CROATIAN OPEN COMPETITION IN INFORMATICS

1^{st} ROUND

SOLUTIONS

COCI 2009/2010 **Task NOTE**

1st round, 24. October 2009. **Author:** Marko Ivanković

This is a simple ad-hoc task. After reading the sequence comparing it to "1 2 3 4 5 6 7 8" and "8 7 6 5 4 3 2 1" yields the result.

Necesarry skills:
array input

Tags:
ad-hoc

This task can be solved by solving the mathematical problem behind it and simply calculating the solution using that formula. However, the constraints were small enough that a more programming oriented solution exists. First note that if we take a particular domino we can turn it so that the first number on the domino is the smaller one (if they are both equal it's doesn't matter anyhow). Now we can sort all tiles by their smaller number first, and break ties using larger number if necessary. Since each tile appears only once, this gives us a simple ordering which we can traverse. Going tile by tile and summing dots gives us the following code:

```
For each i from 0 to N
    For each j from i to N
        solution = solution + i + j
.
```

Necesarry skills:
math, recognizing implicitly defined rules

Tags:
mathematics

At first, you could notice there are 26 ways one can substitute '_' with a letter. Trying all combinations would not be solvable in the constraints given in this task. However there are only 3 classes of substitution:

- substitute with a vowel
- substitute with the letter 'L'
- substitute with a consonant different from 'L'

The first class contains 5 letters, the second class 1 letter and the third class 20
letters.

The constraints were small enough that you can now solve the problem by checking all possible combinations of these three classes for each '_'. Of course you need to take care not to count illegal combinations. A good way to do that is to monitor the last two characters and weather or not L has appeared.

Necesarry skills:

combinatorics

Tags:

dynamic programming

Let's try to solve the following subproblem: Given two sequences A and B, what is the optimal pairing for these two sequences? Let Amax be the largest number in sequence A (if more than one exists choose any one), and Bmin be te smallest number in sequence B. Choose some pairing which contains the air (Amax, Bmin). Is this optimal? There are two cases we need to address: (Amax + Bmin) is the largest sum in the chosen pairing. If this is the case, it is quite obvious that no other number from B can reduce this sum. The best you can do is select another instance of Bmin, if more than one is present, and achieve the exact same sum. So, if (Amax + Bmin) is the largest sum in the chosen pairing, there is no way to improve that. (Amax + Bmin) is not the largest sum in the chosen pairing. Let (Ax, Bx) denote the pair with the largest sum. Can we improve this by breaking the (Amax, Bmin) pair? We could try creating pairs (Amax, Bx) and (Ax, Bmin). It is clear that now the sum Ax + Bmin is equal or smaller than Ax + Bx. However, Amax + Bx is now larger or equal than Ax + Bx becase Amax is larger than or equal to Ax. This has now created a new largest sum, larger than or equal to the previous one. So we conclude that it is not possible to improve the solution by not using (Amax, Bmin). This gives as a good starting point for a greedy approach.

Necesarry skills:

recognizing greedy solutions

Tags:

greedy algorithms

Author: F. Barl, G. Žužić, M. Ivanković

We consturct a directed graph with N vertices, labeled by numbers 1 to N. All edges will be of form permutation[k] → k for each k smaller than or equal to N where permutation[] is the shuffle sequence given in the input. It is clear that each vertex has exactly one incoming and one outgoing edge ,because for each x and y permutation[x] differs from permutation[y] is x differs from y. Such graph contains one or more components, where each component is a cycle of length p. Note that this graph uniquely represents the output sequence of Mirkos machine. We now compute cycle lengths for each component. We construct an array cycle[X] that stores the length of the cycle containing vertex X. This can be constructed in O(N) time. We can now determine for any columns C to D the number of rows between two repetitions of the original ordering as P = LCM(cycle[C], cycle[C+1], …, cycle[D]) where LCM is the least common multiple function. We now know that the rows we are interested in are given as 1 + q * P where q is a nonnegative integer. The solution to the original problem is now the number of nonnegative integers q that satisfy:

$$A \leq 1 + q * P \leq B$$

This can be easily solved.

Necesarry skills:

greatest common divisor, combinatorics

Tags:

graph theory

First, we need to find an efficient way of generating the following sequence:

A%B + (2A)%B + (3A)%B + ... (nA)%B **(0)**

Known formula gives us (where [x] is the integer part of X and {x} is the fractional part of X):

[x] + {x} = x **(1a)**
(A%B) / B = {A/B} **(1b)**

Combining **(0)**, **(1a)** and **(1b)** we have:

A*(1+2+...+n) = B*([A/B] + [2A/B] + ... + [nA/B]) + B*({A/B} + {2A/B}
+ ... ... + {nA / B}) **(2)**

From **(2)** we see that we can solve **(0)** if we can solve:

[A/B] + [2A/B] + ... + [nA/B] **(3)**

Note that **(3)** can be interpreted geometrically: How many lattice points are in the triangle (0,0) (n,0) (n, A/B*n) if we exclude lattice points on the X axis.

Let us examine the following two cases:

- A ≥ B

There exists a nonegative integer k and integer r from [0, B-1] such that A = kB + r. We use this and **(3)** to obtain:

[(kB+r)/B] + [2(kB+r)/B] + ... + [n(kB+r)/B] =
= [r/B] + [2r/B] + ... + [nr/B] + k*(1+2+...+n)

This reduces this case to the following case.

- A < B

Let the rectangle (0,0) (n, A/B*n) be labeled P. We can easily calculate the number of lattice points in P. We are interested in the number of lattice points beneath it's diagonal. This can be calculated by subtracting the number of points above the diagonal from the total number of points. The number of points above the diagonal can be found simpler than the number of point beneath it.
Now, by relabeling axises x and y . We reduce the second case back to the first case. These reductions I) -> II) -> I) can only be performed a finite number of times because the sum A + B decreases each time we solve the first case. By following this we can compute members of **(0)** in O(lg n) Now we just need to store the array in a fast data structure. It turns out using interval / tournament tree gives the total complexity O(n lg2 n). Enough to solve the task.

Necesarry skills:
advanced data strucutres, combinatorial geometry

Tags:
data structures, mathematics, ad-hoc