Gran Premio de México 2019

Segunda Fecha

May 11th, 2019

Contest Solutions

This document contains expected solutions for the 12 problems used during the Contest session.

v1.8

Problem A – Assigning company branches.

Author: Juan Pablo Marín

Suppose we take two points p_1 and p_2 and draw the line such that it touches both of them, we can slightly rotate this line from that point in a way that p_1 and p_2 are on the different sides of it. Then, for these two points, there are two ways to draw the line, in the first case p_1 is on the left side of the line and p_2 to the right, on the other case p_1 is on the right side and p_2 on the left side.

To determine which side of the line from $p_1 = (x_1, y_1)$ to $p_2 = (x_2, y_2)$ a point $p_x = (x, y)$ falls on, compute the value: $d = (x - x_1)(y_2 - y_1) - (y - y_1)(x_2 - x_1)$ If $d < 0$ then the point lies on one side of the line, and if $d > 0$ then it lies on the other side.

A special case comes when point $d = 0$ for p_x given p_1 and p_2, we can see if p_x, a value of $d = 0$ means p_x is in the same line, see that if p_x is not between points p_1 and p_2 in the line, then, the side of p_x is the same as p_1 or p_2 based on the closest of these points to p_x. If p_x is between p_1 and p_2, then, there are two lines: p_1 to p_x, and p_x to p_2 that may not have points in between and to which we can find a side for any given point in the input, whose will be processed in the case where $p_2 = p_x$ and when $p_1 = p_x$ respectively, then, we can ignore the lines that contain points in between.

Now, for each pair of points p_1, p_2 it is possible to compute in $O(N)$ what would be the best of the two possible ways to draw the line that has p_1 on one side and p_2 on the other side. Choosing the one with minimum from all possible pairs can be done in $O(N^3)$ as there are $O(N^2)$ possible pairs to choose points p_1,p_2.

It is important to do all the computations in integer numbers.

Problem B – Buggy visit counter.

Author: Moroni Silverio

A first method to find the answer is to simulate the visit counter, in this we move from 0 to C times obtaining the next value, this has $O(C)$ complexity and given C is very large this will not run in time.

A more efficient approach is to count the number of elements the visit counter generates each time it returns to a negative number. We can see for the first time it will generate numbers from 1 to L, on the second one from -1 to $L+1$, the third time it will go from -2 to $L+(1+2)=L+3$. Lets see a graphic representation of this:

```
            |         |                                   row
          0 | 1 2 ... | L                                   1
       -1 0 | 1 2 ... | L L+1                               2
    -2 -1 0 | 1 2 ... | L L+1 L+2 L+3                       3
 -3 -2 -1 0 | 1 2 ... | L L+1 L+2 L+3 L+4 L+5 L+6           4
            |         |
```

Generally speaking we can see the $i-th$ row starts in the value $-(i-1)$ and ends in the value $L+\frac{(i)(i-1)}{2}$. Let's suppose a function $elems(i)$ returns the number of elements in row i, we can see if we find the smallest value k such that $\sum_{i=1}^{k} elems(i) > C$ we know that the value we are looking for is in the k-th row. So this solution involves finding the value for k which can be done summing the number of elements on each row until we can not sum another row, and then our solution $v = -(k-1)+C-\sum_{i=1}^{k-1} elems(i) = C+1-k-\sum_{i=1}^{k-1} elems(i)$. This solution while more efficient than the first one does not run in time.

To optimize this solution we can make the sum of elements faster, notice from the table above, that the left side of the first bar up to row k contains $1+2+3+....+k=\frac{(k)(k+1)}{2}$ elements, the part between the bars contain each $L-1$ elements then it has $k*(L-1)$ total in the k rows, and the part at the right of the last bar contains $k+\sum_{i=0}^{k-1}\frac{(i)(i+1)}{2}$ elements on the k rows, then, to count all the elements we need to sum these three values:

$S(k) = \frac{(k)(k+1)}{2} + \sum_{i=0}^{k-1}\frac{(i)(i+1)}{2} + k + k*(L-1) = \sum_{i=0}^{k}\frac{(i)(i+1)}{2} + k*L$

$S(k) = \frac{(k)(k+1)(k+2)}{6} + k*L$

Using this formula, we can then use binary search to find the value of $k-th$ row we have summed in the previous solution and apply the same maths described to find the value for v. This solution is at most $O(LogC)$ and will run in time.

Problem C – Credit card PIN number.

Author: Saraí Ramírez

In order for Febo to create the NIP with the given restrictions, he can select for the first digit of the PIN any of the 10 digits $0, 1, 2, ..., 9$. For the second digit of the PIN, Febo can select any digit but the one taken in the first digit, then, on the second position he can select only one from 9 possible digits, the same applies for all other PIN digits.

The different PIN numbers Febo can select, is the product of the different choices he can make on each position, then for a pin of K digits, he can select $10 \times 9^{K-1}$ different PIN numbers.

Implementations that run in the time limit will use a fast exponentiation method to answer each query $O(logK)$, or memorize the value of all powers of 9 up to 10^6 to answer each query in $O(1)$

Problem D – Delivery schedules.

Author: Moroni Silverio

Lets consider the Floyd Warshall algorithm. In this algorithm in the $k - th$ iteration, the matrix of distances has the shortest path for all pair of vertices considering all paths that contain the first k vertices in the paths. When no iteration has been done, the distances matrix contains all the shortest paths without any city in the path (edges of the graph). Then, if an employee does not want to take the paths that go through some city c, we can have the paths the employee will take in the iteration of the Floyd Warshall algorithm just before processing city c.

The way the employees information is given in the input, guarantees that the last employee in the list is the one that has the longest list of cities to not visit, then, we can sort the vertices in such way that they are processed using the Floyd Warshall algorithm in a way that we can answer the paths for the employees from the last one to the first one in the list.

As an example consider the list of cities in the input to be :

```
Employee    List in input    Cities employee does not want to go
1           7                 {7}
2           6 5               {5 6 7}
3           4                 {4 5 6 7}
4           3 2               {2 3 4 5 6 7}
```

It can be seen if we process the vertex 1 then we can answer the paths for employee 4, then we process vertices 2 and 3, and can answer the paths for employee 3, then we process vertex 4 and answer for employee 2, last we process vertices 5 and 6 to answer for employee 1.

As this required only a ordering or relabeling of the vertices before running Floyd Warshall algorithm, in the worst case it takes $O(E^3)$ to have all the paths of all employees, plus O(1) to count each of the employees paths, as there are at most 500 jobs per employee, there will be no more than 500*E paths to consider to answer for all employees.

Problem E – Exponential points game.

Adapted by: Juan Pablo Marín

The original idea of this problem was taken from the problem ”TheXGame”, used as TopCoder's SRM 304, Division 1, Level 3.

As prerequisite it is required to understand the game of Nim, some information and a demonstration of the winning strategy can be found here: https://en.wikipedia.org/wiki/Nim

If we can prove that the problem has the same winning strategy as in the game of Nim then, we can find if Santiago will win with the first move.

Let's consider an unmarked tile in the game correspond to the stones in the Nim problem. The contiguous sequences of unmarked tiles correspond to the heaps. Marking some consecutive tiles is as taking some stones from one heap, but, this can also lead to split the stone in two heaps, which is a move not included in the Nim problem. Adding such move does change the xor sum strategy to win, as the move of taking some stones from a heap of size x and getting two heaps of sizes y and z, brings us to the same xor sum as we would if we removed (x - (y xor z)) stones from the heap.

Let's prove by induction that if you have n stones in total at turn m and the xor sum of the remaining heaps is 0, then, Santiago will win by a minimum of $\frac{n}{2} * 2^{m-2}$ points. The base case is when the game ends in two moves after Santiago's first move, then on turn $m = 2$ you are faced with two heaps, and if the xor sum of the sizes of the heaps is 0 then the two heaps have equal size, so, both of you took $\frac{n}{2}$ stones, then, you get $\frac{n}{2}$ points, and Santiago gets $\frac{n}{2} * 2$, he has $\frac{n}{2} * 2^{2-2}$ points more than you, which satisfies the induction hypothesis.
Now, if it is not the last two moves in the game, you can take at most $\frac{n}{2}$, because for the xor sum to be 0, the largest heap in the game can't be of size bigger than $\frac{n}{2}$, suppose you take the $\frac{n}{2}$ stones (the maximum points you can get in the turn, leaving Santiago with the minimum points he can take), and Santiago's take x stones after that, this leads to a smaller game with $\frac{n}{2} - x$ stones. By our induction hypothesis, Santiago will win this game by at least $\frac{\frac{n}{2}-x}{2} * 4 * 2^{m-2}$ points, then, from the previous turns Santiago earned $x * 2^{m-2} * 2$ points, and Santiago will earn at least: $x * 2^{m-2} * 2 + \frac{\frac{n}{2}-x}{2} * 4 * 2^{m-2} - \frac{n}{2} * 2^{m-2}$, more points than you, the value $\frac{n}{2} * 2^{m-2}$ is substracted since that is the points you got from your move at turn m. Then:
$x * 2^{m-2} * 2 + (2 * \frac{n}{2} - 2x) * 2^{m-2} - \frac{n}{2} * 2^{m-2}$
$x * 2^{m-2} * 2 + (n - 2x) * 2^{m-2} - \frac{n}{2} * 2^{m-2}$
$x * 2^{m-2} - 2x * 2^{m-2} + n * 2^{m-2} - \frac{n}{2} * 2^{m-2} = \frac{n}{2} * 2^{m-2}$ Which proves our induction hypothesis.

Now to solve the problem we can make every possible first move of Santiago and see if the nim problem has the xor sum equal to 0, keep track of the minimum number of tiles in the move and output it, if no first move for Santiago turns in a nim problem where the xor sum equals 0, then, there is no first move Santiago can make to win.

Problem F – Forgotten PIN number.

Author: Saraí Ramírez

Let F be the forgotten PIN, N the new PIN, and k the number of digits in the PIN. P_i is the property, that, in the new PIN, the digits F_i and F_{i+1} (making an invalid PIN) appear together for all i $1 \leq i \leq k-1$, and let A_i the set of PINs that satisfy condition P_i. As we have k digits, we can have $k-1$ pairs of contiguos pairs in the PIN.

By the inclusion and exclusion principle, the number of PINs that satisfy at least one of the P properties is:

$$\bigcup_{i=1}^{k-1} |A_i| = S_1 - S_2 + ... + (-1)^k S_{k-1}$$

Where:
$S_1 = |A_1| + |A_2| + ... + |A_k|$
$S_2 = |A_1 \cap A_2| + |A_1 \cap A_3| + ... + |A_{k-2} \cap A_{k-1}|$
$S_3 = |A_1 \cap A_2 \cap A_3| + |A_1 \cap A_2 \cap A_4| + ... + |A_{k-3} \cap A_{k-2} \cap A_{k-1}|$
...
$S_{k-1} = |A_1 \cap A_2 \cap ... \cap A_{k-1}|$

Lets focus on obtaining the values for each S_i. Suppose we take property P_i. To count the PINs that can be formed satisfying at least that property, we will see both F_i and F_{i+1} as a single element. This way there are $k-1$ digits that can be added in any of the $k-1$ positions of the NIP, where no repetitions are allowed, which can be counted with $(k-1)!$. Considering that there are $k-1$ possible properties, then:

$$S_1 = (k-1) * (k-1)!$$

For S_2, lets take properties P_i and P_j such that $i \neq j$. There are two possibilities, the pairs for both conditions are disjoint sets, or, both share exactly one element. In the first case there are $k-4+2$ digits to put in $k-2$ positions. In the second case, there are $k-3+1$ digits for the same $k-2$ positions. Since we can choose $\binom{k-1}{2}$ pairs of propertiee then:

$$S_2 = \binom{k-1}{2} * (k-2)!$$

Generally speaking, taking h different properties, if all them are disjoint, then, there are $k-2h+h = k-h$ digits to be positioned in the invalid PIN. If, there are l properties that share a digit, we have $k-h+l$ digits to be positioned, however we need to take l, the number of properties that are omitted when taking the non disjoint properties, therefore, regardless the number of digits the h properties share, there are $k-h+l-l = k-h$ digits to be positioned in $k-h$ positions for the invalid PIN. As there are $\binom{k-1}{h}$ ways to take the h properties, then:

$$S_h = \binom{k-1}{h} * (k-h)!$$

Then, the number of invalid PINs is:

$$\sum_{i=1}^{k-1} (-1)^{i+1} \binom{k-1}{i} (k-i)!$$

Each element of this series can be represented as follows:

$$\binom{k-1}{i}(k-i)! = \frac{(k-1)!}{i!(k-1-i)!}(k-i)! = (k-1)!\frac{k-i}{i!}$$

Finally:

$$(k-1)! \sum_{i=1}^{k-1} (-1)^{i+1} \frac{k-i}{i!}$$

To properly compute this value it is required to compute the multiplicative inverse of $i!$ with respect to the modulo $10^9 + 7$ in order to perform the division.

Problem G – Gato currency converter.

Author: Moroni Silverio

This is a very easy problem. To convert from n pesos to gato just take the integer part of dividing n by 5, and if the division is not exact add 1 to the result.

It is important in the implementation to properly reference the number Jaime queries and to consider that a product can be queried more than once.

Problem H – Hidden number.

Author: María Celeste Ramírez

This problem is a variation of the subset sum problem. An approach can be, to use an array S to set $S[i] = true$ if, i can be represented as the sum of a subsequence in Santiago's list. Then we can setting all values of S to false, $S[0] = 1$ and then for each number L_i from the list set $S[i + L_i] = true$ for all i such that $S[i] = true$. The answer is the smallest value i such that $S[i] = false$.This requires S to have at least $MAXSUM + 1$ elements (in worst case 10^{12}), where $MAXSUM$ is the sum of all elements in the list. This solution takes $O(N * MAXSUM)$ time, and will not run in time.

See that, if the list does not contain the value 1, then, the smallest number that can not be represented as a sum of any subsequence in the list is 1. Also, in general, a positive number n represented as sum of positive numbers can be represented only by numbers less or equal to n then, to verify if this number n can be represented as the sum of a subsequence in L we need only the values smaller or equal to it. Suppose, we know all the values from 1 to k can be represented as the sum of a subsequence of the first i numbers of the list L sorted in increasing order, then, we can obtain all the values $1 + L_{i+1}, 2 + L_{i+1}, ..., k + L_{i+1}$. If, $L_{i+1} > k + 1$, then we know $k + 1$ can not be obtained as the sum of any subsequence in L. This solution involves sorting the list and then iterate on each value of the sorted list in the worst case, complexity would be of the sorting algorithm wich can be O(N) using bucket sort.

Problem I – ICPC training.

Author: Juan Pablo Marín

Suppose you can't train on a given day, the only thing you can do is to rest. Otherwise you have to choice between to train or to rest. Let's suppose you already trained $M - 1$ days, if you choose to rest this day instead of training, then, you are not training in the minimum number of days to achieve the M train days. Therefore, the solution is to train if it is possible and to rest if it is not possible to train.

Doing a simulation of the previous method is sufficient to get an answer that runs in the given time limit. Two conditions should be considered to print -1, the first, is, if already $N + 1$ days have been simulated, the other is when $t > T$, since you can not train at least 1 day. The simulation will take at most $O(N)$ which runs in the given time limit.

Problem J – Jaime's multiplications.

Author: Moroni Silverio

First thing to notice, is that, for a number N to be a perfect square, all prime factores that compose N should have an even exponent.

Consider two numbers $A = p_1^{\alpha_1} * p_2^{\alpha_2} * p_3^{\alpha_3} * ... * p_{10}^{\alpha_{10}}$ and $B = p_1^{\beta_1} * p_2^{\beta_2} * p_3^{\beta_3} * ... * p_{10}^{\beta_{10}}$, the multiplication of these numbers is $A * B = p_1^{\alpha_1+\beta_1} * p_2^{\alpha_2+\beta_2} * p_3^{\alpha_3+\beta_3} * ... * p_{10}^{\alpha_{10}+\beta_{10}}$, then, for $A * B$ to be a square number is needed that all values: $\alpha_1 + \beta_1$, $\alpha_2 + \beta_2$, $\alpha_3 + \beta_3$, ... , $\alpha_{10} + \beta_{10}$ to be even.

Remember that the sum of two odd or two even numbers is always even, while the sum of an even and an odd number is always odd, using this property we can consider only the parity of the exponents and represent the multiplication of the numbers as the xor of a bitmask that represent each number where the $i - th$ bit is on if the exponent of the $i - th$ prime number is odd, and the bit is off the exponent is even.

As an example, consider the numbers:

```
A = 2183671875 = 3 * 5 * 7 * 7 * 11 * 11 * 11
prime      2, 3, 5, 7, 11
exponent   0, 1, 1, 2,  3
bitmask    0, 1, 1, 0,  1
```

And

```
B = 36 = 2 * 2 * 3 * 3
prime      2, 3, 5, 7, 11
exponent   2, 2, 0, 0,  0
bitmask    0, 0, 0, 0,  0
```

Then $A * B = 2183671875 * 36$, will result in a number with the following parity in the exponents:

```
   0    1    1    0    1
^  0    0    0    0    0
-------------------------
   0    1    1    0    1
```

Therefore, $A * B$ is not a perfect square, doing the same with the numbers $245 = 5 * 7 * 7 = 00100$, $375 = 3 * 5 * 5 * 5 = 01100$ and $3 = 0100$

```
   0    0    1    0    0                   245
^  0    1    1    0    0                   375
   0    1    0    0    0                     3
-------------------------
   0    0    0    0    0
```

As the result contains all values in 0 in the bitmask it means all exponents from the prime representation are even and therefore $245 * 375 * 3$ is a perfect square.

The result can be found using dynamic programming, with the recursive function $f(i, b)$, which stores, the length of the longest subsequence that ends at index i and results in bitmask b which is the maximum of the following two cases

- 1 + f(index-1, bitmask Ŝ[index]) -¿ case where the *index* value is taken
- f(index-1, bitmask) -¿ case where *index* element is not taken

As there are 10 prime numbers, there are $2^{10} = 1024$ possible bitmasks to test for each value of *index*, then, the solution is $O(2^{10} * N)$.

Problem K – Keep it healthy.

Author: Moroni Silverio

A first solution for the problem would be to check every possible slice of cake, for this solution you can keep two points from the cake p_1 and p_2, that represent the section at the top left of the slice and the section at the bottom right of the slice, these two points describe the limits of the rectangle, then, consider the perimeter of the rectangle if no section in the slice have raisins. This solution is slow and will not run in time.

A more efficient solution can be found if instead of going through all pair of points, we fix two columns c_1, and c_2 of the matrix and find the maximum slice of cake that contains such that the column for p_1 is c_1 and the column for p_2 is c_2 in the slice. If we find such slice for all possible values c_1 and c_2 then, the maximum of all them is the maximum slice we can get. We have C^2 possible pairs of choosing c_1 and c_2, so we look for an efficient way to answer the largest slice of cake for a given pair of columns. Start on the first row and take only the sections of the cake that lies between c_1 and c_2 inclusive, if no section contains raisins, then, it can be part of a slice of cake Jaime would like to eat. To do this in an efficient way consider the matrix A[R][C], that stores in the position A[r][c] the number of sections between A[r][1] and A[r][c] that do not contain raisins, then, for row r, if $A[r][c_2] - A[r][c_1 - 1] = c_2 - c_1 + 1$, then, no section between c_1 and c_2 in that row contain raisins.

To find the maximum height of a slice of cake with the fixed columns c_1, and c_2, it is required to find the maximum number of consecutive values for r that satisfy the previous condition using the matrix A, this can be done in $O(N)$, resetting the height to 0 and keeping the maximum found each time the condition is not met, for each pair of columns c_1, c_2 then the maximum perimeter found will be $2 \times (c_1 - c_2 + 1 + maxHeight(c_1, c_2))$, keep the maximum between all possible values for c_1, c_2 and that is Jaime's slice of cake. This solution takes $O(R * C^2)$ and will run in time.

Problem L – Land distribution.

Author: Félix Arreola

The problem can be solved using a backtracking solution.

First, we get all different rectangle shapes possible for the different areas a programmer can own, for example, if a programmer has an area $A = 12$, then, the different rectangles he can own have dimenssions $= (1, 12), (2, 6), (3, 4), (4, 3), (6, 2), (12, 1)$.

The backtracking strategy is as follows, for the $i - th$ programmer, test each possible rectangle with the programmers assigned land size, in all the different positions the rectangle can be put such that the rectangle contains the $i - th$ programmers house, mark the rectangle in the map and try to put the rectangle for programmer $i + 1$. For any programmer i a valid rectangle does not intersect with the rectangle of any other programmer that has been already added to the land. If we can not set a rectangle for the $i - th$ programmer, we get back to the previous programmer, unmark the land and try another rectagle / position. If the N rectangles have been added to the map a solution was found.

Some heuristics can be taken to find a solution faster in case the solution exists, you can first process programmers with larger areas. Also, you can store the top left and bottom right positions that represent the land for a programmer, making the mark and unmark operations in O(1) and querying for instersection in O(N).