CROATIAN OPEN COMPETITION IN INFORMATICS

4th ROUND

SOLUTIONS

COCI 2010/11	Task AVION
4th round, January 15th, 2011	**Author:** Bruno Rahle

This problem is straightforward. For each of the five lines of input, we check whether it contains a substring "FBI". We can use existing functions: *pos* in Pascal, *strpos* in C, and *string.find()* in C++. We must also detect if the substring is not present in any of the given strings. This is done easily. At the beginning, we set some flag to zero. If we find the substring "FBI", we set that flag to one. At the end, if flag is still zero, we output "HE GOT AWAY!".
This problem can be solved using loops, but this is not required since the number of given strings is always 5.

Necessary skills:

String manipulation

Tags:

Strings, ad hoc

Notice that the number of red blocks, **R**, cannot be smaller than any of the two dimensions of Ivica's room. Since **R** is not greater than 5000, both numbers **L** and **W** are also not greater than 5000. Therefore, it is possible to check all the possible combinations for **L** and **W**.

For any **L** and **W**, it is easy to find corresponding **R** and **B**:

$$R = 2L + 2W - 4$$
$$B = (L-2)(W-2)$$

If those numbers are the ones we are looking for, we output **L** and **W**.

Necessary skills:

Basic arithmetic operations, area and circumference of a rectangle

Tags

Ad hoc

First, convert all timestamps to minutes. Now, find the minute at which both stars flash. How? Let **A** and **B** be minutes at which the stars flashed and let **P** and **Q** be the periods (in minutes) between consecutive flashes, accordingly. The first star thus flashes at the following minutes: **A**, **A**+**P**, **A**+2**P**, **A**+3**P**, … More generally: it flashes at minutes **A** + **K*****P** for non-negative integers **K**.

Assume that flash overlapping, if it occurs, will occur for **K** less than 10^6 (it can be proved that taking **K** less than 3000 suffices, which is left as an exercise for the reader). For each such **K**, we determine whether the second star will flash at the minute **A** + **K*****P**. Note that this will happen whenever the equation **A** + **K*****P** = **B** + **L*****Q** holds. Therefore, it is sufficient to check whether **L** = (**A** + **K*****P** **-** **B**) / **Q** is a non-negative integer - in that case the flash occurs, indeed.

Once the first overlapping is found, the corresponding minute needs to be converted to days and hours and outputted. In case none was found for any **K**, we output "Never ".

Necessary skills:

Time conversion (minutes, days and hours)

Category

Ad hoc

COCI 2010/11	**Task PROSJEK**
4th round, January 15th, 2011	**Author:** Goran Gašić

First, let us determine the minimum number of integers needed to obtain the average of **P**.

Let **S** = {**n1**, **n2**, **n3**, **n4**, **n5**} be the solution to the problem and let **n = n1 + n2 + n3 + n4 + n5**. The following holds:

(n1 + 2*n2 + 3*n3 + 4*n4 + 5*n5) / n = P
(n1 + 2*n2 + 3*n3 + 4*n4 + 5*n5) = n * P

It can be inferred that the right side of the equation is an integer.
P can be expressed in the form of a reduced fraction **a/b**, where **a** and **b** are relatively prime.

(n1 + 2*n2 + 3*n3 + 4*n4 + 5*n5) = n * (a/b)

The equation holds if and only if **n** is a multiple of **b**. Therefore, the minimum **n** is **not less than b**. We now show how to obtain a solution where **n = b**.

Observe that with **n** integers we can obtain any sum in the interval **[n**, **5n]**.
Moreover, **a** ≤ **5b**. Considering these facts, the number of each integer can be found with a greedy algorithm. Starting from the greatest one, we pick each integer maximum number of times, such that it is possible that the remaining sum can be obtained using the remaining number of integers.

Necessary skills:

Euclid's algorithm

Category

Number theory, greedy algorithms

COCI 2010/11	**Task DUGOVI**
4th round, January 15th, 2011	**Author:** Adrian Satja Kurdija

First, let us create a graph in the following manner: if person A owes money to person B, we create a directed edge between vertices A and B with cost C. One important property of such a graph is that every vertex has output degree equal to 1 (input degree varies).

Further, let us assume that graph is connected - if we know a solution for such a graph, we can solve general graph case by solving each component separately and summing the results.

Looking at one component we can observe that exists exactly one cycle. We can prove that by induction. Base of the induction is a component with 2 vertices. It is obvious that it contains one cycle. Let us now assume that this claim is true for every component which contains **N** vertices. If we now look at a component with **N**+1 vertices, we can observe that there are two cases - if the input degree of each vertex is equal to 1 it is trivial to show that all of the **N**+1 vertices form a cycle. If there is a vertex with input degree equal to zero, we can erase that vertex - now we have component with **N** vertices which does contain a cycle (according to induction).

Each vertex with input degree of zero has to get the amount of money equal to the total debt of the corresponding person. After that step we can erase each such vertex, but we keep track that neighbor vertices now have a starting budget. These steps are repeated until there are no more vertices with input degree of zero (notice that by erasing a vertex, we change input degree of neighboring vertices).

At this point, there is only one cycle remaining. If there is a vertex which has enough money to return its debt, we return its debt and erase its outgoing edges. Solution for the case when no such vertex exists is described in the next paragraph.

For every vertex **X** we will compute how much money it must get from the city after it gets money from its debtor which lies on the cycle - let **f(X)** denote that amount. We will donate that amount to every vertex **X** on the cycle, except for the one from which debt-returning-chain-reaction is started - that vertex will get the amount it needs, assuming that it gets no money from its debtor - let **g(X)** denote that number. If $\mathbf{X_1}$, $\mathbf{X_2}$, …, $\mathbf{X_n}$ are elements of the cycle, and if $\mathbf{X_k}$ is a starting point of the chain reaction, the total amount of donated money is $\mathbf{f(X_1)} + \dots + \mathbf{f(X_{k-1})} + \mathbf{g(X_k)} + \mathbf{f(X_{k+1})} + \dots + \mathbf{f(X_n)} = \mathbf{S} - \mathbf{f(X_k)} + \mathbf{g(X_k)}$, where **S** is the sum of all **f(X)**, and is a constant. It is clear now that we are looking for the minimum of all **S - $f(X_k)$** + **$g(X_k)$**, which can easily be found by iterating through all vertices on the cycle.

Necessary skills:

Basic graph theory

Category

Ad hoc

COCI 2010/11	**Task HRPA**
4th round, January 15th, 2011	**Author:** Goran Žužić

The idea of solving this kind of problems consists of creating a brute force solution for small input instances (**N** $\leq$ 1 000) and observing solution patterns for solving big cases.

Solution for small **N** uses a dynamic programming approach. Let dp[**x**][**k**] denote if a player can win regardless of its opponent's actions (winning position) if there are x pebbles remaining, and he can take at most 2***k** rocks in the current turn. These are the possible transitions:

dp[**x**-1][1], dp[**x**-2][2], dp[**x**-3][3], …, dp[**x**-2***k**][2***k**]

under the assumption that there are enough rocks (2***k** $\leq$ **x**), otherwise it is allowed to take x rocks at most. If every neighbouring state is a losing position, we can conclude that our current state is a winning position. From these observations we can quickly create a O($\mathbf{N}^3$) algorithm.

Observing the outputs for small **N** we can notice that solutions we are looking at are in fact Fibonacci numbers. We can also observe that most of the solutions are not too big (except for the Fibonacci numbers and numbers which can be written as sums of Fibonacci numbers - their solutions are big). From these facts our intuition tells us that we might be able to find a solution if we try to disassemble the number **N** as a (unique) sum of non-consecutive Fibonacci numbers - the solution is then in fact the smallest number in the sum. That algorithm can be implemented in O(log **N**) complexity.

For those who want to learn more about this type of problem - try to look for the math game called "Fibonacci Nim".

Necessary skills:

Game theory, dynamic programming, mathematics

Category

Game theory