Latvia University Contest 2

March 9, 2019

Artem Vasilev and Ivan Smirnov

Hello Muscat Programming Bootcamp 2019

A. Thomas the Tank Engine

You have an undirected graph and a "train" of length k. What are all possible positions for the head of the train?

A. Thomas the Tank Engine

You have an undirected graph and a "train" of length k. What are all possible positions for the head of the train?

The graph is a *cactus*: every vertex is located on at most one simple cycle.

A. Thomas the Tank Engine

Detect all cycles in graph using a single DFS. For each vertex find the length of the cycle.

A. Thomas the Tank Engine

Detect all cycles in graph using a single DFS. For each vertex find the length of the cycle.

Then, run any graph search algorithm (BFS or DFS) from the locomotive vertex (mark all vertices of the train as used, so you don't visit them during BFS).

If any cycle of length $\geqslant k$ is reachable, then every vertex of the graph is reachable.

Otherwise, output all vertices visited by BFS. Complexity of this solution is $O(n + m)$.

B. Broken line

Given N points on the perimeter of a convex polygon. Find a shortest polygonal chain through all these vertices.

B. Broken line

Order these points so they form a convex polygon (either by finding a convex hull, or just sorting).

B. Broken line

Order these points so they form a convex polygon (either by finding a convex hull, or just sorting).

Now, calculate a DP: $f_{i,j}$ is the minimum cost to connect all points in range $[i, j]$, such that the chain starts at i. $f_{j,i}$ is similar, but the chain must start at j.

B. Broken line

Order these points so they form a convex polygon (either by finding a convex hull, or just sorting).

Now, calculate a DP: $f_{i,j}$ is the minimum cost to connect all points in range $[i, j]$, such that the chain starts at i. $f_{j,i}$ is similar, but the chain must start at j.

How to calculate $f_{i,j}$: either make a segment from point i to point $i + 1$, then take $f_{i+1,j}$, or make a segment from i to j, then take $f_{j,i+1}$.

All these calculations take $O(n^2)$ time.

B. Broken line

When you have calculated $f_{i,j}$, it's time to consider some cases to get to the final answer.

B. Broken line

When you have calculated $f_{i,j}$, it's time to consider some cases to get to the final answer.

Fix any point (say, the n-th one). It can be the end vertex of a chain, or an interior one.

If it's an endvertex, check two neighbors. If it is an interior vertex, check all possible other points to be connected to the n-th one, and take the minimum.

C. Columns

Given an $n \times m$ grid, find the number of divisions into *oriented columns*.

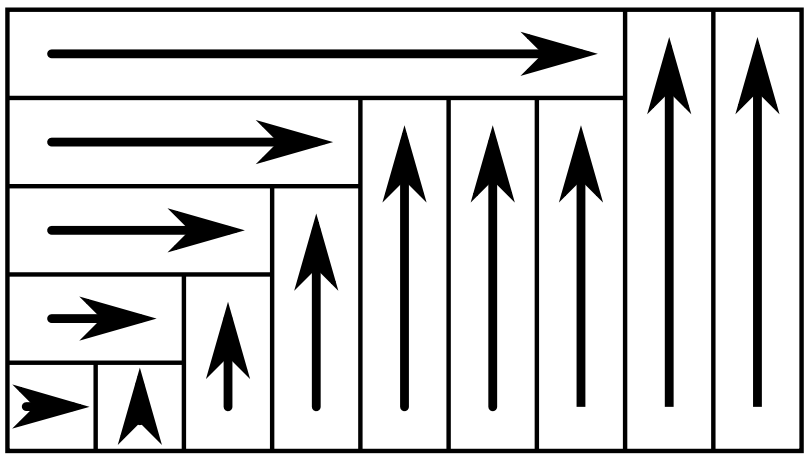
C. Columns

We restrict the allowed divisions to several simpler patterns and calculate the number of ways to make these patterns.

C. Columns

We restrict the allowed divisions to several simpler patterns and calculate the number of ways to make these patterns.

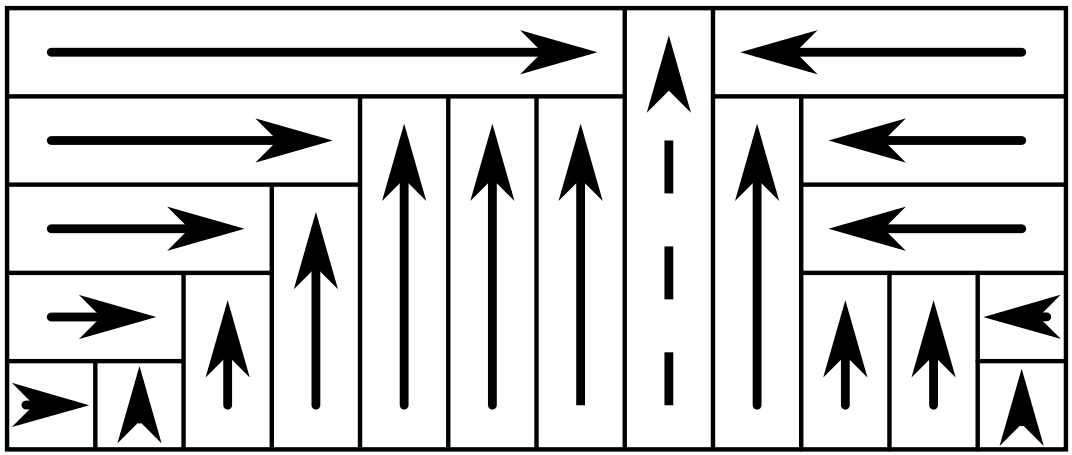
First auxiliary pattern ("2-diff"): only two different directions are allowed (e.g. "right" and "up").



The number of such patterns is C_{w+h}^{h}.

C. Columns

Second auxiliary pattern ("3-diff"): three different directions allowed (e.g. "right", "left" and "up"). At least one "up" tile must touch the top of the rectangle (dashed in the picture).

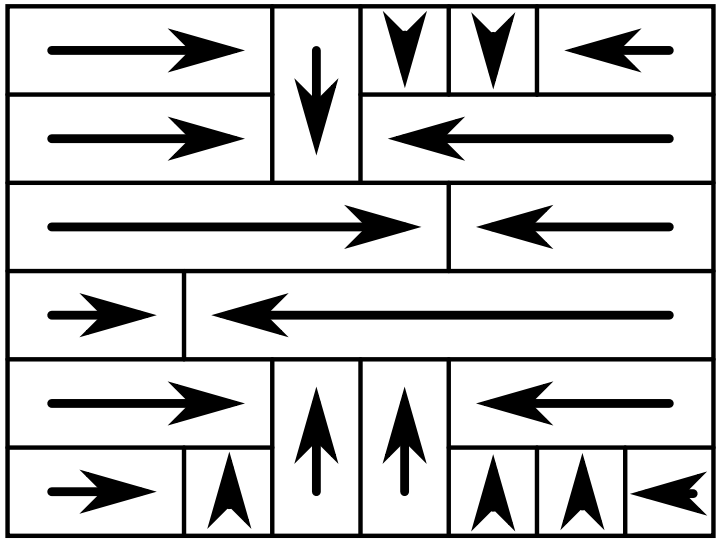


To calculate the number of such patterns, fix the position of the leftmost "up" column touching the top. Now the rectangles to the right and to the left contain exactly "2-diff" patterns.

Computation is done in $O(w)$.

C. Columns

Patterns with horizontal separator: one or several lines containing only horizontal columns.

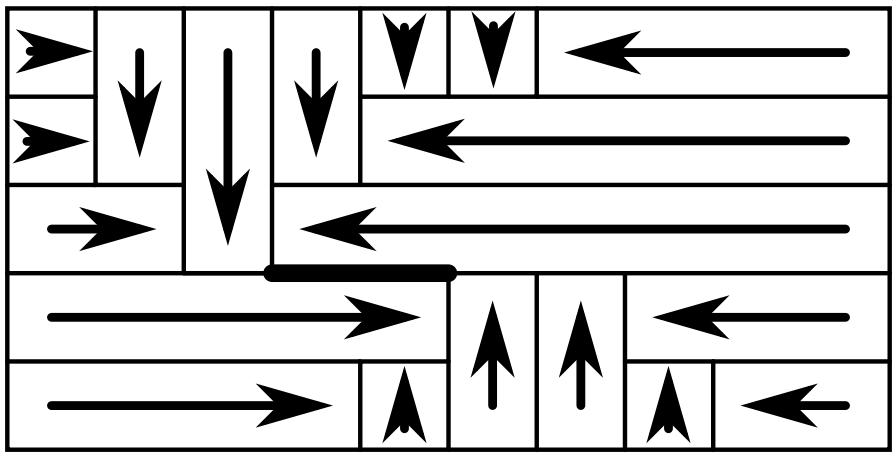


To calculate the number of ways, fix the first and the last separator line. Let there be s of them. Patterns to the top and to the bottom are "3-diff", and the number of ways to fill separator lines is $(w+1)^s$.

Computation is done in $O(h^2)$

C. Columns

General case: neither vertical nor horizontal separators exist.



C. Columns

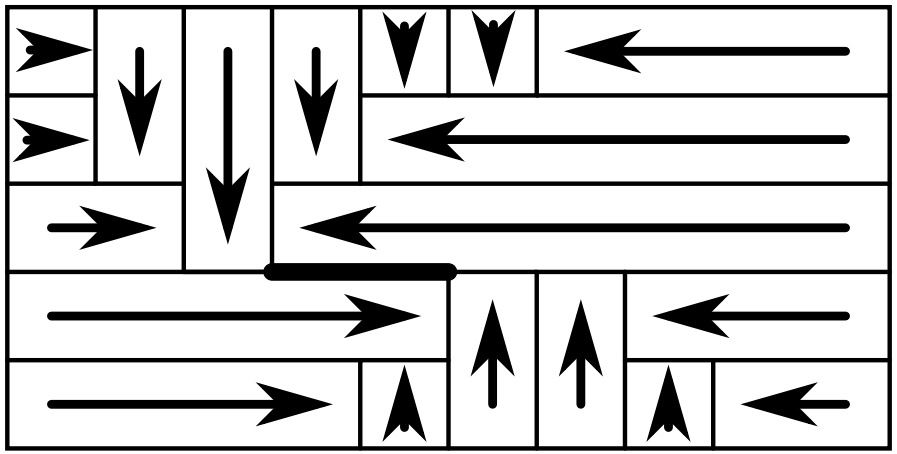
General case: neither vertical nor horizontal separators exist.

Three cases are possible.

- Two horizontal opposite-facing columns touch (depicted in the previous slide)
- Two vertical opposite-facing columns touch
- None of the above happens

We calculate the first two cases allowing the touch length to be zero and then subtract the result for the third case because we double-counted it.

C. Columns



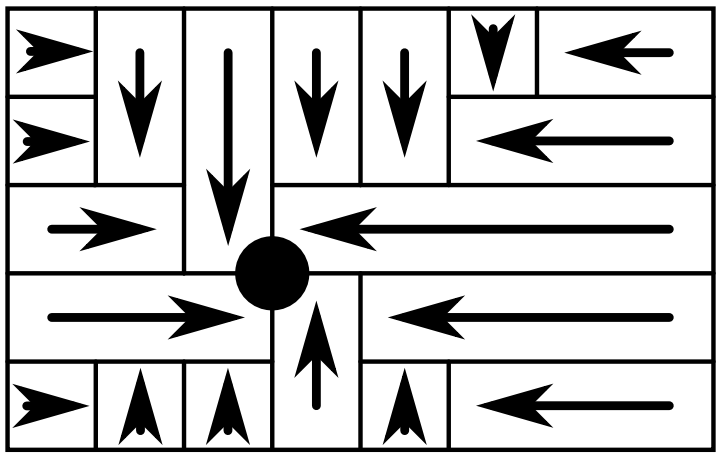
We always consider the pattern look like in the picture: two horizontal columns touch and the one facing left is at the top.

We fix the height and iterate through the position of the first "full" vertical column in the bottom part. Now the left and the right parts of the bottom parts resemble "2-diff" pattern with fixed dimensions. We use prefix sums to count all possibilities for the top part, which are also look as combinations of two "2-diff"-s.

Complexity: $O(wh)$.

C. Columns

General case where the length of the touch is zero.



We iterate through the position of the center and use "2-diff" pattern to calculate the number of ways in each of the four corners.

Complexity: $O(wh)$.

C. Columns

Outline:

- Add cases where either horizontal or vertical separator line(s) exist
- Add cases where no separator exist and two opposite-facing lines touch by a (possibly empty) segment
- Subtract cases where lines of four different directions meet at the same point

E. Zebra

Given a polygon, and a zebra-like coloring of a plane, find the white area inside the polygon.

E. Zebra

Since every polygon can be represented as a sum of signed trapezoids (include some, exclude some), it's sufficient to solve the problem for a trapezoid.

E. Zebra

Since every polygon can be represented as a sum of signed trapezoids (include some, exclude some), it's sufficient to solve the problem for a trapezoid.

We will select trapezoids that have bases parallel to zebra lines. Areas of intersections of whites line with this trapezoid form an arithmetic progression, sum of which can be easily found.

We consider n trapezoids, each in $O(1)$ time.

F. Fractal NV

Traverse a colored fractal polyline efficiently.

F. Fractal NV

As the case with fractals, create a recursive function for every edge. Thoughtful implementation required.

F. Fractal NV

As the case with fractals, create a recursive function for every edge. Thoughtful implementation required.

Represent the path as a sequence of superedges with levels. Superedge of level 0 is just an edge, superedge of level k can expand into a cross with superedges of level 0 and $k-1$. Skip superedges and subtract number of colored edges quickly.

F. Fractal NV

As the case with fractals, create a recursive function for every edge. Thoughtful implementation required.

Represent the path as a sequence of superedges with levels. Superedge of level 0 is just an edge, superedge of level k can expand into a cross with superedges of level 0 and $k-1$. Skip superedges and subtract number of colored edges quickly.

To increase level, increase level of all superedges with maximum level in your current path.

G. Pluses and minuses

Given a sequence of n “+” and “-”. Let’s replace “+” by 0 and “-” by 1.

G. Pluses and minuses

Given a sequence of n "+" and "-". Let's replace "+" by 0 and "-" by 1. Now, do the following n times: take XOR of adjacent elements in the sequence and output the first element in the sequence.

G. Pluses and minuses

Given a sequence of n "+" and "-". Let's replace "+" by 0 and "-" by 1. Now, do the following n times: take XOR of adjacent elements in the sequence and output the first element in the sequence. Restore the original sequence given the output.

G. Pluses and minuses

Notice that this transformation is bijective: transformation from the first row to the first column is the same as the inverse one.

G. Pluses and minuses

Notice that this transformation is bijective: transformation from the first row to the first column is the same as the inverse one.

We need to calculate the transform described in problem statement.

G. Pluses and minuses

Notice that this transformation is bijective: transformation from the first row to the first column is the same as the inverse one.

We need to calculate the transform described in problem statement.

You can prove that after the i-th stage the first element is equal to XOR of all elements j where j is a submask of i.

$$result_i = \bigoplus_{j \subset i} input_j$$

G. Pluses and minuses

It is possible to calculate this array in $O(n \log n)$ time using an algorithm from Fast Subset Convolution.

Might require some optimizations, since time limit is really tight.

Towers of Hanoi

Find the minimum number of ways to solve the Hanoi Towers puzzle from the given starting configuration.

Towers of Hanoi

Iterate from the largest disks to smallest ones. Keep track of the current target rod.

Towers of Hanoi

Iterate from the largest disks to smallest ones. Keep track of the current target rod.

If the current disk is on the target rod, skip it. Otherwise, you have to move everything to the third rod, then move the current disk to target rod, then move all others into the target rod.

This takes $2^x + y$ operations, where x is the number of current disk, and y is the cost of moving all disks to a new target rod.

Towers of Hanoi

Iterate from the largest disks to smallest ones. Keep track of the current target rod.

If the current disk is on the target rod, skip it. Otherwise, you have to move everything to the third rod, then move the current disk to target rod, then move all others into the target rod.

This takes $2^x + y$ operations, where x is the number of current disk, and y is the cost of moving all disks to a new target rod.

In other words, just add 2^x each time the current disk is not on target rod, then switch targets. $O(n)$ time.

I. Interesting sequence

Find p-th occurrence of n in the given sequence.

I. Interesting sequence

As usual, in problems like this, it's enough to calculate values of the following kind: how many numbers starting with binary prefix P have $a_x = n$?

I. Interesting sequence

As usual, in problems like this, it's enough to calculate values of the following kind: how many numbers starting with binary prefix P have $a_x = n$?

Notice that $a_{2^k-1} = 1$ for every k. Use DP on binary digits to get the result.

I. Interesting sequence

As usual, in problems like this, it's enough to calculate values of the following kind: how many numbers starting with binary prefix P have $a_x = n$?

Notice that $a_{2^k-1} = 1$ for every k. Use DP on binary digits to get the result.

By the way, a_i is the bit count of standard Gray code:
$a_i = \text{bitcount}(i \oplus \lfloor \frac{i}{2} \rfloor)$

J. Transformation on numbers

Find the number of operations of adding 10^y and changing a digit to transform one given integer into another.

J. Transformation on numbers

Pad one of the numbers with zeros so that they have equal length. Consider all digits, starting from the least significant.

J. Transformation on numbers

Pad one of the numbers with zeros so that they have equal length. Consider all digits, starting from the least significant.

Keep track of two values: minimum cost of operations to make two integers be equal up to current digits if there was no carry, and if there was a carried 1.

J. Transformation on numbers

Pad one of the numbers with zeros so that they have equal length. Consider all digits, starting from the least significant.

Keep track of two values: minimum cost of operations to make two integers be equal up to current digits if there was no carry, and if there was a carried 1.

Try all possibilities to get a non-carry/carry situation from the current state (non-carry/carry) and current digits.

J. Transformation on numbers

Pad one of the numbers with zeros so that they have equal length. Consider all digits, starting from the least significant.

Keep track of two values: minimum cost of operations to make two integers be equal up to current digits if there was no carry, and if there was a carried 1.

Try all possibilities to get a non-carry/carry situation from the current state (non-carry/carry) and current digits.

When you process all digits, the answer for a state with no carry is the final result.

K. Kingdom division

Split a tree into three connected components A, B and C such that $|A| = |B|$. Maximize $|A|$.

K. Kingdom division

Solution outline: consider many cases and win.

K. Kingdom division

Solution outline: consider many cases and win.

One possible way of doing that: root the tree at arbitrary vertex. Iterate over all possible vertices in the rooted tree, let that vertex u be the highest vertex in a tree that belongs to C.

K. Kingdom division

Solution outline: consider many cases and win.

One possible way of doing that: root the tree at arbitrary vertex. Iterate over all possible vertices in the rooted tree, let that vertex u be the highest vertex in a tree that belongs to C.

If u is not a root, then everything above it is either A, or $A \cup B$. In the first case you have to check if u has a subtree of size $n - size_u$, in the second case you have to check whether the $A \cup B$ part can be split into two equal subtrees.

K. Kingdom division

Solution outline: consider many cases and win.

One possible way of doing that: root the tree at arbitrary vertex. Iterate over all possible vertices in the rooted tree, let that vertex u be the highest vertex in a tree that belongs to C.

If u is not a root, then everything above it is either A, or $A \cup B$. In the first case you have to check if u has a subtree of size $n - size_u$, in the second case you have to check whether the $A \cup B$ part can be split into two equal subtrees.

If u is a root, you need to find two subtrees of equal sizes. This all can be done, if you store a multiset of all subtree sizes for every vertex and merge them small-to-large.