Day 2 Contest Analysis, Division AB

July 8, 2019

Filipp Rukhovich

Discover Vladivostok 2019

A. Cleaning Robot

Given a grid $w \times h$, $1 \leqslant w, h \leqslant 20$, and a robot inside it. The robot is able to execute commands 'f'(move forward if possible), 'l'(turn left), 'r'(turn right), and each command depends on previous 10 commands written in memory. We can set the dependence in such a way that independently of initial cell, direction of facing and state of memory, the robot will visit each cell at least once.

A. Cleaning Robot

Let $code(x)$, $0 \leqslant x < 128$, is 7-bit binary representation of x with symbol 'l' instead of 0 and 'r' instead of 1.

If robot execute $code(x)$ from some x as sequence of commands, then robot will turn 90 degrees left or right (because 7 is odd). Then, there is a command $cmps(x)$ which is 'l' or 'r' and executing of $code(x)$ and then $cmps(x)$ is equivalent to executing nothing.

The idea of the solution is that robot should execute some fixed sequence S of commands cyclically. This sequence will be a concatenation of strings $code(0)$, $cmps(0)$, $comm(0)$, $code(1)$, $cmps(1)$, $comm(1)$ ..., $code(z-1)$, $cmps(z-1)$, $comm(z-1)$, $z \leqslant 128$. Here, to each $comm(x)$ we assign 'f', 'ff' or 'lf'.

A. Cleaning Robot

By the construction, each command of S can be defined uniquely from previous ten commands (prove it!); so we can describe S in terms of robot program. Of course, it doesn't cover some of 3^{10} instructions in the program; but it's easy to define it in such a way that starting from any memory state, no more than 10 steps later memory state will be substring of S (for example, take a biggest suffix of state which is a substring of S and assign a command as symbol after the occurrence in S).

So, without loss of generality, we can assume that we start from arbitrary cell and facing direction and perform operation of S cyclically.

In fact, commands in $code(x)$ and $cmps(x)$, $0 \leqslant x < 128$ can be removed from S without changing of route of robot; so, we are to find a cycle of 128 or less commands of type 'f', 'lf' or 'ff' in such a way that repeating them cyclically makes robot visiting all cells.

A. Cleaning Robot

Such a sequence C can be calculated in the following way:

- suppose that at the beginning, robot occurs in down left corner, and direction of its facing is down;
- robot goes over borders of squares with corner in down left corner of the field and side length 2, 3, 4, . . . , 10;
- after that, robot goes from left down corner straight to right down corner and stop without rotations.

Obviously, during the second iteration of C robot visit all vertices in 10×10 square with corner in right down corner of the field and stop into right up corner; so, during four iterations, robot visits the whole table. But it is in assumption that robot starts from the left down cell and facing into the down wall. But it can be easily seen that after no more than 4 iteration of C, robot will be in needed state.

A. Cleaning Robot

The last question is: how many commands do we need to go over such a borders?

Obviously, if we want to go over a border of square with length l, we need $4 * \lceil \frac{l}{2} \rceil$. Then, the whole C can consist of $4 * (1 + 2 + 2 + 3 + 3 + 4 + 4 + 5 + 5) + 10 = 126 < 128$. It means that our solution is absolutely correct.

H. Rolling a Dice

In this problem, we are to roll a dice from lower left corner of $a \times b$ grid to upper right one and find the sum of numbers imprinted on the plane.

Before solving a problem mathematically, discuss technical details:

- enumerate faces of dice from one to six in such a way that at initial position, front face is 1, left is 2 etc. Number of face is NOT connected with number written on it;
- let's describe any sequence of rolling up and right as a permutation of faces and numbers $c_1, c_2, \ldots, c_6$; c_i will be number of times face number i is imprinted on the table during the performing of the sequence in assumption that we started making the sequence from initial position;
- if we have such a descriptions for sequences A and B then in constant time we can get a description for concatenation AB of sequences;

H. Rolling a Dice

- if we have such a description for sequence A then we can find an interpretation of concatenation of n sequences equal to A for any natural integer n in constant time due to low (no more that 24) period of permutations we work with;
- for any such a description, we can find a description of inverse transformation.

Having such an infrastructure, we can solve the problem using some mathematics. What should it be?

H. Rolling a Dice

Formulate the process in the following way. Suppose that $a \leqslant b$, and b is a horizontal size of rectangle. We know that the diagonal line connecting $(0, 0)$ with (b, a) intersect vertical line $x = i$ in the point $(i, \frac{a}{b}i)$. Then, for $i = 0, 1, \ldots, b - 1$ we should do the following:

- if $\lfloor \frac{a}{b}i \rfloor < \lfloor \frac{a}{b}(i + 1) \rfloor$ then we should roll dice up and then right;
- otherwise we should just rool dice right.

After that, last two rolls should be cancelled.

H. Rolling a Dice

Note that $\lfloor \frac{a}{b} i \rfloor < \lfloor \frac{a}{b}(i+1) \rfloor$ if and only if $(ai)\%b + a \geqslant b$; it give us a possibility to reformulate the problem in the following way:

- let r_b be a zero;
- for $i = 0, 1, \ldots, b-1$ do the following:
  - if $r_b + a \geqslant b$ then roll the dice up;
  - roll the dice right;
  - $r_b := (r_b + a)\%b$.
- At the end, cancel last two actions.

So, we've described a process in terms of moving "from vertical to vertical".

H. Rolling a Dice

Then, try to describe the process in terms of moving "from horizontal to horizontal". For $j = 0, 1, \ldots, a-1$, do the following:

- roll dice right d_j times, $d_j = \lfloor \frac{b}{a}(j+1) \rfloor - \lfloor \frac{b}{a}(j) \rfloor$;
- roll dice up.

After that cancel two last rolls (they will differ from those from "vertical" analysis).

H. Rolling a Dice

Note that $d_j = \lfloor \frac{b}{a} \rfloor$ if $(bj)\%a < (b(j+1))\%a$ and $d_j = \lfloor \frac{b}{a} + 1 \rfloor$ otherwise.

It gives us the following reformulation of the algorithm:

- let $r = b\%a$, and $r_a = 0$;
- for $j = 0, 1, \ldots, a-1$ do the following:
  - if $r_a + r \geqslant a$ then roll the dice right;
  - roll the dice right $\lfloor \frac{b}{a} \rfloor$ times and then up at once;
  - $r_a := (r_a + r)\%a$.
- At the end, cancel last two actions.

H. Rolling a Dice

As we can see, reformulation of second method is very similar to reformulation of first one and differs only in the following details:

- instead of numbers (a, b), we work with $(b\%a, a)$;
- rolling up is replaced by rolling right;
- rolling right is replaced by some fixed sequence of operations;
- r_b is renamed by r_a :)

These observations give us a possibility to implement some recursive function $f(a, b, ACTION\_1, ACTION\_2)$ and solve the problem with complexity of Euclidean algorithm - $O(log\ n)$.

C. Where Do Identity Permutations Come From

Given a permutation P of length n. We can perform some number of swaps of adjacent elements; after performing swaps, we can replace any element of the permutation to its index. How many operations do we need to perform to transform given permutation to the identity?

C. Where Do Identity Permutations Come From

Consider an optimal sequence of swaps and replacings. Connect two adjacent positions in the permutation iff there was at least one swap of elements on these positions.

Then, consider a component of connection with size 2 or more; let it be a subsegment $[l, r]$, $r > l$. This subsegment contains $r - l + 1$ element; so, there were at least $r - l$ swaps inside it.

If there were more than $r - l$ swaps then we can remove all these swap from the sequence; instead of that, we add $r - l$ or less replacings of elements in $P[l, r]$, and it doesn't make the sequence longer. So, we may assume that only subsegment with $r - l$ inverses can be interesting for optimal sequence. Moreover, after these swaps all numbers in $P[l, r]$ should be equal to its index (otherwise we should do at least one replacing so make replacings without swaps is optimal again).

C. Where Do Identity Permutations Come From

Written above implies that $P[l, r]$ can be component of connection if and only if $P[l, r]$ contains exactly numbers from l to r and has $r - l$ inversions (because exactly x swaps are needed to perform a permutation with x inversions into identity). Conditions about numbers is satisfied if and only if the following conditions are satisfied:

1. $\sum_{i=l}^{r}(P[i] - i) = 0$;
2. $\max_{i=l}^{r} P[i] \geqslant r$.

Denote segments $[l, r]$ satisfying these two conditions *good*.

Let $b[i] = P[i] - i$, $i = 1, 2, \ldots, n$, and $partb[0..n]$ is an array of partial sums of b. Also let $pb[i]$, $1 \leqslant i \leqslant n$ is equal to maximal possible $j < i$ such that $P[j] > P[i]$ or 0 if there are no such a j. Arrays b, $partb$ and pb can be calculated in $O(n)$ time.

C. Where Do Identity Permutations Come From

Then, segment $[l, r]$ is good if and only if $partb[l-1] = partb[r]$, and $pb[r] < l$. Fix some r, and let l_r be a maximal possible value $l \leqslant r$ such that $partb[l-1] = partb[r]$ if such an l exist.

Lemma 1: only segments $[l_r, r]$, $1 \leqslant r \leqslant n$, can be component of connection.

Prove Lemma 1. Suppose that for some r, there exists $l < l_r$ such that $[l, r]$ is component of connection. Obviously, then both $[l, l_r - 1]$ and $[l_r, r]$ are good segments, so swaps in $[l, r]$ can be done independently in $[l, l_r - 1]$ and $[l_r, r]$; it gives a contradiction with connectivity of $[l, r]$.

C. Where Do Identity Permutations Come From

All l_r-s can be found in $O(n)$ time using std::unordered_map. Create an array $a[1..n]$ and segment tree over it and fill a by zeroes. Iterate over $i = 1, 2, \ldots, n$. For each i, increment $a[P[i]$; then, let $dp[i]$, $0 \leqslant i \leqslant n$, be minimum number of operations needed to transform $P[1..i]$ to identity. The $dp[i]$ can be equal to $1 + dp[i-1]$ or $i - l_i + dp[l_i - 1]$ if l_i exists and $P[l_i, i]$ contains exactly $i - l_i$ inversions. If $Inv[j]$, $0 \leqslant j \leqslant n$, is a number of inversions of $P[1..j]$ then number of inversions in $P[l_i, i]$ can be found in $O(log\ n)$ as
$Inv[i] - Inv[l_i - 1] - (i - l_i + 1) * (a[l_i + 1] + a[l_i + 2] + \ldots + a[i])$.

So, the answer $dp[n]$ can be found in $O(n\ log\ n)$ time.

E. Partition

Given a connected figure consisting of n cells on rectangular grid. We are to erase no more than $\lceil \frac{3}{2}\sqrt{n} \rceil$ cells in such a way that all each of the remaining connected components contains no more than $\lceil \frac{n}{2} \rceil$ cells.

The solution will be probabilistic. Choose a random angle α from 0 to π and fix a direction with polar angle α. Then, sort all cells by its projection on the direction orthogonal to fixed one; then, take $\frac{n}{2}$-th cell in this order and draw a line l through the center of it. We should remove exactly cells which are intersected by line l.

To make the solution more reliable, repeat it by some number of times and choose more optimal one.

F. Matrix Consistency

Given an $n \times n$-matrix A over the field $\mathbb{Z}_2$. Let $x, y \in \mathbb{Z}_2^n$ are uniformly random row and column vector. What is the probability that $xAy = 1$?

If xA is zero vector then $xAy = 0$ with probability 1. Otherwise, such a probability is $\frac{1}{2}$ (if, for example, first coordinate of xA is 1, then all y-s can be divided into pairs of vectors with the only difference in first position; for each such pair (y', y''), $xAy' \neq xAy''$).

So, the answer is $(1 - p)/2$ for p as probability of the event that xA is a zero vector. It's known from linear algebra that such a probability is $1/2^{rkA}$. rkA can be found in $O(n^3/32)$ by Gauss algorithm; but note that if we did at least ~~TAMBOURINE~~ BUBEN $= 40$ iterations in Gauss algorithm then $rkA \geqslant BUBEN$, and then answer is 0.5 with very good precision, so we can stop the Gauss.

The complexity of the solution is $O(BUBEN * n^2/32)$.

G. MST of Random Points

We are to build a minimal spanning tree on Euclidean distance graph with n random vertices.

As in problem E, choose some random direction and sort points by projection of point to this direction. Then, for each point p_i (with numeration in sorted order), choose *BUBEN* (about 12-15) closest points among points p_j, s.t. $j > i$. We do it in straightforward way bit if distance between projection of p_i and current p_j is more than distances from p_i to current closest *BUBEN* points then we can stop the process.

After that, we connect p_i with these *BUBEN* closest points. Having done this for all points, just run Kruskal's algorithm with DSU.

I. Statistics

There are n different objects and a game which consist of an infinite number of rounds. During each round, each object is touched with probability p.

What is the expected number of touched object at the moment after the first round after which first object was touched exactly k times?

Let $f(k)$ be a probability that second (or 3rd, or 4th, or etc) object will be touched at the moment described in the statement. Due to linearity of mathematical expectation, answer to the problem is $1 + (n - 1) * f(k)$ (first object was touched with probability 1).

I. Statistics

Fact 1: $f(k) = 1 - (1 - f(1))^k$ (due to the fact that second object should not be touched before first touch of first object, the before second, then before 3rd etc).

Fact 2: $f(1) = (1 - p) * (p + (1 - p) * f(1))$ (just considering of events that second object is not touch in first round, and first is touched or not).

From equation in fact 2 one can find $f(1)$ and then calculate $f(k)$ and answer.

D. Secret Community Card Game

Given two rectangle $n \times m$ - tables A and B of 'x' and '.'; A can also contain '?'. If number of 'x' in rows of B is not non-increasing function then we are to print "NO". Otherwise we are to replace all '?' and then reorder rows of A in such a way that A becomes equal to B.

To do that, consider rows of A and B as vertices of bipartite graph; row of A should be connected with row of B iff first row can be transformed into second one. It means that we has reduced the problem to finding a perfect matching in the built graph which can be done in $O(n^3)$.

B. Tree Coloring

Given a rooted tree with n vertices. The root of the tree has exactly three children, and each non-root-or-leaf-vertex has exactly two children. Each leaf is colored in one of three colors. We are to paint all non-leaves in three colors in such a way that adjacent vertices are colored differently or determine that there is no such coloring.

Let $dp[v][c]$, $1 \leqslant v \leqslant n$, $1 \leqslant c \leqslant 3$, is a boolean value equal to true if and only if we can color v and its subtree in such a way that v is colored into color c. Such a $dp[v][c]$ is true iff for any child v' of v there exist a color $c' = c'(v')$, $c' \neq c$ that $dp[v'][c']$ is true. It means that all dp-s can be calculated in $O(n)$ using simple DFS. By second DFS, we can restore the colors in $O(n)$ too.