Editorial

In this contest the problems were more or less sorted by difficulty. It is sometimes hard to compare problems because difficulty does not mean the same for everyone.

It seems the contest was a bit too hard as the best participant solved 3 problems. I will try to reduce the difficulty for subsequent rounds. Again, it is not always easy to measure difficulty.

The goal of an editorial is not give the complete details of the solution by rather to sketch the main ideas. We encourage you to try to solve the problems and fill in the details on your own after reading this.

You can already submit here: `http://domjudge.info.ucl.ac.be`.

Problem A

You are given a grid and you need to know if there is a path of length T between two points of the grid. Let d be the Manhattan distance between the two points.

Clearly if $T < d$ then it is impossible as we cannot even reach the target. This was a detail that some of you forgot and thus got wrong answer.

If $T \geq d$ we need to be able to waste the remaining $T - d$ steps. Every step that goes away from the target requires another step to get closer again since we need to end up at the target in the end. This means that we can do it if and only if the number of excess steps, $T - d$, is even.

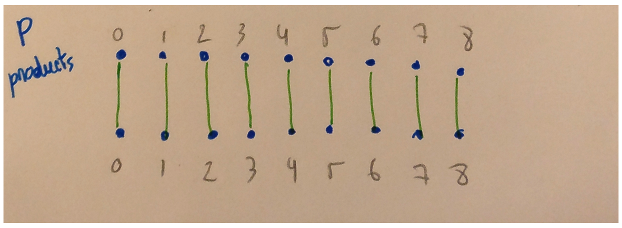
Problem B

Finding the optimal cost is a classical problem. Since we want to minimize the product, it is not hard to see that the best we can do it sort times increasingly and the costs decreasingly and assign the jobs in order.

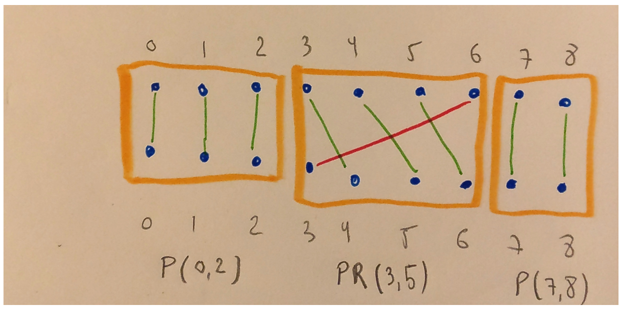
Now we need to preprocess the data to be able to efficiently answer each query.

From now on we assume that the input data was already sorted sorted (times on the top from small to big and costs on the bottom from big to small). In practice you have to map the indexes from the query to the sorted arrays to make it work.

In this case the optimal solution looks like this (we match i to i):

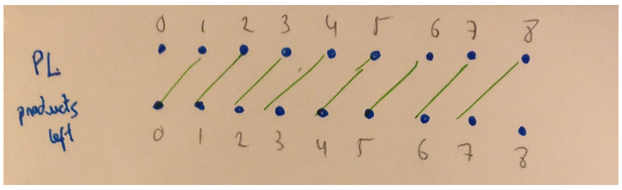


Now lets assume that the query asks to match 6 to 3. The it is easy to see that the best way to do so is the following:

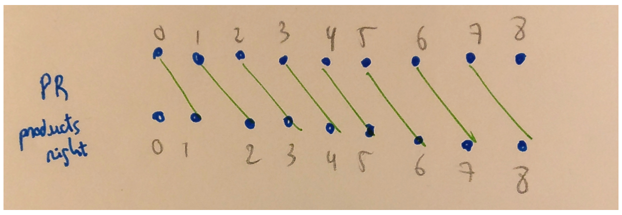


As we can see, before ans after the query region (indexes before 3 and after 6) remain the same. The others are shifted to the right. It would be to the left if the top index as smaller than the bottom one.

Therefore we will precompute the left shifted sums of products.



And the right shifted sums of products.



By doing cumulative sums over these sums of products we can answer in $O(1)$ the value of the sum of products over any given range. Thus we can combine the left, middle and right parts of the answer in $O(1)$.

We leave the details as exercise. For more about cumulative sums you can read here:

`https://www.geeksforgeeks.org/range-sum-queries-without-updates/`

Problem C

The number of cubes hit by a laser shot at (x, y, z) is

$$x + y + z - \gcd(x, y) - \gcd(x, z) - \gcd(y, z) + \gcd(x, y, z)$$

The pairwise gcd correspond the number of times the beam hits edges of the cubes and $\gcd(x, y, z)$ is the number of vertexes hit by the beam.

If you have trouble convincing you of this, start by seeing what happens on a $2D$ grid. The number of squares hit by a laser shot at (x, y) is $x+y-\gcd(x, y)$. $x + y$ is the number of edges that the beam has to cross to each point (x, y) but sometimes the beam will hit a grid point so it will cross two at at the same time, therefore we need to subtract $\gcd(x, y)$, the number times the beam will it a grid point.

In short we need to maximize the function

$$f(x, y, z) = x + y + z - \gcd(x, y) - \gcd(x, z) - \gcd(y, z) + \gcd(x, y, z)$$

It is easy to see that the maximum is $f(n - 2, n - 1, n)$.

We leave the details as an exercise but in short first notice that we cannot have two equal coordinates, say $x = y$, since otherwise we loose $\gcd(x, y) = \gcd(x, x) = x$. Thus $(n-2, n-1, n)$ are the largest all different coordinates. With these we have small pairwise gcd's, $\gcd(n - 2, n - 1) = 1$, $\gcd(n - 2, n) \in \{1, 2\}$ and $\gcd(n - 1, n) = 1$. The only thing that could be better is to have all of them equal to 1. In that case we only increase the value by 1. But that means decreasing one of the coordinates by at least 1 so it is not worth it. Notice that the triple gcd is always smaller than the minimum pairwise gcd so we cannot really gain from there.

Problem D

I forgot to mention in the problem statement that the log is base 2...

We can output any answer that has boat size $k \leq k_{min} + \log(k_{min})$. So if $k_{min} \geq 2$ then we can use boat size equal to $k_{min} + 1$.

A vertex cover of a graph is a subset of nodes such that when removed, no edges remain in the graph. Therefore the first trip must contain a vertex cover since otherwise we would leave a conflict on the left side. This means that $k_{min} \geq vc$, the minimum size vertex cover.

We claim that we can always solve the problem with a boat of size $vc + 1$: just put the vertex cover of size vc on the boat and take the others one by one to the other side never unloading any vertex from the vertex cover. Then when everyone is on the right side just drop the vertex cover.

This is a solution with $k \geq k_{min} + 1$. This is accepted provided that $k_{min} \geq 2$. If $k_{min} = 1$ then $\log(k_{min}) = 0$ so we need an optimal solution.

It is quite easy to list the graphs that have a solution with $k_{min} = 1$. There can be at most two edges both linked the same node. There can be any number of nodes of degree 0. This means that the graph is either

- all nodes have degree 0 (no edges); or
- a path of size 2 and all other nodes have degree 0; or
- a path of size 3 and all other nodes have degree 0.

Thus you need to first detect whether you are in one of these cases. If so you solve it with a boat of size 1. Otherwise you compute the vc and performs the trips as described above.

You need to be careful to not exceed n^2 trips. This means that for the special cases mentioned above you really need to make the minimum number of trips (the most naive solution will do an extra trip).

For you information k_{min} is a well known concept and is called the *Alcuin number* of the graph.

Problem E

This problem is about finding about finding a path in a cactus graph.

`https://en.wikipedia.org/wiki/Cactus_graph`

Each edge of the graph belongs to at most one cycle. Let's label the cycles from 0 to the number of cycles and then their edges with the same labels.

This can be done using a DFS: every time you find a back edge you can label the cycle. This costs $O(V + E)$.

`https://en.wikipedia.org/wiki/Depth-first_search`

Then you can prove that any path from s to t will always visit the set set of cycles (in the same order). This is because if you can find two paths that visit a different set of cycles then you can prove that some edge belongs to more than one cycle.

Therefore you simply compute *any* path from s to t and look at the sequence of cycles that are visited. Then you loop over the cycles and for each of them you can traverse it in two ways. You simply choose the longest one for each of them and you get the longest path.

Look at the pictures in the problem statement to see what I mean by two ways of traversing each cycle in case you are confused.