CROATIAN OPEN COMPETITION IN INFORMATICS

2012/2013

ROUND 1

SOLUTIONS

COCI 2012/2013	**Task DOM**
1st round, October 20th, 2012	**Author:** Adrian Satja Kurdija

Using a for loop, we traverse the input string and output each letter if it is neither ‘C’, nor ‘A’, nor ‘M’, nor ‘B’, nor ‘R’, nor ‘I’, nor ‘D’, nor ‘G’, nor ‘E’.

If we want to avoid writing nine conditions, we can use a variable containing the string CAMBRIDGE and a nested loop over that string checking whether any letter matches the current letter from the input string.

Necessary skills: for loop, string manipulation, branching

Category: strings

COCI 2012/2013	Task F7
1st round, October 20th, 2012	**Author:** Nikola Dmitrović

Let us sort the drivers in descending points order and select the $\mathbf{K}^{th}$ driver in that sorted order. We need to determine whether (s)he can be the World Champion.

We can construct the best possible final race scenario for that driver. In such a race, (s)he would earn **N** points, while the current champion would earn only 1 point, the current runner-up only 2 points and so on. Generally, the driver in position $\mathbf{p} < \mathbf{K}$ earns **p** points; drivers in positions $\mathbf{p} > \mathbf{K}$ are irrelevant since they cannot have more points than driver **K** in the constructed scenario.

Can driver **K** be the World Champion in this scenario? The answer is 'yes' if (s)he has a sum greater than all other drivers, that is, if the relation

$$\mathbf{a}[\mathbf{K}] + \mathbf{N} \geq \max\{\ \mathbf{a}[1] + 1, \mathbf{a}[2] + 2, \ldots, \mathbf{a}[\mathbf{K} - 1] + \mathbf{K} - 1\ \}$$

is satisfied, where $\mathbf{a}[\mathbf{p}]$ is the number of points that driver **p** in the descending sequence had before the final race.

The solution that computes the maximum above (let us denote it by $\mathbf{mx}(\mathbf{K})$) for every **K** from scratch has complexity $O(\mathbf{N}^2)$, which is too slow. Fortunately, the maximum is simple to compute gradually as needed: whenever the observed **K** is incremented, we can compare the previous maximum value with the current point value and update the maximum if needed. To formalize,

$$\mathbf{mx}(\mathbf{K} + 1) = \max\{\ \mathbf{mx}(\mathbf{K}), \mathbf{a}[\mathbf{K}] + \mathbf{K}\ \}$$

This reduces the complexity of finding the solution to $O(\mathbf{N})$. The total complexity is $O(\mathbf{N} \log \mathbf{N})$ because of sorting.

Necessary skills: mathematical problem analysis

Category: ad-hoc

COCI 2012/2013	**Task KONCERT**
1st round, October 20th, 2012	**Author:** Goran Gašić

Notice that the solution is trivial if the number of tickets is greater than or equal to the number of guys. We can simply use a sequence of give commands until each guy has a ticket. After that, all guys can enter.

Otherwise, the key observation is: if there are at least two tickets, two people can enter the venue. After that, one person can give their ticket to the other, who can then exit the venue. After this sequence, there is one more person in the venue, while there are still at least two tickets outside.

It follows that we can choose a guy or girl to act as a "courier", i.e. enter the venue with other guys, take their tickets and give them to guys outside. We can repeat this procedure until all guys are in the venue, perhaps save for the courier (if a male one was chosen), who can simply enter with the ticket(s) he has.

Necessary skills: algorithm design

Category: ad-hoc

COCI 2012/2013	**Task LJUBOMORA**
1st round, October 20th, 2012	**Author:** Adrian Satja Kurdija

Let **X** be the lowest possible envy level, and **Mx** the largest number of marbles of a single color. It is not difficult to show that for all numbers **Y** between **X** and **Mx**, inclusive, we can attain an envy level of **Y**.

It follows that binary search can be used to find the requested solution **X**. We can begin with the lower bound of 1 and upper bound of **Mx**. In each step, for a number **Y** halfway between the current bounds, we can check whether it is larger or smaller than **X**, i.e. whether it is possible to attain an envy level of **Y**.

How can we check that? We can iterate over all colours and divide each of them into as many portions of size **Y** as possible. More precisely, if we have **K** marbles of a given colour, we will make **K** div **Y** portions of size **Y**, as well as one more portion if we have any leftover marbles (if **Y** does not divide **K**). After we have distributed all colours in the manner above, if the total number of portions is at most **N**, we have successfully attained an envy level of **Y**, concluding **Y** ≥ **X**. Otherwise, we have to conclude **Y** < **X**, since we have made too many portions, which means that at least one child must get more than **Y** marbles.

Necessary skills: binary search

Category: binary search

An important observation is that, for any large **N**, the following number in the sequence – the smallest positive integer that doesn't divide it – is very small. Thus, there are very few possible numbers following any large number, hence it is natural to approach the problem as follows: for each of the possible followers **K** we can count the numbers between **A** and **B** followed by **K**, making it easy to compute the sum of their strengths – they al have a strength of strength(**K**) + 1.

How can we count the numbers for a given **K**? What are the conditions for a number **N** to be followed by **K**? It has to be divisible by all numbers less than **K** (otherwise one of them would be the follower of **N**), which means it is also divisible by their least common multiple, *LCM*(1, 2, …, **K** – 1). Furthermore, it must not be divisible by **K** itself. These two conditions are obviously both necessary and sufficient.

Thus, let us count the numbers between **A** and **B** divisible by *LCM*(1, 2, …, **K** - 1). From that number we need to subtract the numbers divisible by **K**. The numbers in the intersection of the two sets (divisible by both *LCM*(1, 2, …, **K** - 1) and **K**) are also divisible by *LCM*(1, 2, …, **K** - 1, **K**), so it is easy to find and subtract their count between **A** and **B**. Generally, we can count the numbers between **A** and **B** (inclusive) divisible by **D** using the formula (think about why it is correct): **B** div **D** - (**A** - 1) div **D**.

Finally, for which numbers **K** do we need to carry out the computation? From the discussion above, it is obvious that as soon as *LCM*(1, 2, …, **K** - 1) becomes greater than **B**, we are done: there are no numbers between **A** and **B** with such (or any greater) **K** (since they aren't divisible by *LCM*(1, 2, …, **K** – 1), being smaller than it). It turns out that the largest **K** will be less than 50. It is easy to compute the strength for such **K**, increment it by 1 and add it to the total sum as many times as there are numbers between **A** and **B** whose follower is **K**.

The last thing to consider is calculating *LCM*(1, 2, …, **K** - 1, **K**). Notice that

LCM(1, 2, …, **K** - 1, **K**) = *LCM*(*LCM*(1, 2, …, **K** - 1), **K**),

so if we have calculated **V** = *LCM*(1, 2, …, **K** – 1) in the previous step, then *LCM*(**V**, **K**) can be obtained using the formula **V** * **K** / *GCD*(**V**, **K**), which is valid for any two positive integers. *GCD*, the greatest common divisor, is easily obtained using Euclid's algorithm.

In the accompanying code, the *LCM* was computed using a different method. It is also worth taking a look at snaga_alternative.cpp which has a completely different solution.

Necessary skills: mathematical problem analysis

Category: number theory

We will describe two solutions based on dynamic programming.

For the purposes of both solutions, we will call sets of positions in the sequence {1, 2}, {3, 4}, {5, 6}, … the 2-structures, sets {1, 2, 3, 4}, {5, 6, 7, 8}, … the 4-structures, sets {1, 2, 3, 4, 5, 6, 7, 8}, … the 8-structures and so on. Two bacteria in the same 2-structure will be called siblings; if they are not siblings, but are in the same 4-structure, we'll call them first cousins; if they are neither siblings nor first cousins, we'll call them second cousins etc.

The first solution (devised by Luka Kalinovčić while solving COCI)

Imagine that we are adding bacteria into the sequence from left to right; then the state is described by the position in the sequence and the number of the bacterium placed in that position.

At first glance, these two parameters appear insufficient, but it can be shown that they are in fact sufficient. Let **p** and **p** + 1 be two adjacent positions in the sequence, with a bacterium marked **prev** placed in position **p**. Let us show that we can, without knowledge about the rest of the sequence constructed before position **p**, unambiguously determine which bacteria are candidates for placement in position **p** + 1 (thus giving us the set of possible following states, making the state transition in dynamic programming possible). With that goal, let us consider the following cases:

1) **p** and **p** + 1 are in the same 2-structure. It is clear that the only candidate for **p** + 1 is the sibling of **prev**.

2) Case 1) doesn't apply, but **p** and **p** + 1 are in the same 4-structure. Here candidates for position **p** + 1 are the two first cousins of **prev**.

3) None of the previous cases apply, but **p** and **p** + 1 are in the same 8-structure. Here candidates for position **p** + 1 are the four second cousins of **prev**.

...

K) None of the previous cases apply, but **p** and **p** + 1 are in the same 2**K**-structure (spanning all positions). Here candidates for position **p** + 1 are the 2**K** - 1 most distant cousins of **prev**.

Notice that this reasoning leads to the exact set of candidates for the next position, since none of the candidates could have been used before (take a moment to think and convince yourself).

The complexity of the algorithm is the number of states, O($\mathbf{N}^2$), multiplied by the transition complexity (number of candidates), which, at first glance, totals O($\mathbf{N}^3$). Since the number of candidates is 1 in ½ of cases, 2 in ¼ of cases, 4 in ⅛ of cases and so on, the actual complexity is even lower: O($\mathbf{N}^2$ log **N**).

The second solution

Let **dp**(**L**, **R**) be the smallest possible length of a subsequence starting with bacterium **L**, ending with bacterium **R**, and including precisely the bacteria which are descendants of the first (lowest) common ancestor of **L** and **R** (denoted by **LCA**(**L**, **R**)). The agenda is as follows:

1) Compute **dp**(**L**, **R**) for all **L**, **R** that are siblings.

2) Compute **dp**(**L**, **R**) for all **L**, **R** that are first cousins.

3) Compute **dp**(**L**, **R**) for all **L**, **R** that are second cousins.

...

K) Compute **dp**(**L**, **R**) for all **L**, **R** that are most distant cousins.

The final result will, of course, be the smallest of the numbers obtained in the last, **K**th step, since they cover all sequences containing all 2**K** bacteria.

Let us denote with **rep**(**x**, **y**) the repulsion between bacteria **x** and **y**. **dp** values in the first step are trivial to obtain, since **dp**(**L**, **R**) = **rep**(**L**, **R**). Other steps are a bit more complex, so bear with us.

For given **L** and **R**, we can try out all selections of bacteria **a** and **b** that "divide structures **L** and **R**", i.e. are exactly in the middle of the subsequence we're building between **L** and **R**, where **a** is the cousin closer to **L** than **R**, while **b** is the cousin closer to **R** than **L**. They cannot be too close cousins, since we have to fit half of all descendants of **LCA**(**L**, **R**) between **L** and **a**, and analogously for **b** – in other words, **LCA**(**L**, **a**) and **LCA**(**b**, **R**) must be the two children of **LCA**(**L**, **R**).

For fixed **L**, **R**, and a selection of bacteria **a** and **b**, the smallest possible length of the segment between **L** and **R** is obviously

$$\mathbf{dp}(\mathbf{L}, \mathbf{a}) + \mathbf{rep}(\mathbf{a}, \mathbf{b}) + \mathbf{dp}(\mathbf{b}, \mathbf{R}).$$

Therefore, **dp**(**L**, **R**) is equal to the minimum of this expression over all valid choices of bacteria **a** and **b**.

We need to find a fast method of computing this transition. If, for given **L** and **R**, we select some bacterium **a**, we need to minimize

$$\mathbf{rep}(\mathbf{a}, \mathbf{b}) + \mathbf{dp}(\mathbf{b}, \mathbf{R})$$

for all legal choices of **b**. Notice that this minimum doesn't depend on **L**. Let us denote it by **best**(**a**, **R**).

We can thus, before computing values of **dp** for the current step, precompute values of **best** needed for that step (where **best**(**a**, **R**) is obtained by trying out all valid values of **b**) and then use the **best** values to obtain **dp** values for that step more efficiently.

The complexity is obtained by summing the complexity for computing **best** values and for computing **dp** values. Both are at most O($\mathbf{N}^3$) since they compute O($\mathbf{N}^2$) values, each of which requires one for loop. Since this loop is often short because of relatively few valid choices, mathematically inclined readers are encouraged to determine whether this complexity is really O($\mathbf{N}^3$) or somewhat smaller, as in the previous solution.

Necessary skills: dynamic programming, binary trees

Category: dynamic programming