NOP

Examine the program sequentially from left to right. If an instruction (capital letter) is at an address not divisible by 4, then we need to add NOP instructions just before it until the address becomes divisible by 4.

It is possible to literally add characters into the program string, but not necessary. It suffices to keep track of the number of inserted NOP instructions; the actual address of any instruction is its original address plus the number of NOP instructions inserted before it.

MAJSTOR

We use the algorithm given in the problem statement to calculate Sven's score. For each round, compare Sven's symbol with the other symbols and add the appropriate number of points to the score.

The largest number of points Sven could have scored can be calculated by trying all three symbols in every round and using the one that scores best.

This is easily done if we write a helper function points(r, symbol) that takes two arguments, the round number s and the symbol Sven supposedly shows in that round. If A and B are the two numbers we want to calculate for a round (the actual score and the largest possible score), we can calculate them as:

A = A + points(r, Sven[r])

B = B + max{points(r, 'S'), points(r, 'P'), points(r, 'K')}

CIJEVI

First write four auxiliary functions left(r,c), right(r,c), up(r,c) and down(r,c) telling us if gas can flow in each direction from empty cell(r,c). For example, the function left(r,c) can look like this:

```
return true if Map(r,s−1) = '+' or Map(r,s−1) = '-' or Map(r,s−1) = '1' or Map(r,s−1) = '2'
```

The task can be solved in one pass through the map. For every empty cell(r, c) we run the following checks:

```
if left(r,c) and right(r,c) and up(r,c) and down(r,c) then output r c '+'
else if left(r,c) and right(r,c) then output r c '-'
else if up(r,c) and down(r,c) then output r c '|'
else if right(r,c) and down(r,c) then output r c '1'
else if right(r,c) and up(r,c) then output r c '2'
else if left(r,c) and up(r,c) then output r c '3'
else if left(r,c) and down(r,c) then output r c '4'
```

TABLICA

Directly implementing the algorithm given in the problem statement gives a solution of time complexity $O(K \cdot N^2)$ and space complexity $O(N^2)$, scoring 30 points.

A better solution exploits the fact that we don't need to know the positions of all numbers, just the K numbers that appear as X in the moves. We go about simulating the moves as before, but only move numbers whose positions will be significant sometime later. The time complexity of this solution is $O(K^2)$ and the space complexity $O(K)$.

RELJEF

The task asks us to simulate the stick fight.

In every step, find the chunk of mineral that the thrown stick hits, delete it and check for any clusters floating in the air. If there is a floating cluster, we need lower all the chunks it contains until one of the chunks lands on the ground or another cluster. Clusters can be found using breadth-first search (BFS) or depth-first search (DFS).

CVJETICI

Simulating the growth of plants by maintaining a two-dimensional table is of time and space complexity $O(N^2 + N \cdot M)$, where M is the maximum coordinate. This solution scores 30 points.

Let A(x) be the number of plants for which a flower may grow at coordinate x (but hasn't yet). The sequence A initially contains all zeros and changes when plant (L, R) grows:

- Flowers will grow at coordinates L and R, a total of A(L) + A(R) of them, so we output that number. After these flowers have grown, there are no more available plants at coordinates L and R, so we reset A(L) and A(R) to zero.
- The horizontal segment [L+1, R−1] is now available to grow flowers, so we increase A(x) by one for each of those coordinates.

Directly implementing the above algorithm yields a solution of complexity $O(N \cdot M)$, scoring 45 points.

For full score, we need a data structure that supports the following operations:

- Set A(x) = 0;
- Increase A(x) by one for each x in [L-1, R+1].

Some of the data structures that do this are interval trees and Fenwick trees for O(log N) per query, or splitting the sequence A into blocks of size about sqrt(N), for O(sqrt(N)) per query.

The official source code splits the sequence A into blocks of size 256. See task JAGODA (COCI 2009, round 5) for details on this structure.