

CROATIAN OPEN COMPETITION IN INFORMATICS

6th ROUND

SOLUTIONS

COCI 2010/11	Task OKUPLJANJE
6th round, March 5th, 2011	Author: Marko Ivanković

The solution is ad-hoc. It's easy to see that our estimate for the number of people at the party is the product of the given party area, and the number of people per m². Now we just read the 5 input numbers one by one, and output the requested difference for each of them. We can use some type of loop, but we don't have to since there are always exactly 5 numbers.

Necessary skills:

Simple arithmetic, number input

Tags:

Ad-hoc

COCI 2010/11	Task USPON
6th round, March 5th, 2011	Author: Matija Osrečki

As the task clearly states, we have to find the largest climb. Climb is defined as a strictly increasing subsequence of consecutive numbers, and it's size as a difference between the last and the first number. We traverse the sequence once, keeping track of the largest climb found so far, and the starting point of the climb we are currently at. Of course, it is possible that we are currently not on any climb, if the previous point was not lower than the current one. We can mark this, for example, by setting the starting point of the current climb to -1.

Necessary skills:

Array traversal

Tags:

Ad-hoc

COCI 2010/11	Task RAZINE
6th round, March 5th, 2011	Author: Adrian Satja Kurdija

Notice that there is no point in reducing the number of points for the last level. Now let's take a look at the level right before the last one. If this level is already worth less points than the last one, we don't need to reduce. If this level is worth more points, we reduce to the number of points the last level is worth decreased by one. We can now ignore the last level, and proceed in the same manner until we reach the first.

It's not difficult to see that obtained sequence will be strictly increasing, and also that we will be reducing by minimal possible number of points. Indeed, we didn't reduce anything that we absolutely didn't have to!

Necessary skills:

Array traversal

Tags:

Ad-hoc

COCI 2010/11	Task ABECEDA
6th round, March 5th, 2011	Author: Luka Kalinovčić

Consider every pair of consecutive words such that neither word is prefix of the other. Let k be the first position at which the words differ. Let a be the k -th letter of the word that appears later in the input, and b the k -th letter of the other word. It follows that a comes after b in alphabetical order.

Let's define the binary relation *greater than* on the given set of letters. We'll say that a is *greater than* b if a comes after b in alphabetical order. Notice that this relation is transitive (if $a > b$ and $b > c$, then $a > c$). We can easily compute its transitive closure using the Floyd-Warshall algorithm.

If the transitive closure suggests that $a > a$ for some letter a , the ordering does not exist.

Next, if there are two letters a and b such that neither $a > b$ nor $b > a$ holds, the ordering is not unique.

Otherwise, the ordering does exist and it is unique. Let the number k be equal to the number of letters b such that $a > b$ for some letter a . The letter a will take the k -th place in the sorted sequence of all letters in the alphabet, indexed from zero.

Necessary skills:

Transitive closure, Floyd-Warshall algorithm

Tags:

Graph theory

COCI 2010/11	Task STEP
6th round, March 5th, 2011	Author: Frane Kurtović

Let's say that we have two choreographies. Let's see what information about them we have to keep, in order to know the value of their concatenation. Resulting value can be either the value of the first choreography, value of the second, or new alternating subsequence can be formed using some suffix of the first one, and prefix of the second choreography. Of course, we are only interested in the longest suffix and prefix. Also, the last letter in the first choreography must differ from the first letter in the second. So, for each choreography, we must store: value, length of the longest alternating suffix and prefix, and first and last letter. Note that we can also obtain all of this for the resulting choreography.

This leads to efficient solution using a *tournament (interval) tree*. In each node of the tree, we store the information described above.

Task can also be solved using an existing *set* structure found in STL. We keep in the set all alternating subsequences. These subsequences are disjoint. Modifying a letter can lead either to splitting an existing interval, or to joining of the two intervals.

Both solutions have complexity $O(Q \log N)$.

Necessary skills:

Tournament tree or balanced binary tree (C++ set)

Tags:

Data structures

COCI 2010/11	Task VODA
6th round, March 5th, 2011	Author: Goran Žužić

We will solve the task by using dynamic programming. We fill the grid row by row (row-major order). Let's assume we have completely covered the first three rows. Notice that we don't care exactly how is the upper part of the grid filled, we only need to know some information, as the dynamic programming paradigm suggests.

For each column, we keep track of whether there is a pipe opened towards the fourth row. To be more precise, we will describe our current state as a tuple with number of elements equal to the number of columns. Each element can be either of:

- 0 → there is no pipe placed in this cell, or this pipe is not pointing towards the next row
- 1 → pipe is pointing down, and it is connected to the water source. Notice that there is at most one cell of this type at any moment.
- 2 and above → pipe is pointing down, but it's not connected to the water source. We are labeling connected components in this way. Notice that each number can appear either 0 or 2 times in some tuple.

In order to efficiently implement this solution, we have to normalize our tuples. Notice that the tuples (1, 5, 2, 0, 5, 2) and (1, 2, 3, 0, 2, 3) are equivalent, and we have to find a way to hash them to the same value.

So, we use memoization and count the requested number of ways. It's not easy to derive a precise complexity analysis. Taking into consideration the planarity of the pipe network, it's possible to prove that the number of tuples will be less than $6n^3 \cdot 3^n$, which leads to $3 \cdot 10^8$ for $N=10$. In reality, the number of states is considerably less.

Necessary skills:

dynamic programming

Tags:

advanced dynamic programming, hashing