



## Editorial

Tasks, test data and solutions were prepared by: Gabrijel Jambrošić, Marin Kišić, Pavel Kliska, Vedran Kurdija, Daniel Paleka and Paula Vidas.

Implementation examples are given in attached source code files.

### Task Crtanje

Prepared by: Marin Kišić

Necessary skills: 2D arrays

We need to simulate what's written in the statement. The only issue is maintaining the given matrix, because the row index may be negative. One common trick is to add a constant (say, 100) to each row index, so that everything is nonnegative and we can use a standard 2-dimensional array.

### Task Odašiljači

Prepared by: Vedran Kurdija

Necessary skills: binary search, dfs

First, let's discuss how to check for some fixed radius  $r$  if all antennas can communicate. Two antennas can communicate directly if their distance is at most  $2r$ . We can build a graph over antennas, and add edges for all pairs of antennas that are closer than  $2r$ . Then, we need to check if the graph is connected, which can be done for example with dfs.

The optimal  $r$ , that is the minimal  $r$  for which the constructed graph is connected, can be found using binary search. For the upper bound, we can put for example  $10^9$ . The complexity is  $O(n^2 \log_2 10^9)$ .

### Task Euklid

Prepared by: Daniel Paleka and Paula Vidas

Necessary skills: math, ad-hoc

The first few subtasks can be solved in many ways; e.g. for  $h = 2$ , the simplest way is  $(a, b) = (2g, g)$ . The fourth subtask is just calculating  $R(a, b)$  for  $a, b \leq 1500$ ; we can check that this covers all  $g, h \leq 20$ .

For the large subtasks, the main issue is to find solutions not larger than  $10^{18}$ .

Notice that, if the problem can be solved for the given constraints, it must have a solution where few division steps occur in executing Edicul's algorithm. More precisely, the product of the numbers on the sand is a nice monovariant, which decreases by a factor of  $\frac{1}{b}$  in each step. Since  $b \geq h$  up to the last step, there can be at most 5 division steps for  $h = 200000$ .

We can play with finitely many possibilities to find a construction similar to the following one.

Let  $K$  be a positive integer such that  $h^K > q$ . Construction:

$$b = g \left\lceil \frac{h^K}{g} \right\rceil$$
$$a = hb + g$$

Notice that  $b = h^k + \text{something small}$ , such that  $g$  divides  $b$ . Also, we have  $b \leq a \leq \max(g, h)^3 \leq 10^{16}$ .

It's easy to see  $\text{GCD}(a, b) = g$ . Let's prove that  $R(a, b) = h$ .



We have  $\lfloor \frac{a}{b} \rfloor = h$  because  $g < h^K \leq b$ . Hence,  $R(a, b) = R(h, b) = R(b, h)$  since  $h \leq b$ .

However,  $h^{K+1} > b \geq h^K$ , thus  $R(b, h)$  becomes  $R(1, h) = h$  after  $K$  steps.

The time complexity is  $O(1)$  per testcase.

## Task Sjekira

Prepared by: Pavel Kliska

Necessary skills: trees, disjoint set union, ad-hoc

To solve the first subtask, since  $n \leq 10$ , it will be possible to try all permutations of connection cutting and determine each permutation price using dfs or a disjoint set union structure.

The main caveat to solving other subtasks is that it will always be optimal to cut the connections around the maximum node in the tree first. Proof sketch: on the path between the hardest node of value  $M$  and the second hardest node of value  $m$ , we have to cut some connection at some point, and pay  $m + M$ . If the connection is closer to  $M$ , the cut is more cost effective because more nodes will be in the subtree whose maximum is  $m \leq M$ .

So, there will always be a solution that finds the maximum in the tree, cuts all the connections around it, finds maxima in newly formed trees, and recursively descends into them. It only remains to implement finding the maximum efficiently.

The second subtask, in which the graph is a chain, can be efficiently solved using the segment tree structure, which allows you to quickly search for the maximum at an interval (or subtree) with time complexity  $O(n \log n)$ .

In the third subtask ( $n \leq 1000$ ) it is enough to search for the maximum using dfs, with time complexity  $O(n^2)$ . Now we solve the main subtask.

**Claim:** The solution is equal to

$$\sum_{i=1}^n t_i - \max_{1 \leq i \leq n} t_i + \sum_{i=1}^{n-1} \max(t_{x_i}, t_{y_i}).$$

*Proof:* Induct on  $n$ . Base case  $n = 1$  is clear. When we cut a connection  $a-b_j$  around the maximum vertex  $a$ , we add  $t_a + t_{a_j}$ , where  $a_j$  is the maximum vertex in the subtree of  $b_j$ . Now adding up the formulas for the subtrees finishes the induction step.

## Task Svjetlo

Prepared by: Gabrijel Jambrošić

Necessary skills: trees, dynamic programming

Let the optimal path start in  $x$  and end in  $y$ . We call the shortest path from  $x$  to  $y$  the primary path, and we call the subtrees that remain when we remove the primary path secondary subtrees. In optimal solution, we will go through the primary path, and solve secondary subtrees along the way (turn all bulbs on). It's obvious that it's optimal to solve a secondary subtree when we are on the primary node that connects to it, it doesn't pay off to return to it later.

We will now describe the algorithm for solving a secondary subtree, rooted in its primary node. When we are done with the subtree, we need to be in the root, to be able to exit. Since each move changes the parity of the number of bulbs that are off, and each edge will be traversed an even number of times, we can see that the parity of the number of bulbs that will be off in the end is determined. If it's even, it's optimal to turn all bulbs on, and if it's odd, it's optimal to have root turned off, and all others turned on.

That brings us to the following algorithm: we recursively solve subtrees of child nodes, and if some child remains off, we move down and up (that will flip the child and the current node). We do the same thing



when moving along the primary path. It's easy to see that this gives the minimal number of steps. If  $y$  is turned off in the end, then there is no solution that starts in  $x$  and ends in  $y$ .

We will use dynamic programming to solve the problem. Root the initial tree in any node. Let  $dp[i][j][k]$  be the minimal cost (number of nodes in the sequence), where  $i$  is the current node,  $j$  is 0 if when after solving the subtree the node  $i$  will be off, and 1 if it will be on, and  $k$  marks if the primary path goes through  $i$ .

For the case  $k = 0$  (without the primary path), we just sum up the  $dp$  states of children with  $k = 0$  and add the necessary number of crossings of edges between  $i$  and its children. We also need to keep track of the number of times we flipped node  $i$ , so we know its final state.

For the case  $k = 1$  we have two possibilities, we can take over the primary path from some child, or start the primary path in node  $i$ . The transition is the sum of  $dp$  states of children with  $k = 0$  for all except one child (that has  $k = 1$ ), or the sum when all children have  $k = 0$ .

For each primary path we can find a node such that the path goes through it, and both endpoints belong to its subtree. We can try for each node to be such a node. One possibility is to take its  $dp$  for  $k = 1$  and add the number of steps needed to solve the part of the tree "above" (i.e. the whole tree minus its subtree), in this case the node is the endpoint of the primary path. Second possibility is to take two  $dp$ -s of children that have  $k = 1$ , and others to have  $k = 0$ .

Total complexity is  $O(n)$ .