Solutions to the previous round

February 26, 2016

The Perfect Factorial

What’s the smallest N, so that N! (1 * 2 * .. * N) is divisible by P ^ K.

(P and K are input)

The Perfect Factorial Solution

Since for any factorial, the next one will have all its divisors and some more, the number function f(N) = 1 if N! % P^K == 0 else 0, will look like:

[0,0,0,0,0,1,1,1,1…]

So let’s binary search the result!

The only thing left is how to calculate f. Let’s calculate how many Ps fit in N!, and if they are more or equal to K then the function is 1, if it’s less it’s 0.

How many numbers are divisible to P once in N! ? N/P. How many numbers are divisible by P two times? N/(P^2) etc. which gives us Sum(N/P^i for i)

Circular Max Sum

Find the sequence with the maximum sum in a circular array.

Example: [-1 2 -4 3 5] returns 9, the result being [-1 2 -4 3 5]

Circular Max Sum Solution

There are two possible kinds of solution, the normal one in which the sequence solution does not cycle, in which case we do the normal max sequence sum algorithm.

In the case that it cycles, the solution will look something like:

[X X X X X X X X X X X X], green begin part of the solution red not being.

Let’s represent this solution by the red sequence. The result will be Sum(array) - Sum(red). Since for any such solution the first sum will be the same we want to have the smallest possible red sum. So we can find that sum by calculating the Min Sum Sequence in the array.

Final solution: calculate both above solution and print the best

Maximum Matrix Sum

You are given a matrix of integers, find the rectangle with the biggest sum.

Example (green is the solution)

-1 2 3 -10

1 3 -4 5

5 5 -4 -4

1 -10 -2 8

Maximum matrix sum *pseudo* solution

The first solution that we can think of is fixing the 4 corners of the rectangle and calculating the sum. This will give you a O(N^4) solution for fixing the corners times O(N^2) for calculating the sum for a final complexity of O(N^6). Though if you do 2D partial sums, you'll have a complexity of O(1) for calculating the sum for a O(N^4) final solution.

In theory this should be too slow to pass, but sadly it did because Hackerrank was faster than expected :(

Maximum matrix sum solution

Let’s say we want to calculate the best rectangle that starts on line i and ends on line j. This will be equivalent to calculating the maximum subsequence sum for an array v, where v[k] is the sum of elements on column k between lines i and j. We can calculate this vector in linear time using pre-computed prefix sums on each column, and there are O(N^2) possibilities to fix the two lines, which gives us a final complexity of O(N^3).

For lines 1..3 in the example example, the array is [4,10,-5,-9]

```
-1 2 3 -10
 1 3 -4 5
 5 5 -4 -4
1 -10 -2 8
```

Game of Algorithms

You have an array v of length N. You want to find the minimum K such that you can construct K subsets of {1,2,..N} with N-1 elements, and each element from 1 to N appears in at least v[i] arrays.

For example if we have 2 2 as the array, the answer is K = 4, with the following subsets:

[1], [1], [2], [2].

This was the only unsolved problem in the extended time!

Game of Algorithms solution

For the input, if we have a solution with X rounds, then X+1 rounds will also have a solution. (Just copy the X solution, and add a random valid array)

From this we get the idea of binary searching the result. Now let's see how for a fixed X number of rounds we can test if everyone's number will appear the correct number of times.

Edge case: if there is an v[i] that is bigger than X, then X won't be good (since even if he appears in all of the rounds, it still won't be enough).

Now the only constraint that might cause X to be too small is the fact that in each round we skip an element. If we can somehow find X elements that can be skipped, X will be good. How many times can we make i the judge ? Well he needs to be in v[i] rounds, so we can make him in X - v[1]. Same for 2,3…N

So if the number of rounds we can find a judge for (sum(X - v[i])) is bigger than the number of rounds, X is a valid solution. Final complexity O(log N) for binary search times O(N) for checking solutions.

Eating Ice Cream

You are given a circular array v of length N, and are given M queries of the type:

How long is the biggest array that starts in x and has sum less than or equal to y?

Eating Ice Cream Solution

Since there are a lot of queries, we should respond to each of them pretty quickly.

For a given query, we have 2 options. The solution won’t cycle, in which case we’ll search for a j from i..N. Or it will cycle, in which case we subtract from the query sum(i...N) and search for that solution in [1...i-1].

Let’s make partial sums. Since all the numbers are positive we can binary search for the furthest away j. Thus giving us a complexity of O(logN) per query.