Solution Sketches

CS@Mines High School Programming Competition 2024

20 April 2024

Colorado School of Mines

Parking Pandemonium (Sumner Evans)

Summary

Calculate the amount of time it takes for Hannah to find a parking spot given that she circles the lot C times at M minutes per circle.

The solution is to read the first and third lines of input and multiply them. The only difficult thing here is ignoring the second line of input which is the temperature of the parking lot in Kelvin.

M-Climb Road (Ethan Richards)

Summary

You walk along a road that is $5280W$ feet long. You get sprayed with water every N feet. You do not get sprayed at the start, but it possible you could get sprayed exactly at the end.

Output $\left\lfloor \frac{5280W}{N} \right\rfloor$.

The notation $\lfloor x \rfloor$ means round down.

Abby’s Absolutes (Levi Sprung)

Summary

Calculate the number of apples that Abby will buy in K trips. Given that on the i^{th} trip you want her to buy A_i apples, but she will buy either 1 or N apples on each trip, whichever is closer to your request (or 1 in cases of ties).

Loop over each of the trips. During the i^{th} trip, if $A_i - 1 \leq N - A_i$, then Abby will buy 1 apple, so print 1. Otherwise, she will buy N apples, so print N.

QWERTY (Byron Sharman)

Summary

Riley typed some text with the keyboard ”ABCDEFGHIJKLMNOPQRSTUVWXYZ”. Translate this message to what it would be if she had typed using a ”QWERTYUIOPASDFGHJKLZXCVBNM” keyboard.

Create a map/dictionary to translate from each character typed using the ABCDEF keyboard, to what it would be on a QWERTY keyboard. Build a new string by passing each character through this map.

There are many other (shorter) ways to solve this problem.

Exact Change (Jayden Pahukula)

Summary

Umbertoland has bill denominations $1, 5, 15, 30, 150$.
Express N dollars using as few bills as possible.

Greedily use the largest bills possible.

The number of 150 dollar bills is $\lfloor \frac{N}{150} \rfloor$, and update $N \leftarrow N - 150 \times \lfloor \frac{N}{150} \rfloor$.

Repeat until you get down to the 1 dollar bills, and output your answer.

Win Streak (Sumner Evans)

Summary

Calculate the longest win streak over *N* games given the scores of your team and the opposing teams.

You need variables to keep track of the *current win streak length* and the *maximum win streak length*. Then, for each game:

- If your team wins, increment the current win streak by 1.
- If your team ties or loses, check if the current win streak is longer than the maximum win streak. If it is, set the maximum win streak to the current win streak.
  Then, reset the current win streak to 0.

Clock Catchup (Ethan Richards)

Summary

Given a start time and end time of the same day, calculate how many times each of the second, minute, and hour hands move onto the the 12.

Let H_s, M_s, S_s and H_e, M_e, S_e be the hour, minute, and second of the start time and end time, respectively.

The hour hand cross once if $H_s < 12 \leq H_e$, otherwise it does not cross at all.

The minute hand crosses $H_e - H_s$ times.

The second hand crosses $60 \times (H_e - H_s) + (M_e - M_s)$ times.

No Stragglers (Keenan Schott)

Summary

Given security logs for number of students, faculty, and visitors entering or exiting Mines Market, determine if there are still people inside, and if so, how many?

Notice that the distinction between students, faculty, and visitors is irrelevant to the problem and can be ignored.

Initialize some variable to 0 that will be used to track how many people are currently in Mines Market. Now process the logs sequentially, for “IN” events, add to the total, and for “OUT” events, subtract from the total.

If at the end the total is 0, output “NO STRAGGLERS”, otherwise output the total.

Purchasing Perishables (Nathan George)

Summary

Given a list of meal prices for *N* days, find an optimal interval for Katie to purchase *N* meals such that the total cost is minimized, and output this minimum cost.

The solution is to try each potential purchasing interval from 1 to *N* days and compute the cost of meals for that interval. If the interval is *k* days, then the cost can be expressed as:

$$\sum_{i=0}^{\lceil \frac{N}{k} \rceil - 1} p_{i \cdot k+1} \cdot \min(k, N - i \cdot k)$$

But *N* can be as large as 10^5. Is this fast enough?

Purchasing Perishables (Nathan George) (continued)

Yes, for each interval length k, we can compute its cost in $\mathcal{O}\left(\frac{N}{k}\right)$ time. Then, the total time to compute the cost over all intervals is

$$\sum_{k=1}^{N} \frac{N}{k}$$

We can show that this runs in $\mathcal{O}\left(N \log(N)\right)$ with the following comparison

$$\sum_{k=1}^{N} \frac{N}{k} \approx \int_{1}^{N} \frac{N}{x} dx = N \ln(N)$$

Closing Early (Kelly Dance)

Summary

Each pizza has S slices. There are R slices ready. N customers will arrive sequentially and the i^{th} customer will order A_i slices. When is the earliest time we can stop serving customers and have 0 slices left over?

We want to find the first $k \geq 0$ where $A_1 + A_2 + ... + A_k = R + fS$ for some other integer f representing the number of full pizzas cooked. To do this we consider both sides of this equation modulo S.

$$(A_1 + A_2 + ... + A_k)\%S = R$$

Closing Early (Kelly Dance) (continued)

$$(A_1 + A_2 + ... + A_k)\%S = R$$

To find k we can maintain a running total T. Now, we can loop over the customers until $T = R$. With each customer we update $T \leftarrow (T + A_i)\%S$.

Be careful about the $k = 0$ case, or the case when there is no such k.

This solution runs in $\mathcal{O}(n)$ time. $\mathcal{O}\left(n^2\right)$ solutions are too slow.

Winning Wagers (Umberto Gherardi)

Summary

Given a set of *N probability items* (coin, die, cards) and the amount of money you would lose if you incorrectly predict their outcomes, calculate the break-even point for the wager.

Let T_i be the number of possible outcomes for the i^{th} event. The probability of correctly predicting all N events is:

$$\frac{1}{D} = \prod_{i=1}^{N} \frac{1}{T_i}.$$

Note that since all T_i are integers, D is also an integer. Then set the the *expected value* of the wager to 0:

$$\frac{1}{D} \times W - \left(1 - \frac{1}{D}\right) \times L = 0.$$

Winning Wagers (Umberto Gherardi) (continued)

Solving for W, we get:

$$\begin{aligned}\frac{1}{D} \times W - \left(1 - \frac{1}{D}\right) \times L = 0 \\ \frac{1}{D} \times W = \left(1 - \frac{1}{D}\right) \times L \\ W = D \times \left(1 - \frac{1}{D}\right) \times L \\ W = (D - 1) \times L.\end{aligned}$$

This calculation *does not require any division*. No floating point math necessary.

If you attempted to use an equation that included division, you would likely have run into precision issues with your calculations.

Warehouse Stocking (Ethan Richards)

Summary

You are given a set of `PUT`, `TAKE`, and `FIND` operations on a warehouse. For all `FIND` operations, output all of the locations of the item being searched for in the warehouse.

The key here is to have *two* dictionaries: one to store a mapping of locations to the item they contain (L), and another to store a mapping of items to a list of the locations in which that item can be found (T).

Then, it's a matter of accounting for the state updates caused by each operation.

Warehouse Stocking (Ethan Richards) (continued)

For each of the `PUT` operations, update $L[\texttt{loc}] \leftarrow \texttt{item}$ and $T[\texttt{item}] \leftarrow$ append `loc` to $T[\texttt{item}]$.

For each of the `TAKE` operations, update $T[L[\texttt{loc}]] \leftarrow$ remove `loc` from $T[L[\texttt{loc}]]$ and set $L[\texttt{loc}] \leftarrow$ nil.

For each of the `FIND` operations, if $T[\texttt{item}]$ exists, sort and output it. If it does not exist, output `NOT FOUND`.

Procrastination (Ethan Richards)

Summary

Kelly has M hours to complete some tasks. Task i takes T_i hours and will increase his grade by G_i. Kelly opts to complete the shortest tasks first. He will break ties by picking the task with the greater grade points. What will be the total benefit to his grade?

Sort the tasks by increasing T_i and break ties with decreasing G_i. This can be done using a built in sorting algorithm on pairs/tuples if G_i is placed second in the tuple and negated.

Initiaize $F = 0$ to track his total grade. Now loop over the tasks with this new order as long as $M \geq T_i$. For each task update $M \leftarrow M - T_i$ and $F \leftarrow F + G_i$.

Output the final grade F.

Mines Motor Company (Mete Saka)

Summary

Given a sequeunce of locations to visit on a 2D grid of workstations, calculate the total distance you must travel.

Locations are given in order as 2-character strings. Examples: "AA" represents $(1, 1)$, "ZA" represents $(26, 1)$, "GX" represents $(7, 24)$.

Loop over locations starting from the second location (since the first is the starting location).

Now we want to add up the distance between successive pairs of locations. Convert the characters for the X direction of each location to a number 1-26 (or ASCII), then take the absolute value of their difference and add this to the total distance. Repeat for the Y direction.

Trolley Troubles (Jonathon Robel)

Summary

Given a map of N parallel L-length tracks with cracks, find a path that minimizes the number of cracks crossed.

This is a graph modelling problem. We want to construct a graph such that finding the shortest path provides us with a path through the tracks that minimizes the number of cracks crossed.

- **Nodes** are track segments ($N \times L$ total) labeled (t, s) where t is the track number and s is the segment number.
- **Edges** are decisions we can make at each track segment (move forward or switch tracks)
- **Edge Weights** are the number of cracks crossed to go to the next track segment (0 if no crack or 1 if there is)

Trolley Troubles (Jonathon Robel) (continued)

For each node (t, s), create the following edges depending on the segment type of (t, s):

- `=` pieces — create an edge to $(t, s + 1)$ with weight 0.
- `H` pieces — create an edge to $(t, s + 1)$ with weight 1.
- `^` pieces – create an edge to $(t, s + 1)$ and an edge to $(t - 1, s)$ both with weight 0.
- `v` pieces – create an edge to $(t, s + 1)$ and an edge to $(t + 1, s)$ both with weight 0.

It may be helpful to create a sigular *finish* node that all (t, L) nodes have a zero-cost edge to. This allows you to find the shortest path to a single node.

Trolley Troubles (Jonathon Robel) (continued)

Now, we need to actually find a path through the graph from $(k, 0)$ to the finish node. There are a couple ways to do this:

- **Dijkstra's Algorithm** — This algorithm uses a priority queue to visit nodes in a uniform manner radiating from the start node.
- **0-1 BFS** — This algorithm performs a normal BFS with the following modifications:
    1. Use a deque (double-ended queue) instead of a queue.
    2. If the edge has weight 1 enqueue to the *back* of the deque, if the edge has weight 0 enqueue to the *front* of the deque.
    3. When updating the distance matrix, use the edge weight rather than hard-coding 1.

Trolley Troubles (Jonathon Robel) (continued)

In both algorithms, you need to track the node from which each node is discovered, then perform a **traceback** from the finish node to construct the path.

If the parent of a node (t, s) is $(t - 1, s)$, then the path will include s`d` indicating a downward switch at column s. The parent has a lower track number than its child, meaning that it is *above* the child node.

Conversely, if the parent of a node (t, s) is $(t + 1, s)$, then the path will include s`u` indicating an upward switch at column s.

You will have to perform the traceback starting from the finish node, and output in the *reversed* order.

New Professor (Mete Saka)

Summary

Blake has S_i shirts of the i^{th}. He wants to avoid wearing the same color shirt multiple times within the same work week (5-day weeks). How many days can he go before he must wear a shirt color he has already worn that week, or he runs out of shirts?

We will simulate choosing which color shirts to wear each week. When choosing the colors for a week, we will pick the 5 colors that have the most shirts remaining. Remember to update the number of remaining shirts of these colors.

When there are $k < 5$ distinct shirt color remaining, we can go exactly k more days before finally repeating our self or running out of shirts.

New Professor (Mete Saka) (continued)

Summary

Blake has S_i shirts of the i^{th}. He wants to avoid wearing the same color shirt multiple times within the same work week (5-day weeks). How many days can he go before he must wear a shirt color he has already worn that week, or he runs out of shirts?

This approach can be made to run in $O(CM\log(C))$ where M is the maximum number of any shirt color and C is the number of shirt colors by utilizing a priority queue, but we set the bounds low enough that repeatedly sorting the shirt color frequency list would still be fast enough ($O(C^2M\log(C))$).