CROATIAN OPEN COMPETITION IN INFORMATICS

Round 2

ALGORITHM DESCRIPTIONS

COCI 2011/2012	Task NAJBOLJIH 5
Round 2, November 19th, 2011	**Author:** Nikola Dmitrović

It is possible to solve this task in various ways. Expect for the most obvious solution where one looks for the 5 wanted numbers with 5 nested loops, we could just search for the 3 numbers with minimum sum and use 3 nested loops, which simplifies the code.

```
min=total; //sum of all numbers

for i=1 to 6 do
      for j=i+1 to 7 do
            for k=j+1 to 8 do
                  if score[i]+score[j]+score[k] < min tada
                  begin
                  min:=score[i]+score[j]+score[k];
                  first:=i;
                  second:=j;
                  third:=k;
                  end;

print (total-min);

for i=1 to 8 do
     if (i != prvi) and (i != drugi) and (i != treci) then
      print (i);
```

Required skills: nested loops

Category: ad hoc

COCI 2011/2012	Task OKRET
Round 2, November 19th, 2011	**Author:** Adrian Satja Kurdija

There are dead-ends inside the given city if and only if there are road surface cells that are adjacent to only one other road surface cell.

Let’s prove this. If there is a road surface cell (let’s call it A) that has only one neighbor (B), then by moving from B to A we get stuck in A, i.e. our only way out is by a 180 degrees turn, which means that dead-end exists.

If there is no such cell, than we can exit every cell using some direction other than the one by which we came in. Let’s start at some road surface cell using any direction and proceed in the following matter: leave the cell we are currently in by using direction other than the one by which we came in. Since there are finite number of free cells, we will sooner or later enter some cell which we already visited. When this happens, one of these two statements are true: either we returned to the starting point, or we can return to the starting point using the same path. In both cases it’s possible to get back to the starting point, which concludes our proof.

Described solution is very easily implemented: for each road surface cell count the number of free cells.

Required skills: matrices manipulation
Category: ad hoc

COCI 2011/2012	**Task ZADAĆA**
Round 2, November 19th, 2011	**Author:** Ivan Katanić

Greatest common divisor of two integers can be defined as the product of their common prime factors, as following:

$$A = p1^{a1} * p2^{a2} * \ldots * pn^{an} \qquad\qquad B = p1^{b1} * p2^{b2} * \ldots * pn^{bn}$$
$$GCD(A, B) = p1^{\min(a1,b1)} * p2^{\min(a2,b2)} * \ldots * pn^{\min(an,bn)}$$

where p1..pn are the prime factors and a1..an, b1..bn are corresponding exponents.
We can get the factorization of large numbers A and B by factorizing every of their given factors and summing the prime number exponents over some prime in all factorizations. Next step is computing the GCD using the expression given above. For details check out the attached code.

Alternative solution would be to find GCD of all pairs of numbers A_i, B_j and it to the result (multiply), and divide the numbers A_i, B_j with the same number to prevent adding it to the result several times (in the next iterations).

Required skills: factorization, prime numbers

Category: number theory

COCI 2011/2012	Task KOMPIĆI
Round 2, November 19th, 2011	**Author:** Adrian Satja Kurdija

First thing to notice is that for each of the input values only set of it's digits is of importance. We are not interested in order in which digits appear or repetition of digits. Therefore, each value can be represented with sequence of 10 binary digits - 1 if that digit is present, and 0 if it isn't.

There are at most 2^{10} = 1024 different sequences. For each sequence, we can easily calculate how many input values yield exactly that sequence, and store these results into some array.

For each pair of sequences, it's easy to tell if they share some digit - they do if there is a position at which both sequences have ones. If they don't share a digit, there are no pals here. If they do, than we can form a pair of pals by choosing any value that yields the first sequence, and any value that yields the second sequence. Total number of such pairs is:

number_of_values[sequence1] * number_of_values[sequence2].

Finally, we must count the number of pals that have the same sequence:

number_of_values[sequence] * (number_of_values[sequence] - 1) / 2

We must go through every possible pair of sequences, so complexity is O(1024^2).

Required skills: binary number system

Category: ad hoc

COCI 2011/2012	Task FUNKCIJA
Round 2, November 19th, 2011	**Author:** Goran Gašić

First observation we can make is that loops actually represent a system of inequalities:

X1 ≤ a ≤ **Y1**

X2 ≤ b ≤ **Y2**

…

XN ≤ <***N***-*to*> ≤ **YN**

Solution to our problems is the number of integer solutions of the given system of inequalities modulo 1000000007.

Lets build a graph with **N** nodes, each node representing one inequality. From a node which represents inequality of **var1** we'll put an edge towards node of **var2** if upper or lower bound of **var2** is equal to the same-kind bound of **var1**.

This graph is disjunct union of directed rooted trees. Since variables in different trees are independent of each other, the number of solutions for inequality system is equal to the product of number of solutions for each of the trees. Let us demonstrate how to calculate the solution for only one tree.

Let f(**root**) is equal to the number of solutions of the tree rooted at **root**. Let g(**node**, **number**) equal to the number of solutions of a subtree rooted at **node**, if variable bound for that node inequality is equal to **number**. We'll now make the following claims

$$f(root) = \sum_{i=X(root)}^{Y(root)} \prod_{child \in children(root)} g(child, i)$$

$$g(node, number) = \sum_{i=number}^{Y(number)} \prod_{child \in children(node)} g(child, i)$$

for inequality with variable lower bound (it is analog in the case of variable upper bound). This algoritm has complexity of O(**NM**^{2}), where **M** is a limit on the upper bound, which is good enough for 70% of the points. Further, we can notice that following holds (for the variable lower bound case - as before it is analog with variable upper bound):

$$g(node,number) = g(node,number+1) + \prod_{child \in children(node)} g(child,number)$$

Using this observation, complexity becomes O(**NM**) and this wins 100% points.

Required skills: dynamic programming

Category: dynamic programming, graph theory

Let's say that we are given some permutation (order in which pizza's are made) π_i, and we wish to calculate the tip we'll get:

$$\sum_{i=1}^{n}(R\pi_i - \sum_{j=1}^{i} V\pi_j) = \sum_{i=1}^{n} R\pi_i - \sum_{i=1}^{n}(n-i+1)V\pi_j$$

We can now conclude that order with regard to deadlines is not important, and that order with regard to durations must be increasing, so that pizza's that are baked longer are multiplied by smaller coefficient.

Brute force solution that sorts the pizzas after each update and calculates the above sum has complexity O(N log N) or O(N) and is worth 50 points.

Constraint given to $\mathbf{V_i}$ was sort of a hint that can lead to a simple solution: in some array **cnt** we can use **cnt[x]** to store the number of pizzas having baking time **x**. For $\mathbf{V_i}$ under 1000 we calculate every query by traversing the **cnt** array. This approach was worth 80 points.

We can use some data structure like segmented array (BIT) for implementing the **cnt** array, and aditional sequence that keeps the $\mathbf{V_i}$ sums (**sum[x]** = **cnt[x]*x**). To remove duration $\mathbf{V_p}$ for some pizza **p**, we must increase our solution by

$$V_p \cdot \sum_{i=0}^{V_p} cnt[i] + \sum_{i=V_p+1}^{maxV} sum[i]$$

Now we decrease **cnt[$\mathbf{V_p}$]** by 1 and **sum[$\mathbf{V_p}$]** by $\mathbf{V_p}$. In order to insert $\mathbf{V_p}$ into our structure, we do the same thing, but with inversed signs and order (change **cnt** and **sum** first and then decrease the solution). $\mathbf{R_i}$ sums are trivial to calculate and don't require any data structures.

We can answer each query in O(log maxV), and total complexity is O((N+P) log maxV) which is good enough for obtaining maximum points for this task.

There is alternative solution that doesn’t depend on maxV. Idea is to maintain the sorted sequence of all the baking durations, along with the sums and corresponding coefficients. This can be achieved by using balanced tree (online solution), and by using segmented array built on top of the sorted sequence of all the durations (offline solution).

Complexity of this approach is O((N + P) log (N + P)). Offline implementation can be found in official solutions.

Required skills: mathematic problem analysis, data structures

Category: data structures