 6 22 89 121 180 202, etc.

0 (low) n-1 (high)

Each time, we guess halfway between our possible range, cutting the search space in half.

Etch Problem

Imagine array storing:

F(0), f(1), f(2), f(3), …f(86400)

Where f(x) is RHS of that equation for plugging x seconds.

Based on the function property, this array is sorted.

So basically, if I know what target answer I am searching for, I can do the same exact thing.

In fact, it's probably a bit easier since x is a real number and I can simulate this process without worrying about off by one errors.

Hallmarks of when binary search is a useful technique

1. Problem is hard to figure out backwards (given f(x) what is x?)
2. Problem is easy to figure out forwards (given x find f(x)
3. Function I am searching within is increasing or decreasing.

My tips for real valued binary searches

1. Run for a fixed number of iterations. (Usually 100 is enough.)
2. mid = (low+high)/2 using real number division
3. Pay careful attention to your initial values of low and high. Make sure they are really true bounds and that plugging them into your function doesn't cause some math overflow error or something of that nature.
4. Pay careful attention to whether you set low = mid or high = mid.

Example Problem Airport Shuttle

Hard to know when to send a shuttle off if we are trying to minimize max wait time.

But…if we knew the max time each staffer was willing to wait (same number for all) then we could calculate # of staffers necessary to transport all the kids. **And this is a decreasing function. (non-increasing.)** As the time increases, the number of staffers decreases…

Searching for the minimum integer t such that we need k or fewer staffers to transport all the kids when each staffer waits no more than t minutes.

Low = 0, high = maxT – minT.

My tips for real valued int searches

1. Run until low == high (when we cut out) while (low < high)

2. mid = (low+high)/2 **OR** (low+high+1)/2 using integer division, **plug in low = 2, high = 3**

3. Pay careful attention to your initial values of low and high. Make sure they are really true bounds and that plugging them into your function doesn't cause some math overflow error or something of that nature.

4. There are quite a few options for setting low and high depending on how you characterize your function, all of these, in theory are possible:

Low = mid, low = mid+1, high = mid, high = mid-1

To determine between these, think about what your comparison really means…

Careful Approach

A. What order should the planes land? → not clear

B. Even if we know the order, it's not clear what exact time we should land the planes?

To "fix" A, just try all permutations since 8! isn't too big.

10 50, it probably makes sense to land this at 10…

10 60, but what about this one???

5 25

15 40

20 45

Easier question: Given an order of planes AND a forced minimum wait time between planes, can we land all the planes?

Can we do wait time of 5?

10	50,	**10**	**10**
10	60,	**15**	**20**
5	25	**20**	**oops!**
15	40	**25**	
20	45	**30**	
		W=5	**w=10**

New example wait = 10

10	30	**10**
5	25	**20**
40	50	**40 (next time = max(old+10, start)**

For a fixed order of planes, binary search the maximum gap achievable.

Bones Batteries

Higher the charge length, more stuff is reachable

Lower the charge length, less stuff is reachable

Run a floyd warshalls on the original graph so you have the shortest distance between all pairs of vertices.

Binary search the charge length.

Add in the edges of length 1 between all pairs of vertices that have a shortest distance of X or less where X is the charge length.

On this new graph run Floyd's again and make sure that none of the pairwise charging distances is greater than k.