The 2025 ICPC Asia Hong Kong Regional Contest

Analysis

Nov. 30 2025

Overall Situation

Expected Hardness

$HJ < BIK < GF < AD < CEL$

H. Longest Common Prefix

Description

You're given a string s. Let t_i be the string after remove position i from s. Find $i \neq j$ such that $lcp(t_i, t_j)$ is maximized.

Solving Situation

- AC before frozen: 136
- First Solve: CQOI Flames 4min

H. Longest Common Prefix

Solution

Note that the answer is at least $n-2$ by choosing $i = n-1$ and $j = n-2$.
Meanwhile, the answer is at most $n-1$ since the length of any substring is at most $n-1$.
Checking whether the answer can be $n-1$ is equivalent to checking whether there are two identical adjacent letters in the string.

J. Re: Becoming the Programming Champion

Description

You are given, for each of three contestants, their leaving time from the contest and the set of problems they need to solve.
Your task is to determine whether there exists a schedule such that each contestant can finish all of their required problems before leaving, and the total penalty time is strictly less than that of the current champion.

Solving Situation

- AC before frozen: 55
- First Solve: Fear WA, all in implementation 38 min

J. Re: Becoming the Programming Champion

Solution

We consider the schedule in reverse time and decide, for each moment, which problem each contestant should solve. One can easily observe that, in order to minimize the total penalty time, each contestant must solve their longest problem last.
If we sum up the time costs of all problems, we obtain the total duration of the contest. Then, reasoning backwards is equivalent to letting the three contestants enter the contest one by one in reverse time. Each time a contestant "enters," we choose, from the set of problems that are still unsolved and belong to the contestants currently present, the one with the largest time cost and mark it as solved. This selection process can be efficiently maintained using a heap.

K. Cyclic Shift

Description

Given a sequence, consider all of its cyclic shifts. For each shift, take the sequence of prefix maximums. Your task is to find the cyclic shift whose prefix-maximum sequence is lexicographically largest.

Solving Situation

- AC before frozen: 81
- First Solve: Nice Nature 15 min

K. Cyclic Shift

Solution

For each element in the sequence, we connect it to the first element after it (in cyclic order) that is strictly larger than it. This is equivalent to linking each value to the next value that appears in the corresponding prefix-maximum sequence.
It can be observed that the resulting graph is a tree, and any path along this tree will not wrap around the original cyclic sequence more than once.
Once we have this tree, we perform hashing on it: the hash of each node is defined as the concatenation of

- the hash of its parent, and
- the hash of a pair (value,length),

where value is the value of the node, and length is the distance in the original sequence between the node and its parent. Intuitively, this length represents how many times the current value appears consecutively in the prefix-maximum sequence.

K. Cyclic Shift

Solution

Next, we apply binary lifting on this tree to precompute "jump hashes" for powers of two.
For any two prefix-maximum sequences (corresponding to two cyclic shifts), we can locate their starting nodes in the tree, and then use binary lifting on the hashes to find the length of their longest common prefix.
This tells us the first position at which they differ.
At this first mismatching position, there are two possible cases:

- The values differ. Then we can directly compare these values to determine which sequence is lexicographically smaller.
- The values are the same, but the lengths differ. In this case, we take the position with the shorter length, jump from that node to its parent, and then continue comparing the values from there.

Using this procedure, we can compare any two prefix-maximum sequences in $O(\log n)$, leading to an overall time complexity of $O(n \log n)$.

I. Dfs Order - Extra Stage

Description

Given m dfs orders of the same tree. Determined how many distinct trees can produce all m dfs orders.

Solving Situation

- AC before frozen: 21
- First Solve: Rinsed to Remote 62 min

I. Dfs Order - Extra Stage

Solution

A necessary and sufficient condition for an interval in the first permutation to be a valid subtree is that its elements form a contiguous block in every permutation and share the same first element across all of them. We refer to such intervals as valid intervals, and all of them can be precomputed in $O(n^2 m)$ time.

I. Dfs Order - Extra Stage

Solution

Let $f_{l,r}$ be the number of trees that can produce the interval $[l, r]$ of the first permutation, $g_{l,r}$ be the number of continuous several subtrees that can produce interval $[l, r]$. then:

$$f_{l,r} = \begin{cases} g_{l+1,r} & \text{if } [l, r] \text{ is valid} \\ 0 & \text{Otherwise} \end{cases}$$

$$g_{l,r} = \sum_{k=l+1}^{r} f_{l,k-1} \times g_{k,r}$$

The answer is $f_{1,n}$. Overall it's $O(n^2 m + n^3)$.

B. Travelling

Description

Given a tree with weight on each edge. You should find a path start from node 1, visiting every node, and go back to node 1. Each edge can be passed at most twice.
The cost of each move is the weight of the passed edge, and you can make k consecutive moves free. For each k from 0 to $2n-2$, find the mininum total cost.

Solving Situation

- AC before frozen: 23
- First Solve: SUSTech-Lyaki 39 min

B. Travelling

Solution

Consider to choose the path that is free, containing exactly k moves.
Because we can pass an edge at most twice, once we pass an edge for the second time, all nodes on its subtree should be visited.
That means if we want to make an edge free twice, we should make all edges on its subtree free twice.
Hence, all the edges that are made free exactly once always form a simple path p. The edges that are made free twice is chosen from complete subtrees of such edges (u, v), which u is on path p and v is a child of u.

B. Travelling

Solution

Let $f_{i,j,k}$ indicates the maximum total weight if we made j moves free in the subtree of i, with k $(0 \leq k \leq 2)$ path made free once, starting from i and terminating at a descendent of i.
This value can be calculated in $O(n^2)$ using Dynamic Programming.
The answer of k is $2\sum_{1 \leq i < n} w_i - \max_{1 \leq x \leq n}(f_{x,k,2})$.

G. Watering System

Description

Given a watering system. Each watering machine can water left or right and have the same water range L. You should find the minimal L so that we can water every pot in $[l, r]$.

Solving Situation

- AC before frozen: 1
- First Solve: DeepSleep 235 min

G. Watering System

Solution

The operations are equivalent to

- Watering Left: set $a_i = \min(a_i, x - i),\ i \leq x$
- Watering Right: set $a_i = \min(a_i, i - x),\ i \geq x$

lemma

There's at most $O(\log n)$ intervals need to be updated in a single modification.

G. Watering System

Solution

proof

Considering all the same-direction watering machines, it is clear that updates will only occur between the newly inserted watering machine and the next same-direction watering machine. For a reverse-direction machine, if it intersects with the newly inserted machine, the next intersecting machine must be at least twice the distance away. So there's at most $O(\log n)$ intervals needs to be updated.

Use a segment tree to maintain the minimal value and maximal value of $a_i - i$ and $a_i + i$. We can find the next interval in $O(\log n)$. Overall it's $O(n \log^2 n)$.

bonus

Prove the solution is $O(n \log n)$ when $q = O(n)$.

F. Find the Circuit

Description

This is a communication problem.
First Run:
You are given a connected undirected graph with n vertices and m edges and a simple cycle $p_1, p_2, \ldots, p_k$ in the graph.
You must assign a direction to each edge of the graph.
Second Run:
You receive the resulting directed graph (with permuted vertex labels).
You must output the permuted labels of the original cycle vertices in the correct directed cyclic order.
$n, m \leq 5 \cdot 10^5$

Solving Situation

- AC before frozen: 5
- First Solve: DeepSleep 105 min

F. Find the Circuit

Solution

First Run:
Arbitrarily define a topological order a from 1 to n such that $a_i = p_i$ (for the cycle vertices), and direct all edges (except the edge (p_1, p_n)) according to this topological order (i.e., from the earlier vertex to the later vertex in the order).
After this orientation, the largest cycle in the resulting directed graph will be exactly $p_1, \ldots, p_k$.

F. Find the Circuit

Solution

Second Run:
First, extract the set of vertices in the cycle using Tarjan's algorithm or topological sorting.
Then, repeatedly find a vertex v with in-degree 1. Suppose v is reached from u, then the edge (u, v) must be part of the cycle. We can delete all outgoing edges from u except (u, v).
Initially, p_2 must have in-degree 1. It can be easily proved by mathematical induction that this process will eventually yield the complete cycle.
The total time complexity is $O(n + m)$.

A. Bipartite Graph Matching Problem

Description

Given a bipartite graph, for every interval of right part, find it's maximal match with the whole left part.

Solving Situation

- AC before frozen: 0
- First Solve: N/A

lemma

The set of right vertices that can have a perfect match with left part is a linear matroid.

Proof: Let n dimension vector v_i be:

$$v_{i,j} = \begin{cases} x_{i,j} & \text{if } (i,j) \in \mathrm{E} \\ 0 & \text{Otherwise} \end{cases}$$

Which $x_{i,j}$ be an element in polynomial field. Then v_i is the matroid we want.

A. Bipartite Graph Matching Problem

Solution

We process the right-part vertices by scanning from $r = 1$ to $r = n$, maintaining a set of currently matched vertices. For each new vertex r, we first attempt to find an augmenting path to increase the size of the maximum matching. If successful, r is added to the matched set. Otherwise, we check whether r can replace the smallest-index matched vertex while preserving the matching property. This can be done by finding the smallest vertex in augment tree.
The answer for a fixed r is the number of matched vertices within the interval $[l, r]$ after processing vertex r.

D. Dumb Problem

Description

Given an integer sequence ,you can perform the following operation exactly once:

- Choose a permutation $p_1, p_2, \ldots, p_n$ of $1, 2, \ldots, n$.
- Then, in order from $i = 1$ to n, execute:

$$a_{p_i} \leftarrow a_{p_i} + a_{p_i-1} + a_{p_i+1}$$

where $a_0 = 0$ and $a_{n+1} = 0$.

Maximize the sum of all elements.

Solving Situation

- AC before frozen: 0
- First Solve: N/A

D. Dumb Problem

Solution

Let q_i be the inverse of p. i.e $q_{p_i} = i$. One can find that the answer is only related by the relative order of q_i and q_{i-1}.
Moreover, consider a fixed number a_i. Which positions it can "spread" only depends on a continuous increasing or decresing q_i start from i. So we can come up with a $O(n^2)$ dp: precompute $s_i = \sum_{j=1}^{i} a_j, s'_i = \sum_{j=1}^{i} a_j \cdot j$.
let $f_{i,0/1}$ be the answer where $q_i > q_{i-1}$ or $q_i < q_{i-1}$. Enumerate $j < i$:

$$\begin{cases} f_{i,1} \leftarrow f_{j,0} + 2 * (s'_{i-1} - s'_{j-1}) - (s_{i-1} - s_{j-1}) \cdot (2j-3) - a_j + a_i \\ \cdot (i-j+1) + a_{i+1} \cdot (i-j) \\ f_{i,0} \leftarrow f_{j,1} + (s_i - s_j) \cdot (2i+3) - 2 \cdot (s'_i - s'_j) + a_j \cdot (i-j) + a_{j-1} \cdot (i-j) \end{cases}$$

Optimize this dp by Li-Chao tree or Line Container.

C. Stonebag

Solution

lemma

Let $S = \sum a_i$, b_i be the final sequence, then it's valid if and only if: $b_i \bmod n < a_i$, or $n - \max b_i \bmod n + \sum b_i \bmod n \leq S$

Assuming all $b_i \bmod n$ are known, the number of possible schemes is the number of ways to partition $\lfloor \frac{(S-\sum(b_i \bmod n))}{n} \rfloor = M$ into $n+1$ piles (the last pile is the remainder), which equals $\binom{M+n}{n}$.
Note that if $M \neq 0$, there are no restrictions on b_i. Categorize the discussion as follows:
If $M \neq 0$, the constraint is: $\sum(b_i \bmod n) \leq S - n$ Consider $f(x) = \left(\frac{1-x^n}{1-x}\right)^n$, then: ans $= \sum_{i=0}^{S-n} [x^i] f(x) \cdot \binom{\frac{S-i}{n}+n}{n}$
Enumerate intervals where the combinatorial numbers are the same, and use inclusion–exclusion to compute the coefficients.

C. Stonebag

Solution

If $M = 0$, the constraint is:
$S - n < \sum(b_i \bmod n) \leq S - n + \max(b_i \bmod n)$
Let $g[i](x) = \left(\frac{1-x^i}{1-x}\right)^n$, with $g[0] = 0$. Define $f[i] = g[i+1] - g[i]$, which is the generating function where the maximum value is exactly i. Then, for each $f[i]$, the answer is: $\sum_{j=S-n}^{S-n+i}[x^j]f[i] = \sum_{i=0}^{n-1}\sum_{j=S-n}^{S-n+i}[x^j](g[i+1] - g[i])$
Considering that $[x^j]g[i]$ has a coefficient of $+1$ at $(i-1, j)$ and -1 at (i, j), contributions only occur at $j = i + S - n$ (for $0 \leq i < n-1$) and at $i = n$. The contribution at $i = n$ can be computed brute-force in $O(n^2 \log n)$.
Now consider how to compute: $\sum_{i=0}^{n-1}[x^{i+S-n}]g[i]$
Compute $h = \sum g[i] \cdot x^{n-i}$, i.e.: $h = \frac{\sum(1-x^i)^n \cdot x^{n-i}}{(1-x)^n}$ Both the numerator and denominator can be computed in $O(n^2)$.

Second case: Similarly, let $S = \sum a_i$. Consider when $n - \max(b_i \bmod n) > S - \sum(b_i \bmod n)$, i.e.: $S < \sum(b_i \bmod n) + n - \max(b_i \bmod n)$ Assume $\max(b_i \bmod n)$ is enumerated.
The constraint becomes: $S - n + \max(b_i \bmod n) < \sum(b_i \bmod n)$
Let $\max(b_i) = P$. Note that $\sum(b_i \bmod n)$ clearly should not exceed S, so we only care about the coefficients in the interval $[S - n + 1, S]$. Consider the polynomial $f[i]$ as the generating function for $a_i - (b_i \bmod n)$ in the i-th pile. From the inequality: $0 \leq b_i \bmod n \leq \min(a_i, P)$ we know the number taken c_i satisfies: $a_i - \min(a_i, P) \leq c_i \leq a_i$

C. Stonebag

Solution

Now, sort a_i. Then, for a certain prefix of a_i, $0 \leq c_i \leq a_i$, and for the remaining suffix, $a_i - P \leq c_i \leq a_i$. Note that since there are only n terms, brute-force convolution is equivalent to computing prefix sums, with a complexity of $O(n)$ per operation. Therefore, any brute-force method is acceptable.

Additionally, if there are x terms in the latter part, each brute-force convolution will require x operations. After this, the lowest-degree coefficient of the polynomial increases by at least x (since P decreases by 1, the lowest term increases). The prefix part only requires a total of n convolutions, so the overall complexity of this part is $O(n^2)$.

L. Cyclic Shift II

Description

You are given two integer sequences A and B of lengths $2n$ and $2m$ respectively, where each absolute value from 1 to n (for A) or 1 to m (for B) appears exactly twice.

You can perform the following operations on A:

- Cyclic shift
- Delete a prefix pair $(x, -x)$
- Insert a new pair $(x, -x)$ at the front (where x does not appear in the current sequence)
- Reverse the sequence and negate all elements
- Rearrange across a cut (two cases depending on whether the chosen pair has same or opposite signs)

Two sequences are *equivalent* if there exists a sign-preserving bijection f with $f(-x) = -f(x)$ that maps one sequence to the other.

Determine if A can be transformed into a sequence equivalent to B using these operations.

L. Cyclic Shift II

Solution

We model each sequence as an edge-labelled polygon that defines a closed surface.

Consider A of length $2n$:

- Take a $2n$-gon.
- Label its vertices $0, 1, \ldots, 2n-1$ in cyclic order.
- The j-th edge is the directed edge from vertex j to vertex $(j+1) \bmod 2n$. Put label a_j on it.
- If $a_j = +i$, the orientation of this edge matches the boundary (from j to $j+1$).
- If $a_j = -i$, the orientation is opposite to the boundary (from $j+1$ to j).

Each absolute value $i \in \{1, \ldots, n\}$ appears exactly twice, so there are exactly two edges with labels i. We identify (glue) these two edges together to form a single real edge, preserving their arrow directions:

- Suppose one edge is $u \to v$ and the other is $u' \to v'$;
- When we glue them, we identify $u \sim u'$ and $v \sim v'$.

L. Cyclic Shift II

Solution

After all pairs are glued, we obtain a connected closed surface (no boundary). The mapping f only renames edge labels and does not change which edges are glued, so it does not change the resulting surface.
All the given operations (cyclic shift, insert/delete $(x, -x)$ at the front, reverse-and-negate, cut-and-rearrange) are classical "cut-and-paste" moves on the polygon; they do not change the topological type of the resulting surface.
Therefore, The problem is equivalent to asking whether the two sequences A and B define homeomorphic closed surfaces.
From the classification theorem of connected closed surfaces, such a surface is uniquely determined by:

1. Whether it is orientable;
2. Its Euler characteristic χ.

Thus, we only need to compute for A and B:

$$(\text{orientable}_A, \chi_A) \quad \text{and} \quad (\text{orientable}_B, \chi_B)$$

L. Cyclic Shift II

Solution

In this "polygon with each label appearing twice" model, there is a classical criterion:

> *The resulting surface is orientable if and only if, for every i, the two occurrences of $\pm i$ are exactly one $+i$ and one $-i$.*

Intuitively, if a pair appears as $(+i, +i)$ or $(-i, -i)$, then the corresponding edge is glued in the same direction on both sides, creating a Möbius-type twist and making the surface non-orientable.

All operations in the problem preserve the pattern of signs for each absolute value. The renaming map f only changes names, not sign patterns. Hence, the orientability is a true invariant.

In practice:

- For each i, count how many times $+i$ and $-i$ appear.
- If, for all i, we have exactly one $+i$ and one $-i$, then the surface is orientable.
- Otherwise, it is non-orientable.

We do this once for A and once for B.

L. Cyclic Shift II

Solution

For any closed surface we have:

$$\chi = V - E + F$$

where V is the number of vertices, E is the number of edges, and F is the number of faces.
In our construction:

- We started from one $2n$-gon, so after identification there is still exactly one face: $F = 1$.
- Each absolute value i creates one real edge, so $E = n$.
- The only non-trivial part is the number of vertices V after gluing.

We can count V via DSU.

E. Bipartite Graph Weighting Problem

Description

Given a bipartite graph. Answering several questions (a, b, k).

Solving Situation

- AC before frozen: 0
- First Solve: N/A

E. Bipartite Graph Weighting Problem

Solution

For every S and $N(S)$, we regard $(|S|, |N(S)|)$ as a point in 2d plane. Let the total sum of p in left part be x, and the sum of p in the right part be y. (x, y) lies in the convex hull of all $(|S|, |N(S)|)$. Moreover, each question is equivalent to using a line with negative slope to cut the convex hull.
So if we can find out the lower convex hull of the point set, we can answer each question in $O(\log n)$. Note the the upper convex hull is not solvable since it's a NPC problem.
Finding every possible $(|S|, |N(S)|)$ is also a NPC problem. But we can find the lower convex hull but finding a positive slope line's tangent with the convex hull.

E. Bipartite Graph Weighting Problem

Solution

Consider a line $Ax + By + C$ where $A, B > 0$. Finding the tangent C is equivalent to finding the Maximum Weight Closed Subgraph, where each left part vertex weights A and right part vertex weights $-B$.
So we can find a tangent point in $O(dinic)$. Let's show how to find the whole convex hull in $O(size \times dinic)$. initially there's two points $(0, 0)$ and (n, n) in convex hull. Each time we choose two adajcent point in the convex hull and find the tangent of the convex hull and the line pass through the two points. If a new point is found, keep doing this procedure. Otherwise we find an edge of the convex hull.
After finding the lower convex hull, the problem can be solved in $O(q \log n)$. Overall it's $O(n^{\frac{2}{3}} \times flow(n, m))$.

Thanks