CS 9

Week 4 Problems

Andrew Benson
Ian Tullis
Problem 4-1: Consecolumns

- In spreadsheets, columns are labeled A, B, ..., Z, then AA, AB, ..., ZZ, and so on.
- Write a function that, given a string (length up to 10000), determines whether that string consists of exactly two consecutive column labels, in order. (And be careful!)

<table>
<thead>
<tr>
<th>String</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>HI</td>
<td>True</td>
</tr>
<tr>
<td>XW</td>
<td>False</td>
</tr>
<tr>
<td>XERXES</td>
<td>True</td>
</tr>
<tr>
<td>ABABC</td>
<td>False</td>
</tr>
</tbody>
</table>
Problem 4-2: Trendy Dessert

- Some new dessert place just dropped in Palo Alto, and there's a huge line.
  - Is it really that good though?
    - OK it's probably pretty good

- There are K servers (2 <= K <= 1000); the i-th server takes $K_i$ minutes to help a customer, then opens up for the next one. Customers go to the lowest-numbered available server.

- The store just opened. You are position N in line (1 <= N <= $10^9$). Which server will help you?
Example

e.g., $K_1 = 20$, $K_2 = 10$, $K_3 = 15$: three servers
$N = 6$: you are customer 6 in line

0 min:
  Store opens. Customers 1, 2, and 3 go to servers 1, 2, and 3.
10 min:
  Server 2 finishes. Customer 4 goes to server 2.
15 min:
  Server 3 finishes. Customer 5 goes to server 3
20 min:
  Servers 1 and 2 finish. You go to server 1. Customer 7 goes to server 2.
Problem 4-3: Average Joe

- You want to cast the lead of a movie called "Average Joe".
- A total of $K \times N$ actors have applied.
- You have asked each of your $K$ ($2 \leq K \leq 1000$) casting directors to audition $N$ ($1 \leq N \leq 10000$) of the actors, and give them each a score ($1 \leq S_{ij} \leq 10^9$). Each director gives you a list of their results, in score order from highest to lowest. (There might be ties.)
- You think it would be most authentic to cast a person with the median score. Write a program to find such a person efficiently.

Example:
$K = 3$, $N = 3$. Lists are [987654321, 97, 64], [99, 99, 83], [85, 43, 1]. We should cast the first person in the third list.
• My build was working yesterday (at changelist 100000000)

• It's not working now (at changelist 100500000)

• In that interval, someone else's changelist broke it, and it stayed broken
  ○ so who do I yell at (nicely)?
Binary search to the rescue!

<table>
<thead>
<tr>
<th>low</th>
<th>high</th>
<th>mid</th>
<th>build OK?</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000000000</td>
<td>1005000000</td>
<td>1002500000</td>
<td>No</td>
</tr>
<tr>
<td>1000000000</td>
<td>1002500000</td>
<td>1001250000</td>
<td>Yes</td>
</tr>
<tr>
<td>100125000</td>
<td>1002500000</td>
<td>100187500</td>
<td>Yes</td>
</tr>
<tr>
<td>100187500</td>
<td>1002500000</td>
<td>100218750</td>
<td>No</td>
</tr>
<tr>
<td>100187500</td>
<td>100218750</td>
<td>100203125</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>etc.</td>
</tr>
</tbody>
</table>
Binary search is awesome!

- Takes $O(\log N)$ time, where $N$ is the size of the range of values searched
  - Some huge bound like $10^9$ on input data is often a clue that the solution involves binary search...

- Also works on a pre-sorted list of values!
  - or you may be able to limit the possible values to a specific set of candidates...
The whole internet loves binary search, a lovely algorithm that seems easy to code! *5 seconds later* We regret to inform you the way you coded it loops forever and is also wrong.
Binary search is awful!

- Infamously easy to mess up when coding it on the spot
  - or even e.g. in CS161 HW with less time pressure...
  - do not let an interview be the first time you code it up!

- Only works if values in the range are of the form True True ... True False False... or vice versa – i.e a single switch from True to False (or vice versa)

- Example situations where binary search won't work:
  - find largest prime number less than 100000000
  - find minimum of unimodal f(x) on [1, 100000000]
Problem 4-2: Trendy Dessert

- Some new dessert place just dropped in Palo Alto, and there's a huge line.
- There are $K$ servers ($2 \leq K \leq 1000$); the $i$-th server takes $K_i$ minutes to help a customer, then opens up for the next one. Customers go to the lowest-numbered available server.
- The store just opened. You are position $N$ in line ($1 \leq N \leq 10^9$). Which server will help you?

How can we use binary search here?
Guess a time and check

e.g., $K_1 = 20$, $K_2 = 10$, $K_3 = 15$, $N = 10$

assume time in range $[0, 20 \times 10] = [0, 200]$. Midpoint 100

- $20 \times 10$ is a safe upper bound since the slowest server takes 20 minutes

- After 100 minutes:
  - Server 1 will have helped $\lceil 100/20 \rceil = 5$ customers
  - Server 2: $\lceil 100/10 \rceil = 10$
  - Server 3: $\lceil 100/15 \rceil = 7$
  - Total customers helped: 22. Too many!
Guess a time and check

e.g., $K_1 = 20$, $K_2 = 10$, $K_3 = 15$, $N = 10$

new range $[0, 100]$. Midpoint 50

- After 50 minutes:
  - server 1 will have helped $\text{ceil}(50/20) = 3$ customers
  - server 2: $\text{ceil}(50/10) = 5$
  - server 3: $\text{ceil}(50/15) = 4$
  - total customers helped: 12. Still too many!
Guess a time and check

e.g., $K_1 = 20$, $K_2 = 10$, $K_3 = 15$, $N = 10$

new range $[0, 50]$. Midpoint 25

- After 25 minutes:
  - server 1 will have helped $\text{ceil}(25/20) = 2$ customers
  - server 2: $\text{ceil}(25/10) = 3$
  - server 3: $\text{ceil}(25/15) = 2$
  - total customers helped: 7. Too few!
That's the idea!

- Binary search until you find the time when you get served. Determine which server opened up and served you at that time
- Make sure to handle edge cases where there are ties (lots of other customers served at same instant as you)
- $O(K)$ work per check, $O(\log (NK_{\text{Max}}))$ checks
Don't let Ian forget to say the password

(and the solution to 4-3 is on the next slides!)
There are KN actors in total, so we can solve this in $O(KN \log KN)$ time by combining all the lists and using an algorithm such as merge sort, then taking the median.

Or, we can merge the K lists like merge sort does: maintain pointers to the start of every list, then keep identifying and removing a largest score out of those pointed to, until we've seen half the actors. But if we have to look at the starts of all K lists each time to find the current largest score, this is $O(K) \times KN = O(K^2N)$...

- To find the largest score efficiently, we need to use something like a priority queue (remember last week's discussion?) to store the top K elements. Now finding the current maximum is $O(\log K)$, which cuts the time to $O(KN \log K)$.

Or, we can combine all the lists and then use a linear-time selection algorithm to solve in $O(KN)$ time. (The details of linear-time median finding are complicated; there's an overview article here: [https://rcoh.me/posts/linear-time-median-finding/](https://rcoh.me/posts/linear-time-median-finding/). This also comes up in CS161.)

- But is $O(KN)$ a tight bound here? Do we really need to look at all the data? 😐
There is also a solution sort of like the one in 4-2! In this case it involves a **double** binary search... 😲
- Outer: Binary search (over the range \([1, S]\)) for the median score \(S_m\).
- Inner: For each guess, for each of the \(K\) lists, binary search on the list to see how many scores are less than \(S_m\). Then take the total of these numbers.
  - If it's less than \(KN/2\), try making \(S_m\) bigger.
  - If it's greater than \(KN/2\), try making \(S_m\) smaller.
  - If it's \(KN/2\), or if we've found some \(S_m < KN/2\) and \(S_m + 1 > KN/2\), we're done.

Each check is \(O(K \log N)\), so the algorithm is \(O(K \log N \log S)\)