SIBICE

Since the box is of rectangular shape, the longest possible match is the same length as the diagonal of the box. A match fits in the box if $W^2 + H^2 \geq L^2$, where L is the length of the match.

SKENER

Using two for loops over the dimensions of the enlarged image (call the indices i and j), we print the character at position i/ZR, j/ZS in the original image (integral division, truncates towards zero).

PRSTENI

The number of times the i-th ring turns depends only on its radius and the radius of the first ring. The sought ratio is R_i / R_1. Reducing the fractions can be done by dividing the numerator and denominator by their greatest common divisor, given by the recursive formula:

$$GCD(a,b) = \begin{cases} a & , when\ b = 0 \\ GCD(b, a \bmod b), & when\ b \neq 0 \end{cases}$$

ZBRKA

This problem is solved using dynamic programming. Let f(n, k) be the number of sequences of length n with confusion k.

The number of such sequences that start with the number 1 is f(n-1, k) because the 1 does not affect the confusion of the rest of the sequence and it makes no difference if we use numbers 1..n-1 or 2..n.

If 1 is the second number, then f(n, k) = f(n-1, k-1) because whichever element is first, it will form a confused pair with the 1. It's easy to see that the complete relation is:

$$f(n,k) = \sum_{i=0}^{n-1} f(n-1, k-i).$$

The time complexity of a direct implementation of this formula (using dynamic programming) would be $O(N^2 \cdot K)$, which is too slow.

We need to note that f(n, k) = f(n, k-1) + f(n-1, k) – f(n-1, k-n), which leads to a $O(N \cdot K)$ solution. It is also possible to cut down on the memory used by keeping only two rows of the matrix used for calculations at any time.

JOGURT

There's more than one way to solve this problem, we present one of them.

Let's try to extend a tree of depth n for which the property is satisfied to a tree of depth n+1 for which the property is satisfied. Have the root node of the new tree contain the number 1 and put two copies of the old tree as its left and right subtrees, only multiplied by 2. Now the subtrees contain only even numbers and each appears twice between the two subtrees. The property is still satisfied in the subtrees; multiplying the elements by two caused the differences to be multiplied by two, but the depths of those nodes increased by one so this is good.

Add 1 to all leaf nodes in the left subtree and to all non-leaf nodes in the right subtree. Now each element appears only once in the entire tree. The property is still satisfied in the subtrees because adding a constant to all nodes on the same level in a subtree does not change the differences. The property holds for the root node because we added 2^n to the left subtree and $\sum_{i=0}^{n-1} 2^i = 2^n - 1$ to the right subtree so the difference is 1.

ISPITI

We're looking to design a data structure that holds pairs of numbers and can answer queries of the form "Which pair has both numbers larger than a given pair, but so that the second number differs by as little as possible?". One such data structure is an order-statistic tree (also called tournament tree).

In general, a tournament tree is a complete binary tree whose leaves hold data, while each node higher in the tree holds some desired statistic about the leaf nodes contained in the subtree rooted at that node: for example the smallest number, largest number, sum of numbers etc.

In this problem we'll have the nodes hold the largest number in their subtree. The root node thus holds the largest number of all the leaves, its left child holds the largest in the first half of the sequence etc. The tree is stored in an array so that the children of node x are at indices 2x and 2x+1. The parent of node x is x/2 (division truncates).

When adding the pair (a, b) to the tree, we write the value a on the b-th leaf node, and climb up the tree, updating the higher nodes.

The tree can tell us the first number larger than the one in leaf node x, located in the tree to the right of node x. The algorithm is:

- Start from the root node (consider the entire tree)
- Repeat:
    - If x is not in the subtree rooted at the current node, then none of these nodes are right of x so there is no solution in that subtree
    - If x is in the right subtree, move to that subtree (no nodes in the left subtree are right of x)
    - Otherwise (x is in the left subtree), if the left subtree contains a number larger than x, try to find the first one right of x (recursive call) – otherwise, find the leftmost number larger than x in the right subtree

It takes some convincing, but working out the cases reveals that the complexity of one query is O(log N), so the complexity of the entire algorithm is O(N·log N).

One final observation is that the numbers b are too large to be held in the tree. This can be remedied by sorting all students by the number b, then replacing all numbers b with the indices in the sorted sequence: the smallest number b is changed to 1, the second smallest by 2 etc. We can do this because the rest of the algorithm only cares about their relative ordering, not the absolute values. This procedure is referred to as compressing the input data.