The 2017 Moroccan Collegiate Programming Contest

Problem Analysis

October, 2018

MCPC 2017 was held in October 2017 at Université Internationale de Rabat. The problem set contained 12 problems with varying difficulties and topics. The solutions or ideas proposed here are assumed to be detailed enough to enable the reader to solve the problem by merely doing an implementation effort. Please feel free to contact us if you find an error or for any clarification.

Overview

Problem	A	B	C	D	E	F	G	H	I	J	K	L
Attempted	30	217	1	0	112	85	203	1	10	183	17	224
Solved	7	68	0	0	21	10	68	1	2	62	0	2

Problem A. Young Adleman

Let $g_2(p)$ denote the number of occurences of character p in S_2 and $g_1'(p, s, e)$ the number of occurences of character p in the subsegment of S_1 starting at s and ending at e. The gist of this problem is to keep two pointers, which point to the first and last elements of a subsegment of S_1 that contains all the letters in S_2 at least once.

First, we find an endpoint e such that the substring $S_1(0, e)$ satisfies the desired condition. If e does not exist, there is no solution. Then we find a maximal s such that $S_1(s, e)$ satisfies the condition. From now on, increment the pointer e, and everytime, make sure you advance s to its maximal possible position (*i.e* the position such that $g_1'(S_1[s], s, e) = g_2(S_1[s])$), so that it is a potential candidate. The substrings generated by this process are *some* of the substrings containing all the letters in S_2 at least once, so each time we generate a new string, we keep track of its size and we chose the smallest one.

Complexity: $O(N_1 + N_2)$.

Problem B. Self Describing Numbers

This was probably the easiest problem in the problem set. All you need to do is count the occurences of every digit in the entry number (seen as a string) and check if the conditions of self-description are satisfied.

Complexity: $O(\log(N))$.

Problem C. Numberica

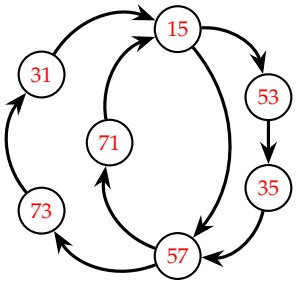
The first thing that should be blindingly obvious in this problem is that it should be modelized as a graph. Of course, you still need some other neat observations to solve it.

At first glance, one could think of the integers in the entry sequence A as vertices of a directed graph and this way, the task would be to find a hamiltonian path in this graph. As you certainly know, the problem is NP-complete and given the constraints, that certainly will not do it. We need to see the entry sequence differently and build an easier graph to treat.

Let's consider the directed graph $G = (V, E)$ defined as follows :

$$V = \left\{ x \in [\![10^{D-1}, 10^D - 1]\!] \mid \exists s \in [\![1, n]\!] / (s \times 10^{D-1} + x) \in A \right\}$$

$$E = \{(u, v) \in V \mid (u \times 10 + (v \mod 10)) \in A\}$$



Practically, the graph for the last sample entry list would look like the one on the right. It is easy to show that the entry list A can be reordered in a way that it is connected, if and only if the directed graph we defined above has an Eulerian directed path (take some time to think about it). Now one needs to check if the graph associated to the list A is Eulerian, which is done in linear time using the well-known characterization :

- At most one vertex has (out-degree) − (in-degree) = 1, at most one vertex has (in-degree) − (out-degree) = 1, every other vertex has equal in-degree and out-degree.
- All of the graph's vertices with nonzero degree belong to a single connected component.

The last step is to run a graph traversal to store the eulerian trail; a DFS with backtracking should do it.

Complexity: $O(N \log(N))$.

Problem D. Cycles in a String

The graph cycles as described in the statement are disjoint, which is why one should work on them separately, that is, for every cycle, find the lexicographically smallest subsequence and the corresponding number of moves to make the transformation (*e.g* by running a DFS). Therefore, the total cost would be the sum of the costs and the wanted final string would be the combination of the ones found for each cycle. Alright, now how do we find the answer for a given cycle ?

To do this, we are going to need a classic algorithm. But let's first see why we need it. For a given cycle, what we want is to find the lexicographically smallest substring by doing a number of "rotations". In other terms, denoting by S_i the string associated to the current cycle, we have to find the lexicographically smallest substring of length $|S_i|$ in the string $S_i \cdot S_i$, and we need to do it efficiently. In order to do that, we need to build a *suffix array* and traverse it, evaluating LCPs. If you're not familiar with these notions, there are a bunch of ressources online where you can practice them, following which, this problem becomes just a matter of implementation.

Complexity: $O(N \log^2(N))$.

Problem E. Donald's Riddle

This is a simple problem which can be solved using a greedy approach. It is enough to iterate through all the indices and, for each index i, transform $S[i]$ into $T[i]$, or $T[i]$ into $S[i]$ with the minimum cost possible.
Given two letters p and q such that $p \neq q$ (case-insensitive), the minimum cost to transform p to q is :

$$\begin{cases} \min(b, d+a+c) & \text{if } u(p) \text{ and } u(q) \\ \min(a, c+b+d) & \text{if } l(p) \text{ and } l(q) \\ b+d & \text{if } u(p) \text{ and } l(q) \\ c+b & \text{if } l(p) \text{ and } u(q) \end{cases}$$

where $u(p)$ indicates that p is an uppercase letter and $l(p)$ that it is a lowercase letter.

The above formulas convey the idea that in order to make p and q equal, one can apply one or two different sequences of operations, and thus should retain the minimum of the associated costs. Besides, the minimum cost to transform p to q if they represent the same letter but have a different case is c if $l(p)$ and d if $u(p)$.

NB: *The resulting total cost may not fit in 32 bits.*

Complexity: $O(N)$.

Problem F. Sort Segments

This is a straightforward data-structure problem. The important thing to notice here is that all you actually need is the minimum of every segment $[i, i+M]$. In fact, the final array will look like this : $[u_1, u_2, \cdots, u_m, ...]$ where u_i is the minimum over segment $[i, i+M]$ and the remaining elements are sorted. So, what we need is a way to find these minimums efficiently. It turns out that *heaps* are the best match for that. We iterate through the starts of the aforementioned segments and keep track of their minimums. We discard all the found minimums until we reach the index k; the corresponding minimum is the answer.

Complexity: $O(N \log(N))$.

Problem G. Geometry is Beautiful

The volume of a cuboid with side lengths u, v and w is $u \times v \times w$. Therefore, the volume of a cuboid with a square face of side-length a and a depth d is $a^2 d$. The volumes of the three cuboids are then $a^2 d$, $b^2 d$ and $(a^2 + b^2)d$ using the Pythagorian theorem. These volumes remain to be sorted.

Complexity: $O(1)$.

Problem H. Point on a Circle

We can solve this problem using dynamic programming. Let's define $f(i, d, p, c)$ as the minimal number of moves to make the difference between positive cycles and negative cycles equal to d in the substring $S[1, i]$ when the current cycle's position is $p \in \{E, S, W, N\}$ and we have made $-4 < c < 4$ steps towards completing the cycle. When $c = 0$, the current position is N and we are starting a new one). For convenience, let's map E, S, W and N to $1, 2, 3$ and 0, respectively. Then we can make transitions as follows :

For every $0 \leq p' \leq 3$, putting $s = 0$ if $p' = S[i+1]$ and 1 otherwise :

- if $|p' - p| \equiv 0 \mod 2$, $f(i+1, d, p, c) = s + f(i, d, p, c)$, meaning that we ignore the next direction.
- if $p' = (p+1 \mod 4)$, $f(i+1, d+\delta, p', (c+1) \mod 4) = s + f(i, d, p, c)$, meaning that we move in clockwise direction.
- if $p = (p'+1 \mod 4)$, $f(i+1, d-\delta, p', (c-1) \mod 4) = s + f(i, d, p, c)$, meaning that we move in counterclockwise direction.

where $\delta = 0$ if $c+1 < 4$, $\delta = 1$ if $c+1 = 4$, $\delta = -1$ if $c-1 = -4$.

Initializing f with $f(0,0,0,0) = 0$, the answer is therefore $\min_{p,c} f(n,k,p,c)$.

Complexity: $O(N^2)$.

Problem I. Laser Gun

This problem is about probabilities and computing an expected value. Uncommonly for this topic, the approach to solve this problem is quite straight-forward, which is why it can be considered an easy one.

Let X_k be the random variable denoting the number of laser beam reflections before it deviates from the mirrors range, when the laser gun is positionned at the k^{th} green point. We first need to compute $P(X_k = d)$ for all possible values of d. When a gun is at the k^{th} green point, the first beam will either hit the opposite mirror at a point in the range $[\![1, k-1]\!]$ or $[\![n-k, n]\!]$. Hence,

$$P(X_k = d) = \left(\frac{k-1}{n-1}\right) P(X_k = d | A_{<k}) + \left(\frac{n-k}{n-1}\right) P(X_k = d | A_{>k})$$

where $A_{<k}$ (resp. $A_{>k}$) is the event that the first beam hits the lower (resp. higher) part of the opposite mirror. Now, $(k-1)P(X_k = d|A_{<k}) = \left\lfloor \frac{k-1}{d} \right\rfloor - \left\lfloor \frac{k-1}{d+1} \right\rfloor$ and $(n-k)P(X_k = d|A_{>k}) = \left\lfloor \frac{n-k}{d} \right\rfloor - \left\lfloor \frac{n-k}{d+1} \right\rfloor$, so

$$P(X_k = d) = \frac{1}{n-1}\left(\left\lfloor \frac{k-1}{d} \right\rfloor - \left\lfloor \frac{k-1}{d+1} \right\rfloor + \left\lfloor \frac{n-k}{d} \right\rfloor - \left\lfloor \frac{n-k}{d+1} \right\rfloor\right)$$

and finally,

$$\mathrm{E}[X_k] = \frac{1}{n-1} \sum_{d=1}^{\max(k-1, n-k)} d\left(\left\lfloor \frac{k-1}{d} \right\rfloor - \left\lfloor \frac{k-1}{d+1} \right\rfloor + \left\lfloor \frac{n-k}{d} \right\rfloor - \left\lfloor \frac{n-k}{d+1} \right\rfloor\right) = \frac{f(k-1) + f(n-k)}{n-1},$$

putting $f : n \longmapsto \sum_{i=1}^{n} i\left(\left\lfloor \frac{n}{i} \right\rfloor - \left\lfloor \frac{n}{i+1} \right\rfloor\right) = \sum_{i=1}^{n} \left\lfloor \frac{n}{i} \right\rfloor$.

Now it all comes to evaluating f efficiently. This is quite classic but let's show how this is done anyway. First, putting $D = \left\{ \left\lfloor \frac{n}{i} \right\rfloor \ / \ 1 \leq i \leq n \right\}$, notice that $|D| \leq 2\left\lfloor \sqrt{n} \right\rfloor$. So it suffices to assemble the values of D and evaluate their contribution to the above sum. This way, the sum would only contain around $O(\sqrt{n})$ terms. Formally, we can rewrite $f(n)$ as

$$\begin{aligned} f(n) &= \sum_{p \in D} \sum_{\substack{i=1 \\ \lfloor \frac{n}{i} \rfloor = p}}^{n} i = \sum_{p \in D} p \sum_{\substack{i=1 \\ \lfloor \frac{n}{i} \rfloor = p}}^{n} 1 = \sum_{p \in D} p \sum_{i=\left\lfloor \frac{n}{p+1} \right\rfloor + 1}^{\left\lfloor \frac{n}{p} \right\rfloor} 1 \\ &= \sum_{p \in D} p\left(\left\lfloor \frac{n}{p} \right\rfloor - \left\lfloor \frac{n}{p+1} \right\rfloor\right) \end{aligned}$$

and we are done.

NB: *The above method takes $O(\sqrt{n}\log(n))$ which is enough for this problem, but we can do better by noticing that*

$$f(n) = 2f\left(\left\lfloor \sqrt{n} \right\rfloor\right) - \left\lfloor \sqrt{n} \right\rfloor^2$$

An alternative approach is instead of solving the problem for (N, k) (shooting from point k to one of the $N-1$ points), we can solve the problem for $(k, 1)$ + $(N-k+1, 1)$ (either the laser gun shoots to the right, to $i < k$ or to the left, to $i > k$).
The new problem is the following : the gun is at position 1 and will shoot to some point i, $1 < i \leq x$. The average number of bounces is then : $\frac{1}{x-1} \cdot \sum_{i=1}^{x} \left\lfloor \frac{x-1}{i} \right\rfloor$

Complexity: $O(\sqrt{N}\log(N))$ or $O(\sqrt{N})$.

Problem J. Descartes Coordinates

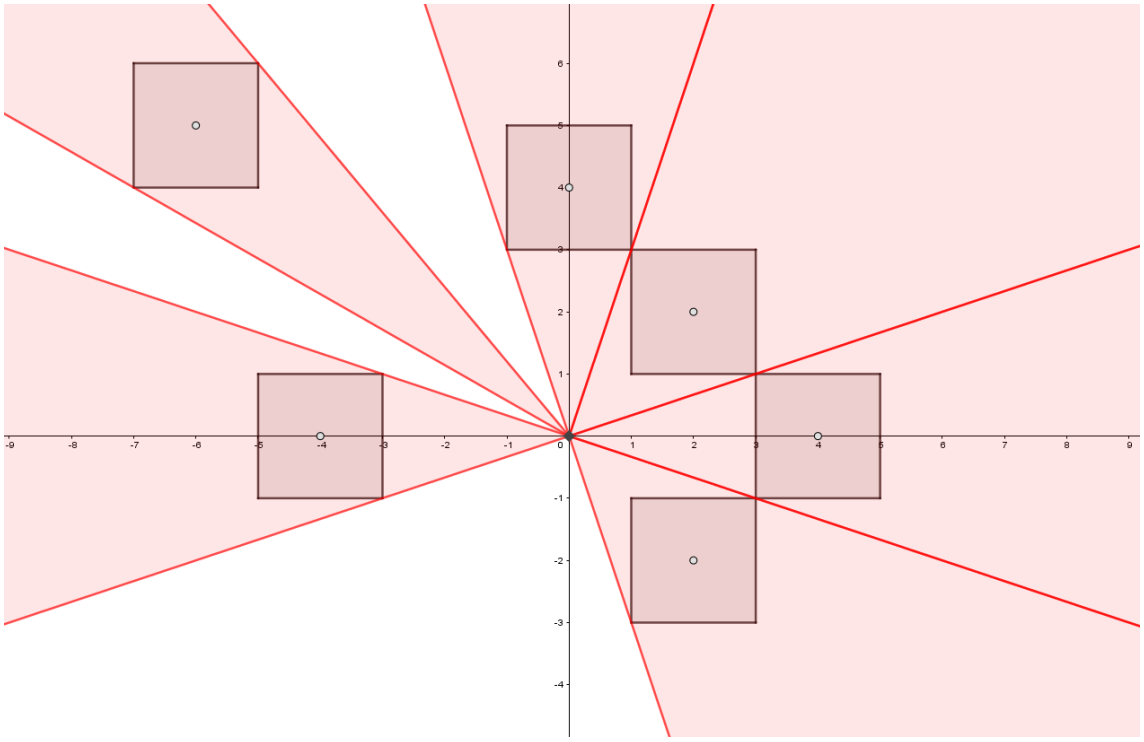
Since an intersection can only happen between non-parallel lines, the intersections to consider are those between the $N-K$ non pairwise parallel lines on the one hand (say I_1), and on the other hand, the intersections between the K parallel lines and these $N-K$ non-parallel lines (say I_2). It is easy to see that $I_1 = \frac{(N-K)(N-K-1)}{2}$ and $I_2 = K \times (N-K)$.

NB: *The resulting total cost may not fit in 32 bits.*

Complexity: $O(1)$.

Problem K. IBM Balloons

Let's binary search the minimum time. We set the upper bound of the search to be *e.g.* 10^7, as the answer won't exceed it. In order to binary search the answer, we need to be able to check every time wether the vision is blocked or not. One way to do it is by computing, for a given value of the elapsed time t, the angle range covered by each balloon. One then iterates over the angles to verify their *connectedness*.



At time t, the side of a balloon i with origin (x_i, y_i) has length rt. After treating all the possible situations for a square, we get $(u_{i,1}, v_{i,1})$ and $(u_{i,2}, v_{i,2})$ the respective coordinates of the corners marking out the range that the balloon blocks, then, again, handling the different configurations, we can get the angle range for balloon i. The angle's main term is $\pm \arcsin\left(\frac{|v_{i,1}|}{v_{i,1}^2+u_{i,1}^2}\right)$ adding eventually π or 2π.

Complexity: $O(N \log(\epsilon^{-1}))$.

Problem L. Vowely Pairs

For every string s_j we encounter, we need to find $d(s_j)$, the number of distinct strings s_i such that $i < j$ and $|s_i|_v = |s_j|_v$ where $|s|_v$ denotes the number of vowels in s. In order to do that, we keep track of $w_j(p) = \#\{s_i \,/\, |s_i|_v = p \wedge i < j\}$ for every $p \in [\![1, 26]\!]$ and we update $d(s_j)$ for every $j \in [\![1, n]\!]$ in the following way :

- At index j, if it is the first occurence of s_j we encounter, set $d(s_j)$ to be equal to $w_j(p)$ then increment $w_j(p)$ (so that $w_{j+1}(p) = w_j(p) + 1$).
- otherwise, just set $d(s_j)$ to be equal to $w_j(p)$.

Denoting by P the set of the entry strings, the answer to the problem is

$$\sum_{s \in P} d(s)$$

Complexity: $O(N)$.