KORNISLAV

The solution is to sort the numbers A, B, C and D and output the product of the first and third numbers.

We need to separate the four numbers into two sets, one containing horizontal moves $\{x_1, x_2\}$, the other containing vertical moves $\{y_1, y_2\}$. The area is then $\min(x_1, x_2) \cdot \min(y_1, y_2)$. This expression is maximized when the two smallest numbers are in the same set.

RESETO

The algorithm is given in the problem statement; all that is needed is to implement it. Every time a number is crossed out, the counter K must be decreased by 1. When the counter becomes 0, we output the last number crossed out and end the program.

PERKET

The number of ingredients is small enough to try every subset of them and find which of them yields the smallest absolute difference between sourness and bitterness.

The recursive function takes three arguments:

- The index of the next ingredient for which we will decide whether to put it in the meal or not;
- The total sourness so far;
- The total bitterness so far.

In each recursive call we make one decision, branching into two new recursive calls. In one branch we don't use the current ingredient, in the other we do (updating the bitterness and sourness).

SVADA

Suppose that the output (how many seconds passed between the arrival of the first type and second types of monkeys) is S. Then we know that monkeys of the first type spent S seconds in the garden and monkeys of the second type spent $T-S$ seconds.

We can calculate how many coconuts the first type can gather in S seconds. Let this amount be $X(S)$.

Also, we can calculate how many coconuts the second type can open in $T-S$ seconds (assuming that there is an infinite number on the ground). Let this amount be $Y(S)$.

If $X(S) > Y(S)$, this means that the actual output is less than S because monkeys of the second type do not have enough time to open all the coconuts.

If $X(S) \leq Y(S)$, this means that the output can be S or more because there is time to open all coconuts.

The correct output is the largest S for which $X(S) \leq Y(S)$. We can use binary search to find it.

SETNJA

Suppose that the root of the tree is not labeled 1, but with an arbitrary label x.

For some x, let $f_i(x)$ be the value of all paths in the set described by the substring of the input starting with the i-th character.

The following recursive relations hold:

1. $f_i(x) = f_{i+1}(x)$; if the i-th character is 'P' (we stay in the root of the tree, with value x)
2. $f_i(x) = f_{i+1}(2 \cdot x)$; if the i-th character is 'L' (we move to the left child, with value $2 \cdot x$)
3. $f_i(x) = f_{i+1}(2 \cdot x+1)$; if the i-th character is 'R' (we move to the right child, with value $2 \cdot x+1$)
4. $f_i(x) = f_{i+1}(x) + f_{i+1}(2 \cdot x) + f_{i+1}(2 \cdot x+1)$; if the i-th character is '*'

The base case is the empty substring – $f_{N+1}(x) = x$.

The recursive formulas are linear, meaning that we only add constants or multiply by them. Because of this every term $f_i(x)$ can be written in the form $f_i(x) = A_i \cdot x + B_i$.

The above recursive formulas can then be rewritten as:

1. $f_i(x) = A_{i+1} \cdot x + B_{i+1}$
2. $f_i(x) = A_{i+1} \cdot (2 \cdot x) + B_{i+1} = (2 \cdot A_{i+1}) \cdot x + B_{i+1}$
3. $f_i(x) = A_{i+1} \cdot (2 \cdot x+1) + B_{i+1} = (2 \cdot A_{i+1}) \cdot x + (A_{i+1}+B_{i+1})$
4. $f_i(x) = A_{i+1} \cdot x + B_{i+1} + A_{i+1} \cdot (2 \cdot x) + B_{i+1} + A_{i+1} \cdot (2 \cdot x+1) + B_{i+1} = (5 \cdot A_{i+1}) \cdot x + (A_{i+1}+3 \cdot B_{i+1})$

The output is $f_1(1) = A_1 \cdot 1+B_1$.

We can use dynamic programming to calculate the sequences A and B. Because the terms in the sequences can get large, it is required to implement big integer arithmetic supporting long addition.

CAVLI

First we need to sort by x-coordinate and then by y-coordinate in order to find for each nail its neighbour in each of the four directions.

Now we simulate the removing of nails in order to find the order of removal.

Then we retrace our steps. Starting with the remaining two nails we add back nails, keeping track of the convex hull and calculating its area.

We can keep track of four extreme nails on the hull – top, bottom, left and right. This is helpful because they divide the hull into four parts in which the hull behaves predictably and adding a nail is simple.

Because of the way we removed nails, every nail we add will become extreme. After adding a nail, we remove nails near it until the hull is convex again, calculating the change in area along the way.