CROATIAN OPEN COMPETITION IN INFORMATICS

5th ROUND

SOLUTIONS

At least one of the three juices will always be fully used. If we know which one, using that we can easily determine the amount of cocktail they can make. Let us determine which juice will be fully used. If we knew that they made **X** liters of cocktail, how much of each juice would they have left over? Obviously **A** - **X** * **I** orange, **B** - **X** * **J** apple and **C** - **X** * **K** pineapple. Using some simple math we easily see that they can make at most **A / I** cocktail if the apple juice is the one that will be fully used. This leads us straight to the solution. Simply finding the smallest number from **A/I**, **B/J** and **C/K** and inserting it as **X** in the aforementioned formulas.

Necessary skills:
simple mathematics

Tags:
ad-hoc

Author: Marko Ivanković, Bruno Rahle

A brute force search will work quite well. For each possible parking position, we count the number of buildings and increment counters accordingly.

Necessary skills:
simple string operations

Tags:
brute force search

If we could normalize different locks to some form that would wield the same form for equal keys we could solve this task by comparing all normalized locks and counting the number of different forms. Fortunately, such normalization exists. First, let us examine how rotations come into play. Because the outline of the keyhole is composed of lines parallel to the edges of the lock, only rotations of 0, 90, 180 and 270 degrees are viable. All of these rotations can be simply accounted for. Rotation of 0 and 180 degrees is the same sequence read in the left-to-right or right-to-left order. Rotations of 90 and 270 degrees can be performed by subtraction and addition from the length/width of the lock. The normalization algorithm is as follows: first find the lowest point of the keyhole. This is our referent point. Now construct a new sequence from the lower edge sequence subtracting the referent point. This, paired with another sequence containing the widths of all keyhole segments (subtract the lower edge from the upper edge) gives us a perfect normal form for this task.

Necessary skills:
normalization

Tags:
normalization

We can use dynamic programming to solve this task. Let $\mathbf{A}_i$ be the color of the i^{th} marble. Let our state be describe with 3 items: (l, r, count) that gives the minimal number of marbles we need to insert into subsequence {$\mathbf{A}_l$, $\mathbf{A}_{l+1}$, ..., $\mathbf{A}_r$} so that it disappears, with the added bonus that we can insert **count** copies of marble $\mathbf{A}_l$ on the beginning of the sequence.
For example let **A** = {*1, 3, 3, 3, 2*, 3, 1, 1, 1, 3}. State [*0*][*4*][**3**] denotes the minimal number of marbles we need to insert into {**1**, **1**, **1**, 1, 3, 3, 3, 2} so that the entire subsequence disappears.

There are three transitions for each state:

- Add a duplicate of the first marble to the beginning of the sequence. This gives [l][r][X] = [l][r][X+1] + 1.
- If X = **k**-1, we can delete all marbles on the beginning of the sequence, including the first marble of our subsequence. This gives [l][r][X] = [l+1][r][0] for X = k-1.
- Third and most complex case is merging the first marble with some later marble. We expect to merge our first marble ($\mathbf{A}_l$) with some marble $\mathbf{A}_j$ assuming $\mathbf{A}_l$ = $\mathbf{A}_j$. This merger gives [l][r][X] = [l+1][j-1][0] + [j][r][X+1].

For each state we must of course find the minimum of all possible transitions.

Tags:
dynamic programming

Let us start with the brute force solution. We simulate the calls to the `something` function. Using appropriate structure (for example tournament tree) we answer the sequence sum question in O(1) time.

Now for the first improvement. If there are multiple times that number Y appears in the input (X times for example): there is no need to call the `something` function X times. Instead we can call a modified version of the function once, and have her increment the fields by X. This eliminates all duplicates in the input sequence.

What is the complexity of this solution? The worst case for the brute force algorithm is **K** ones leading to O(**N*****K**) complexity. But our new modified function solves such case in one pass. Obviously, we can have only one number 1 in the worst case. Next worst thing is to enter number 2. Than 3. Than 4. And so on. So the worst case is now {1,2,.....**K**}. Our modified function will perform **N**/1+**N**/2+**N**/3+...+**N**/**K** steps. But it can be easily shown that this is at most 20***N**. This is solvable in the given time limit.

Necessary skills:
complexity analysis

Tags:
data structures

First note that using row/column rotations we can place arbitrary elements on given rows or columns. For example let sequence S = $\{p_1, p_2, ..., p_R\}$ of R elements we want to place in the first column. For i^th element we:

- If p_i is already in the first column, we place it out rotating the row by one.
- We rotate the column containing p_i so that it is in the i^th row.
- Rotate the row containing p_i so that it is placed into the first column

Using this algorithm we can place any chosen R elements in the first column and any chosen S elements in the first row. We can now negate all elements in the first row/column. If we repeat this procedure it follows that we can choose arbitrary $a \cdot R + b \cdot S \leq R \cdot S$ elements of the matrix and negate them.

The best solution is of course to negate all negative elements. Since this is not always possible, we need to choose such *a* and *b* that results in the smallest possible solution. Sorting the elements of the array and using dynamic programming easily yields the best *a* and *b*.

Note that for each element we use at most three rotations, and two negations this ensures that the number of operations is smaller than $5 \cdot R \cdot S$.

Coding this in $O((max\{R, S\})^4)$ solves the problem.

Necessary skills:
tricky coding

Tags:
mathematics