CROATIAN OPEN COMPETITION IN INFORMATICS

4th ROUND

SOLUTIONS

This is a simple ad-hoc task. The task can be summed up as "Find the first letter of each last name present in a long variant, that is the solution". If you look at the constraints, you will see that we can easily identify these first letters, they are the **only** uppercase letters in the input. So, we simply sift the input looking for uppercase letters.

Necesarry skills:
string input, char-by-char string parsing, recognizing uppercase chars

Tags:
ad-hoc

Midpoint displacement algorithm is an actual algorithm used for landscape generation. This task shows however, a very real problem which arises in practical use. After 16 iterations the number of vertices exceeds 2^{31}, even if no duplicates are stored (and doing this actually complicates the rendering process). If each vertex stores only its 3D coordinates as 3 doubles, after 16 iterations more than 50 MB of memory are needed to store the landscape alone, with no color, texture or shade data.

To solve this task, we reduce it to a subproblem. First note that the total number of vertices will always be the square of number of vertices in the first row (or column). If we can determine this number, obtaining the total is simple.

If we look at a single step, starting with **X** vertices, midpoint displacement will add one vertex between each of the two neighbouring vertices. Since there are **X** - 1 neighbours, we add **X** - 1 new vertices so at the end of the step we now have 2**X** - 1 vertices in total. This leads to a recursive formula where $\mathbf{N}_i$ denotes the number of vertices after i steps:

$$N_i = 2N_{i-1} - 1 \, \forall \, i > 0$$

With the special case of $\mathbf{N}_0$ = 2. Using induction we can prove that the number of vertices in the first row (or column) after **x** steps is $N_x = 2^x + 1$.

Necesarry skills:

math, recognizing implicitly defined rules

Tags:

mathematics

Let us examine one arbitrary prime, **X**. How many times can you divide the sequence with **X**? Obviously, the smallest number of times you can divide each number in sequence with **X**. Suppose we know for each number in sequence number b_i which indicates the number of times we can divide the i^{th} number in sequence by **X**. Dividing one number with **X** and multiplying some other number with **X** now turns into decreasing/increasing corresponding b-s. Since our goal is to maximize the smallest b, it's quite obvious that the best thing to do is always take one away from the largest b and add it to the smallest, until all b-s are equal or almost equal (they can be off by at most one). It can be easily seen now that the solution for each **X** is equal to the sum of all b-s for the corresponding **X** divided by the number of elements, rounded down.

Using Eratostens Sieve for fast prime number generation solves this task under given constraints.

Necesarry skills:

prime numbers, factorisation, Eratostens Sieve

Tags:

math

This task can easily be reduced to a histogram problem, solvable in three steps:

1. For each column determine the maximal height reachable if the left part of the brush is touching the column. Using monotone queue this can be solved in O(N).
2. For each column determine the maximal height reachable at all. This also determines the surface area left unpainted. Again, using a different monotone queue this can be solved in O(N).
3. For each column use greedy algorithm to determine weather or not to perform a stroke. This can also be done in O(N).

Using O(N log N) structures was also acceptable, if coded correctly.

Necesarry skills:
logarithmic data structures, monotone queue

Tags:
greedy, data strucutes

Best way to approach this task is using dynamic programming. We ask ourselves the following question: How many ways are there to fold the left part of the strip into a spiral with L segments with the smallest non-folded piece having length K (*K* ≥ *A*). And then the same question for the right part. If we know how to answer that, we use the information to answer this: how many ways can be fold the left (or right) part of the strip into a spiral with the last two non-folded pieces length D. Summing on D gives us the number of ways to fold the strip into a spiral of length L with at least two equal pieces having the last two pieces of length at least one. We can now extend this even more to: How many ways are there to fold a spiral of length L with the last two non-folded pieces equal, that doesn't explode (i.e. the smallest segment is at least A or B). Note this number as dpA[L] and dpB[L].

Now note that there are 3 cases:

1) left or right end of the strip is folded into a spiral of length L with the last two segments equal, and the other end is not folded at all

2) both ends are folded into two spirals with the last two segments equal, and there is a non-folded segment of length M between them.

3) the strip is not folded at all

For 1) we sum on all lengths so the total is $\sum_{L=1}^{N}(dpA[L]\cdot 1 + 1\cdot dpB[L])$. For 2) we sum on the length of M so the total is $\sum_{M=0}^{N}\sum_{L=1}^{N}(dpA[L]\cdot 1\cdot dpB[N-M-L])$. For 3), there is only one way to to that. Sum of both these sums plus one is the solution.

Necesarry skills:

logarithmic data structures, monotone queue

Tags:

greedy, data strucutes

First, let us presume that Ana doesn't need to buy all ingredients and set up diference equations that will help us calculate the number of tours. Let $X_i[t]$ be the number of tours ending in crossroad i at time t. To set up the equations we examine all roads leading to i. For each road (edge j,i), on the right side of the equation we add the case of Ana not entering in the shop, $x_j[t-1]$, and the case where she did, $x_j[t-2]$. For each crossroad this gives equations in the form $X_i[t] = x_a[t-1] + x_a[t-2] + x_b[t-1] + x_b[t-2] + \ldots$
Let S[t] be the number of tours ending in vertex 1 with time t. We have $S[t] = S[t-1] + X_1[t]$. The total number of tours ending in vertex 1 with time T is S[T]. Using binary exponentiation and matrix multiplication we can obtain S[T]. However, S[T] now also contains invalid paths. To get the number of valid paths, we use the inclusion/exclusion formula. First we subtract all path that do not contain one ingredient, than add all that do not contain two, then subtract all that do not contain three etc. The number of paths that do not contain one or more ingredients can be easily obtained by omitting $x_j[t-2]$ in all equations where the shop sells the wanted ingredient.

Skills:
matrix, binary exponentiation, diference equations (not diferential), inclusion/exclusion theorem

Tags:
matrix diference euqation