CROATIAN OPEN COMPETITION IN INFORMATICS 2012/2013

ROUND 6

SOLUTIONS

COCI 2012/2013	Task BAKA
6th round, March 9th, 2013	**Author:** Adrian Satja Kurdija

We have to do a for-loop through the string and for each character (letter) determine on which digit of the phone it is found.

Let us make an auxiliary string where we keep the starting letters on the digits (A, D, G, J, M, P, T, W). For a given letter we will iterate through the auxiliary string and locate the last letter in it that is less than or equal to the given letter (for example, if a given letter is O, the corresponding letter in the auxiliary string is M). We can now replace the given letter with the found letter since it is a letter that belongs to the same digit. Now we easily determine this digit **Z** using the position of the found letter in the auxiliary string and increase the final solution by **Z** + 1.

Neccessary skills: string operations, nested for-loop

Category: ad hoc

COCI 2012/2013	**Task SUME**
6th round, March 9th, 2013	**Author:** Adrian Satja Kurdija

We will leave the case **N** = 2 as a practice to the reader. If **N** is at least 3, observe the first three (unknown for now) elements of the array and their mutual sums (read from the matrix):

a + **b** = **s**1,
b + **c** = **s**2,
c + **a** = **s**3.

Summing these equations we get 2**a** + 2**b** + 2**c** = **s**1 + **s**2 + **s**3, and then **a** + **b** + **c** = (**s**1 + **s**2 + **s**3) / 2.

Now **a** = (**a** + **b** + **c**) - (**b** + **c**) = (**s**1 + **s**2 + **s**3) / 2 - **s**2.

Knowing the first element of the array, we easily determine the others: **i**th element is equal to the sum of the first and **i**th element (this sum is read from the matrix) decreased by the first element.

If we want to avoid the above mentioned math, limitations for the elements given in the task allow us to try all possible values for the first element of the array. For each of these posibilities we generate other elements of the array as shown in the previous paragraph and test the correctness of the array by summing any two elements different from the first. The question of why it works is left as an exercise to the reader.

For practice, consider this task where the cases that there is no solution or that there are infinitely many solutions are possible and to be detected.

Necessary skills: mathematical analysis of the problem

Category: math

COCI 2012/2013	Task DOBRI
6th round, March 9th, 2013	**Author:** Ivan Mandura

A naive solution, trying all possible combinations of three elements for every **i**, is too slow. Its complexity is O(**N**^{4}) and is worth 40% points.

Since the values in the array are small, we can have an array **P** which tells us if there exists a certain value in the array before the current position **i**. More precisely, **P[x] = 1** if there is a value **x** in the array **A** before the position **i**, else **P[x] = 0**. Using that, we can improve our starting solution. Instead of trying every possible combination of three elements, we try every possible pair of positions (**j**, **k**) less than **i** and ask if there is a value **A[i] - A[j] - A[k]** in the array before **i**. We have that information in the array **P** on the position **A[i] - A[j] - A[k]**. After processing the position **i**, we set **P[A[i]]** = **1**. We have thus achieved the complexity of O(**N**^{3}) and 70% points.

For 100% points we need an algorithm with a time complexity of O(**N**^{2}). Instead of asking if there is a value **A[i] - A[j] - A[k]** for each pair (**j**, **k**) in the array before **i**, we can ask for every position **j** if there is a pair of values before **i** such that their sum is equal to **A[i] - A[j]**. We can again use the array **P** to answer that, because the sum of two small numbers is also a small number. After processing the position **i**, for every pair (**i, j**) with **j** ≤ **i** we set **P[A[i] + A[j]]** = **1**. Using this optimization we get a solution that is fast enough and that achieves full points.

Notice that the space complexity of the algorithm is O(max $\mathbf{A_i}$), but if there were larger numbers in the task, we could use a balanaced tree instead of the array **P**. In C++ we can use set and map. We would thus get a solution of a space complexity O(**N**) and time complexity O(**N**^{2} log **N**) which was worth 70% points in this task.

Necessary skills: array operations

Category: ad hoc

COCI 2012/2013	Task BUREK
6th round, March 9th, 2013	**Author:** Adrian Satja Kurdija

Let us take a look only at the cuts x = **c** (solution for cuts y = **c** is analogous).

Notice the pastries that are cut by a line x = **c** are not completely left nor completely right to the line. The solution for this line is therefore:

N - number_of_pastries_left(**c**) - number_of_pastries_right(**c**).

We will calculate the function values of number_of_pastries_left(**x**) and number_of_pastries_right(**x**) before reading the cuts (therefore answering to each cut in a constant time complexity). We calculate the values using the following relation:

number_of_pastries_left(**x**) = number_of_pastries_left(**x** - 1) + number_of_pastries_with_a_rightmost_point_equal_to(**x**)

and an analogous relation for the second function. We read the values of the auxiliary function number_of_pastries_with_a_rightmost_point_equal_to(**x**) from an array whose elements we increase during the input of the pastries.

An alternative solution uses a sweep-line algorithm and is left as an exercise to the reader.

Necessary skills: precomputing

Category: sweep

COCI 2012/2013	Task JEDAN
6th round, March 9th, 2013	**Author:** Anton Grbin

The relief can be seen as a histogram. A histogram is **good** if it can be made using the moves shown in the task.

Notice that every histogram is good if and only if it satisfies the following three conditions:

- the first and the last column have a height of 0
- the difference between two adjacent columns is at most 1
- no column has a negative height

Proof. It is easily seen that a move in the task does not disrupt these three conditions. The first because we never increase the first or the last column, the second because we increase the column by 1 on the interval where every height is the same and we don't change the borders of the interval. The third because we only increase the relief.
Since the initial situation satisfies the conditions and a move does not disrupt them, we have proved one direction of the claim.

We now have to prove that every good histogram can be achieved by a series of moves.
Let us choose every column of a height 0 in a good histogram. There are at least two of them: the first and the last. Between every two non-adjacent zeros we make a "reverse move" by reducing every column between those two by one. Since the second condition is valid, we have at least doubled the number of zeros in the histogram using the moves we have made. We repeat this procedure until every column in the histogram has a height of 0. The moves we have made bring us to the starting histogram. Thus, the statement is proved.

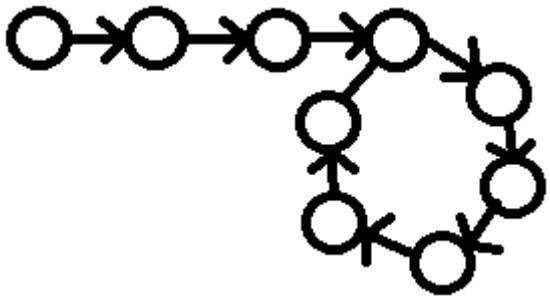
Solution
The solution is possible to construct in a quadratic complexity by calculating the number of ways to set hights of the first **K** columns so that the last column has a height of **H**, for each state (**K**, **H**). Since the height of this column depends only on the height of the previous column, the relation is of constant complexity. The solution is memory efficient if implemented iteratively, remembering **O**(**n**) states of the previous iterations.

Necessary skills: mathematical analysis of the problem, dynamical programming

Category: dynamical programming

HONI 2012/2013	**Task BAKTERIJE**
6th round, March 9th, 2013	**Author:** Ivan Katanić

Take a look at a bacterium: its state can be described using three parameters **X**, **Y** and **C**, where **X** and **Y** present a row/column mark of the cell in which the bacterium currently is, and **C** is the direction bacterium is facing. If we simulate the movement of the bacterium, we can notice the bacterium will eventually encounter a state it had been in before and, since the rules of moving do not change, the bacterium has closed the loop in which it will cycle forever (or by the end of the game). The path of the bacterium will be as shown in the picture:



Circles present the states of the bacterium, and arrows its movement. Part of the path which is not in the loop is called "tail". Let $\mathbf{L_i}$ be the number of states in the loop of the **i**-th bacterium, and $\mathbf{T_{i,X,Y,C}}$ the second in which the bacterium first encountered a state (**X**, **Y**, **C**). If the state (**X**, **Y**, **C**) is on the "tail", this second is the only one in which the bacterium is in a state (**X**, **Y**, **C**), and if it is in the loop, it will be there in seconds $\mathbf{t} = \mathbf{T_{i,X,Y,C}} + \mathbf{k} * \mathbf{L_i}$, $\mathbf{k} \geq 0$.

Assume that in the final second of the game the bacterium ***i*** will be in a state ($\mathbf{X_e}$, $\mathbf{Y_e}$, $\mathbf{C_i}$), where $\mathbf{X_e}$, $\mathbf{Y_e}$ marks a trapped cell. If some bacterium is in that state for the first and the only time (regardless of the game end), then it is trivial to check if other bacteria will be in the final state in that second. If not, we have a system of **K** equations:

$$\mathbf{t} = \mathbf{T_{i,X,Y,}}_{C} + \mathbf{k_i} * \mathbf{L_i},\ \mathbf{k_i} \geq 0,$$

which can also be presented using congruences:

$$\mathbf{t} \equiv \mathbf{T_{i,X,Y,C}} \pmod{\mathbf{L_i}},\ \text{with}\ \mathbf{t} \geq \mathbf{T_{i,X,Y,C}}.$$

This problem is solved using a Chinese remainder theorem (http://en.wikipedia.org/wiki/Chinese_remainder_theorem). Since the theorem offers a solution if the modules ($\mathbf{L_i}$ in our case) are pairwise comprime, we have to split every $\mathbf{L_i}$ to comprime factors (powers of primes). We thus get a greater number of equations but the modules will

now be pairwise comprime. In case there are two powers of the same prime number **p** among the modules, we can eliminate the smaller one because the remainder of division by it is uniquely determined from the remainder of divison by the greater one (if they do not match, there is no solution).

After we find the solution, we cannot forget the conditions $\mathbf{t} \geq \mathbf{T_{i,X,Y,C}}$: if our solution **t** does not satisfy them, we simply increase it sufficient number of times by the product of all the modules because the Chinese remainder theorem says that all solutions to the system are congruent modulo this product. Since the maximum loop length is 10 000 and every loop is of even length, the product of modules, as well as the solution, fits in a signed 64-bit integer.

Now we just try all $4^{\mathbf{K}}$ combinations for the third parameter $\mathbf{C_i}$ of final states and take the one that gives us the smallest solution.

Necessary skills: Chinese remainder theorem

Cathegory: math