

# **CROATIAN OPEN COMPETITION IN INFORMATICS**

**Round 4**

**SOLUTIONS**

<b>COCI 2011/2012</b>	<b>Task KINO</b>
<b>Round 4, February 18<sup>th</sup>, 2012</b>	<b>Author:</b> Adrian Satja Kurdija

If there are no love seats, there are **N**+1 cup-holders. Every pair of love seats decreases the number of cup-holders by one. We can express this fact with the following formula:

$$\text{number\_of\_cup\_holders} = \mathbf{N} + 1 - \text{number\_of\_love\_seat\_pairs}$$

The solution is to output the number of cup-holders, except when there are no love seats in which case we output **N**.

**Necessary skills:** reading strings from input, *for* loop

**Category:** ad-hoc

<b>COCI 2011/2012</b>	<b>Task ZIMA</b>
<b>Round 4, February 18<sup>th</sup>, 2012</b>	<b>Author:</b> Bruno Rahle

First, we traverse the given day temperatures and locate the winter periods. Every winter period starts with a negative number, and lasts until a positive number is encountered.

If there is a winter period starting with day **A** and lasting **T** days, we mark all days from **A-2\*T** to **A-1** (inclusive) in some boolean array. This array tells us during which days are we allowed to announce the winter. We also keep track of the length of longest winter period so far.

Finally, we must choose one of the longest winter periods and allow it's announcing **3\*T** days in advance. We do this by checking for each of the longest periods by how much will our solution increase if we choose that period, and then choosing the one that was optimal. This checking is easy to do by using the same array of booleans as before.

**Necessary skills:** array manipulation

**Category:** ad-hoc

<b>COCI 2011/2012</b>	<b>Task KEKS</b>
<b>Round 4, February 18<sup>th</sup>, 2012</b>	<b>Author:</b> Ivan Katanić

After removing **K** digits from **N**-digit number, **N-K** digits will remain. So we can rephrase the problem statement and say that we must choose **N-K** digits that form the maximum possible number.

We will use greedy approach. We find the largest digit that still allows us to add additional digits after it, and place it as the first digit of our solution. We repeat this for second digit, and so on. If there are multiple choices at any time, it's easy to see that we can choose the leftmost one without harming our best solution.

If we didn't choose the largest possible digit at some point, we would obviously end up with a smaller number instead, which means that our algorithm is correct.

**Necessary skills:** math problem analysis

**Category:** greedy algorithm's

<b>COCI 2011/2012</b>	<b>Task OGRADA</b>
<b>Round 4, February 18<sup>th</sup>, 2012</b>	<b>Author:</b> Ivan Katanić

Let's mark the **i**-th smallest board that Mirko has with **A<sub>i</sub>**, and **i**-th board in the final arrangement with **C<sub>i</sub>**.

The niceness of Mirko's fence is equal to the sum of absolute differences between adjacent boards:

$$|C_1 - C_2| + |C_2 - C_3| + \dots + |C_{N-1} - C_N|$$

Since we know the orderings of each two adjacent boards, this sum can be rewritten without absolute values.

If we proceed to reduce this expression to canonical form, every **C<sub>i</sub>** will appear next to integer coefficient from interval  $[-2, 2]$ .

If we put board **A<sub>1</sub>** next to the smallest of these coefficients, **A<sub>2</sub>** next to the second smallest and so on, we would surely end up with the fence as nice as possible.

Now we must arrange boards in this way, but also in a way that assures that Mirko's fence is similar to Slavko's.

We use this algorithm:

**1.) Coefficients +2 and -2.** Boards with these coefficients are larger (smaller) than both boards adjacent to them. There are no adjacent positions with +2 (or -2) coefficient, so we can go and place largest boards next to +2's, and smallest boards next to -2's. It's easy to see that we won't violate any constraint by doing this.

**2.) Coefficients +1 and -1.** Only the first and the last board can have +1 or -1, so we put the largest board that's left from the first step next to +1 and the smallest one next to -1.

**3.) Coefficient 0.** These boards are smaller than one neighbor and larger than the other. By placing any of these boards next to boards already positioned in steps 1 and 2, we won't violate anything. So we only must be careful when placing them next to one another. Consecutive sequences of boards with coefficient 0 will be either strictly increasing or strictly decreasing, depending on adjacent boards at the beginning and at the end of sequence. We place them in any way that satisfies this condition and we are done.

**Necessary skills:** math problem analysis

**Category:** ad-hoc

<b>COCI 2011/2012</b>	<b>Task BROJ</b>
<b>Round 4, February 18<sup>th</sup>, 2012</b>	Author: Goran Gašić

To solve this for large values of **P** we will use modification of the sieve of Eratosthenes. Size of our sieve will be  $10^9 / \mathbf{P}$ . Integers in the sieve represent multiples of **P**. During the execution of this algorithm we can find smallest prime factors or mark only multiples of prime numbers smaller than **P** as in the official solution.

For smaller values of **P** we can binary search through  $[1, 10^9 / \mathbf{P}]$ , again looking at these numbers as the corresponding multiples of **P**. For some number we must find the number of integers not greater and relatively prime with that number. We can do this by using inclusion-exclusion principle with prime numbers less than **P**.

With careful implementation this solution can work for much larger values of **P** than requested for this subtask.

We can also solve this task for smaller value of **P** by making use of periodic behaviour of smallest prime factors. Let **A(n)** be the smallest prime factor of **n**, **B(k)** the **k**-th prime number, and **T(k)** the product of first **k** primes. For **A(n) ≤ B(k)**, **A(n + T(k)) = A(n)** holds. So it's enough to know **A(n)** for **n ≤ T(k)** in order to find the **N**-th prime who's smallest prime factor is **B(k)**

**Necessary skills:** sieve of Eratosthenes, inclusion-exclusion principle

**Category:** number theory, combinatorics

<b>COCI 2011/2012</b>	<b>Task KRIPTOGRAM</b>
<b>Round 4, February 18<sup>th</sup>, 2012</b>	Author: Goran Žužić

We can solve this task by modifying Knuth-Morris-Pratt (KMP) string searching algorithm. Let's introduce some notations.

- We will denote corresponding words of encrypted message with  $A[1], A[2], \dots, A[n]$ . Also,  $A[x, y]$  will denote the sentence made up of words  $A[x]$  through  $A[y]$

- $B[1], B[2], \dots, B[m]$  will be words from sentence of the original text, and  $B[x, y]$  sentence made up of words  $B[x]$  through  $B[y]$
- Let  $matches(A[x, x+L], B[y, y+L])$  be boolean function telling us whether  $A[x, x+L]$  can be decrypted into  $B[y, y+L]$ . For example,  $matches("a b a", "c d c") = true$ ;  $matches("a b b", "x y z") = false$ .

As in standard KMP, we will calculate the prefix function  $P[1, 2, \dots, m]$ , but with slightly different meaning.  $P[x]$  will be equal to largest possible  $L$  such that:

$$matches(B[1, L], B[x-L+1, x]) = True^1$$

After finding  $P$ , we must find  $B$  within  $A$ . For each word in  $A$  we are interested in largest suffix that corresponds to some prefix of  $B$ . If we encounter a mismatch, we continue with the largest possible prefix of  $B$ , which we lookup in  $P$ .

We must also find a way to efficiently evaluate  $matches$  function. We will transform our messages using the following transformation:

$$\begin{aligned} T(X)[i] &= -1 && \text{if } X[i] \text{ doesn't appear in } X[1, i-1] \\ T(X)[i] &= j && \text{if } j \text{ is the largest index such that} \\ &&& j < i \text{ and } X[j] = X[i] \end{aligned}$$

By using  $A' = T(A)$  i  $B' = T(B)$  we can calculate  $matches$  in linear time which is sufficient for obtaining the maximum number of points. Total complexity is also linear.

**Necessary skills:** Knuth-Morris-Pratt algorithm

**Category:** string searching

---

<sup>1</sup> Only difference between this definition and the original one is that we use our  $matches$  function instead of standard string comparison.