International Collegiate Programming Contest Asia Hangzhou Regional Contest

Analysis

Zhejiang University

12.04.2022

A. Modulo Ruins the Legend

Description

Given an array a of length n, add an arithmetic progression to it so that the sum after modulo m is minimized.

A. Modulo Ruins the Legend

Solution

Observation: Adding a arithmetic progression to the sequence is equivalent as adding its "median" element n times. Note that if n is even, the "median" element can be $\frac{x+y}{2}$. Hence, d can be restricted to 0 or 1 without loss of generality.

The problem now changes into solving the equation $an + bm = k$. Since n is small, we can enumerate b instead of using Extended Euclid Algorithm.

Total time complexity is $\Theta(n)$.

B. Useful Algorithm

Description

Given two integer sequence c, d of length n and a sequence w of length m, each number c is less than 2^m. There are also q modifications, each modification will change c_i, d_i at one position. Calculate $\max\limits_{1\le i,j\le n}\{\max\{\max\{w_x | x \in S(c_i, c_j)\}, 0\} \cdot (d_i + d_j)\}$ for the $q+1$ versions, where $S(a, b)$ is the set of bits that a carry occurred when calculating $a + b$ with their binary representations.

B. Useful Algorithm

Solution

When calculating $x+y$ with their binary representation, if the carry occurred at the i-th bit, we have $(x \mod 2^{i+1}) + (y \mod 2^{i+1}) \geq 2^{i+1}$. Let $D_x = \max\{d_i | c_i = x\}$. Then we enumerate the bit where carry occurred, say t-th bit, then we are calculating $\max\{D_i + D_j | i + j \geq 2^{t+1}\}$.

B. Useful Algorithm

Solution

Let $D'_x = \max\{d_i | c_i = 2^{t+1} - x\}$, the formula now is $\max\{D_i + D'_j | i \geq j\}$. We can use `std::multiset` or `std::priority_queue` to maintain D_x and D'_x. And use segment tree to maintain $\max\{D_i + D'_j | i \geq j\}$. The time complexity is $\Theta(nm \log n + qm^2 + qm \log n)$.

There is also a solution using sqrt decomposition. The time complexity is $\Theta(nm \log n + q2^{\frac{m}{2}} + qm \log n)$, which can also pass.

C. No Bug No Game

Description

Given n items, find a permutation to wear them all such that the k-limited magic buff can offer the most points of bonus power.

C. No Bug No Game

Solution

When $\sum_{i=1}^{n} p_i \leq k$, the answer is $\sum_{i=1}^{n} w_{i,p_i}$.
Otherwise, we can always find an item that is partially upgraded.
Fix the i-th item to be the partially upgraded one, iterate its upgraded size t $(0 \leq t \leq \min(k, p_i))$, then we need to select a subset of items S such that $i \notin S$ and $\sum_{j \in S} p_j = k - t$. The target is to maximize $\sum_{j \in S} w_{j,p_j}$.
Here we have two solutions.

C. No Bug No Game

Solution1

Consider a simple DP solution with $f_{i,j,t}$ denoting the largest points of bonus power when we select items among the first i items, such that the total upgraded size is j, and t $(0 \leq t \leq 1)$ item is partially upgraded.
To make transitions, we need to iterate the size of the partially upgraded item. Fortunately, the size p is at most 10.
The time complexity is $\Theta(nkp)$.

C. No Bug No Game

Solution2

When p is unbounded, we can still solve this problem efficiently. Consider a divide and conquer solution with $solve(l, r)$ denoting the partially upgraded item is indexed in $[l, r]$:

- When $l = r$, just iterate the upgraded size of the l-th item and update the answer.
- Otherwise, let $mid = \lfloor \frac{l+r}{2} \rfloor$, divide the problem into two subproblems $solve(l, mid)$ and $solve(mid + 1, r)$. Here we need to add all the items indexed in $[mid + 1, r]$ into the left knapsack, and add all the items indexed in $[l, mid]$ into the right knapsack.

The time complexity is $T(n) = 2T(\frac{n}{2}) + \Theta(nk) = \Theta(nk \log n)$.

D. Circle Money

Description

There are n people sitting in a cycle, during each round, each person will give half of his money to the next one in order. Find the number of money each people have after 2022^{1204} rounds.

D. Circle Money

Solution

The sum of money won't change. Assume the sum of money is S and $w = \frac{S}{n+1}$. The number of rounds is large enough that the amount of money each person have will converge to $2w, w, w, w, \ldots, w$. Then we just have to calculate S and w, the time complexity is $\Theta(n)$.
The proof is left as exercise for the readers.

E. Oscar is all you need

Description

You are given a permutation of length n. Find the permutation with the smallest lexicographical order you can get with the following operation: cut the sequence into three consecutive non-empty parts, and swap the first part and the last part.

E. Oscar is all you need

Solution

When $n = 3$, we only have to check whether we have $p_1 > p_3$.
When $n \geq 4$, we will try to sort the sequence. Assume when we do the operation on the sequence, the size of the three parts are $x, n - x - y, y$, we call this $op(x, y)$.

E. Oscar is all you need

Solution

Let's use insertion sort to sort the sequence. First we will try to put 1 to the beginning of the sequence. If 1 is not the first two elements of the sequence, and 1 is at position p, we will just do $op(1, n-p+1)$. Otherwise if 1 is the second element of the array, then we will do $op(2,1), op(1,1)$. Now 1 is at the beginning of the sequence. Assume now a prefix of length $i \geq 1$ is sorted, and we will insert the last element of the sequence into its position, say after the j-th$(1 \leq j \leq i)$ element, then we will just do $op(j,2), op(1,j)$.

E. Oscar is all you need

Solution

But we can only do this when $j < n-2$, this means we can't use it to sort $1, 2, \ldots, n-2, n, n-1$. However, sorting it is similar to sort $1, 2, 4, 3$, we can use BFS to find a solution for such situation. One possible solution is: $op(1,1), op(n-2,1), op(2,n-3), op(n-3,1), op(1,n-2)$. The total number of operations is $2(n-2)+5 = 2n+1$, which will fit in with the constraints.

We can also do this by selection sort, but need a few more classified discussion. The time complexity is $\Theta(n^2)$, which can be optimized to $\Theta(n \log n)$ by using splay, but not needed.

F. Da Mi Lao Shi Ai Kan De

Description

You are given the messages of n message groups, for each group forwarding all messages containing substring `bie` to group 0 once, or report there is no such message.

F. Da Mi Lao Shi Ai Kan De

Solution

Do what the problem asks. Compare the strings in any way you like. The time complexity is $\Theta(\sum |s|^2)$ or $\Theta(\sum |s| \log n)$.

G. Subgraph Isomorphism

Description

Given a connected undirected graph, judge whether all spanning trees are isomorphic.

G. Subgraph Isomorphism

Solution

If the graph is a tree, then the answer is "Yes".
If there are more than one cycle in the graph, then the answer is "No". The proof is left as an exercise.
If there is exactly one cycle in the graph, the rooted trees attached to the cycle must be of the form `abababab` or `aaaaaaa`. The proof is also left as an exercise.
We can use tree hash or carefully implemented sort to detect whether the rooted trees are isomorphic.
The total time complexity is $\Theta(n)$ or $\Theta(n \log n)$, depending on implementations.

H. RPG Pro League

Description

There are three roles in the game. Only two types of teams are valid:

- One Damager, two Synergiers, and one Buffer.
- Two Damagers, one Synergier, and one Buffer.

We are given the information of n players. Each player can only play a certain set of roles. We need to invite players to form the maximum number of teams such that the total cost is minimized.
The cost of each player can be updated.

H. RPG Pro League

Solution

First let's find the maximum number of teams.
Build a bipartite graph with $4 + 7$ vertices.

- 4 vertices in the left part, denoting 4 slots in a team: a Damager slot, a Synergier slot, a Buffer slot, and a flexible slot that can be either a Damager or a Synergier.
- $7 = 2^3 - 1$ vertices in the right part, denoting the set of roles a player can play, respectively.
- For each pair of possible vertices, add arcs between them according to the meaning.

Binary search for the maximum number of teams, and check whether the left part can entirely match the right part.

Instead of running MaxFlow, we can do it more efficiently using Hall's Theorem:

Theorem

Let G be a bipartite graph with partition X, Y.
G contains a matching of X if and only if $|N(S)| \geq |S|$ for all $S \subseteq X$.

According to Hall's Theorem, $cnt = \min_S \lfloor \frac{|N(S)|}{|S|} \rfloor$. Here S is a non-empty subset of the left 4 vertices, and $N(S)$ denotes how many players the set is linked to.
Hence we can find the maximum number of teams cnt in $\Theta(2^4)$.
In addition, given all the values of $N(\cdot)$, we can check whether it is possible to reach cnt teams in $\Theta(2^4)$.

H. RPG Pro League

Solution

Next let's find the cheapest way to invite players such that they can form *cnt* teams.
Initially, invite all the players, and we need to cancel some players to cut down the cost.
Take the player with the highest cost, try to cancel him, there are two possible results:

- If it's impossible to form *cnt* teams, we must invite him.
- If it's still possible to form *cnt* teams, we must cancel him, because he is the most expensive one.

Hence we can get the answer by sorting all the players according to the cost, and canceling players greedily.

H. RPG Pro League

Solution

The last question is how to perform cost updates efficiently.
Actually canceling players is the dual matroid of the matching matroid.
So when a cost update comes, we may replace one invited player with one uninvited player.
Group all the invited and uninvited players according to the role set they can play. Obviously, only the player with the lowest/highest cost in each group may be replaced.
Since there are only 7 groups, try them all and choose the best decision.
The time complexity is $\Theta(q \log n)$.

I. Guess Cycle Length

Description

There is a cycle with n ($n \leq 10^9$) vertices, and the vertices are numbered from 1 to n in a fixed order. You should guess n in 10^4 queries.
Walk forward x vertices and receive your position after walking, where x is your output and $1 \leq x \leq 10^9$.

I. Guess Cycle Length

Solution

Firstly, if the query limit is 10^5, then you can use BSGS.
Since the indices of vertices are not used in the method, we can modify the method a bit to utilize the indices.
We first use 3333 operations to randomly walk on the cycle and obtain a sample of the graph. Assume that the maximum index during the walk is m, then the probability of $m \leq n \leq m + 10^7$ is more than $1 - 10^{-14}$. So we can use BSGS to solve the remaining part.

I. Guess Cycle Length

Solution

The final solution is:

1. walk 3333 steps randomly, assume the maximum index during this step is m.
2. walk 3333 steps with length 1.
3. walk 1 step with length m.
4. walk 3333 steps with length 3333.

If we walked on the same index twice in steps $2 \sim 4$, then we can know the cycle length by finding the total travelled length between two operations.

The total number of queries is $\Theta((n \log \frac{1}{\varepsilon})^{\frac{1}{3}})$, where ε is the failing probability.

J. Painting

Description

Draw n segments one by one. When drawing the i-th segment, we will start at point $(0, a_i)$, and end at (W, b_i). When we touch any segment drawn before, stop at the intersection point immediately.
Report where we will stop for each step.

J. Painting

Solution

Each segment will stop at the intersection of another segment or the wall. Build a tree rooted at the wall. Each segment's parent on the tree is the segment or the wall it hits.

When drawing a new segment, in order to find its parent, first find the segment whose start point is closest to it on both sides via `std::set`, say u and v. Now the path on the tree from u to v is $u - LCA(u, v) - v$, which forms a convex hull. Hence we can binary search on the path and find the intersection.

In order to do binary search and find LCA, binary lifting is needed.

The time complexity is $\Theta(n \log n)$.

K. Master of Both

Description

Given a sequence of n strings, answer q queries: how many inversions are there in the sequence with the specified alphabet.

K. Master of Both

Solution

Consider how we compare two strings s, t, assume their longest common prefix is l, then we will compare s_{l+1}, t_{l+1}. Hence we only have to count $c_{x,y}$, which means how many $1 \leq i < j \leq n$ satisfy that $s_{i,LCP(s_i,s_j)} = x, s_{j,LCP(s_i,s_j)+1} = y$. Then with the given alphabet, if character $a < b$, then we will add $c_{b,a}$ to answer. We can do the counting by inserting all strings into a Trie one by one. Do not forget to count the number of pairs that s_j is a prefix of s_i. The time complexity is $\Theta(\sum |s| \cdot 26 + q \cdot 26^2)$.
Bonus: try to solve it in $\Theta(26q)$.

L. Levenshtein Distance

Description

Given two strings S and T. Find the number of substrings of T whose edit distance between S is exactly i for every possible integer i $(0 \leq i \leq k)$.

L. Levenshtein Distance

Solution

Fix a suffix T' of T, then the goal is to find the number of its prefixes whose edit distance between S is i for $i = 0, 1, 2, \ldots, k$.
Let $f_{i,j} = x$ denotes the maximum value of x such that $S[1 \ldots x]$ can match $T'[1 \ldots x + j]$ after editing i times, because if x can match, then all the values below x can also match.
Note that we can always match pairs of same characters greedily, so $f_{i,j} = f_{i,j} + LCP(S[f_{i,j} + 1 \ldots |S|], T'[f_{i,j} + j + 1 \ldots |T'|])$, here LCP is the longest common prefix and can be calculated in $O(1)$ via suffix array and RMQ.
After that, we need to edit a character, so we can update $f_{i+1,j-1}$, $f_{i+1,j}$ or $f_{i+1,j+1}$.
Note that $i \leq k$ and $-i \leq j \leq i$, hence we can compute all the values of f in $\Theta(k^2)$.

L. Levenshtein Distance

Solution

For a state $f_{i,j} = |S|$, we can know the edit distance between $T'[1 \ldots |S| + j]$ and S is no more than i.
However, some prefixes may be ignored when LCP is too long.
Fortunately, assume edit distance between $T'[1 \ldots i]$ and S is $dis(i)$, we have $dis(i) = \min\{dis(i), dis(i+1)+1\}$.
Hence we can find the answer using $dis(\cdot)$.
The time complexity is $\Theta(nk^2)$.

M. Please Save the Pigeland

Description

Given a weighted tree with n vertices and k special vertices $c_1, c_2, \ldots, c_k$, you need to choose some vertex x and a parameter d and minimize $\sum_{i=1}^{k} \frac{dis(x,c_i)}{d}$ and $d | dis(x, c_i)$ for all i.

M. Please Save the Pigeland

Description

When we chose vertex x, it's obvious we should choose $d = \gcd dis(x, c_i)$.
So now we have to choose vertex x to minimize $\frac{\sum dis(x,c_i)}{\gcd\{dis(x,c_i)\}}$.
Let's iterate vertex x, and try to calculate $\sum dis(x, c_i)$. This can be done by dynamic programming. We can calculate the number of special vertices in the subtree of u and the sum of distance from u to all special vertices in its subtree. Then we can do another DFS to calculate the answer.

M. Please Save the Pigeland

Description

Actually, we can calculate $\gcd\{dis(x, c_i)\}$ similarly. We can calculate the $\gcd$ of all distance from x to the special vertices in its subtree, and use the similar way to compute the answer. The only thing we have to do is maintaining the $\gcd$ of a set $S = \{a_1, a_2, \ldots, a_m\}$. We need to support merging two sets and add some integer w to all elements in the set. We represent the $\gcd$ of the set by maintaining two integers $f = a_1, g = \gcd\{a_2 - a_1, a_3 - a_1, \ldots, a_m - a_1\}$.
Then adding w to all elements we can change (f, g) to $(f + w, g)$, merging two sets $(f_1, g_1), (f_2, g_2)$ result in $(f_1, \gcd\{g_1, g_2, f_2 - f_1\})$.
The time complexity is $\Theta(n \log w)$.

Thanks!