PATULJCI

Instead of writing seven nested for-loops to choose the seven candidate dwarves and check, we can choose two dwarves considered not in the group and check if the proposition is valid.

We add up the nine numbers and find two numbers that we need to subtract to get 100, using two nested for-loops. We then output the other seven numbers.

NPUZZLE

For each letter we calculate where it's supposed to end up at. We then sum the distances from the initial positions of all letters to their final positions.

The row and column indices of a letter's final position can be found using integral division by 4 (both the quotient and remainder) from the ASCII value of the letter, or using a hand-computed table.

TROJKE

We first find all letters in the input grid and make note of their positions. After that, for each triplet of letters, we check if there is a line going through all of them.

There is more than one way to check if there is such a line, one of which is to calculate the area of the triangle formed by the three points; if the area is zero, then the points lie on one line.

TENKIĆI

Suppose that we lax the constraints a bit so that more than one tank may occupy a single square. With that assumption in place, any tank may take any route to his final destination (it doesn't have to look out for other tanks).

Now notice that we can consider rows and columns separately, distributing all tanks to separate rows first, then to separate columns. We distribute the tanks into rows (and later columns) greedily – sort the tank by rows, send the first tank to the first row, the next to the second etc. It's not hard to prove that such an arrangement is optimal.

Reintroducing the constraint that tanks may not occupy the same square and using the same strategy, we only need to make sure that the tanks are moved so that no two tanks run into each other. We can achieve that by moving all tanks that need to be moved up in top-down order, all tanks that need to be moved down in bottom-up order etc.

BICIKLI

We model the network of cities and roads with a directed graph.

Note that vertices unreachable from the start node and vertices from which the end node is not reachable are of no use in constructing the bicycle route – we can remove them from the graph.

An infinite number of routes exist only if there is a cycle in the new graph. If there is a cycle, we can go from the start node to the cycle, go about the cycle any number of times and then continue on to the end node.

If there are no cycles, then the number of routes is finite and the graph is said to be acyclic (*directed acyclic graph*, DAG). Such a graph has what is called a topological ordering: if there is a path from vertex A to vertex B, then A comes before B in the topological ordering. We can find one topological ordering of the graph using a depth-first-search.

Once we've found a topological ordering, we consider vertices in that order and for each vertex V we calculate the number of routes from vertex 1 to that vertex: examine the edges going into V and add up the numbers of ways to reach the preceding vertices (those numbers have already been calculated by now).

LISTA

There are three parts to the solution.

The first part requires us to simulate Mirko's moves. As suggested by the problem statement, a doubly-linked list is to be used, supporting three operations ("erase", "insert before" and "insert after").

The second part requires us to find the smallest set of nodes to be moved to bring the list back to its starting state. Note that this is equivalent to finding the largest set of nodes that will not be moved, or the longest increasing subsequence (not necessarily contiguous). The longest increasing subsequence is found as follows

Let M[d] be the smallest number that can end an increasing subsequence of length d. M is then a (not necessarily strictly) increasing sequence – if an increasing subsequence of length d+1 can end in x, then so can a subsequence of length d.

Let M[0] = 0 and M[d>0] = infinity to start with. Iterate the list left to right, and for each number x:

The number x can extend any subsequence ending in a number less than x. We use binary search to find the largest d so that M[d] < x. Appending x to that subsequence, the length becomes d+1, so we set M[d+1] =x.

The length of the longest increasing subsequence is the largest d for which M[d] is finite. The solution is then N-d.

In the third part we find the sequence of moves that restores the list's original state. Extract the numbers that form the longest increasing subsequence into a separate sequence A (sorted in increasing order). The following algorithm generates the desired sequence of moves:

```
for each number x from 1 to N not in A
        let y be the smallest number in A such that y > x
        if there is no such y then
                move x to the end of the list
        else
                move x in front of y
```