2019 Virginia Tech High School Contest Solution Outlines

The Judges

Dec 14, 2019

A – Field Trip – First solved at 0:05

Problem

Given a list of n numbers, check if they can be split into 3 groups with the same sum.

A – Field Trip – First solved at 0:05

Solution

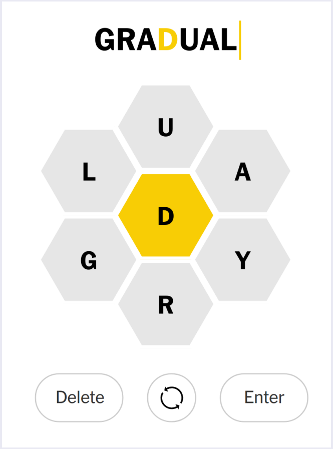
- Compute the total sum s. If not divisible by 3, output -1. Else greedily assign numbers to groups (busses). Ensure that 1st and 2nd groups are filled with exactly $s/3$ students.

C – Spelling Bee – First solved at 0:05

Problem

Given 7 letters of which one is a "center letter," and given a list of words, list all words that

- consists only of letters in the given set of 7
- are at least 4 characters long
- contain the center letter.



C – Spelling Bee – First solved at 0:05

Solution

- Implement as stated.
- Use either a set of a nested loop for establishing the subset relationship.

E – Musical Trees – First solved at 0:08

Problem

Given a set of rules of how n people pick from m trees, how many people do not get a tree.

E – Musical Trees – First solved at 0:08

Solution

- For every person, find the closest tree to them by distance and if there are multiple closest trees, pick the one with the smallest t_i.
- If someone else has already claimed that tree, then this person will not get a tree.
- Complexity: $\mathcal{O}(nm)$.

M – Broken Calculator – First solved at 0:08

Problem

Simulate a calculator with a register for last result and some redefined operations instead of addition, subtraction, multiplication, and division, simulate the redefined operations and compute results for a series of input operations.

M – Broken Calculator – First solved at 0:08

Solution

- Implement as stated.
- Be careful to use at least 64-bit longs for all operations to avoid overflow.

B – Escape Wall Maria – First solved at 0:16

Problem

Given a 2D grid in which horizontal or vertical movements are possible, with some constraints on some tiles (cannot be entered, can be entered only from right/left/above/below), determine the length of the shortest path to an open position on the edge of a grid if it exists. Output its length if it is less than a given value, else `IMPOSSIBLE`.

B – Escape Wall Maria – First solved at 0:16

Solution

- This is a standard BFS (breadth-first search) problem on the directed graph induced by the grid
- Model grid as a graph: pairwise directed edges between any two cells that Eren can travel in unrestricted, and a single directed edge for the special cases.
- Implementation: need a FIFO queue (`java.lang.ArrayDeque` or `std::queue`) or similar to keep track what still needs to be explored.
- Need to keep track of shortest distance to reach any square. (Can be done with a map or perhaps simplest with a 2D array that records that shortest distance from start to each cell).
- Process cells until queue is exhausted or a cell at the edge is reached.

H – Annoyed Coworker – First solved at 0:28

Problem

Given a set of coworkers, their initial annoyance levels, and how much their annoyance level increases every time they help you, minimize the highest annoyance level after asking for help h times.

H – Annoyed Coworker – First solved at 0:28

Solution

1. Greedy assignment via priority queue.
   - We only care about their annoyance level *after* they help you.
   - Hence, always ask the coworker whose annoyance level after they help you is the smallest compared to all others. That is, you should ask coworker i for help if for all other coworkers j, $a_i + d_i \leq a_j + d_j$.
   - This can be done with a min priority queue in $\mathcal{O}(h \log c)$ by sorting each coworker i by $a_i + d_i$ (sorting them by their annoyance after they help you).
2. Binary search for target value
   - Let each coworker help as often as possible to stay at or below target. Complexity: $\mathcal{O}(c \log(\max(a)h))$
   - Note: the "maximum annoyance level *any* coworker has" means that coworkers who do not help you still need to be considered. Some may already be very annoyed at you even if you don't ask them for help.

I – Reconstructing Tape Art – First solved at 0:34

Problem

- Given an array of numbers representing colors, output a set of instructions such that:
    - Each instruction a, b, c represents coloring the range $[a, b]$ with color c
    - Each color c is only colored once
    - The result of executing the instructions in the order given results in the input array

I – Reconstructing Tape Art – First solved at 0:34

Naive Solution

- iteratively remove colors that can be removed. this can be done by iterating over the current state of the array and removing any colors such that no other colors occur between the first and last occurrence.
- in the worst case, this is $\mathcal{O}(n^2)$ which is too slow!

I – Reconstructing Tape Art – First solved at 0:34

Stack Solution

- Recognize this quite similar to checking for balanced parentheses, which uses a stack (Link)
- For those not familiar with regular bracket sequences the giveaway is the fact that overlapping tape portions mimic stack behavior
- To implement this you do a sweep from left to right:
    - At each first occurrence of a color -> push
    - If current color matches the top of the stack -> continue
    - If current color does not match the top of the stack -> fail
    - At last occurrence of each color -> pop
- Output instructions encompassing range (first, last) appearance of each color left to right.
- This is $\mathcal{O}(n)$ and was the overwhelming solution of choice by contestants.

I – Reconstructing Tape Art – First solved at 0:34

List Solution

- You can also treat each contiguous subarray of the same color as a node in a linked list.
- You can then extract any color that only occurs once in the list
- Each time you remove you can merge the newly neighboring elements if they are of the same color
- If you ever have no colors that only occur once -> `IMPOSSIBLE`
- Otherwise output in the same fashion as the stack approach
- Also $\mathcal{O}(n)$

K – Random Digital Exponentiation – First solved at 1:06

Problem

Given 2 numbers a, b, where a has n digits a_i so that $a = \sum_{i=0}^{n-1} 10^i a_i$, find a sequence e_i of length n such that $b = \sum_{i=0}^{n-1} 10^i a_i^{e_i}$. You are guaranteed that e_i exists and is unique.

K – Random Digital Exponentiation – First solved at 1:06

Solution

- Idea: use recursive backtracking. We know that $a_0^{e_0} \equiv b \pmod{10}$. Try all exponents e_0 for which this is the case (and that are small enough, e.g. $a_0^{e_0} < b$, then recurse and find e_1 for the pair $((a - a_0)/10, (b - a_0^{e_0})/10)$.
- Base case: $(0, 0)$.

J – Jack The Lumberjack – First solved at 1:27

Problem

Given a list of n ($n \leq 1000$) piecewise linear functions of the form

$$f_k(t) = \begin{cases} 0 & \text{if } t < b_k \\ s_k + (t - b_k)i_k & \text{if } b_k \leq t \leq b_k + y_k \\ \max(0, s_k + y_k i_k - (t - b_k - y_k)i_k) & \text{if } t \geq b_k + y_k \end{cases}$$

where $t \in \mathbb{N}$, find the maximum of $\sum_{k=1}^{n} f_k(t)$.

J – Jack The Lumberjack – First solved at 1:27

Solution

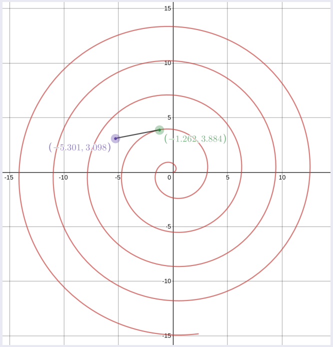
- Approach (1): The maximum of the sum must coincidence with the maximum of one of the functions. Since n is small, can try all $t_j = b_j + y_j$ years and compute $\sum_{k=1}^{n} f_k(t_j)$, then take the maximum. Complexity: $\mathcal{O}(n^2)$.
- Approach (2): The sum of piecewise linear functions is itself a piecewise linear function $F(t) = a_j t + b_j$ where $a_j = \sum_{j=1}^{n} \sigma_j y_j$ and σ_j either 0 or ± 1. Perform a line sweep through all points where a_j changes and update the cumulative value of a_j and $F(t_{j+1}) = F(t_j) + a_j(t_{j+1} - t_j)$. Need 3 events:
  1. a new species appears
  2. a species peaks and reverses slope
  3. a species dies out. Care must be taken with crossing 0.

  Complexity: $\mathcal{O}(n)$.

F – Archimedes – First solved at 1:37

Problem

Given an Archimedean spiral and a point $T = (t_x, t_y)$ in the plane, find the point on the spiral whose tangent goes through T when continued in the direction the spiral is moving - see illustration.



F – Archimedes – First solved at 1:37

Solution

- Step 1: Express the spiral in the complex plane as a complex function $z(\phi)$. Using Euler's identity, $z(\phi) = re^{i\phi} = b\phi e^{i\phi}$.
- Step 2: Find the derivative wrt ϕ using product rule

$$\begin{aligned} z'(\phi) &= be^{i\phi} + b\phi i e^{i\phi} \\ &= e^{i\phi} b(1 + \phi i) \end{aligned}$$

- Step 3: Find phase angle ϕ_0 of target T.
- Step 4: Perform a bisection or binary search in $[\phi_0 - \frac{\pi}{2}, \phi_0]$ to find ϕ_s such that the points T, polar$(b\phi_s, \phi_s)$, and polar$(b\phi_s, \phi_s) + z'(\phi_s)$ are collinear.
- Instead of performing Step 2, a finite difference approximation with sufficiently small dx is accepted as well.

F – Archimedes – First solved at 1:37

Implementation Hints

- When using C++, the derivative can be implemented directly using `std::complex`, e.g. `polar(b, phi) * std::complex{1.0, phi}`
- To implement the remaining geometry (e.g. crossproduct to establish collinearity), use operations described here: [URL].
- When using traditional 2D geometry, recognize that multiplying by $e^{i\phi}$ implies a 2D rotation by angle ϕ. Thus, the formula of the derivative is equivalent to rotating the vector $(1, \phi)$ by ϕ and scaling it by b in the complex plane. [URL].

G – Perfect Skyline – First solved at 1:52

Problem

Given a set of blocks with heights and a set of building with heights, output a valid assignment of blocks to buildings such that the sum of the blocks is the height of the building. If no such assignment exists then print -1.

G – Perfect Skyline – First solved at 1:52

Solution

- The first subproblem to solve is which sets of blocks can correctly create a given building.
- Since the building heights are large and the number of blocks is small can be done with subset sum.
- This portion of the problem takes $\mathcal{O}(n * 2^n)$

G – Perfect Skyline – First solved at 1:52

Solution

- Now the possible solution for each building must be combined into a single solution for all buildings
- This is possible with dynamic programming as well
- This can be implemented by iterating and maintaining information for the current prefix of buildings
- Then, for each subset it is necessary to track the transition used to arrive at this subset

G – Perfect Skyline – First solved at 1:52

Solution

- Transitions are executed with subsets of subsets DP.
- For each subset of blocks in the current prefix, is there a subset of this subset that is valid for the previous prefix and the difference generates the current building.
- By storing transitions we can reverse engineer the subsets used at each building and output the solution
- Final complexity $n * 3^n$

D – I.O.U. – First solved at 4:54

Problem

- Find and cancel all cycles in a directed weighted graph.
- Given a directed weighted graph with positive edge weights, find a cycle and subtract from each edge in the cycle the minimum of the cycle's edge weights.
- Remove any edges that have now zero edge weights.
- Repeat until there are no more cycles in the graph.

D – I.O.U. – First solved at 4:54

Solution

- Implementation follows problem statement.
- Finding a cycle can be done, among other methods, via a DFS search, keeping a stack of nodes on the current path. If a node that is already on the current path is encountered by the DFS, you have found a cycle.
- Cancel it and repeat until there are no more cycles.
- Complexity: $\mathcal{O}(nm)$.

L – Counting Codes – Not solved during contest

Problem

Count the number of complete codes (matrix filled with nonzero decimals from 1 to 9) subject to a few constraints:

1. no digit repeats within any row
2. for each digit l, except for those in the topmost row and rightmost column, let u be the digit above it and let r be the digit to its right in the matrix. Then one of the following must be true:
    - u is the product of l and r
    - u is the sum of l and r
    - u is the difference of l and r or r and l
    - u is the quotient of l and r or r and l

L – Counting Codes – Not solved during contest

Solution

- Naive backtracking should time out.
- Need backtracking with some pruning.
- Find the position (i, j) of the first zero from the top. Replace the value at (i, j) with any digit that is not in row i and maintains an arithmetic relationship between digits above $(i - 1, j - 1)$ and to the left $(i, j - 1)$. Backtrack to find other solutions, and return the total.