ACM ICPC 2010-2011 NEERC Moscow Subregional Contest

Maxim Akhmedov
Moscow State University, Yandex

January 27th, 2017

1 Problem A. Alien Visit

Let $C_1, C_2, \ldots, C_n$ be the circles in order from left to right. Simply calculate $S_{tot} = S(C_1) + S(C_2 \setminus C_1) + S(C_e \setminus C_2) + \ldots + S(C_n \setminus C_{n-1})$.

2 Problem B. Big Number

First of all, we have to make sure the length of the final number is as large as possible. This means that we are going to remove all strings of lengths smaller than x and several of the strings of length exactly x for some critical length x.

Now one may probe the following greedy solution (as usual, the formal proof is a homework for a curious reader): we need to remove several strings of length x. Perform all removals sequentially, doing each remove greedily, i.e. on each step remove the string that leaves the number as large as possible (to compare different possibilities of removing some substring of length x from the original string, you may use hashing and binary search as a comparison predicate).

3 Problem C. Contest

100 teams always satisfy the conditions of the problem, so just iterate over all possible numbers of teams between 1 and 100 and find the minimum valid number by checking if each of the given percentages in the input may have actually happened.

4 Problem D. Distance

Consider the pair of people that will meet in an optimal answer. Their path looks like the following: A first contestant starts from the vertex a, a second contestant starts from the vertex b, and they meet inside the edge (x, y) in such way that a walked through x and b walked through y. If contestants meet exactly in the vertex, consider edge (x, y) to be the last edge taken by one of the contestants.

Note that a should be closest vertex to x and b should be closest vertex to y, otherwise we could have asked the person that is closer to (let's say) x to go to x, he would be there before a and he could meet the second person earlier.

This means that if we calculate for each vertex of the graph the start vertex that is closest to it (in case of tie, the one with smaller index), we may just iterate over all edges (x, y) after that and make the answer be equal to the minimum $d(a, x) + d(x, y) + d(y, b)$.

In order to find the closest vertices, just run a Dijkstra's algorithm starting from all start vertices simultaneously. The running time of such algorithm is equivalent to the running time of Dijkstra.

5 Problem E. Efficient Cartography

Consider the $A \times B$ grid that contains all the tilings in an optimal answer. Denote the horizontal shift of that grid as dx and the vertical shift as dy.

Fix some dy between 0 and $A - 1$. Consider some stripe. Denote the x-coordinate of a leftmost pixel in that stripe as sl and the x-coordinate of a rightmost pixel as sr. Then for some values of dx it will require $\left\lfloor \frac{sr-sl+1}{B} \right\rfloor$ tiles to cover this stripe, and for the rest of them it will require $\left\lceil \frac{sr-sl+1}{B} \right\rceil$ tiles. Each kind of values corresponds to a contiguous segment of possible dx's modulo B.

Now the solution looks like the following. Calculate $sl_i = \min(l_i, l_{i+1}, \ldots, l_{i+A-1})$, $sr_i = \max(r_i, r_{i+1}, \ldots, r_{i+A-1})$ using the queue with minimum or any logarithmic data structure. Iterate over all possible dy between 0 and $A - 1$, iterate over all $O(M/A)$ possible stripes, for each stripe find out those two segments of dx that correspond to two possible answers for this stripe and perform a modification query on those segments inside some data structure that will contain the $answer_{dx}$ for all dx by the end of our procedure. This data structure should be able to handle the "increase values by Δ on segment $[l, r]$" queries, such a data structure may be a segment tree or just an array that stores adjacent differences of elements (such an array changes in $O(1)$ position after each such query).

The overall complexity is $O(M + N)$.

6 Problem F. Finance

Just read the statement **very** carefully and implement exactly what is written.

7 Problem G. Golden Spire

We need to find out the total surface area of a figure that is a union of several balls with centers on a vertical line. The surface of that union consists of several parts, each of those parts is a spherical segment corresponding to a some of the spheres.

A great fact to know is that the surface area of a spherical segment is $2\pi rh$ where the h is the height of a spherical segment, i.e. it does not depend on the exact form of a spherical segment.

Hence, all we need to know is the set of balls that are present on the surface (some of the balls might be too small and be hidden by the rest of the balls) and the heights of their intersection circles.

That is actually a 2-dimensional problem a bit similar to the problem A. It may be done in a linear time using the stack. Sort all circles by the center height. Keep the stack of all pairs (circle, one of its intersection points with previous circle). Now when a new circle appears, it removes some of the previous circles by covering them completely, pop those circles until the top circle intersection point with a previous is uncovered by the new circle. After that, add the new circle to the stack.

Calculate the answer depending on which circles are left in the stack. The running time of the algorithm is linear.

8 Problem H. Hometask

- $n^2 \mid n! \Leftrightarrow n \mid (n-1)!$;
- if n is prime, $(n-1)!$ does not contain any term that is divisible by n, thus $n \nmid (n-1)!$;
- if $n = ab$ where $1 < a, b < n$ and $a \neq b$, then $n \mid (n-1)!$ since both a and b appear in $(n-1)!$;
- n allows an expression of the form above if n contains at least three primes (not necessarily different) in its representation; thus the last remaining cases are $n = p^2$ and $n = 1$;
- if $n = 1$, $n \mid (n-1)!$;
- if $n = p^2$ where $p > 2$, p and $2p$ appear among $(n-1)!$, that's why $n \mid (n-1)!$;
- if $n = 2^2 =$, $n \nmid (n-1)!$.

9 Problem I. Interest Targeting

Do exactly what is written in the statement.

10 Problem J. Joke

The ways to get WA:

- divide in doubles and then round: double stores only about 15 significant digits, while this problem requires at least $2 \cdot 9 = 18$;
- divide in doubles and add a random epsilon: no way that can work :);
- use Python but still operate with floats: floats in Python are not of arbitrary precision, they are exactly the same as doubles in C++/Java;
- use long doubles but do not add epsilong: it may result that x divides by y producing .5 as a fractional part, but the long double representation of x/y will actually be $x/y - \varepsilon$ leading the round function round towards the smaller value.

The ways to get OK:

- write an integral solution that multiplies both number by 10^9 and stores both of them in a 64-bit integral type;
- write a solution using long doubles in C++ and making a proper rounding with "$+\varepsilon$";
- write a solution in Python that uses `fractions`/`decimal` module.

An interesting fact: there are 89 tests in this problem. For 28 of them at least one team has produced WA on that test. Indices of those tests are 1, 2, 4, 7, 8, 9, 10, 11, 12, 13, 15, 16, 24, 25, 28, 29, 30, 34, 42, 43, 49, 50, 53, 54, 66, 72, 76, 79.

11 Problem K. KMC Attacks

Consider some moment. By this moment unit is shifted from its original position by some vector (dx, dy) that is equal to the sum of all its movements up to this moment. Also we know, that during its movement the maximum value of its shift in horizontal direction was r, the maximum value of shit in vertical direction was u, the minimum value of its shift in horizontal direction was l and the minimum value of its shift in vertical direction was d. All these values may be easily updated when a new move movement happens.

The solution is to fix each possible starting position (sx, sy) of the unit and then ask if the unit now stays at $(sx + dx, sy + dy)$. We are not allowed to ask about the positions such that they can not contain the unit according to the information we have, so we skip a query if $sx + r > M$, $sy + d > N$, $sx + l < 1$ or $sy + u < 1$ (this includes the case when $(sx + dx, sy + dy)$ is outside the field).

12 Problem L. Lanes

Denote the old amounts of people on the lanes as x_i and new ones as the x'_i. Let $s_1 = x_1$, $s_2 = x_1 + x_2$, ..., $s_n = x_1 + \ldots + x_n$ and $s'_1 = x'_1$, $s'_2 = x'_1 + x'_2$, ..., $s'_n = x'_1 + \ldots + x'_n$.

The number of lane changes may be calculated as the sum of crossings over each barrier between the adjacent lanes. If previously there were s_i people to the left of the barrier between i-th lane and $(i + 1)$-st lane and $s_n - s_i$ people to the right of it, and now there are s'_i people to the left and $s'_n - s'_i = s_n - s'_i$, then exactly $|s_i - s'_i|$ people should have crossed the barrier between the lanes in one of either directions depending on which of s_i and s'_i is larger.

This leads us to the following dynamic programming solution: $D[i][j]$ = minimum number of lane barrier crossings on all lanes up to i-th if $s'_i = j$. All transitions from this state of DP are defined by the number k of people on an $(i + 1)$-st lane, i.e. this DP has the complexity of $O(n \cdot S \cdot S)$ that is too much (at least was too much back in 2010).

The final idea is that actually there are no more then $O(\sqrt{S})$ meaningful values of k for any pair (i, j). Indeed, if there exist two x'_a and x'_b such that $|x'_a - x'_b| = d$ then the sum of all x' between a-th and b-th is at least $0 + 1 + 2 + \ldots + d = O(d^2)$ since all numbers between x'_a and x'_b should be present and their sum is at least the sum of all integers between 0 and d. Hence, d is bounded by a constant $C = O(\sqrt{S})$. Note that at least one x'_i should be equal to the average value $\lfloor s_n/n \rfloor$ meaning that k should belong to the range of $\lfloor s_n/n \rfloor - C$ to the $\lfloor s_n/n \rfloor + C$.

So, the complexity of solution is reduced to $O(nS^{1.5})$.