Day 1 Contest Analysis, Division AB

July 7, 2019

Artem Vasilev, Ivan Smirnov, Filipp Rukhovich

Discover Vladivostok 2019

A. Engine called Thomas

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

A. Engine called Thomas

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. Engine called Thomas

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

A. Engine called Thomas

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 DFS/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 neighbours. 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*.

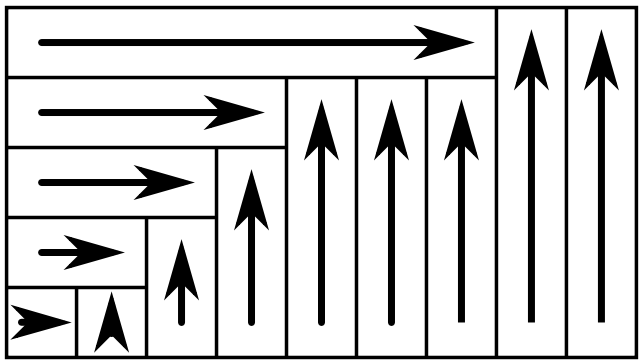
C. Columns

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.

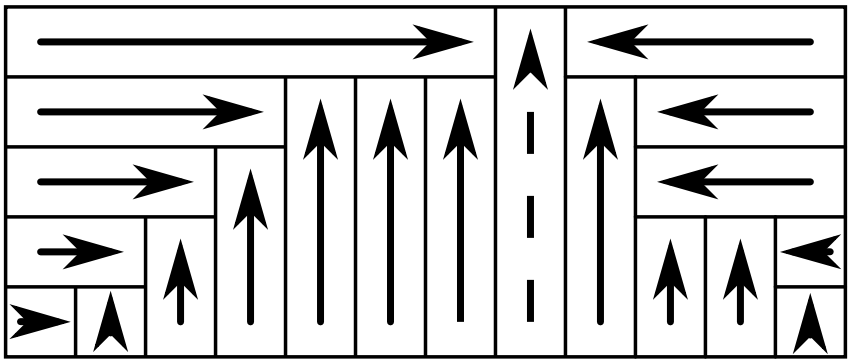
First auxiliary pattern ("2-diff"): only two different directions are allowed (e.g. "right" and "up").



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).

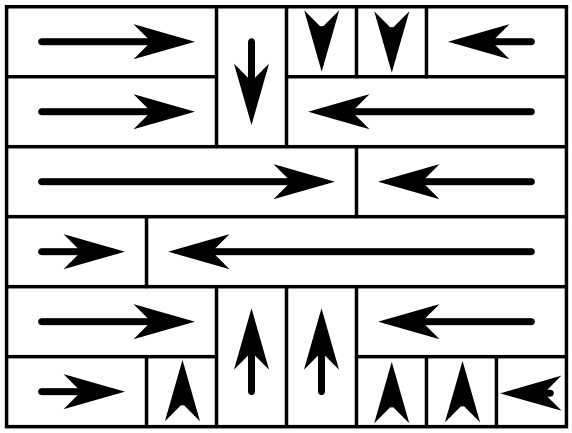


To calculate the number of such patterns, fix the position of the leftmost "up" column touching the top. Now the rectangles to the right and to the left contain exactly "2-diff" patterns.

Computation is done in $O(w)$.

C. Columns

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

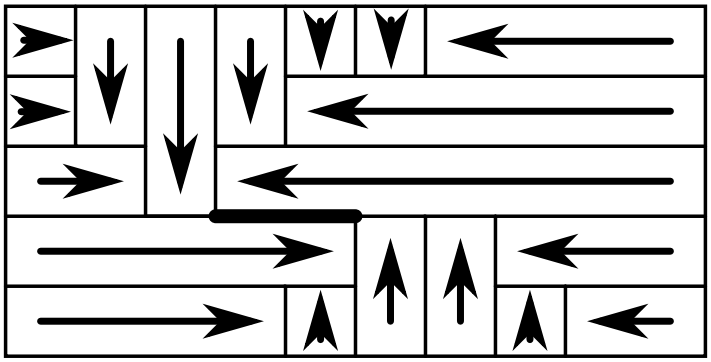


To calculate the number of ways, fix the first and the last separator line. Let there be s of them. Patterns to the top and to the bottom are "3-diff", and the number of ways to fill separator lines is $(w+1)^s$.

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

C. Columns

General case: neither vertical nor horizontal separators exist.



C. Columns

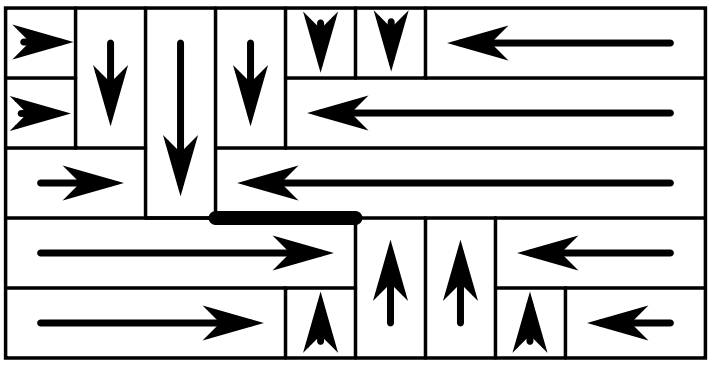
General case: neither vertical nor horizontal separators exist.

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



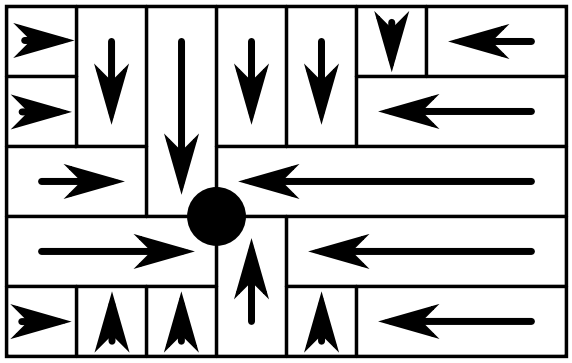
We always consider the pattern look like in the picture: two horizontal columns touch and the one facing left is at the top.

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

D. Zebra

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

D. 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.

D. 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.

E. Pluses and minuses

Given a sequence of n "+" and "-". Let's replace "+" by 0 and "-" by 1.

E. 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.

E. 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.

E. Pluses and minuses

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

E. Pluses and minuses

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

We need to calculate the transform described in problem statement.

E. Pluses and minuses

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

We need to calculate the transform described in problem statement.

Fact 1: after 2^k-th stage, the zeroth element is equal to XOR of elements with numbers 0 and 2^k (can be proved by induction).

E. Pluses and minuses

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

We need to calculate the transform described in problem statement.

Fact 1: after 2^k-th stage, the zeroth element is equal to XOR of elements with numbers 0 and 2^k (can be proved by induction).

Fact 2: after the i-th stage the first element is equal to XOR of all elements j where j is a submask of i:

$result_i = \bigoplus_{j \subset i} input_j$ (consequence of Fact 1).

E. 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.

E. 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.

But how to solve it if you don't know anything about Fast Subset Convolution?

E. Pluses and minuses

First of all, we know that zeroth element of i-th row is a XOR or some elements of zeroth row. But what are these elements?

E. Pluses and minuses

First of all, we know that zeroth element of i-th row is a XOR or some elements of zeroth row. But what are these elements?

The answer contains in binary Pascal's triangle; then, you can print the triangle at the computer and note the fact about zeroth element in 2^k-th row.

E. Pluses and minuses

First of all, we know that zeroth element of i-th row is a XOR or some elements of zeroth row. But what are these elements?

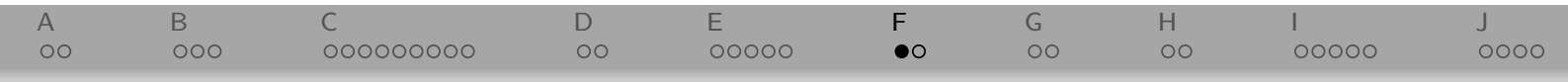
The answer contains in binary Pascal's triangle; then, you can print the triangle at the computer and note the fact about zeroth element in 2^k-th row.

Then, suppose that $n = 2^l$ for some integer l (if not, add some number of zeroes to the end of 0th row). Then, use divide-and-conquer technology:

- calculate first, "upper", half of 0th column and 2^{l-1}-th column recursively;
- calculate second half of 0th column as bitwise XOR of first half of 0th column and 2^{l-1}-th column (consequence of Fact 1).

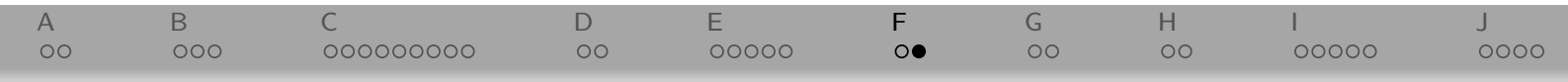
G. Pluses and minuses

The complexity of such a method is $O(n \ log \ n)$; but if you store columns in std::bitset (or your custom version) then it will work in $O(n \ log \ n/32)$ which should be fast enough even for $n = 5 * 10^6$)))



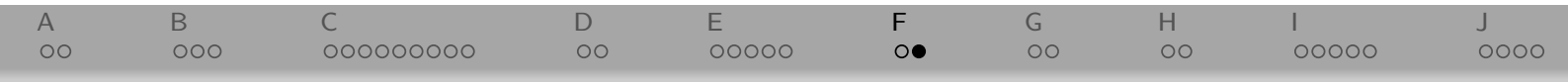
F.Towers of Hanoi

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



F. Towers of Hanoi

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



F. 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.

F. 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.

G. Interesting sequence

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

G. Interesting sequence

First of all, notice that a_i is a number of maximal by inclusion series of equal consecutive bits in binary representation. For example, $a_{13} = a_{1101_2} = 3$, $a_8 = a_{1000_2} = 2$, $a_{15} = a_{1111_2} = 1$.

G. Interesting sequence

First of all, notice that a_i is a number of maximal by inclusion series of equal consecutive bits in binary representation. For example, $a_{13} = a_{1101_2} = 3$, $a_8 = a_{1000_2} = 2$, $a_{15} = a_{1111_2} = 1$.

Then, let $dp[l][k]$, $1 \leqslant k \leqslant l \leqslant 63$, is a number of 0,1-sequences of length l with exactly k series of equal consecutive bits (Why 63? Because the bit length of answer is not more than 63).

G. Interesting sequence

First of all, notice that a_i is a number of maximal by inclusion series of equal consecutive bits in binary representation. For example, $a_{13} = a_{1101_2} = 3$, $a_8 = a_{1000_2} = 2$, $a_{15} = a_{1111_2} = 1$.

Then, let $dp[l][k]$, $1 \leqslant k \leqslant l \leqslant 63$, is a number of 0,1-sequences of length l with exactly k series of equal consecutive bits (Why 63? Because the bit length of answer is not more than 63).

Obviously, $dp[l+1][k] = dp[l][k] + dp[l][k+1]$ (first bit can be equal to second one or not). Using this dynamics, one can find p-th 0,1-sequence with n series starting from one by the same way as p-th combination of k elements of set $\{1, 2, 3, 4, \ldots, n\}$ can be found.

H. Transformation on numbers

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

H. Transformation on numbers

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

H. 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.

H. 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.

H. 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.

I. Kingdom division

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

I. Kingdom division

Solution outline: consider many cases and win.

I. 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.

I. 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.

I. 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.

I. Kingdom division

Such a solution works in $O(n\ log^2\ n)$ time with std::multiset and in $O(n\ log\ n)$ using std::unordered$_m$*ultiset*.

I. Kingdom division

Such a solution works in $O(n \log^2 n)$ time with std::multiset and in $O(n \log n)$ using std::unordered$_m$ultiset.

However, there is a way to solve it in $O(n \log n)$ without mergeable multisets! All you need is to calculate $size_u$ and binary lifts for each vertex u. Also let $cnt[k]$, $k = 1, \ldots, n$, be a number of such a vertices u that $size_u = k$.

I. Kingdom division

Such a solution works in $O(n \log^2 n)$ time with std::multiset and in $O(n \log n)$ using std::unordered$_m$ultiset.

However, there is a way to solve it in $O(n \log n)$ without mergeable multisets! All you need is to calculate $size_u$ and binary lifts for each vertex u. Also let $cnt[k]$, $k = 1, \ldots, n$, be a number of such a vertices u that $size_u = k$.

Then, to get three components, we should erase two edges. They may be ancestor and descendant - or not. Consider these two cases.

I. Kingdom division

Such a solution works in $O(n\ log^2\ n)$ time with std::multiset and in $O(n\ log\ n)$ using std::unordered$_m$ultiset.

However, there is a way to solve it in $O(n\ log\ n)$ without mergeable multisets! All you need is to calculate $size_u$ and binary lifts for each vertex u. Also let $cnt[k]$, $k = 1, \ldots, n$, be a number of such a vertices u that $size_u = k$.

Then, to get three components, we should erase two edges. They may be ancestor and descendant - or not. Consider these two cases.

Suppose that we want to erase an edge from known vertex u to its parent, and also we want to erase and edge from some unknown vertex v to its parent in such a way that v should be ancestor of v.

I. Kingdom division

Obviously, $size_v$ should be equal to $2 * size_u$, $n - size_u$ or $n - (n - size_u)/2$; existing of such a v on the path from u to root can be established in $O(log\ n)$ for each u using binary search on binary lifts.

I. Kingdom division

Obviously, $size_v$ should be equal to $2 * size_u$, $n - size_u$ or $n - (n - size_u)/2$; existing of such a v on the path from u to root can be established in $O(log\ n)$ for each u using binary search on binary lifts.

The other case is that for some known u, we want to erase an edge from u to parent and also an edge from such v to its parent that $size_v \geqslant size_u$ and v is not an ancestor of u.

I. Kingdom division

Obviously, $size_v$ should be equal to $2 * size_u$, $n - size_u$ or $n - (n - size_u)/2$; existing of such a v on the path from u to root can be established in $O(log\ n)$ for each u using binary search on binary lifts.

The other case is that for some known u, we want to erase an edge from u to parent and also an edge from such v to its parent that $size_v \geqslant size_u$ and v is not an ancestor of u.

To do that, $size_v$ should be equal to $size_u$, $n - 2 * size_u$ or $(n - (n - size_u))/2$. Suppose that we are checking some value $S \geqslant size_u$ to be $size_v$.

I. Kingdom division

Obviously, $size_v$ should be equal to $2 * size_u$, $n - size_u$ or $n - (n - size_u)/2$; existing of such a v on the path from u to root can be established in $O(log\ n)$ for each u using binary search on binary lifts.

The other case is that for some known u, we want to erase an edge from u to parent and also an edge from such v to its parent that $size_v \geqslant size_u$ and v is not an ancestor of u.

To do that, $size_v$ should be equal to $size_u$, $n - 2 * size_u$ or $(n - (n - size_u))/2$. Suppose that we are checking some value $S \geqslant size_u$ to be $size_v$.

If $size_u = S$ then we should just check whether $cnt[size_u] \geqslant 2$ or not.

I. Kingdom division

I. Kingdom division

Otherwise, we should see on $cnt[S]$ too. If $cnt[S] = 0$ then v does not exist; if $cnt[S] \geqslant 2$ then good v definitely exists; if $cnt[S] = 1$ then we should check whether the corresponding v is an ancestor of u or not; it can be done using the same binary search (or arrays *tin* and *tout*)))).

J. Domino sets

Given a set of m-tiles of dominoes; each one is of one of v types. We can drill new holes in any tile and are to calculate maximum number of full m-sets we can make.

We will solve the problem using binary search by answer. So, given positive integer k, is it possible to make at least k m-piles?

J. Domino sets

Suppose for each possible type (i, j) of tile that $i \leqslant j$. Before binary search, sort all given types by i, and in case of equal i-s - by j.

During binary search, we should transform tiles into at least k $(0, 0)$-tiles, k $(0, 1)$-tiles, ..., (m, m)-tiles.

Suppose for beginning that we should build only tiles with $i = 0$. Then, create a stack ST of pairs (type of tile, number of such tiles). After that, iterate over all j from 0 to m and try to make k $(0, j)$-tiles by the following way:

J. Domino sets

- if there are z $(0, j)$-tiles in the initial tile, add pair $((0, j), z)$ into ST;
- after that, ST contains all tiles which can be transformed into $(0, j)$-tiles. If ST contains less than k elements then we cannot form k m-tiles; otherwise, it's optimal to do it from top k elements of ST, and we should remove them from ST.

It works in $O(m + v)$ time; but not that if input does not contains, for example, tiles $(0, l), (0, l + 1), \ldots, (0, r)$ for some segment $[l, r]$ then iterations $l, l + 1, \ldots, r$, can be performed as one iteration of removing $k * (r - l + 1)$ tiles from ST. This idea reduces the complexity to $O(v)$.

J. Domino sets

Then, learn how to solve a problem without assumption about $i = 0$.

First of all, tile with $i = 0$ can be made only from tiles with $i = 0$; try to build it by the way described above.

After that, ST may contain some tiles we can use later; for each such tile, increase i by one (and j if it is needed) and return it back to the initial set. Then repeat the algorithm for $i = 1, 2, \ldots, m$. It gives us a full check in $O(vm)$ time.

It is possible to reduce the time to $O(v^2)$ by the way similar to optimization of $[l, r]$. So, the total complexity of the solution is $O(v^2 * log\ MAX\_POSSIBLE\_ANSWER)$.