International Collegiate Programming Contest Asia Hong Kong Regional Contest

Analysis

Zhejiang University

01.14.2023

A. TreeScript

Description

Given a tree of size n, you need to create nodes such that just before you create i, the p_i must be still in a register, find the minimum number of registers needed.

Create the "small" subtrees first. When creating the subtrees except the "biggest" one, you need an extra register to store the root. So you can get $dp_u = max(dp_{v_1}, dp_{v_2} + 1)$, where v_i is child of u and $dp_{v_1} \geq dp_{v_2} \geq \cdots$.

B. Big Picture

Description

Given a $(n+1) \times (m+1)$ matrix. For each row, a prefix of random length will be colored with a given probability, as well as each column. Calculate the expected number of orthogonally connected regions.

B. Big Picture

Solution

It could be easily noticed that the number of the colored connected regions must be 1.
For each uncolored connected region, there is exactly 1 grid whose right and bottom are both colored.
So we only need to calculate the expected number of uncolored grids whose right and bottom are both colored, and this could be done by calculating the probability for each grid respectively.
The time complexity is $\Theta(nm)$.

C. Painting Grid

Description

Color a $n \times m$ grid in black and white so that:

1. The number of white cells is equal to the number of black cells.
2. There are no equal rows.
3. There are no equal columns.

If both n, m are odd, then no solution exists.
If $n > 2^m$ or $m > 2^n$, then by the pigeonhole principle, no solution exists.
If a solution exists, then at least one of n, m is even. Without loss of generality, let's assume that m is even.

C. Painting Grid
Solution

For the first $\lceil \log_2(m) \rceil$ rows, we can make sure that they satisfy the following properties:

1. Each row have the same number of black and white cells.
2. There are no equal rows.
3. There are no equal columns.

This can be done by divide and conquer. An example of $m = 10$ is shown below. Note that the first 5 columns can be flipped 0→1 and 1→0 to change into the last 5 columns.

0000011111
0101010101
0011011001
0000111110

For the remaining rows, we can view each row as a binary sequence, and pair up binary sequence s with $\neg s$. If we put both of the sequences inside the grid will result in the same number of black and white cells. If one of the two sequences in a pair has appeared in the previous rows, we shall temporarily abandon them (at least one row is abandoned). By putting the remaining paired rows into the grid as much as possible, we will have some remaining unfilled rows in the grid while the filled rows satisfy the three conditions. In the last step, we fill in the unfilled rows using the previously abandoned rows. The three conditions will still be satisfied after filling.

D. Shortest Path Query

Description

Given a DAG with black and white edges. In each query, report the shortest path from 1 to x_i if we regard the length of each black edge as a_i and regard the length of each white edge as b_i.

Consider a simple DP solution with $f_{i,j}$ denoting the minimum number of white edges we need to walk through from vertex 1 to vertex i if we have walked through j black edges.
There will be $\Theta(n^2)$ states, which are unfortunately too slow to pass.

Lemma

For a fixed vertex i, if we draw each state $f_{i,j}$ as a point $(j, f_{i,j})$ on the plane, only states on the convex hull will affect the answer.

Lemma

For a fixed vertex i, there will be at most $\Theta(n^{\frac{2}{3}})$ states on the convex hull.

Hence we can only store the states on the convex hull for each vertex, and answer the query by just iterating all the points.

The time complexity is $\Theta((m + q)n^{\frac{2}{3}})$.

E. Goose, goose, DUCK?

Description

Given an array a of length n, count how many intervals satisfy that no element exists exactly k times.

E. Goose, goose, DUCK?

Solution

Let's enumerate the right end of the interval, say r. Then we will maintain for all $l \leq r$, if l is a legal left end.

For each element x, let's assume the existing positions of this element in $[1, r]$ is $p_1, p_2, \ldots, p_m$ and $m \geq k$, then the illegal left end is an interval which is $[p_{m-k} + 1, p_{m-k+1}]$. Noticing that when r change from $r - 1$ to r, only the interval of a_r will change, so we can try to maintain for each l, how many elements is illegal, and the problem is reduced to add 1 or -1 to some interval and count the number of global minimum value. This can be done with a segment tree.

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

F. Sum of Numbers

Description

Given n digits without '0', add some '+' to make the expression minimum.

Obviously, the difference in the length of the adjacent numbers must be -1, 0, or 1. So there are 3^k ways. But only about $\frac{1}{k+1}$ of the ways are possible which depends on the value of n modulo $(k+1)$, so there is an $O(\frac{3^k}{k+1}n)$ solution. You can add some prunings to make it fast but the naive implementation is enough.

There are two segments AB and BC, where AB rotates around A and BC rotates around B and the angle is limited, calculate the area swept by AB and BC.

When $\beta \leq \frac{\pi}{2}$, the answer is $(l_1 + l_2)^2\alpha + l_2^2\beta$.
Otherwise, the "lowest point" of BC is not necessary the whole stick, but one point on BC. So there are some extra area under the sector. Assume that ϕ reaches β now, the following two conditions have to be classified discussed:

- 2α and $\angle CBA$
- $\angle BCA$ and $\frac{\pi}{2}$

So there are four situations. Be careful when you are calling inverse trigonometric function, the result might be `nan` due to precision errors.
The time complexity is $\Theta(1)$.

The duck has a killing skill which its cool down can be arbitrary chosen between $[l, r]$, and he meets a goose every b seconds, how much time can the duck kill exactly k geese?

It is always better to let the cool down to be l seconds. The answer is $\lceil \frac{l}{b} \rceil \cdot b \cdot k$.

The time complexity is $\Theta(1)$.

Given n points in the Euclidean plane. In each query, report the closest pair of points if only points indexed in $[l, r]$ are available.

Let's create $\Theta(\log 10^8)$ cell systems. The k-th cell system will help us to find pair of points whose distance is in $[2^k, 2^{k+1})$. In the k-th cell system, a point (x, y) will be put in the cell $(\lfloor \frac{x}{2^{k+1}} \rfloor, \lfloor \frac{y}{2^{k+1}} \rfloor)$.

Answer queries offline, moving the right endpoint to the right and storing the answer for each left endpoint.

Consider one step of moving the right endpoint to position i, which denotes the point p_i. We need to find some points indexed before i, and update the answer. For each cell system, only points in the cell adjacent to the cell p_i will be put into can affect the answer. So for each cell system, iterate all the points in the adjacent 3×3 cells, and add p_i into the system.

To control the number of points in each cell, if we find a point p_j whose distance from p_i is lower than 2^k, remove all the points indexed not greater than j from the k-th system. Since the size of each cell is $2^{k+1} \times 2^{k+1}$, and each existing point is at least 2^k far from others, there will be $\Theta(1)$ points in each cell.

Hence we will find $\Theta(\log 10^8)$ pair of points to update the answer when we moving the right endpoint by a step.

Now we need a data structure to do $\Theta(n \log 10^8)$ updates and $\Theta(q)$ range queries. We can use Fenwick Tree ($\Theta(\log n) - \Theta(\log n)$) or Sqrt Decomposition ($\Theta(1) - \Theta(\sqrt{n})$). The time complexity of the whole algorithm is either $\Theta(n \log n \log 10^8 + q \log n)$ or $\Theta(n \log 10^8 + q\sqrt{n})$, respectively.

Given n, calculate $\frac{1}{n^2} \cdot \sum_{i=0}^{n-1} \max(\sum_{j=0}^{n-1} i \oplus j, i \cdot n)$. Answer T queries.

Let $C_i(l, r) = \sum_{i=l}^{r} [i \mod 2^{i+1} \geq 2^i]$, which is the number integers having 1 on the i-th bit, this can be calculated in $\Theta(1)$, since it has a circular section of 2^{i+1}.

Let $f(i) = (\sum_{j=0}^{n-1} i \oplus j) - i \cdot n$ and rewrite the formula as $i \cdot n + \sum_{i=0}^{n-1} \max(f(i), 0)$.

We have $f(i) = \sum_{j=0}^{29} d_j C_j(0, n-1) \cdot 2^j$, where $d_j = 1$ when the j-th bit of i is 1, and $d_i = -1$ otherwise.

Let the most significant bit of n is p, and $w_j = d_j C_j(0, n-1) \cdot 2^j$. we have $\frac{w_k}{w_{k-1}} \geq \frac{2^{p-k+1}}{2^{p-k}+1}$ for $k < p$. So without considering the MSB, if the higher bits are decided, the sign of $f(*)$ are almost decided.
So there aren't many maximal intervals that $f(*)$ have same sign. Actually there are about at most 20 intervals when $n \leq 10^9$. We can use binary search or divide and conquer to find the intervals and calculate the contribution.

Now we just have to know for an interval $[l, r]$, if the sign of $f(*)$ is same. We can calculate the maximum value and minimum value of $f(*)$ on this interval. One possible way is dynamic programming, or you can also make $[l, r]$ always be $[p \cdot 2^k, (p+1) \cdot 2^{k+1})$ and calculate it with some pre-calculation.

The time complexity for one query is $\Theta(k \log^2 n)$ or $\Theta(k \log n)$, where k is the number of intervals.

You have an array a_i. You can do any number of the following operation on the array: Choose an element a_i and a positive integer x, change a_i into $(a_i \bmod x)$. 0 cannot appear in the array after any operation. Maximize the GCD of the array after the operations.

K. Maximum GCD

Solution

The maximum value of the answer is equal to the minimum element of the array.
We need to check if the minimum element of the array a_{min} can be the answer.
A number a_i can be changed into any integer in $[1, \lfloor \frac{a_i - 1}{2} \rfloor] \cup a_i$. If a number is even, then it is a multiple of $\frac{a_i}{2}$.
If x is the answer, then all elements in the array must satisfy at least one of the following two conditions:
1. a_i can be changed into x, that is, $x \in [1, \lfloor \frac{a_i - 1}{2} \rfloor] \cup a_i$.
2. a_i is a multiple of x.

If $x = a_{min}$ cannot make the array satisfy the condition, then the answer will always be $\lfloor \frac{a_{min}}{2} \rfloor$ because $x = \lfloor \frac{a_{min}}{2} \rfloor$ will always make the array satisfy the condition.

L. Delete Permutation

Description

Given a permutation a, answer whether you can turn it into another sequence q by performing the following operations: Choose an interval of length l_i and delete the maximum element.

L. Delete Permutation
Solution

First we can notice that we can delete elements one by one from large to small. Assume the i-th element deleted is d_i, then if $d_i > d_{i+1}$, we can swap d_i and d_{i+1}, the deleting sequence is still legal.
Now Let's consider elements from large to small, if i is not deleted, then none of the further operating intervals can cover i, splitting the sequence into two sequences. Otherwise if the sequence which i is in have x remaining elements, then we have to perform an operation with $l_i \leq x$.
This can be maintained with `std::set` and fenwick tree. We just have to store all such x and check if they can pair with given l_i, this can be solved greedily.
The time complexity is $\Theta(n \log n)$.

Thanks!