CROATIAN OPEN COMPETITION IN INFORMATICS

5th ROUND

SOLUTIONS

COCI 2010/11	Task GLJIVE
5th round, February 5th, 2011	**Author:** Adrian Satja Kurdija

Notice that distance between the numbers **X** and 100 is equal to:

- **X**-100, if **X** is greater than 100
- 100-**X**, otherwise

We add numbers one by one starting from the first. If the running sum becomes greater than 100, our solution is either the previous sum we had, or the current sum. We choose the sum closer to 100.

If the sum of all the numbers is not greater than 100, we output this sum.

Necessary skills:

Addition, substraction, and comparision of integers

Tags:

Ad-hoc

COCI 2010/11	**Task KOSARKA**
5th round, February 5th, 2011	**Author:** Goran Gašić

For each second in the interval [0, 48*60], we store what team (if any) scored in that second. To fill this array, we must convert the given events from MM:SS format into the number of seconds elapsed from the start of the game, MM*60+SS.

Now we traverse this array keeping track of the current score for each team. For each second during which the score wasn't tied, increase the counter for the winning team.

Necessary skills:

while or *for* loop, integer arrays

Tags:

Ad-hoc

COCI 2010/11	Task BRODOVI
5th round, February 5th, 2011	**Author:** Adrian Satja Kurdija

First, let's decrease all given day indices by 1. This way, a ship with visit period **K** will come to the harbour in days 0, **K**, 2**K**, 3**K**, …

Let **A** be the first entertaining day after the day 0. It is clear that there is a ship with visit period **A**. This ship will also arrive to the harbour in days 2**A**, 3**A**, etc.. Therefore, we no longer need to worry about entertaining days divisible by **A**, since this ship will visit them all.

We repeat this procedure, ignoring the days covered by previously found ships.

When there are no more entertaining days that we must take care of, we are done and the solution is equal to the number of ships found.

Complexity is O(**N**^{2}), because for each ship found (**N** ships max.) we must traverse all the entertaining days (**N**).

Necessary skills:

Basic modular arithmetic

Tags:

Number theory

COCI 2010/11	**Task HONI**
5th round, February 5th, 2011	**Author:** Stjepan Glavina

Let $\mathbf{A}_1$, $\mathbf{A}_2$, …, $\mathbf{A}_\mathbf{N}$ denote the sets of tasks with weights 1, 2, … **N**, respectively. Additionally, let $\mathbf{B}_2$, $\mathbf{B}_3$, ...$\mathbf{B}_\mathbf{N}$ denote the sets of tasks with weights 1 or 2, 2 or 3, …, **N**-1 or **N**, respectively. Finally, let $\mathbf{B}_1 = 0$.
The problem can now be solved using a dynamic programming approach; starting from the first weight (the least one) we choose a task for each weight. When choosing a task with the weight equal to **T**, we need to check whether another task from the set $\mathbf{B}_\mathbf{T}$ has already been taken. Therefore, it is sufficient to have a table of the form `dp[t][b]`.

This yields the following relations:
`dp[0][0]` = 1
`dp[0][1]` = 0
`dp[i][0]` = `dp[i-1][0]` * ($\mathbf{A}_i$ + $\mathbf{B}_i$) + `dp[i-1][1]` * ($\mathbf{A}_i$ + $\mathbf{B}_i$ - 1)
`dp[i][1]` = `dp[i-1][0]` * $\mathbf{B}_i$ + `dp[i-1][1]` * $\mathbf{B}_{i+1}$

Necessary skills:

Dynamic programming

Tags:

Dynamic programming

COCI 2010/11	**Task DVONIZ**
5th round, February 5th, 2011	**Author:** Goran Žužić

Let us freeze the middle of an interesting sequence in all possible ways. It can easily be noticed that if there exists a sequence of length 2**N**, there also exists a sequence of length 2**M** < 2**N** with the same middle value for each integer **M** less than **N**. This suggests a binary search approach to be used to find the longest interesting sequence for every position of the middle element, which yields an algorithm with O(**N** log **N**) time complexity.

For each element, we first search for the longest sequence which starts with that element. To solve that problem, we process elements from right to left. Among all interesting sequences [$\mathbf{a_i}$, $\mathbf{b_i}$] which start before the position of the current element we are processing we search for the one for which the value ($\mathbf{b_i}$-$\mathbf{a_i}$+1)-2(**X**-$\mathbf{a_i}$) = $\mathbf{b_i}$+$\mathbf{a_i}$+1-2**X** is maximal. The value of $\mathbf{a_i}$ can be found as follows: as we process the elements, we keep track of the maximum and we try to improve that maximum by considering interesting sequences which start at the current position.
When we move to the right, we decrease that maximum by 2 (since a move **X** := **X**+1 decreases this value by 2).
The above algorithm can easily be implemented with O(**N** log **N**) time complexity, which solves the problem.

Necessary skills:

Binary search, range operations

Tags:

Searching, simple data structures

COCI 2010/11	**Task SLIKA**
5th round, February 5th, 2011	**Author:** Goran Gašić

First, observe that we can get rid of all SAVE and LOAD commands with O(**M**) time complexity. We process the sequence of commands in reverse order, starting from the last one down to the first one. Whenever a LOAD command is to be processed, we ignore all commands between this command and the corresponding SAVE command. With this approach, we obtain an equivalent sequence of commands which consists of only PAINT commands.

This problem can now be solved using a sweep-line algorithm. Traversing the painting row by row, we keep two tournament trees; a red one and a white one, for convenience. The red tree will contain the checkerboard patterns in which even squares are painted in the current row, while the white tree will contain the patterns in which odd squares are painted in the current row. As we traverse rows, we swap those two trees before advancing to the next row. The tree nodes will contain the sets of patterns which cover the corresponding intervals, sorted by the time of painting.

Note that insertion and removal of a pattern has O(log **N** * log **M**) time complexity, while a query for any square has O(log **N**) time complexity, which yields the total time complexity of O($\mathbf{N}^2$ * log **N** + **M** * log **N** * log **M**).

Alternatively, the task can be solved with the union-find algorithm, in time complexity O(**N** * **M**). This is left as an exercise for the reader.

Necessary skills:

Tournament tree, union-find algorithm

Tags:

Advanced data structures