CROATIAN OPEN COMPETITION IN INFORMATICS

2013/2014

ROUND 2

SOLUTIONS

COCI 2012/2013	Task VOLIM
2nd round, November 9th, 2013	**Author:** Nikola Dmitrović

The solution of this task is to simulate the described game. For each query, the total game duration should be increased by the duration of that query. If the answer to that query was "T", then the score of the player "K" should be increased by 1. (The emerging problem is what to do if the box reaches a player with a label greater than 8, which can be solved easily with remainders.)
We keep repeating this until the total game duration is less or equal to 210 seconds (3 minutes and 30 seconds). Notice that the events "incorrect" (N) and "skipped" (P) do not make change the player's turn and shouldn't be taken into account. From the task conditions it is obvious that the query can continue even after the box has exploded. Notice that it is not a problem to end the program before all data has been read, what is important is the correct output. Take a look at the pseudocode for one of the possible solutions of this task.

```
input (K)
input (N)
game := 0 // game duration, initially zero

while (game <= 210) do
{
    input (T, Z)
    game := game + T // always increase game duration by T
    if (Z = 'T') and (game <= 210) then
        K = (K % 8) + 1
}
output(K)
```

Necessary skills: while loop

Category: simple simulation

COCI 2012/2013	Task MISA
2nd round, November 9th, 2013	**Author:** Adrian Satja Kurdija

We check if each seating place inside the church is available or not. If it is available, we check for the number of its non-available neighbours, to find out whether Mirko will sit there. If it is not available, yet again we need the number of its non-available neighbours in order to count them in the total number of handshakes.

How to check an item's (row, column) 8 neighbours simply? Its neighbours are following:

(row - 1, column - 1)
(row - 1, column)
(row - 1, column + 1)
(row, column - 1)
(row, column + 1)
(row + 1, column - 1)
(row + 1, column)
(row + 1, column + 1)

For each item we must check whether it is a part of our seating matrix (that is, if its coordinates are greater than zero and not greater than **R** or **S**) and whether it is already occupied. Manually checking of each condition can be time-consuming, so we use two arrays to help us:

array move_by_row = (-1, -1, -1, 0, 0, 1, 1, 1),
array move_by_column = (-1, 0, 1, -1, 1, -1, 0, 1).

Using a for loop, we will iterate from 1 to 8 and calculate for each **i**:
row_neighbours = row + move_by_row[**i**]
column_neighbours = column + move_by_column[**i**]

If you think about it, you will see that this way we iterate through all 8 of our current element's neighbours. Inside the for loop we still have to check the aforementioned conditions.

The total number of handshakes during the Mass, without Mirko, is equal to the sum of calculated handshakes of all neighbouring non-available elements divided by two, because we counted each handshake twice. We still have to add the number of Mirko's handshakes, which is represented by the maximum number of neighbouring items of the current item.

Necessary skills: matrix operations

Category: ad hoc

COCI 2012/2013	**Task SLOM**
2nd round, November 9th, 2013	**Author:** Marin Tomić

Given the current word, it is easy to reconstruct what that word looked like before one blink. Let's call it a reverse blink. We can find the initial word by repeating the reverse blink **X** times. This solution is of the complexity O(**NX**), which is good

enough to gain 50 points.

After a bit of scribbling on paper, it is easily noticed that the reverse blinking is periodical, meaning that after **P** reverse blinks, the word will transform into the initial form. Hence, if we repeat the reverse blink k * **P** + l times, it is the same as we repeated it just l times.

The solution requires only finding the number **P** (which can be done by simply repeating the reverse blink until the word is transformed into the initial form) and then repeat the reverse blink **X** mod **P** times.

It is shown that **P** is always less than **N**, meaning the total complexity of this algorithm is at most O($\mathbf{N^2}$), which is fast enough for 100% of points.

Necessary skills: for loop, perceptive skills

Category: ad-hoc

COCI 2012/2013	**Task PUTNIK**
2nd round, November 9th, 2013	**Author:** Adrian Satja Kurdija

Any sequence of towns fulfilling Mirko's condition can be constructed in the following way. Firstly, town 1 is added into the sequence. Then we add the town 2 to the left or to the right of town 1. Then we add town 3 on the left or right end of the current sequence of towns. Next, town 4 is aded on the left or the right end, and so on. Such sequence will fulfill Mirko's condition, and vice versa, any sequence fulfilling Mirko's condition can be constructed in the described manner.

Now the task can be solved using dynamic programming. The state is described with the left and right end of the current sequence. This is enough to tell us which town can be added next. Why? The town we added last is either on the left or right end, so its label is actually the maximum of those two numbers. The city being added next, of course, has a label increased by 1. We choose whether to put it on the left or right end: for each of those possibilities, we calculate the total flight duration as the duration of the chosen flight increased by the value of the next state in the dynamic. Moreover, we choose the more favorable solution as the current state value. The complexity of this algorithm is O($\mathbf{N^2}$).

Necessary skills: dynamic programming

Category: dynamic programming

COCI 2012/2013	Task PALETA
2nd round, November 9th, 2013	**Author:** Domagoj Ćevid

If we represent the coloring book as a graph, the images act as the nodes and two nodes are connected with an edge if they cannot be colored in the same color.

Firstly, solve the case when all the input numbers are different. Then the graph consists of disjoint cycles and f(n) represents the number of ways we can color a cycle consisting of n nodes. The images are painted sequentially. The first image can be painted in k ways. The second can be painted in k-1 way because it has to differ from the first image. Analogously, it is concluded that in the $k(k-1)^{n-1}$ colorings there will exists some colorings in which the image numbered 1 is of the same color as the image numbered n. Connecting the nodes 1 and n, if they're of the same color, a cycle sized n-1 is created with differently colored nodes. We can conclude that the recursive relation applies: $f(n) = k(k-1)^{n-1} - f(n-1)$. Therefore, we can compute this function easily in linear complexity.

If the input numbers are not different, our cycles will have edges that are, in fact, trees. Let us imagine that we have colored the cycles and are now coloring those edges. The node connected directly with the cycle can be colored in k-1 ways (it has to be different than the one connected to it), the node connected to it can be painted in also k-1 ways and so on. It is a valid conclusion that for each node not belonging to the cycle, we have to multiply our current solution by k-1.

We traverse the graph using DFS and decide how many cycles there are of different sizes.

Necessary skills: graphs, recursive formulas, DFS

Category: graphs

COCI 2012/2013	Task LINIJE
2nd round, November 9th, 2013	**Author:** Domagoj Ćevid

Let's call a line valid if it is paralell with x or y-axis and there is a point located on it.

The lines and points can be represented by a bipartite graph. The nodes on left side of the graph represent valid lines parallel with x-axis, and the nodes on the right side represent valid lines parallel with y-axis. For each input point (x,y) we can set an edge between the line x from the left side and the line y from the right

side.

Now the game can be presented like this:

- in the first move Mirko chooses a node and removes it from the graph
- after that, Slavko removes any node not removed so far connected to the previously removed node
- next, Mirko does the same and they repeat this alternately
- the loser is the one who cannot remove anything more

Since this is related to bipartite graphs, it is expected that the solution will be connected with maximum matchings.

Exactly!
If there is a perfect matching (where each node from the graph is paired with exactly one other node with which it shares an edge), Slavko will win. Otherwise, Mirko wins.

Why?
Let us presume that there is a pefect matching in our graph. It doesn't matter which node Mirko chooses – it will always have an existing match on the other side of the graph, making Slavko's next move possible. Therefore, whatever Mirko chooses, Slavko can fight back. This makes Mirko the designated loser.

In case there isn't a perfect matching, we can analyse the maximum one. Mirko can in the beginning choose whichever nonmatched node. Now the node Slavko chooses will surely have a pair (if it doesn't have a pair, we could improve the maximum matching by connecting Mirko's and Slavko's played out nodes – contradiction), so Mirko can use the same tactics as Slavko in the previous paragraph.

Necessary skills: maximum bipartite matching

Category: combinatorics, game theory, graphs