AtCoder World Tour Finals 2019

writer: rng_58

February 21st, 2018

A: Magic

Suppose that the magician moved the treasure in the order $y_1 \to y_2 \to \cdots \to y_{K+1}$. Here $y_i \neq y_{i+1}$ for each i because it doesn't make sense to move the treasure to the same box. Then, your output sequence must contain this sequence as a subsequence. Otherwise, if the treasure was initailly in y_1, it was moved to y_2 immediately before Snuke named y_1, moved to y_3 immediately before Snuke moved y_2, and so on, he never finds the treasure. On the other hand, if your sequence contains $y_1 \to y_2 \to \cdots \to y_{K+1}$, it is easy to see that he can always find the treasure in case it was moved this way.

We call a sequence *interesting* if no two adjacent elements are the same. Thus, your task is to find a sequence that contains all interesting sequences of length $K+1$ as subsequences.

Let's define the state $(b_1, \ldots, b_N)$: it means that the sequence must contain all interesting sequences of length b_i that start with i. We can assume that $\max b_i - \min b_i \leq 1$, because all interesting sequences of length L that start with i is contained in some sequences of length $L+1$ that start with j. We start with the state $(K+1, \ldots, K+1)$ and the empty sequence. We append elements to the sequence one by one, and update the state that represents the set of interesting sequences the remaining part must contain.

It turns out that this process proceeds in a very limited way.

- After we add the first element the state becomes a permutation of $(K+1, \ldots, K+1, K)$.
- If we choose x first, then it doesn't make sense to choose x again because it doesn't change the state. Similarly, we can assume that the first $N-1$ elements are pairwise distinct - and each time we add a number, one of $K+1$ changes to K.
- Now we have a permutation of $(K+1, K, \ldots, K)$. The next step is a bit different - we must change the $K+1$, and it changes to $K-1$. Now we have a permutation of $(K, \ldots, K, K-1)$.
- Again we start changing the state in a similar way. We perform "decrement by two" once in each N steps.

This means that we only need to consider sequences of length $(K+1)N - K$, and each of the first N elements, the N-th element to the $(2N-1)$-th element, the $(2N-1)$-th element to the $(3N-2)$-th elements, $\cdots$ form a permutation of $\{1, \cdots, N\}$. In other words, there are $K+1$ permutations, and each adjacent pair of permutations share exactly one element in common.

Let $x_1, \ldots, x_N$ be those shared elements. For each i, $x_i \neq x_{i+1}$ must be satisfied. The upper bounds of the number of occurrences of c in the entire sequence gives the lower bound of its occurrences among $x_1, \ldots, x_N$. This is a well-known problem, and the answer is "Yes" when the sum of those lower bounds doesn't exceed N and none of the lower bounds exceeds $(N+1)/2$.

B: Multiple of Nine

Let's call the point between the i-th element and the $(i+1)$-th elemnt "Point i", and write the prefix sum s_i (the sum of the first i numbers modulo 9) on this point. Then, each constraint says that the values wrriten on the left bound and the right bound of the interval must be the same. Let x_i be the value written on both endpoints of the i-th interval.

How can we compute the number of digit sequences that match the given values of $\{x_i\}$? For simplicity, assume that the $2Q$ endpoints are pairwise distinct, and they are $a_1 < a_2 < \cdots < a_{2Q}$. Then, basically the number of digit sequences is

$$10^{a_1} \times \frac{10^(a_2 - a_1) - 1}{9} \times \cdots \times \frac{10^(a_{2Q} - a_{2Q-1}) - 1}{9} \times 10^{N - a_{2Q}} \tag{1}$$

However, when the values written on a_i and a_{i+1} are the same, the term $\frac{10^(a_{i+1}-a_i)-1}{9}$ increases by one. In other words, the score is multiplied by $\frac{\frac{10^(a_{i+1}-a_i)-1}{9}+1}{\frac{10^(a_{i+1}-a_i)-1}{9}}$.

Thus, the problem reduces to the following problem:

There are Q items, and we want to divide them into 9 groups labeled $0, \ldots, 8$. The score of one grouping is basically 1. However, we are given some pairs of items (p, q) with the coefficient c, and in case we put p and q to the same group the score is multiplied by c. Compute the total score of all 9^Q groupings.

Then, for each subset of items, we know the coefficient when those items form a group. This leads to a standard DP on subsets, and this solution works in $O(3^Q)$ time.

Note that in case the $2Q$ pairs are not pairwise distinct, we first divide them into connected components, and use the number of connected components instead of Q.

C: Triangular Lamps

We simply call integer points with on lamps "black points", and integer points with off lamps "white points". We say two sets of black points are equivalent when one set can be converted to the other by repeating the operations.

The first observation is that we can compress the black points to a single line in a unique way. Let's draw a horizontal line below all black points, and move all points to the line. This is always possible. For example, we can repeat taking the topmost black point, and move it to two points directly below it, until all points reach the line. Suppose that after all black points are moved to the line, the set of their positions becomes P, and by using another way, the set becomes Q. If P and Q are different, P and Q must be equivalent, thus $PXORQ$ (which has black points only on the line) and the empty set must be equivalent. However this is a contradiction, because it's easy to see that no matter how you perform operations, if you start from the empty grid, we can prove that black points remain on at least two rows.

Therefore, we have the following slow solution. Draw a horizontal line and move black points to the line by a simulation. Since it's equivalent to a grid starting from a single black point, the line must match some row of Sierpinski triangle (https://en.wikipedia.org/wiki/Sierpinski_triangle). Recognise its pattern, and you can uniquely determine the initial coordinates of the black point. Note that we can get the coordinates by only the leftmost/rightmost positions of black points on the line.

How to do this without simulations? We can "query" a color of a point on the line in $O(N)$: we can compute the contribution of each point in the input in $O(1)$ because it depends on the parity of number of shortest paths along the grid, and this can be done in $O(1)$ bitmask operations by Lucas's theorem (https://en.wikipedia.org/wiki/Lucas

Suppose that we somehow found one black point (coordinate x) on the line. (Actually this is not trivial. Since Sierpinski triangle is sparse when the coordinates are large, the chance that a randomly chosen point is black is way too small.) Then, in the order $k = 60, 59, \ldots, 0$, we query the color of the point $x - 2^k$, and if it's black, move x to $x - 2^k$. If we analyze the property of Sierpinski triangle, it's easy to show that this way we can always reach the leftmost point. We can do the same to find the rightmost point. In total, this solution works in $O(NlogMAX)$ time.

How to find one black point?

C1.

In this subtask, we are almost done. Just choose the direction of the line carefully - we move black points to the line $x + y = c$ for sufficiently large c. Then, we know that $(c, 0)$ is always black.

C2.

My original idea was C1, but it was found by yosupo during internal testing. Divide the line by modulo 3, i.e. for each $r (0 \leq r < 3)$, count the number of black points on the line whose coordinate x satisfies $x \equiv r \pmod 3$. The key is that we can always find some r that makes this value odd! (By the way, these sums can be computed in $O(NlogMAX)$ time using a simple digit DP.) Thus, we can find some r such that odd number of points with coordinates $r, r + 3, \ldots, r + 3 \cdot (2^k - 1)$ is black, for sufficiently large k. We can divide them into two groups and one of them must contain odd number of black points; and repeat this until we find one black point, like a binary search. This solution works in $O(Nlog^2MAX)$ time.

D: Distinct Boxes

For simplicity, let's allow creating one empty box. Let's do a binary search on the answer K. We want to choose K distinct points from the set $\{(x, y)|x, y \geq 0\}$ and take their sum, and want to get a point with small coordinates.

One way to do this is as follows. Let's take two positive parameters p and q. Then, we take K points (x, y) greedily in the increasing order of $px + qy$, and compute their sum (X, Y). The sum of points chosen this way minimizes the value $pX + qY$, so it is "optimal" in some sense.

Let's plot the points (X, Y) obtained this way, for different parameters of (p, q). We claim that, the answer for the instance (R, B, K) is "Yes" if and only if the point (R, B) lies on the boundary or above the lower convex hull of those points.

Suppose that two vertices $(X1, Y1)$ and $(X2, Y2)$ are adjacent vertices of the convex hull $(X1 < X2, Y1 > Y2)$. Then, we can deduce that the situation around these points must be the following.

When $(p, q) = (Y1 - Y2, X2 - X1 + \varepsilon)$ we get the point $(X1, Y1)$, and when $(p, q) = (Y1 - Y2 + \varepsilon, X2 - X1)$ we get the point $(X2, Y2)$, where ε is a sufficiently small value. When $(p, q) = (Y1 - Y2, X2 - X1)$, there are multiple points with the same value of $px + qy$, and (X, Y) depends on the tie-breaking rule. This is because (X, Y) changes only when the slope of $px + qy = const$ is rational, and its slope corresponds to the slope of the edge of the convex hull.

In order to avoid unnecessary complications let's use some numerical example (it works in the same way in general). Suppose that $p = 3, q = 5$, and we already chose $K - 2$ points with the smallest values of $px + qy$. But four points are tied next: $(2, 10), (7, 7), (12, 4), (17, 1)$, and we need to choose two of them. If we take $(2, 10)$ and $(7, 7)$, we get some point (X, Y). And if we take $(12, 4)$ and $(17, 1)$, we get the point $(X + 20, Y - 12)$. Between them, if we change the tie-breaking rule one by one, we can shift the final point by the vector $(5, -3)$, so we can get all of the points $(X, Y), (X+5, Y-3), (X+10, Y-6), (X+15, Y-9), (X+20, Y-12)$. Furthermore, instead of changing $\{(2, 10), (7, 7)\} \rightarrow \{(2, 10), (12, 4)\}$, let's choose $\{(2, 10), (x, y)\}$ for some (x, y) in the triangle formed by $\{(7, 7), (12, 4), (12, 7)\}$. This way we can get all sums in the triangle region formed by $\{(X, Y), (X + 5, Y - 3), (X + 5, Y)\}$. In a similar way we can prove that all points on the boundary or above the convex hull can be constructed.

It's easier to prove that all points below the hull can't be constructed: that's simply because those points have too small value of $pX + qY$ for some (p, q). Thus, we proved the claim.

We need three nested binary searches to implement this.

- First binary search on the K.
- Inside that, binary search on the angle (i.e., the ratio of p and q). (Some technical note: we can prove that it's sufficient to consider (p, q) that satisfies $p + q = 10^10 + 19$ (or other sufficiently large prime of your choice) because they cover all essentially different angles. This makes the binary search easier.) If we get (X, Y) by this angle, let's divide the plane into four quadrants by $x = X$ and $y = Y$. If (R, B) is in the top-right region the answer is "Yes", if it's in the lower-left region the answer is "No", and in other two cases we continue the binary search.
- Inside that, binary search on the value z, and count points that satisfy $px + qy \leq z$ (We should choose z that makes this number K), and compute the sum of those points. It's possible to do it in logarithmic time, but to make things easier, you can use $O(\min xmax, ymax) \leq 2000$. (In O-notation it's $O(M^{1/3})$, where $M = \max R, B$.

This solution works in $O(M^{1/3} \log^3 M)$, which is fast enough.

E: e

This is a typical situation that happens in Japanese train. One day I wondered what is the ratio of filled sections, when the seats are very long? This value must be somewhere between $\frac{1}{3}$ and $\frac{1}{2}$, but what's the exact value? Then I experimented numerically and got an unexpected result: $\frac{1}{2} - \frac{1}{2e^2}$! How can e appear in this very combinatorial setting? I managed to find an elementary proof of this, and made it a problem. Let's prove the first example.

First, let's rephrase the process a bit. Instead of randomly choosing a comfortable section each time, let's assign a random permutation from 1 to M to the sections. Then, check if a person can take the section assigned 1; takes it if yes; the next person comes to the section assigned 2; takes it if yes; and so on. We can rephrase even more. For each section, let's assign a random real value between 0 and 1, independently from each other. Then, the sections are checked in the increasing order of assigned real number. It's easy to see that this gives the same result to the original problem.

How can we determine if the Section i will be occupied this way? Let x_i be the random value assigned to it. Consider the longest descending runs from i to both directions. That is, find l, r that satisfies $x_{l-1} > x_l < x_{l+1} < \cdots < x_{i-1} < x_i > x_{i+1} > \cdots > x_{r-1} > x_r < x_{r+1}$. In case $l, r \neq i$, Section l will be always occupied, and then $l+2, l+4, \ldots$ will be occupied in this order. Similarly, $r, r-2, r-4, \ldots$ will be occupied in this order. Thus, Section i will be occupied if and only if both $i-l$ and $r-i$ are even, that is, the length of descending runs are even for both directions. This turns out to be true also in the case $l, r = i$.

What is the probability that the length of the descending run to the left is even, given that $x_i = x$? That is,

$$1 - P(x_i > x_{i-1}) + P((x_i > x_{i-1} > xi - 2) - P((x_i > x_{i-1} > xi - 2 > x_{i-3}) + P((x_i > x_{i-1} > xi - 2 > x_{i-3} > x_{i-4}) - \cdots \quad (2)$$

$$= 1 - x + \frac{x^2}{2} - \frac{x^3}{6} + \frac{x^4}{24} - \cdots \quad (3)$$

$$= e^{-x} \quad (4)$$

Here e suddenly appears! Since the value of x_i is already fixed, the event to the left and to the right of it are independent. Thus, the probability that Section i will be occupied is e^{-2x}. The ratio of filled sections is the average of this, that is,

$$\int_0^1 e^{-2x} dx = \frac{1}{2} - \frac{1}{2e^2} \quad (5)$$

And now we explained the first example.

Now, to solve the original problem, we need to convert it to DP. Suppose that the sections are numbered from $-\infty$, and let's compute the probability that sections 0 to $N-1$ matches the given string.

Suppose that we decide the random values from left to right, one by one. We have already decided the random values assigned to sections from $-\infty$ to i. We keep the following values:

- The value i.
- The value assigned to Section i, call it x.
- The parity of the length of the descending run, starting from i and extending to the left.
- If we know the parity of the length of the descending run starting from i and extending to the right, we can uniquely determine the set of occupied sections up to Section i. Additionally we keep two boolean values for that: Do the states up to Section match the given string in case this is even? What if this is odd?

Then this can be regarded as a function of x, so keep its function as $DP[i][boolean1][boolean2][boolean3]$. During the transition from function $f(x)$, we may need functions $\int_0^x f(y)dy$ or $\int_x^1 f(y)dy$. Fortunately, the functions that can appear by the repetitions of these process is quite limited. In particular, the function is always of the following form:

$$P(x) + \frac{Q(x)}{e} + ce^{-x} \tag{6}$$

where P, Q are polynomial with rational coefficients with degree $O(N)$, and c is a rational constant.

This solution works in $O(N)$ time.

F: Paths

Let's say a simple cycle of length 3 is triangle, and a simple cycle of length 4 is quadruple. For each vertex, let's count the numbers of triangles and quadruples that passes the vertex. If you count these numbers, the answer can be easily calculated.

Let v be the vertex with the largest degree. Let's traverse all the vertices whose distance from v is at most 2. It takes $O(\min(D_v^2, M))$ calculations, where D_v is the degree of v. By traversing, you can process all the triangles and quadruples that passes v.

Remove v and continue the above procedure, then you will get answers.

The time complexity is $O(\sum_v \min(D_v^2, M))$. Since $\min(D_v^2, M) \leq D_v\sqrt{M}$, we get $\sum_v \min(D_v^2, M) \leq M\sqrt{M}$.

G: Ranges

Since the queries are offline, we can reduce the problem to the following static problem:

You have a (multi)set S of ranges. You are given only queries with $T_i = 3$. Answer to them.

This reduction can be done by building a segment tree, whose nodes correspond to time intervals, and whose leaves correspond to queries with $T_i = 3$.

Let's solve the static problem. Instead of a range $[l, r]$, we consider a point (l, r). Then the problem is to find the maximum rectangle for each query. Let's do a sweep line algorithm. Let's assume the sweep line is parallel to y-axis, and its x-coodinate is increasing. When the x-coordinate of the sweep line is x, it maintains the function $f_x(y)$ that, given y, returns the upper-left corner of the maximum rectangle for query (x, y). It can be seen that changes of f_x is tractable by some data structure like segment trees. This runs in $O(N'logN')$ time, where N' is the number of points involved in the problem.

The total time complexity is $O(Nlog^2N)$

H: Simple Game

Considering the number of Rng's candies x, an operation in each turn can be written as follows:

$x = \min(S, \max(0, x + B_i))$, where $S = X + Y$, and B_i is A_i if i is odd and $-A_i$ if i is even.

We can consider that Rng initially has S candies, Snuke takes ∞ candies in the (-1)-th turn, and Rng takes X candies in the 0-th turn. Namely, $B_{-1} = -\infty$ and $B_0 = X$.

Now the problem is to process queries to change B_i and S and to find the final state of the game. If we know such i that $\min(S, \max(0, x + B_i)) = 0$ *or* S and $\min(S, \max(0, x + B_j)) = x + B_j$ for all $i < j$, the answer can be easily calculated. This can be done by finding such range $[l, r]$ that $abs(B_l + B_{l+1} + \cdots + B_r) \geq S$ and l is largest. It can be seen that $i = r$ satisfies the above condition. The range $[l, r]$ can be found by a segment tree. The total time complexity is $O(NlogN)$.

1 I: Union 2SAT

Let's build a rooted binary tree representing the merges of silmes. Each node of the tree corresponds to a slime, and children of a node correspond to its original slimes. The root node corresponds to the slime $2M - 1$. We assume that, for each node, the size(number of leaves) of the left child is not smaller than that of the right child. Let's number leaves from 1 through M from left to right. On the root node, we do a binary search to find the maximum R that we can satisfy leaves $1, 2, ..., R$ at the same time. By this imformation, we can determine whether it is possible to satisfy the root node. Moreover, we know satisfiabilities of the left child of the root node, the leftmost grand child,... and so on. Now the remaining taks is to find the satisfiabilities of subtrees whose root is the right child the parent. We can simply calculate them in the same way. Since the size of a right child is not larger than that of a left child, each leaf is involved in a search on a node at most $O(logN)$ times. Since we do a binary search in each root node, the total time complexity is $O(Nlog^2N)$.

If we do an exponential search instead of a binary search and ommit the calculation of satisfiabilities of children of a satisfiable node, we can get $O(NlogN)$ algorithm.