2024 Rocky Mountain Regional Solutions

The Judges

Nov 9, 2024

Credits

- Darcy Best
- Howard Cheng
- Ian DeHaan
- Ryan Farrell
- Zachary Friggstad
- Yosuke Mizutani
- Lawry Sorenson
- Josh Taylor
- Nick Wu
- Charlie Zheng

Problem

Given the price P_i for each flight ticket $1 \leq i \leq N$, find the minimum possible net cost, assuming that you maximize the reimbursement amount and minimize your purchase.

Solution

- The answer can be determined by finding the minimum and maximum prices.
- Notice that answers are non-negative.
- Output $\max\left\{0, \left(\min_{i=1}^{n} P_i\right) - \left(\max_{i=1}^{n} P_i\right)/2\right\}$.

Problem

Given $N \leq 42$ positive integers summing to at most $2^N - 2$, find a value v such that at least two subsets of integers sum to v.

Solution

- **Binary search the answer**: Maintain a range $[L, U)$ such that more $|U| - |L|$ subsets sum to a value in the range. For the middle point M in the search, we know either $[L, M)$ or $[M, U)$ satisfies this.
- To answer the query, it suffices to compute $q(x) =$ number of subsets summing to $< x$. Then $q(B) - q(A)$ is the number of subset sums in $[A, B)$.

Always Know Where Your Towel Is (cont.)

Solution

- To compute a query $q(x)$ quickly, we do some preprocessing:
  - Split the input into two sets of size $n/2$ each. Generate all $2^{n/2}$ subsets of both and store their sums in a sorted array.
  - To compute $q(x)$, we do a two-index sweep through the two sorted arrays in linear time in the size of the arrays.
- Total running time $O(n \cdot 2^{n/2})$.

Big And — First solve: 27min

Problem

Given boolean inputs with switching delays and `AND` gates that have a switching delay of D, create a circuit to minimize the maximum delay between when an input source changes and when the output of the circuit changes.

Solution

- Greedy: pick two input sources with the least delay n_i, n_j and feed them into an `AND` gate. The output has delay $D + \max\{n_i, n_j\}$.
- Replace these two sources with a new single source with this delay. Repeat until there is only one source.
- Use a *heap* or other similar data structure to manage the least-delay signal. $O(nlogn)$.

Emoticons First solve: 12m

Problem

Given a string of N characters, find the minimum and maximum message lengths after certain conversions are made.

Solution

- There are $k = 95$ possible characters (ASCII code 32-126) and $t = 10$ emoticons to check.
  - Only 11 characters are relevant to the given emoticons.
- A simple $O(k^2tN)$-time algorithms would suffice.
  - Consider all possible replacements by Emma: $O(k^2)$.
  - Simulate all conversions by the platform: $O(tN)$.

Enchanted Maze — First solve: 129min

Problem

Given a maze and a pair of twins, find the least number of moves to reach the exits following a number of rules.

Solution

- Represent the configurations as a state graph: states are specified by the locations of the twins, and the status of each switch (on/off). At most $2^4 R^2 C^2$ states.
- Carefully implement all the rules, especially with regards to trying to simultaneously move on to a switch and a corresponding obstacle.
- Use breadth-first search on the state graph.
- Complexity: linear time in the number of states.

Garage Door Code — First solve: 19m

Problem

Given a list of N strings of up to 12 digits, find all of the subsequences of length K that they have in common.

Solution

- Iterate over all numbers up to length K.
- For each number check that it is a subsequence in each input string. Greedy matching works to do this.
- There are more efficient solutions, but this is all that was intended.

Palindromic Word Search — First solve: 131min

Problem

Given an $R \times C$ gird of characters, find the maximum-area rectangular subgrid that has a palindromic row and a palindromic column.

Solution

- Equivalent to computing for each grid cell (i, j), what are the longest horizontal and the longest vertical palindromes spanning (i, j).
- So for each grid cell (and each space "between grid cells") we compute the longest horizontal palindrome centered there in $O(C)$ time and update the "longest horizontal palindrome" information for the cells it spans. Also do this for columns.
- Running time: $O(R^2 \cdot C + R \cdot C^2)$. Faster is possible, but not required.

Pillow Stacking First solve: 24min

Problem

Given a set of possible pillow softness levels and a desired softness level C, determine if you can choose k (not necessarily unique) pillows with softness levels $C_1, C_2, \ldots, C_k$ such that

$$C = C_1 + \left\lceil \frac{C_2}{2} \right\rceil + \left\lceil \frac{C_3}{4} \right\rceil + \ldots + \left\lceil \frac{C_k}{2^{k-1}} \right\rceil.$$

Solution

- We can solve this problem with DP - let $f(t, c)$ denote that softness level c can be reached with t pillows.
- Base Case: $f(0, 0)$.
- Recursive Case: $f(t, c)$ true if $f(t-1, c - \lceil \frac{C_i}{2^{t-1}} \rceil)$ is true for any pillow softness C_i.

Pillow Stacking (cont.)

Solution

- This is too slow. $\mathcal{O}(N \cdot C)$ states, with $\mathcal{O}(N)$ transitions per state, total time $\mathcal{O}(N^2 \cdot C)$.
- After $\log_2(\max C_i)$ rounds, all pillows have softness level 1. So if $f(\log_2(\max C_i), c)$ is true for any $c \leq C$, the answer is YES.
- Running time $\mathcal{O}(N \cdot C \cdot \log(\max C_i))$.

Rectangle Tiling — First solve: 58min

Problem

Perfectly cover an $R \times C$ rectangle using squares whose side lengths are powers of 2.

Solution

- Good to think about special cases: suppose the input is square ($R = C = 2^m$). If you have a $2^m \times 2^m$ tile then just use that. Otherwise split the region into $2^{m-1} \times 2^{m-1}$ tiles and recurse on each.
- Also consider the case having an infinite supply of square tiles. Can show you would pick the largest tile that "fits" and place as many as you can, hugging the top/left edges.
- The final solution is to imagine the infinite-supply solution and then tile these squares (biggest squares first).

Rocky Mountain First solve: 54min

Problem

Given a sequence of $(x_1, y_1), \ldots, (x_N, y_N)$ with a unique peak, find the lowest visible point on the left and right from the peak.

Solution

- To find the lowest visible point on the left, scan left from the peak P one point at a time, keeping track of the leftmost visible point Q.
- If the current point R is colinear with P and Q, or if $P \to Q \to R$ has clockwise orientation, then R is visible.
- Remember the lowest visible point, breaking ties by taking leftmost point if necessary.
- Finding best point on the right is similar.
- Complexity: $O(N)$.

Sandwich Art — First solve: 45min

Problem

Given a directed graph $G = (V, E)$ (for ingredients V and dependencies E), initial ingredients $S \subseteq V$, a set $U_i \subseteq V$ of ingredients for each genre i, and the maximum number M of total ingredients, determine if there exists a genre i such that $U_i \supseteq \text{Reach}(S)$, where $\text{Reach}(S)$ denotes the set of all reachable vertices from any vertex in S.

Solution

- First, compute $\text{Reach}(S)$ by BFS or DFS on G.
- If $|\text{Reach}(S)| > M$, the given instance is a `disaster`.
- For each genre i, test if U_i is a superset of $\text{Reach}(S)$.
  - The sum of $|U_i|$ over all genres is upper-bounded by 10^6.
  - The overall check can be done efficiently using a fixed-size boolean array.

Problem

Given a list of ranges, $[a_i, b_i]$, find the integer values that are in all of the ranges.

Solution

- This problem reduces to finding the intersection of a set of ranges.
- This intersection is a range, $[l, r]$, where $l = \max a_i, r = \min b_i$.
- If $l > r$, the intersection is empty. Otherwise, the number of integers in the range is $r - l + 1$.
- r is the minimum value contained in all input ranges.

Space Elevator — First solve: 179min

Problem

Given a starting location for an elevator and a list of pickup and dropoff requests, find the minimum distance the elevator must travel to fulfill all requests. There is no limit to how many items can be in the elevator at once.

Solution

- Consider the lowest and highest points the elevator must visit to fulfill the requests. We can model the solution at a high level as which point the elevator visits first:
  - start $\rightarrow$ low $\rightarrow$ high $\rightarrow$ end
  - start $\rightarrow$ high $\rightarrow$ low $\rightarrow$ end
- In the second case, note that moving from high to low can fulfill all requests to move to a lower point without additional movement, and the same happens for requests to move upward that have a pickup location at or above the start.

Space Elevator (cont.)

Solution

- This reduces the problem to finding the optimal placement of requests to move upward that have a pickup location below start.
- There are three possible placements of additional requests in the specified model.
  1. **start** $\rightarrow$ **high** Here we dip down to a lower point, b_1, before going to the highest, and the marginal cost is twice the distance between b_1 and the starting point. Note that b_1 is a pickup location if it exists.
  2. **high** $\rightarrow$ **low** In this case, we fully backtrack the length of each request, and the marginal cost is twice the size of the union of all ranges fulfilled this way.
  3. **low** $\rightarrow$ **end** This involves coming back up after visiting the lowest point, and the marginal cost here is the distance between the lowest point and and final dropoff location, b_3.

Space Elevator (cont.)

Solution

- Dynamic Programming can be used to find the optimal placement of requests into these three groups.
  - Sort the remaining requests by increasing pickup location.
  - Try every pickup location as a potential value for b_1. All of the requests starting at the current one will fall into group 1, while the previous requests will be in either group 2 or 3.
  - Use DP to track the size of the union of previously seen ranges, and update b_3 to the latest dropoff location when doing so decreases the cost.
  - Find the minimum total cost over all values seen during this iteration.
- The case of start $\rightarrow$ low $\rightarrow$ high $\rightarrow$ end can be handled by the same code by reflecting the input and running it again.