ICPC Divisional 2022 - Editorial

Your Name

March 14, 2025

1 Problem A: Trivial

Let $A[i]$ be a boolean value indicating whether the i-th column consists of identical characters. Thus, $A[i] = 1$ if all characters in the i-th column are the same. The problem then reduces to finding the longest contiguous subsequence of ones.
Time Complexity: $O(MN)$

2 Problem B: Data Structures, Inversions

After each iteration, the number of inversions for each element decreases by one, as larger elements move toward the end. Thus, the number of passes required is equal to the maximum number of inversions for any element.

Let $L[k]$ represent the number of elements with k inversions. Then, for the k-th round, the answer is:

$$L[k] + L[k+1] + L[k+2] + \dots$$

We compute the number of inversions by compressing the values and storing them in a Fenwick tree.
Time Complexity: $O(N \log N + M)$, where M is the maximum number of inversions for a single element.

3 Problem C: Modified BFS

Given that N is large, we optimize by iterating over adjacent vertices while removing already-visited ones. This gradually reduces the number of vertices that need to be traversed. Using `std::set`, we efficiently manage this reduction.
Time Complexity: $O(N \log N)$

4 Problem D: Prime Factorization

Since N is large, directly computing the binomial coefficient results in TLE and integer overflow. Instead, we determine the number of times each prime appears in the factorization of:

$$(n-k+1)(n-k+2)\dots n \quad \text{and} \quad k!$$

With these values, we compute the final result via multiplication, stopping once it exceeds the required threshold.
Time Complexity: $O(N(M+K)\log(\text{MAX}))$, where MAX is the maximum value in the set.

5 Problem E: Divide and Conquer

To connect the leftmost and rightmost houses, there must be an intermediate city k such that:

$$L[k] \le \text{leftmost} \quad \text{and} \quad \text{rightmost} \le \text{house}$$

We solve the subproblems $[\text{leftmost}, k-1]$ and $[k+1, \text{rightmost}]$ recursively, akin to Quicksort, leading to a worst-case complexity of $O(N^2)$. To optimize, we process both sides of the intervals simultaneously.
Time Complexity: $O(N \log N)$

6 Problem F: Convex Hull, Ternary Search

Define $f(x_t) = |P_1 - P_2|$, where P_1 and P_2 are the polygons formed after cutting along $x = x_t$. Observing that $f(x)$ is a concave function, we apply ternary search to approximate the optimal x_t.

To compute P_1 and P_2 for a fixed x_t: 1. Split the polygon into upper and lower parts. 2. Find intersections of $x = x_t$ with these parts (binary search). 3. Compute the areas using the shoelace theorem.

For easier implementation, rotate the polygon by 90°.
Time Complexity: $O(N \log N)$

7 Problem G: Knapsack

This is a standard knapsack problem.
Time Complexity: $O(NK)$

8 Problem H: Trivial

Follow the problem's instructions directly.
Time Complexity: $O(NPC)$

9 Problem I: 2-SAT

Model the first dish as x_1 and the second dish as $\neg x_1$. We need an assignment satisfying:

$$\neg(x[i] \land x[j])$$

Time Complexity: $O((m+n)L)$, where L is the length of the string.

10 Problem J: Greedy

Assign children closest to the destination the slowest speed that still allows them to escape the wolf.
Time Complexity: $O(N^2)$

11 Problem K: Trivial

Follow the problem's instructions. To simplify implementation, flatten the board into an array.
Time Complexity: $O(N)$

12 Problem L: Tree DP, Game

Fix the starting node. A node is in a losing state if all its children are in a winning state. Since leaves are winning states, we compute the result for node 1 using DFS. By applying re-root DP, we extend this to all nodes.
Time Complexity: $O(N)$