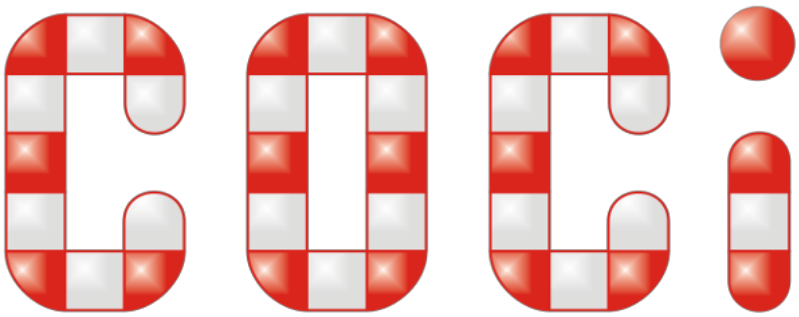

CROATIAN OPEN COMPETITION IN INFORMATICS

4rd round, December 20th, 2014

solutions

Task CESTA	**Author: Ivan Paljak**

The prime factorization of number `30` is `2*3*5`. Therefore, **N** is divisible by `30` if and only if it is divisible by `2`, `3` and `5`.

We know that a number is divisible by 2 if it's last digit is even and it's divisible by 3 when the sum of its digits is divisible by 3 and, naturally, we know that it's divisible by 5 when it's last digit is either 0 or 5. By combining these conditions, we conclude that **N** is divisible by 30 if and only if it's last digit is 0 and the sum of its digits is divisible by 3. Therefore, if the number from the input data doesn't contain the digit 0 or the sum of its digits is not divisible by 3, we output -1.

What is obvious is that it is sufficient to sort the digits of number **N** in descending order to get the required number.

Implementations with time complexity of `O(n')` or `O(n'log(n'))`, where `n'` is the number of digits of **N** were sufficient to score all points on this task.

Necessary skills: divisibility, greedy algorithms
Category: number theory, ad-hoc

Task PŠENICA	**Author: Ivan Paljak**

A naive approach to solving this task would be to simulate every move. More precisely, in each move we would find two smallest numbers if it's Mirko's turn, or two biggest numbers if it's Slavko's turn. Then we would make the corresponding change and simulate this procedure until the game is finished. Because the increase of the number of moves is quadratic considering the number of stalks of wheat and we need `O(n)` time to find the smallest/biggest numbers, the total time complexity of this approach is `O(n^3)`, which was enough to score 50% of total points on this task.

By using smarter storage of data, we can save time needed to find the smallest/biggest stalks. In this purpose, we will use the structure *deque* which enables us to push and pop data from the top or the bottom of the structure in the complexity `O(1)`. Each element of the structure is going to be an ordered pair `(height, amount)` that denotes the number of stalks of wheat for a certain height and the elements are going to be sorted in ascending order according to height (from top to bottom). The search for extremal values now comes down to removing the elements from the top/bottom of the structure. Moreover, the simulation of the game comes down to updating the first two values from the top or the bottom of the structure (depending on who's turn is it. This and similar implementations which have the time complexity `O(n^2)` are sufficient to score 80% of total points on this task.

Finally, let us notice that, given multiple tallest and shortest stalks, some moves will be repeated. Let's illustrate this on an example. Let the current stalks' heights be `{1,1,1,2,3,4,4,4}`. The first few moves (if Mirko starts) are going to be:

- increase 1 to 2
- decrease 4 to 3
- increase 1 to 2
- decrease 4 to 3
- increase 1 to 2

Only now have we decreased the number of different heights by 1. The state of wheat after these moves is `{2,2,2,2,3,3,3,4}`.

Using the ideas from previous paragraphs, we can easily analyze the number of shortest and tallest stalks and deduce what the state of wheat will be after the number of different heights is increased by 1. Therefore, in one iteration of the algorithm we will surely decrease the number of elements in the deque structure by 1. This and similar implementations which have the time complexity `O(n)` are sufficient to score 100% of total points on this task.

For implementation details of all three approaches, consult the official solutions.

Necessary skills: algorithm complexity, deque structure
Category: ad-hoc

Task PRIPREME	**Author: Ivan Katanić**

Let us denote the largest element of the input array with M and the sum of the array with S. Let us observe the following two cases:

a. M <= S / 2

The lower bound to the solution of the task is clearly S and such solution can always be achieved in this case. If we sort the elements of the input array and denote them with integers from 1 (smallest) to N (largest), then one of the optimal arrangements is: Ante {1, 2, ... , N} and Goran {N, 1, 2, …, N-1}.

b. M > S / 2

This inequality can be written as M > S - M or, in other words, the time required for the slowest team is larger than the sum of times of all other teams.

The lower bound to the solution in this case is 2M because both lecturers have to give their lecture to the slowest team and can't do so at the same time. In this case too, it is possible to always achieve the lower bound. Let Ante first give the lecture to the slowest team, while Goran gives his lecture to all the other teams and wait for Ante to finish. After that, Goran takes over the slowest team and Ante the rest of the teams.

Necessary skills: breaking into cases
Category: ad-hoc

Task MRAVI	**Author: Antonio Jurić**

The task states that Bobi has the power to turn on or off the superpower of each super pipe. Notice that it is **always** better to turn on the superpower of a pipe if current amount of liquid flowing through it is larger than or equal to 1, because that way we increase the total amount of liquid in the tree. In the case when the current amount of liquid in the pipe is smaller than 1, we will not use the superpower because that would decrease the total amount of liquid in the tree.

How will we find the minimal **L** that meets the conditions from the task (that each ant gets at least $\mathbf{K}_i$ liters of liquid)?

Let's fixate a concrete **L** and calculate how much liquid each leaf (ant) gets with that flow. We can calculate this by simply traversing the tree and literally simulating the pipe system from the task. Let's assume that for this **L** all ants got a sufficient amount of liquid. This means that for an even bigger value **L** all ants will get a sufficient amount of liquid so we only need to check what are the smaller values of **L** that satisfy the "ant" conditions.

This conclusion leads us to the binary search algorithm with which we can calculate the smallest required **L**.

The experienced eye won't miss that the described algorithm doesn't actually work. More precisely, it doesn't work when implemented on a computer. Imagine this situation: given a chain in a tree we initially have a lot of squaring and then we have a lot of pipes that get only 1% of liquid from the parent node. Firstly, the amount of liquid becomes very large (a lot larger than $2*10^9$ - the upper solution bound), and then the 1-percent pipes decrease it so it fits below the upper bound, but that number is not be precise because, even though the double data type stores large numbers, it doesn't keep absolute precision. This problem can be solved if we logarithmize all numbers so the operations of squaring and multiplying become operations of summation (and thus relative precision is kept).

Since advanced handling of floating point data types doesn't belong to high school competition required knowledge, this solution wasn't expected. Therefore, there were no test data in which the described situation appears.

We leave finding the algorithm in linear complexity for the readers to practise.

Necessary skills: binary search, tree traversal
Category: binary search

Task SABOR	**Author:** Adrian Satja Kurdija

Let us observe the following simple algorithm:

1. Place the MPs to parties in any way.
2. As long as there is an MP **K** for which the required condition is not met, switch **K** to the opposing party.

It is not clear right away whether this algorithm is going to finish or we will keep switching the MPs eternally. But the following is clear: if the algorithm finishes, it finds the required solution.

Luckily, not only does the upper algorithm finish, but it is fast enough. Let's prove it!

Remember that each MP is arguing with at most **five** other MPs. Let's examine the number of **intraparty arguments**. We switch the MP if he does not meet the required condition. In other words, if he causes **at least three** intraparty arguments. By switching him, we remove these arguments, possibly creating new intraparty arguments in the other party, but **at most two** (with his remaining enemies). It follows: with each switch, **the total** number of intraparty arguments **decreases**. Since the initial number of intraparty arguments is at most 5**N**/2, the algorithm must finish after at most 5**N**/2 switches.

Necessary skills: algorithm design and analysis
Category: ad-hoc

Task STANOVI	**Author: Mislav Balunović**

Let's imagine that the sides of the rectangle comprising the apartments are actually hallways.
Firstly, let's prove a statement that is going to help us in solving the task:

Statement: There is a hallway parallel to a side of the building that passes from one edge of the building all the way to the opposite edge.
Proof: Let's assume the contrary, that such a hallway does not exist.
Imagine that we are located at the start of a hallway (so, at the edge of the building) and start moving in the opposite direction. We move using the following algorithm: go straight until you hit an edge of an apartment. In that moment, when we can't continue straight anymore, look to the left and right. In one of these directions we will not hit the building's edge for sure, because that would be contrary to the assumption that there is a hallway from one edge of the building to the other. Start moving in that precise direction.
Notice that we will enter a cycle at one point – from that moment on, we will be walking over the same part of the building. But that would mean that there is an apartment in the

interior of that part of the building and that is impossible because that apartment wouldn't have a window. Therefore, the initial assumption is wrong and the statement is true.

We then proceed to solve the task using dynamic programming.

The current state can be described with dimensions of the building and 4 boolean values that tell us what part of the building we are located in (they represent the directions where the edge of the building is).

If all 4 boolean values are false, we return some large value because we cannot have an apartment in that part of the building.

In the contrary, we have 2 possibilities:

1) The whole of this part of the building is going to be an apartment.
2) We choose a hallway and split the building into two parts and recursively calculate the values of the dynamic for those parts of the building.

For implementation details, consult the official solution.

Necessary skills: combinatorics
Category: dynamic programming