COCI 2016/2017

Round #5, January 21st, 2017

Solutions

Task Tuna	**Author: Nikola Dmitrović**

The only problem in this task is data input. In the case when the absolute difference of two numbers from the same line of the input is less than or equal to X, then the total price is increased by the larger of the two numbers. If the difference is strictly larger than X, we know that we have to input another number from the next line, and then add it to the total price. We repeat the described algorithm N times.

Necessary skills: data input, decision-making commands, for loop, determining the larger of two numbers
Category: ad-hoc

Task Pareto	**Author: Adrian Satja Kurdija**

We will sort the amounts descendingly by size, from the largest to the smallest one. For each “prefix” that consists of K richest clients, we will calculate the corresponding numbers A and B, the share of that prefix in the number of accounts (A = K / N * 100%) and the total share of the money for that prefix: B = (the sum of K largest amounts) / (the sum of all amounts) * 100%. We now check if the difference B - A is the largest so far: if it is, we update the temporary variables *A_best*, *B_best*, the ones we ouput in the end.

Given the size of the array, we need to efficiently sort it, in the complexity O(N log N), and then efficiently calculate the sum of prefixes. The easiest way to do so is to calculate the sum of K-prefix by adding the K^{th} element to the previously calculated (K-1)-prefix.

Necessary skills: sorting
Category: ad-hoc

Task Unija	**Author: Adrian Satja Kurdija**

It is enough to observe only the right half of the image and in the end double the result. The solution is to sum the heights of X-columns for X = 1, 2, …, 10^7, which can be done using a for loop.

How to determine the height of the column at coordinate X? It is larger than or equal to the height of column at coordinate X + 1. It is larger if a rectangle exists that ends at coordinate X and is higher than the column at coordinate X + 1, otherwise it is equal. Therefore, initially, we need to store the X-coordinates where the given rectangles end and their heights, then traverse the X-columns “backwards”, towards the beginning, in order to apply the aforementioned formula.

Necessary skills: arrays
Category: ad-hoc

Task Ronald	**Author: Adrian Satja Kurdija**

For any line A-B we deduce: each Krump's selection of city A or city B changes its existence, so its final existence depends only on the parity of the number of selections of cities A and B. Given this fact, it is sufficient to assume that Krump selects each city zero or one time (selection or non-selection), which greatly simplifies the task.

We investigate two options: Krump either selects city 1, or he doesn't. For each of the possibilities, we will check whether they can lead to a complete graph. If we've fixed the selection (or non-selection) of city 1, for each other city K we can easily determine if it needs to be selected, considering the parity that must enable the existence of flight route 1-K. This way, we determine the selections or non-selections of all cities from 2 to N. Since we've only ensured the existence of flight routes 1-K, we will check the existence of all other lines, and if all of them exist, the answer is DA (Croatian for "yes").

An alternative solution is the following: the answer is DA (Croatian for "yes") if and only if the initial graph consists of exactly two components, each of them being a complete graph (clique). The proof is left as an exercise for the reader.

Necessary skills: analytical reasoning
Category: graphs

Task Poklon	**Author: Dominik Gleich**

We will solve the task using the offline method, where we first input all queries, and then process them. To begin with, let's define two functions, left(x), right(x) that correspond to the position of the first left and the first right element, respectively, of the same value as the one at position x. These two functions are easily computed, traversing from left to right, or from right to left, keeping track of the 'latest' of each value using a hash map, structure Map in C++ or a normal matrix, if we first compress the values. Let's observe a query [L, R] and functions left(x), right(x). It is easily noticed that we will count in a position x for query [L, R] if left(x) < L <= x <= right(x) <= R < right(right(x)). Using natural language: we want the first left appearance of the value to be before the left end of the interval, and the first right appearance before or at the right end of the interval, in addition that there is no other occurrence of the value in the interval. With this, we ensure the counting of each pair of two identical values exactly once. Now the task is reduced to the following problem: for an **x**, increment by 1 the solution of each query where left(**x**) < L <= **x** and where right(**x**) <= R < right(right(**x**)).

In order to execute this query, we need to construct a tournament data structure over all intervals in the following way: for each node of the tournament that covers interval [A, B], we need to ‘insert’ the query [L, R] if A <= L <= R, in other words, L e [A, B]. After this, for each **x**, we need to insert the interval [right(**x**), right(right(**x**))> into all nodes of the tree of which the interval <left(**x**), **x**] is made of (typical query or update operation in a tournament data structure). There are O(lg N) such nodes for each **x**. After inserting queries and intervals into the tournament, we need to calculate the ‘contribution’ of each node to the queries located in that node. We perform this using the sweep technique and leave the implementation details as an exercise for the reader.

The total complexity of the given solution is $O(N \lg^2 N)$. A solution of the complexity O(N lg N) also exists, and is also left as an exercise for the reader.

Necessary skills: sweep, data structures
Category: tournament, data structures

Task Strelice	**Author: Mislav Balunović**

Let’s construct a graph where the nodes are the fields of the board, and the edges are obtained by connecting two nodes of each arrow that is not in the last column, using a directed edge. Since each node has a degree of at most 1, the obtained graph consists of two different types of components:

- Directed tree
- Cycle with “branches” leading to it

Each path from the first column to the last column is actually a path from a tree node to the root of that same tree. Let’s mark the nodes in the first column as “special”. Now the task is actually to color the K nodes of the tree so that each path from a special node to the root contains exactly one colored node.
Let’s apply a trick: we can add an auxiliary node to the graph and an edge from each root to that auxiliary node. Now we have exactly 1 tree in the graph and its solution is checked using dynamic programming.

Let’s calculate the value of function f(x, i, k) that returns whether we can color all subtrees of node x from its ith child onwards. The crucial part is to determine how many nodes we will color in the ith subtree. If we color exactly y nodes, the solution exists if both values of f(x, i + 1, k - y) and f(child[x][i], 0, y) are equal to 1.
In the end, we make sure that we can arbitrarily color the nodes in the components that lead to cycles.
If we iterate over all possible y, the complexity of this algorithm is $O(NMK^2)$.

In order to speed up the solution, we can use bitmasks.
Let’s denote the bitmask that contains bit f(x,i+1,k) in the kth position with F[x][i]. Additionally, we denote with R[x][i] that same mask with the first 50 bits in reverse (the 0th element of F is

the 50th of R). Now f(x,i,k) = (R[x][i] >> (50 - k)) & F[child[x][i]]. We leave it as an exercise for the reader the proof of the corectness of this optimization that enables the speed-up of the aforemention dynamic programming approach to O(NMK).

Necessary skills: dynamic programming, bitmasks
Category: dynamic programming