



Editorial

Tasks, test data and solutions were prepared by: Marin Kišić, Pavel Kliska, Daniel Paleka, Bojan Štetić and Paula Vidas.

Implementation examples are given in attached source code files.

Task Pizza

Prepared by: Paula Vidas

Necessary skills: ad hoc

For each pizza we just check if it contains a topping which Mirko dislikes, and if it doesn't we add 1 to the solution.

Task Vepar

Prepared by: Daniel Paleka

Necessary skills: prime sieve, v_p

For simplicity, replace the expression

$$a \cdot (a+1) \cdots b \mid c \cdot (c+1) \cdots d$$

with the equivalent

$$(c-1)! \cdot b! \mid (a-1)! \cdot d!.$$

For each prime p , define the map $v_p(x)$ which returns the largest power of p which divides x .

The left-hand side divides the right-hand side if and only if: for each p , we have v_p of the left-hand side is at most the v_p of the right-hand side.

It's easy to see $v_p(x \cdot y) = v_p(x) + v_p(y)$. We now want only $v_p(x!)$.

Proposition: We have

$$v_p(x!) = \left\lfloor \frac{x}{p} \right\rfloor + \left\lfloor \frac{x}{p^2} \right\rfloor + \left\lfloor \frac{x}{p^3} \right\rfloor + \dots$$

Proof: The first summand counts multiples of p , the second counts multiples of p^2 , and so on.

The solution is thus: find all primes up to 10^7 , implement $v_p(x!)$. and check the relevant inequality of v_p for each prime.

For implementations, see the official solutions.

Task Hop

Prepared by: Daniel Paleka

Necessary skills: ad hoc

There are several solutions; we describe the shortest one. The idea is: if $a|b$ and $a < b$, then $a \leq 2b$.

Let $\text{greatest_bit}(x)$ equal the position of the leading bit in the binary representation of x .

We give the edge between a and b to frog 1 if $\lfloor \frac{\text{greatest_bit}(a)}{4} \rfloor = \lfloor \frac{\text{greatest_bit}(b)}{4} \rfloor$.

All other edges we give to frog 2 if $\lfloor \frac{\text{greatest_bit}(a)}{16} \rfloor = \lfloor \frac{\text{greatest_bit}(b)}{16} \rfloor$. and the remaining ones we give to frog 3.

Using $\text{greatest_bit}(x) < 64$, we can prove this construction works.



Task Janjetina

Prepared by: Bojan Štetić

Necessary skills: centroid decomposition, Fenwick tree

First subtask can be solved by checking for each pair (x, y) if the condition holds.

In the second subtask, the tree is a chain. We will describe a solution using divide-and-conquer method. Let $f(l, r)$ be the number of valid pairs such that $l \leq x < y \leq r$, and let $m = \lfloor \frac{l+r}{2} \rfloor$. If we calculate the number of valid pairs (x, y) such that $1 \leq x \leq m < y \leq r$, i.e. the pairs whose path goes through the middle edge $m \leftrightarrow m+1$, we can calculate $f(l, r)$ as the sum of that number, $f(l, m)$, and $f(m+1, r)$.

Let $d(x)$ be the distance between nodes x and m , and $v(x)$ be the maximum weight on the path from x to m (we can take $v(m)$ to be $-\infty$). Pair (x, y) is valid if $\max(v(x), v(y)) - (d(x) + d(y)) \geq k$. If we assume that $v(x) \leq v(y)$, the condition is $v(y) - (d(x) + d(y)) \geq k \iff d(x) \leq v(y) - d(y) - k$.

We can sort nodes l, \dots, r ascending by $v(x)$, and go through them in that order. We will use a *Fenwick tree*, where we will store the counts of $d(x)$ for processed nodes. Let y be the current node. We will first query in our Fenwick tree the sum of the prefix $v(y) - d(y) - k$, and then add 1 to the position $d(y)$.

By doing this, we have counted all valid pairs, but also all pairs that are in either the left or the right half that fulfill the condition, which we don't want. So, we will repeat the procedure with nodes l, \dots, m , and with nodes $m+1, \dots, r$, and subtract from the result.

The solution for all points is very similar. Now m will be the *centroid* of the current component. Functions d and v are defined in the same way. Instead of two halves, we now have some number of subtrees. We first calculate the result with all nodes, and then subtract results for each subtree. Finally, we recursively call f for all subtrees.

The complexity is $O(n \log^2 n)$.

Task Patkice II

Prepared by: Pavel Kliska

Necessary skills: 0-1 BFS or Dijkstra algorithm

The map can be thought of as a weighted directed graph. The nodes are cells of the map, and two nodes are connected if the corresponding cells share a side. The weight depends on the direction of the current. If the current in cell A is directed towards cell B (or A is the initial island), the weight of edge $A \rightarrow B$ is zero because we don't have to change the map to be able to go from A to B directly. Otherwise, we have to change one character on the map and the weight is one.

To find the minimum number of changes needed, we just need to find the distance from 'o' to 'x' in the described graph, which can be done with 0-1 BFS. Dijkstra algorithm is a bit slower, but it was also enough to get all points.

The changed matrix can be constructed like this: we start in 'x', and in each step we take an edge whose weight is equal to the difference of the distance of its endpoints to 'o', and we replace the character in the current cell with the appropriate one if the weight is equal to one.