Training Contest 2 Editorial

May 1, 2021

Mikhail Tikhomirov & Gleb Evstropov

Bytedance-Moscow Workshops Training Camp, 2021

A. Another Copy of the Polygon

We are given a convex polygon A given as a sequence of vertices $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$ listed in the order of the traversal and a translation vector d. We have to find the total length of the perimeter of the union of A and B, which is A translated by vector d.

A. Another Copy of the Polygon

First we should deal with the easy case when A and B do not intersect. Then, the answer is just the doubled perimeter of polygon A.

A. Another Copy of the Polygon

First we should deal with the easy case when A and B do not intersect. Then, the answer is just the doubled perimeter of polygon A.

If A and B do intersect we can prove that the part of the border of the union belonging to A forms a continuous segment, as well as the part of the border of the union that belongs to B.

A. Another Copy of the Polygon

First we should deal with the easy case when A and B do not intersect. Then, the answer is just the doubled perimeter of polygon A.

If A and B do intersect we can prove that the part of the border of the union belonging to A forms a continuous segment, as well as the part of the border of the union that belongs to B.

Note this is not necessarily true for two general convex polygons, so we should use the fact that B is A transalted by d. Consider any line l parallel to vector d. Its intersection with A is some segment s_A, while its intersection with B is some segment s_B. Moreover, s_B is s_A translated by d.

A. Another Copy of the Polygon

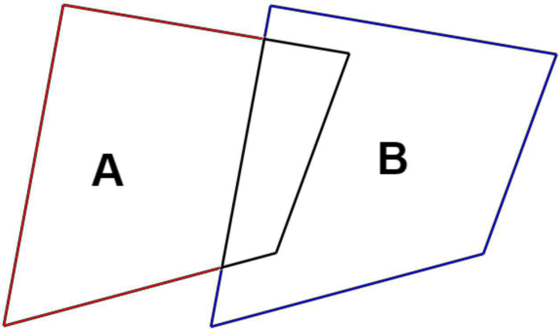
First we should deal with the easy case when A and B do not intersect. Then, the answer is just the doubled perimeter of polygon A.

If A and B do intersect we can prove that the part of the border of the union belonging to A forms a continuous segment, as well as the part of the border of the union that belongs to B.

Note this is not necessarily true for two general convex polygons, so we should use the fact that B is A transalted by d. Consider any line l parallel to vector d. Its intersection with A is some segment s_A, while its intersection with B is some segment s_B. Moreover, s_B is s_A translated by d.

Now we can assume that the set of points of the border of the union belonging to A is not connected and this will result the contradiction with the fact that A is convex.

A. Another Copy of the Polygon



A. Another Copy of the Polygon

To find the exact points of intersections of two borders we need to know the exact pairs of intersecting segments.

A. Another Copy of the Polygon

To find the exact points of intersections of two borders we need to know the exact pairs of intersecting segments.

Bruteforce way to approach this problem is to implement any data structure that allows to process “is point inside convex polygon?” queries.

A. Another Copy of the Polygon

To find the exact points of intersections of two borders we need to know the exact pairs of intersecting segments.

Bruteforce way to approach this problem is to implement any data structure that allows to process "is point inside convex polygon?" queries.

However, we can use the fact that points of A that lie inside B form a consecutive segment, thus we can first find any point inside and any point outside and then apply binary search algorithm and linear time "point inside polygon" check. To get one point of A inside B and one point of A outside B we can take the leftmost and the rightmost points of A if projected on any line parallel to vector d.

B. Byteland Routes

We are given an unweighted tree. *Importance* of a vertex v is $\min(d(v,1), d(v,2))$, where $d(v,u)$ is the distance between v and u, importance of a path is the minimal importance over all vertices in the path. Count total importance for all paths in the tree.

B. Byteland Routes

First, let us compute $a_v = \min(d(v, 1), d(v, 2))$ for all vertices v. This can be done most simply with two DFS'es started from vertices 1 and 2.

B. Byteland Routes

First, let us compute $a_v = \min(d(v,1), d(v,2))$ for all vertices v. This can be done most simply with two DFS'es started from vertices 1 and 2.

The rest of the solution works for arbitrary numbers a_v.

B. Byteland Routes

First, let us compute $a_v = \min(d(v,1), d(v,2))$ for all vertices v. This can be done most simply with two DFS'es started from vertices 1 and 2.

The rest of the solution works for arbitrary numbers a_v.

How many paths in the tree have importance at least x? We can count that as follows:

B. Byteland Routes

First, let us compute $a_v = \min(d(v,1), d(v,2))$ for all vertices v. This can be done most simply with two DFS'es started from vertices 1 and 2.

The rest of the solution works for arbitrary numbers a_v.

How many paths in the tree have importance at least x? We can count that as follows:

- Let us consider an edge (u, v) active if both a_v and a_u are at least x.

B. Byteland Routes

First, let us compute $a_v = \min(d(v, 1), d(v, 2))$ for all vertices v. This can be done most simply with two DFS'es started from vertices 1 and 2.

The rest of the solution works for arbitrary numbers a_v.

How many paths in the tree have importance at least x? We can count that as follows:

- Let us consider an edge (u, v) active if both a_v and a_u are at least x.
- The answer to the question above is the number of paths that use only active edges.

B. Byteland Routes

First, let us compute $a_v = \min(d(v, 1), d(v, 2))$ for all vertices v. This can be done most simply with two DFS'es started from vertices 1 and 2.

The rest of the solution works for arbitrary numbers a_v.

How many paths in the tree have importance at least x? We can count that as follows:

- Let us consider an edge (u, v) active if both a_v and a_u are at least x.
- The answer to the question above is the number of paths that use only active edges.

The latter is equal to $\sum \binom{s_i}{2}$, where the sum ranges over connected components over active edges, and s_i is the size of a component.

B. Byteland Routes

Let us “activate” vertices one by one in order of non-increasing a_v. An edge is activated only when both its endpoints become active.

B. Byteland Routes

Let us "activate" vertices one by one in order of non-increasing a_v. An edge is activated only when both its endpoints become active.

When we activate an edge, we unite its endpoints' components in a DSU, thus we have relevant connected components each time. We can also store the relevant value of $S = \sum \binom{s_i}{2}$.

B. Byteland Routes

Let us "activate" vertices one by one in order of non-increasing a_v. An edge is activated only when both its endpoints become active.

When we activate an edge, we unite its endpoints' components in a DSU, thus we have relevant connected components each time. We can also store the relevant value of $S = \sum \binom{s_i}{2}$.

Claim

The number of paths with importance x is $S_2 - S_1$, where S_1/S_2 is the value of S after activating all vertices with $a_v > x / a_v \geqslant x$.

B. Byteland Routes

Let us "activate" vertices one by one in order of non-increasing a_v. An edge is activated only when both its endpoints become active.

When we activate an edge, we unite its endpoints' components in a DSU, thus we have relevant connected components each time. We can also store the relevant value of $S = \sum \binom{s_i}{2}$.

Claim

The number of paths with importance x is $S_2 - S_1$, where S_1/S_2 is the value of S after activating all vertices with $a_v > x / a_v \geqslant x$.

This solution has complexity $O(n \log n)$ (for sorting the vertices by a_v), but can be optimized to $O(n\alpha(n))$, where $\alpha(n)$ is the inverse Ackermann function.

C. Complete Graph

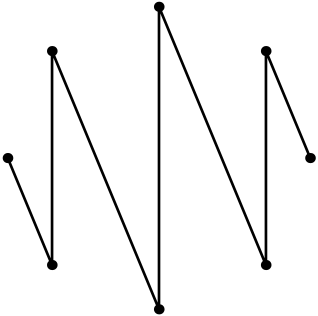
Divide edges of a complete graph on n vertices (n is odd) into $(n-1)/2$ disjoint Hamiltonian cycles.

C. Complete Graph

Let us instead divide a complete graph on $n - 1$ vertices into Hamiltonian *paths*. The answer to the original problem is then obtained by adding one more vertex and looping each Hamiltonian path through the new vertex.

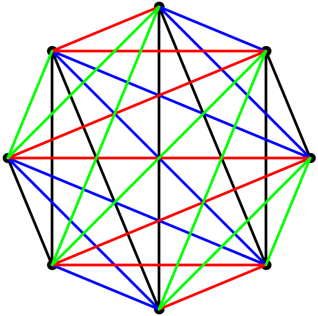
C. Complete Graph

One possible construction of dividing an even-sized complete graph into Hamiltonian paths is as follows. Let us draw the vertices on a circle equally spaced. A single Hamiltonian path can be obtained as follows:



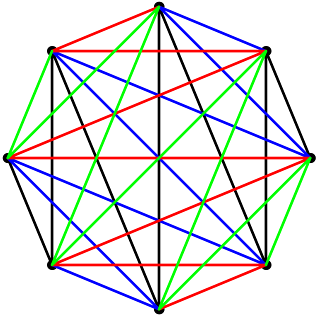
C. Complete Graph

Now all remaining paths can obtained from this path by “rotating” around the center:



C. Complete Graph

Now all remaining paths can obtained from this path by “rotating” around the center:



To understand why no two paths have common edges, notice that all parallel edges belong to the same path, and different slopes belong to different paths.

D. Day For Picnic

There are n distinct special points in the plane. Build a quadrilateral with distinct vertices in special points with area at most l, out of these choose one with maximal area.

D. Day For Picnic

Trivial $O(n^4)$ solution is too slow.

D. Day For Picnic

Trivial $O(n^4)$ solution is too slow.

Notice that each quadrilateral has a diagonal so that the remaining vertices lie strictly to the opposite sides of the line containing the diagonal.

D. Day For Picnic

Trivial $O(n^4)$ solution is too slow.

Notice that each quadrilateral has a diagonal so that the remaining vertices lie strictly to the opposite sides of the line containing the diagonal.

Try all options for endpoints of the diagonal pq . Consider all options for the point to the left of pq. By choosing a valid point r we increase the area by the area of the triangle pqr.

D. Day For Picnic

Trivial $O(n^4)$ solution is too slow.

Notice that each quadrilateral has a diagonal so that the remaining vertices lie strictly to the opposite sides of the line containing the diagonal.

Try all options for endpoints of the diagonal pq . Consider all options for the point to the left of pq. By choosing a valid point r we increase the area by the area of the triangle pqr.

That is, we have a set of numbers (triangle areas) and have to choose one to add to the area.

D. Day For Picnic

Trivial $O(n^4)$ solution is too slow.

Notice that each quadrilateral has a diagonal so that the remaining vertices lie strictly to the opposite sides of the line containing the diagonal.

Try all options for endpoints of the diagonal pq . Consider all options for the point to the left of pq. By choosing a valid point r we increase the area by the area of the triangle pqr.

That is, we have a set of numbers (triangle areas) and have to choose one to add to the area.

Similarly, another set of numbers is available to the other side.

D. Day For Picnic

We now have a problem: choose a number x from set A and a number y from set B so that $x + y \leqslant L$, but otherwise largest possible.

D. Day For Picnic

We now have a problem: choose a number x from set A and a number y from set B so that $x + y \leqslant L$, but otherwise largest possible.

Let us sort A and B. Now the problem can be solved in $O(|A| + |B|)$ time by the "two pointers" approach: for each position i in the first sequence maintain the largest position j such that $a_i + b_j \leqslant L$. As i increases, j can only decrease.

D. Day For Picnic

We now have a problem: choose a number x from set A and a number y from set B so that $x + y \leqslant L$, but otherwise largest possible.

Let us sort A and B. Now the problem can be solved in $O(|A| + |B|)$ time by the "two pointers" approach: for each position i in the first sequence maintain the largest position j such that $a_i + b_j \leqslant L$. As i increases, j can only decrease.

The resulting solution has complexity $O(n^3 \log n)$.

E. Easy Guessing Game

This was interactive problem. Jury's program thinks of one number from 1 to n. Your program can ask whether jury's number is less than x given by you. You are playing this game k times, you have to guess all k numbers.

The number of questions shouldn't exceed $k(\log(n+1) + \frac{1}{10})$

E. Easy Guessing Game

Solution

- One could do binary search, but it only gives us $\lceil \log n \rceil$ queries, in worst case

E. Easy Guessing Game

Solution

- One could do binary search, but it only gives us $\lceil \log n \rceil$ queries, in worst case
- Idea is to build the tree of decisions, so that with high enough probability we get $\lfloor \log n \rfloor$ queries made, instead of $\lceil \log n \rceil$

E. Easy Guessing Game

Solution

- One could do binary search, but it only gives us $\lceil \log n \rceil$ queries, in worst case
- Idea is to build the tree of decisions, so that with high enough probability we get $\lfloor \log n \rfloor$ queries made, instead of $\lceil \log n \rceil$
- So let's build a tree, with even number (n or $n + 1$) leaves, so that every vertex is either on $\lfloor \log n + 1 \rfloor$ or $\lceil \log n + 1 \rceil$ depth

E. Easy Guessing Game

Solution

- One could do binary search, but it only gives us $\lceil \log n \rceil$ queries, in worst case
- Idea is to build the tree of decisions, so that with high enough probability we get $\lfloor \log n \rfloor$ queries made, instead of $\lceil \log n \rceil$
- So let's build a tree, with even number $(n \text{ or } n+1)$ leaves, so that every vertex is either on $\lfloor \log n + 1 \rfloor$ or $\lceil \log n + 1 \rceil$ depth
  - Consider $2^k \leqslant n$, with maximal k chosen

E. Easy Guessing Game

Solution

- One could do binary search, but it only gives us $\lceil \log n \rceil$ queries, in worst case
- Idea is to build the tree of decisions, so that with high enough probability we get $\lfloor \log n \rfloor$ queries made, instead of $\lceil \log n \rceil$
- So let's build a tree, with even number $(n$ or $n+1)$ leaves, so that every vertex is either on $\lfloor \log n + 1 \rfloor$ or $\lceil \log n + 1 \rceil$ depth
    - Consider $2^k \leqslant n$, with maximal k chosen
    - We can get $n - 2^k$ elements and pair them up with some other $n - 2^k$ elements, to make them paired leaves, and leave us with 2^k vertices

E. Easy Guessing Game

Solution

- One could do binary search, but it only gives us $\lceil \log n \rceil$ queries, in worst case
- Idea is to build the tree of decisions, so that with high enough probability we get $\lfloor \log n \rfloor$ queries made, instead of $\lceil \log n \rceil$
- So let's build a tree, with even number $(n$ or $n+1)$ leaves, so that every vertex is either on $\lfloor \log n+1 \rfloor$ or $\lceil \log n+1 \rceil$ depth
  - Consider $2^k \leqslant n$, with maximal k chosen
  - We can get $n - 2^k$ elements and pair them up with some other $n - 2^k$ elements, to make them paired leaves, and leave us with 2^k vertices
  - Then build full binary tree from left elements

E. Easy Guessing Game

Choosing random tree

- Making each vertex to be on $\lfloor \log n + 1 \rfloor$ depth with equal probability makes us to solve the problem

E. Easy Guessing Game

Choosing random tree

- Making each vertex to be on $\lfloor \log n + 1 \rfloor$ depth with equal probability makes us to solve the problem
- To make equal the probability let's all leaves up with its neighboring value

E. Easy Guessing Game

Choosing random tree

- Making each vertex to be on $\lfloor \log n + 1 \rfloor$ depth with equal probability makes us to solve the problem
- To make equal the probability let's all leaves up with its neighboring value
  - If there are $2d$ leaves then d pairs made

E. Easy Guessing Game

Choosing random tree

- Making each vertex to be on $\lfloor \log n + 1 \rfloor$ depth with equal probability makes us to solve the problem
- To make equal the probability let's all leaves up with its neighboring value
  - If there are $2d$ leaves then d pairs made
  - Choose x and get $d - 2^k$ consecutive pairs starting from x in cyclic list of pairs

Proof

- Say n is equal to number of leaves, $m = 2^k \leqslant n < 2^{k+1}$ and $n = m + t$

E. Easy Guessing Game

ByteDance Moscow Workshops

Choosing random tree

- Making each vertex to be on $\lfloor \log n + 1 \rfloor$ depth with equal probability makes us to solve the problem
- To make equal the probability let's all leaves up with its neighboring value
  - If there are $2d$ leaves then d pairs made
  - Choose x and get $d - 2^k$ consecutive pairs starting from x in cyclic list of pairs

Proof

- Say n is equal to number of leaves, $m = 2^k \leqslant n < 2^{k+1}$ and $n = m + t$
- We are to prove: $k(m - t) + (k + 1) \cdot 2t \leqslant (m + t)(\log(m + t) + \frac{1}{10})$

E. Easy Guessing Game

Proof

$$k(m-t)+(k+1)\cdot 2t \leqslant (m+t)(\log(m+t)+\tfrac{1}{10})$$

E. Easy Guessing Game

Proof

$$k(m-t)+(k+1)\cdot 2t \leqslant (m+t)(\log(m+t)+\frac{1}{10})$$
$$k(m+t)+2t \leqslant (m+t)(\log(m+t)+\frac{1}{10})$$

E. Easy Guessing Game

Proof

$$k(m-t)+(k+1)\cdot 2t \leqslant (m+t)(\log(m+t)+\frac{1}{10})$$

$$k(m+t)+2t \leqslant (m+t)(\log(m+t)+\frac{1}{10})$$

$$2t \leqslant (m+t)(\log\frac{m+t}{m}+\frac{1}{10})$$

E. Easy Guessing Game

Proof

$k(m-t)+(k+1)\cdot 2t \leqslant (m+t)(\log(m+t)+\frac{1}{10})$

$k(m+t)+2t \leqslant (m+t)(\log(m+t)+\frac{1}{10})$

$2t \leqslant (m+t)(\log\frac{m+t}{m}+\frac{1}{10})$

Say $t = mx$

E. Easy Guessing Game

Proof

$k(m-t)+(k+1)\cdot 2t \leqslant (m+t)(\log(m+t)+\frac{1}{10})$

$k(m+t)+2t \leqslant (m+t)(\log(m+t)+\frac{1}{10})$

$2t \leqslant (m+t)(\log\frac{m+t}{m}+\frac{1}{10})$

Say $t = mx$

$2mx \leqslant m(1+x)(\log(1+x)+\frac{1}{10})$

E. Easy Guessing Game

Proof

$$k(m-t)+(k+1)\cdot 2t \leqslant (m+t)(\log(m+t)+\tfrac{1}{10})$$
$$k(m+t)+2t \leqslant (m+t)(\log(m+t)+\tfrac{1}{10})$$
$$2t \leqslant (m+t)(\log\tfrac{m+t}{m}+\tfrac{1}{10})$$
Say $t = mx$
$$2mx \leqslant m(1+x)(\log(1+x)+\tfrac{1}{10})$$
$$2x \leqslant (1+x)(\log(1+x)+\tfrac{1}{10})$$

E. Easy Guessing Game

Proof

$$k(m-t)+(k+1)\cdot 2t \leqslant (m+t)(\log(m+t)+\tfrac{1}{10})$$
$$k(m+t)+2t \leqslant (m+t)(\log(m+t)+\tfrac{1}{10})$$
$$2t \leqslant (m+t)(\log\tfrac{m+t}{m}+\tfrac{1}{10})$$

Say $t = mx$

$$2mx \leqslant m(1+x)(\log(1+x)+\tfrac{1}{10})$$
$$2x \leqslant (1+x)(\log(1+x)+\tfrac{1}{10})$$
$$\tfrac{2x}{1+x}-\log(1+x) \leqslant \tfrac{1}{10}$$

E. Easy Guessing Game

Proof

$k(m-t)+(k+1)\cdot 2t \leqslant (m+t)(\log(m+t)+\frac{1}{10})$

$k(m+t)+2t \leqslant (m+t)(\log(m+t)+\frac{1}{10})$

$2t \leqslant (m+t)(\log\frac{m+t}{m}+\frac{1}{10})$

Say $t = mx$

$2mx \leqslant m(1+x)(\log(1+x)+\frac{1}{10})$

$2x \leqslant (1+x)(\log(1+x)+\frac{1}{10})$

$\frac{2x}{1+x}-\log(1+x) \leqslant \frac{1}{10}$

Say $f(x) = \frac{2x}{1+x}-\log(1+x)$

E. Easy Guessing Game

Proof

$k(m-t)+(k+1)\cdot 2t \leqslant (m+t)(\log(m+t)+\frac{1}{10})$

$k(m+t)+2t \leqslant (m+t)(\log(m+t)+\frac{1}{10})$

$2t \leqslant (m+t)(\log\frac{m+t}{m}+\frac{1}{10})$

Say $t = mx$

$2mx \leqslant m(1+x)(\log(1+x)+\frac{1}{10})$

$2x \leqslant (1+x)(\log(1+x)+\frac{1}{10})$

$\frac{2x}{1+x}-\log(1+x) \leqslant \frac{1}{10}$

Say $f(x) = \frac{2x}{1+x}-\log(1+x)$

Let's find $\max\limits_{0\leqslant x\leqslant 1} f(x)$, knowing $f(0)=f(1)=0$

E. Easy Guessing Game

Proof

$$k(m-t)+(k+1)\cdot 2t \leqslant (m+t)(\log(m+t)+\frac{1}{10})$$

$$k(m+t)+2t \leqslant (m+t)(\log(m+t)+\frac{1}{10})$$

$$2t \leqslant (m+t)(\log\frac{m+t}{m}+\frac{1}{10})$$

Say $t = mx$

$$2mx \leqslant m(1+x)(\log(1+x)+\frac{1}{10})$$

$$2x \leqslant (1+x)(\log(1+x)+\frac{1}{10})$$

$$\frac{2x}{1+x}-\log(1+x) \leqslant \frac{1}{10}$$

Say $f(x) = \frac{2x}{1+x} - \log(1+x)$

Let's find $\max\limits_{0\leqslant x\leqslant 1} f(x)$, knowing $f(0)=f(1)=0$

$$f'(x) = \frac{2}{(1+x)^2} - \frac{\log e}{1+x} = 0$$

E. Easy Guessing Game

Proof

$k(m-t)+(k+1)\cdot 2t \leqslant (m+t)(\log(m+t)+\frac{1}{10})$
$k(m+t)+2t \leqslant (m+t)(\log(m+t)+\frac{1}{10})$
$2t \leqslant (m+t)(\log\frac{m+t}{m}+\frac{1}{10})$
Say $t = mx$
$2mx \leqslant m(1+x)(\log(1+x)+\frac{1}{10})$
$2x \leqslant (1+x)(\log(1+x)+\frac{1}{10})$
$\frac{2x}{1+x}-\log(1+x) \leqslant \frac{1}{10}$
Say $f(x)=\frac{2x}{1+x}-\log(1+x)$
Let's find $\max\limits_{0\leqslant x\leqslant 1} f(x)$, knowing $f(0)=f(1)=0$
$f'(x)=\frac{2}{(1+x)^2}-\frac{\log e}{1+x}=0$
$2=(1+x)\log e \Rightarrow (1+x)=2\ln 2 \Rightarrow x=2\ln 2-1$

E. Easy Guessing Game

ByteDance Moscow Workshops

Proof

$k(m-t)+(k+1)\cdot 2t \leqslant (m+t)(\log(m+t)+\frac{1}{10})$

$k(m+t)+2t \leqslant (m+t)(\log(m+t)+\frac{1}{10})$

$2t \leqslant (m+t)(\log\frac{m+t}{m}+\frac{1}{10})$

Say $t = mx$

$2mx \leqslant m(1+x)(\log(1+x)+\frac{1}{10})$

$2x \leqslant (1+x)(\log(1+x)+\frac{1}{10})$

$\frac{2x}{1+x} - \log(1+x) \leqslant \frac{1}{10}$

Say $f(x) = \frac{2x}{1+x} - \log(1+x)$

Let's find $\max\limits_{0\leqslant x\leqslant 1} f(x)$, knowing $f(0) = f(1) = 0$

$f'(x) = \frac{2}{(1+x)^2} - \frac{\log e}{1+x} = 0$

$2 = (1+x)\log e \Rightarrow (1+x) = 2\ln 2 \Rightarrow x = 2\ln 2 - 1$

$f(x) = \frac{2\ln 2 - 1}{\ln 2} - \log(2\ln 2) \approx 0.08607$

E. Easy Guessing Game

Proof

$$k(m-t)+(k+1)\cdot 2t \leqslant (m+t)(\log(m+t)+\tfrac{1}{10})$$
$$k(m+t)+2t \leqslant (m+t)(\log(m+t)+\tfrac{1}{10})$$
$$2t \leqslant (m+t)(\log\tfrac{m+t}{m}+\tfrac{1}{10})$$

Say $t = mx$

$$2mx \leqslant m(1+x)(\log(1+x)+\tfrac{1}{10})$$
$$2x \leqslant (1+x)(\log(1+x)+\tfrac{1}{10})$$
$$\tfrac{2x}{1+x}-\log(1+x) \leqslant \tfrac{1}{10}$$

Say $f(x) = \frac{2x}{1+x} - \log(1+x)$

Let's find $\max\limits_{0\leqslant x\leqslant 1} f(x)$, knowing $f(0)=f(1)=0$

$$f'(x) = \tfrac{2}{(1+x)^2} - \tfrac{\log e}{1+x} = 0$$
$$2 = (1+x)\log e \Rightarrow (1+x) = 2\ln 2 \Rightarrow x = 2\ln 2 - 1$$
$$f(x) = \tfrac{2\ln 2 - 1}{\ln 2} - \log(2\ln 2) \approx 0.08607$$

So $f(x) < 0.1$

F. Fred's Parquet

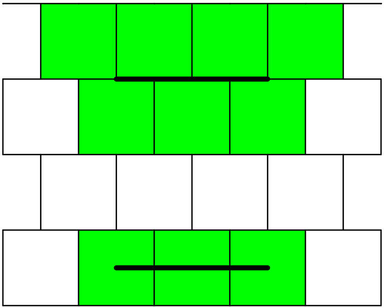
A rectangular border is drawn in the plane with a regular shifted square tiling. Find the number of squares that have a common point with the border.

F. Fred's Parquet

With a reckless approach, the number of corner cases in a solution of this problem can be huge. Let's try to minimize the number of these cases (in fact, we pretty much won't have any).

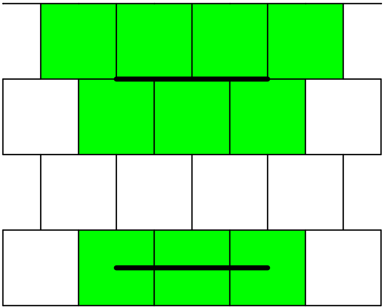
F. Fred's Parquet

Let us consider only the horizontal sides first. For each horizontal segment, the set of concerned squares is one or two horizontal ranges of squares.



F. Fred’s Parquet

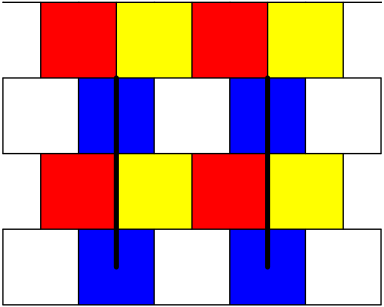
Let us consider only the horizontal sides first. For each horizontal segment, the set of concerned squares is one or two horizontal ranges of squares.



Notice that when the segments are too close a range may correspond to both of them, so we have to eliminate repetitions.

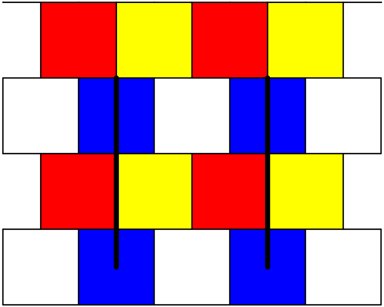
F. Fred's Parquet

We have a similar situation for the vertical sides, except that it is now convenient to define a vertical range as a set of squares with vertical shift 2 from each other (marked with different colors on the picture below).



F. Fred's Parquet

We have a similar situation for the vertical sides, except that it is now convenient to define a vertical range as a set of squares with vertical shift 2 from each other (marked with different colors on the picture below).



Similarly, we have to eliminate repetitions among vertical ranges.

F. Fred's Parquet

Finally, the answer is almost the sum of lengths of all ranges (horizontal and vertical), except for horizontal-vertical intersections.

F. Fred's Parquet

Finally, the answer is almost the sum of lengths of all ranges (horizontal and vertical), except for horizontal-vertical intersections.

Since each square belongs to at most one horizontal and at most one vertical range, we have to consider all horizontal-vertical pairs of ranges and subtract 1 for each intersection (remember to account for parity of vertical ranges).

F. Fred's Parquet

Finally, the answer is almost the sum of lengths of all ranges (horizontal and vertical), except for horizontal-vertical intersections.

Since each square belongs to at most one horizontal and at most one vertical range, we have to consider all horizontal-vertical pairs of ranges and subtract 1 for each intersection (remember to account for parity of vertical ranges).

Time complexity is $O(1)$.

G. Game With Mirroring

Find the number of solutions to $x + rev(x) = z$, where $rev(x)$ is the number obtained from x by reversing its decimal representation.

G. Game With Mirroring

Let us decide the value of l — the length of x (notice that is either length(z) or length$(z) - 1$).

G. Game With Mirroring

Let us decide the value of l — the length of x (notice that is either length(z) or length$(z) - 1$).

Let us decide the leading and the last digit of x simultaneously, denote them d and d'. If the last digit of z is c, we must have either $d + d' = c$ or $d + d' = c + 10$.

G. Game With Mirroring

Let us decide the value of l — the length of x (notice that is either length(z) or length$(z) - 1$).

Let us decide the leading and the last digit of x simultaneously, denote them d and d'. If the last digit of z is c, we must have either $d + d' = c$ or $d + d' = c + 10$.

Note that choosing values of d and d' is unimportant to determine the sum, only $d + d'$ matters.

G. Game With Mirroring

Let us decide the value of l — the length of x (notice that is either $\text{length}(z)$ or $\text{length}(z) - 1$).

Let us decide the leading and the last digit of x simultaneously, denote them d and d'. If the last digit of z is c, we must have either $d + d' = c$ or $d + d' = c + 10$.

Note that choosing values of d and d' is unimportant to determine the sum, only $d + d'$ matters.

Deciding rest of the digits is essentially solving the same problem for $z' = (z - (d + d') \times (1 + 10^{l-1}))/10$ under $\text{length}(x) = l - 2$. The answer to this subproblem has to be multiplied by the number of ways to partition $d + d'$ into digits d and d'.

G. Game With Mirroring

Let us decide the value of l — the length of x (notice that is either length(z) or length$(z) - 1$).

Let us decide the leading and the last digit of x simultaneously, denote them d and d'. If the last digit of z is c, we must have either $d + d' = c$ or $d + d' = c + 10$.

Note that choosing values of d and d' is unimportant to determine the sum, only $d + d'$ matters.

Deciding rest of the digits is essentially solving the same problem for $z' = (z - (d + d') \times (1 + 10^{l-1}))/10$ under length$(x) = l - 2$. The answer to this subproblem has to be multiplied by the number of ways to partition $d + d'$ into digits d and d'.

A bit of care is needed to forbid leading zeros in x.

G. Game With Mirroring

Since length(z) is at most 19, within 10 recursive steps we will determine x completely. Since we have two branches at each step, the complexity is 2^{10} per test case.

G. Game With Mirroring

Since length(z) is at most 19, within 10 recursive steps we will determine x completely. Since we have two branches at each step, the complexity is 2^{10} per test case.

Many other solutions are possible, including digit-wise DP or optimized testing of all possible x.

H. How Long Till Connection?

Given a bipartite graph: the first part has n vertices and the second one has m vertices.

Initially there are k edges in it.

Then you add edge one by one, on i-th step: vertex ($i \bmod n$) from the first part is connected to the vertex ($i \bmod m$) from the second part.

Find out, after how many added edges the graph becomes connected.

H. How Long Till Connection?

Reducing the problem

- Let's make another graph, not necessarily bipartite containing $n + m$ vertices

H. How Long Till Connection?

Reducing the problem

- Let's make another graph, not necessarily bipartite containing $n + m$ vertices
- And on i-th step connect vertices $i \bmod (n + m)$ and $(i + n) \bmod (n + m)$
  - Here we need no more than $n + m$ steps

H. How Long Till Connection?

Reducing the problem

- Let's make another graph, not necessarily bipartite containing $n + m$ vertices
- And on i-th step connect vertices $i \bmod (n + m)$ and $(i + n) \bmod (n + m)$
    - Here we need no more than $n + m$ steps
- So the problem would be the same, as far as we connected the vertices from the same connected components as before

H. How Long Till Connection?

Reducing the problem

- Let's make another graph, not necessarily bipartite containing $n + m$ vertices
- And on i-th step connect vertices $i \bmod (n + m)$ and $(i + n) \bmod (n + m)$
  - Here we need no more than $n + m$ steps
- So the problem would be the same, as far as we connected the vertices from the same connected components as before
- One can see, that before i-th step vertices i and $i \bmod n$ were connected, so were $(i + n) \bmod (n + m)$ and $(i \bmod m) + n$

H. How Long Till Connection?

Reducing the problem

- Let's make another graph, not necessarily bipartite containing $n + m$ vertices
- And on i-th step connect vertices $i \bmod (n + m)$ and $(i + n) \bmod (n + m)$
  - Here we need no more than $n + m$ steps
- So the problem would be the same, as far as we connected the vertices from the same connected components as before
- One can see, that before i-th step vertices i and $i \bmod n$ were connected, so were $(i + n) \bmod (n + m)$ and $(i \bmod m) + n$
- So after connecting $i \bmod (n + m)$ and $(i + n) \bmod (n + m)$, we get i and $(i \bmod m) + n$ connected

H. How Long Till Connection?

Solving the new problem

- So we have a vertices and we connect i with $(i + b) \bmod a$

H. How Long Till Connection?

ByteDance
Moscow Workshops

Solving the new problem

- So we have a vertices and we connect i with $(i+b) \bmod a$
- During the first $a-b$ steps we connect i with $i+b$
  - All pairs $v \bmod b = u \bmod b$ are connected after $a-b$ steps

H. How Long Till Connection?

Solving the new problem

- So we have a vertices and we connect i with $(i + b) \bmod a$
- During the first $a - b$ steps we connect i with $i + b$
  - All pairs $v \bmod b = u \bmod b$ are connected after $a - b$ steps
- So afterwards we can reduce the number of vertices to b
  - And i is connected to $(i + (a \bmod b)) \bmod b$ on i-th step

H. How Long Till Connection?

Solving the new problem

- So we have a vertices and we connect i with $(i+b) \bmod a$
- During the first $a-b$ steps we connect i with $i+b$
  - All pairs $v \bmod b = u \bmod b$ are connected after $a-b$ steps
- So afterwards we can reduce the number of vertices to b
  - And i is connected to $(i + (a \bmod b)) \bmod b$ on i-th step
- Check if after first $a-b$ steps graph is still not connected
  - Get all vertices modulo b and check if the graph connected by initial k edges

H. How Long Till Connection?

Solving the new problem

- So we have a vertices and we connect i with $(i + b) \bmod a$
- During the first $a - b$ steps we connect i with $i + b$
  - All pairs $v \bmod b = u \bmod b$ are connected after $a - b$ steps
- So afterwards we can reduce the number of vertices to b
  - And i is connected to $(i + (a \bmod b)) \bmod b$ on i-th step
- Check if after first $a - b$ steps graph is still not connected
  - Get all vertices modulo b and check if the graph connected by initial k edges
- If it's not connected, then reduce the number of vertices and solve the same problem for b and $a \bmod b$

H. How Long Till Connection?

Solving the new problem

- So we have a vertices and we connect i with $(i+b) \bmod a$
- During the first $a-b$ steps we connect i with $i+b$
  - All pairs $v \bmod b = u \bmod b$ are connected after $a-b$ steps
- So afterwards we can reduce the number of vertices to b
  - And i is connected to $(i+(a \bmod b)) \bmod b$ on i-th step
- Check if after first $a-b$ steps graph is still not connected
  - Get all vertices modulo b and check if the graph connected by initial k edges
- If it's not connected, then reduce the number of vertices and solve the same problem for b and $a \bmod b$
- Otherwise use binary search to find the answer
  - Consider you added t edges, then $a-b-t$ vertices left to connect with initial k edges
  - So each check is always done in $O(k)$

I. Imoaix

In a $(n+1) \times (n+1)$ grid each cell contains a number of coins. We have to make a circular tour of $4n$ steps that visits each of the corners so that each step is a chess king move, while collecting as much coins as possible.

I. Imoaix

If we had to find a single optimal shortest route from one cell to another, we would apply the classic DP solution. Sadly, this problem is not directly reducible to several independent subproblems of this kind since each coin can be picked up at most once.

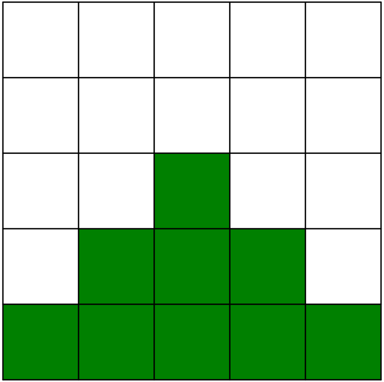
I. Imoaix

If we had to find a single optimal shortest route from one cell to another, we would apply the classic DP solution. Sadly, this problem is not directly reducible to several independent subproblems of this kind since each coin can be picked up at most once.

We can assume that the circular tour visits corner in clockwise order. Indeed, we can reverse the tour if its counter-clockwise, and if the tour includes going from a corner straight to its opposite, then the tour must have a self-intersection in the center of the grid, which allows us to swap parts of the tour (possible with reverse) to get rid of traversing the diagonal.

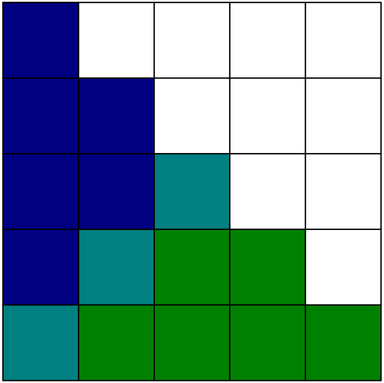
I. Imoaix

For a pair of adjacent corners each shortest path connecting them must lie inside a “quarter-square”. We call finding a path for each of these a *subproblem*.



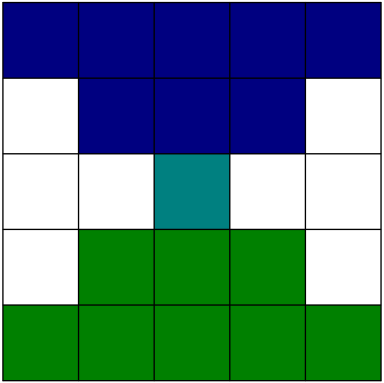
I. Imoaix

Cells reachable from two subproblems corresponding to adjacent sides of the square are half the diagonal (including the center when $n+1$ is odd).



I. Imoaix

And for opposite sides only the center is possibly reachable in both subproblems.



I. Imoaix

Consider a route that uses a non-trivial part of the common diagonal for adjacent subproblems. Let us see that this route can not be optimal.

I. Imoaix

Consider a route that uses a non-trivial part of the common diagonal for adjacent subproblems. Let us see that this route can not be optimal.

Indeed, in one of this subproblems there is no gain from diagonal cells at all (since all of the coins are gathered in the other subproblem). But then the path can be altered to go right next to the diagonal, which is clearly a profit.

I. Imoaix

Consider a route that uses a non-trivial part of the common diagonal for adjacent subproblems. Let us see that this route can not be optimal.

Indeed, in one of this subproblems there is no gain from diagonal cells at all (since all of the coins are gathered in the other subproblem). But then the path can be altered to go right next to the diagonal, which is clearly a profit.

It follows that in each pair of adjacent subproblems one of them doesn't make use of the diagonal at all.

I. Imoaix

Consider a route that uses a non-trivial part of the common diagonal for adjacent subproblems. Let us see that this route can not be optimal.

Indeed, in one of this subproblems there is no gain from diagonal cells at all (since all of the coins are gathered in the other subproblem). But then the path can be altered to go right next to the diagonal, which is clearly a profit.

It follows that in each pair of adjacent subproblems one of them doesn't make use of the diagonal at all.

Similarly, only one of the four subproblems make use of the center.

I. Imoaix

To allow for all possible solutions, let us try all options of allowing diagonal parts to corresponding subproblems, and allowing the center to one of them. In each of these options, the subproblems become independent, and thus can be solved in $O(n^2)$ time.

I. Imoaix

To allow for all possible solutions, let us try all options of allowing diagonal parts to corresponding subproblems, and allowing the center to one of them. In each of these options, the subproblems become independent, and thus can be solved in $O(n^2)$ time.

The number of options is roughty bounded by $2^4 \cdot 4$, hence the solution is fast enough.

J. Journey Of Cat

An $2^n \times 2^{n+1}$ rectangle is divided recursively into 4^n tiles of size 1×2 each. We follow a walk on this rectangle. For each step determine if we've changed a tile on this step or not.

J. Journey Of Cat

For any cell we can easily determine (in constant time) to which one of the four top-level subrectangles it belongs.

J. Journey Of Cat

For any cell we can easily determine (in constant time) to which one of the four top-level subrectangles it belongs.

If the subrectangle is divided further, we find the next subrectangle similarly, and so on until we arrive at the 1×2 tile. When descending, two possible orientations of a rectangle should be treated separately.

J. Journey Of Cat

For any cell we can easily determine (in constant time) to which one of the four top-level subrectangles it belongs.

If the subrectangle is divided further, we find the next subrectangle similarly, and so on until we arrive at the 1×2 tile. When descending, two possible orientations of a rectangle should be treated separately.

Hence, the problem is easily solved in $O(nl)$ time, where l is the length of the walk.

K. King Size

We are given an $r \times c$ table containing decimal digits and q queries "sum in rectangle". As the table is large, each row is given by its period of length no more than $s \leqslant 100$.

K. King Size

One rectangle sum query can be split in four prefix-rectangle queries, i.e.

$$sum(l, u, r, d) = psum(r, d) - psum(l-1, d) - psum(r, u-1) + psum(l-1, u$$

K. King Size

One rectangle sum query can be split in four prefix-rectangle queries, i.e.

$$sum(l, u, r, d) = psum(r, d) - psum(l-1, d) - psum(r, u-1) + psum(l-1, u$$

For each query we can independently compute the sum in all rows with period length equal to x.

K. King Size

One rectangle sum query can be split in four prefix-rectangle queries, i.e.

$$sum(l, u, r, d) = psum(r, d) - psum(l-1, d) - psum(r, u-1) + psum(l-1, u$$

For each query we can independently compute the sum in all rows with period length equal to x.

Consider there are m_x rows with period equal to x. For each row from 1 to r precompute the last row with period x. This can be done in $O(rs)$ time.

K. King Size

One rectangle sum query can be split in four prefix-rectangle queries, i.e.

$$sum(l, u, r, d) = psum(r, d) - psum(l-1, d) - psum(r, u-1) + psum(l-1, u$$

For each query we can independently compute the sum in all rows with period length equal to x.

Consider there are m_x rows with period equal to x. For each row from 1 to r precompute the last row with period x. This can be done in $O(rs)$ time.

For each $psum(a, b)$ and fixed x we know can compute the number of times the whole period will fit into the sum plus the length of the remaining prefix. For each y from 0 to x and i from 0 to m_x compute the sum of prefixes of length y for first i rows with period equal to x. This can be done in $O(m_x \cdot x)$ time, thus in $O(rs)$ in

K. King Size

The total precalculation running time is $O(rs)$.

K. King Size

The total precalculation running time is $O(rs)$.

To process a single query one has to split it in four prefix queries, then for each of them try all valid period lengths and query for the corresponding prefix sum. The running time is $O(s)$ per query.

K. King Size

The total precalculation running time is $O(rs)$.

To process a single query one has to split it in four prefix queries, then for each of them try all valid period lengths and query for the corresponding prefix sum. The running time is $O(s)$ per query.

The overall running time of the whole solution will be $O(s(r + q))$.

L. Laura's Function

For a square $n \times n$ matrix A, the value $f(A)$ is equal to the sum of absolute values of differences of all n^4 ordered pairs of elements of A. Find the sum of $f(A)$ over all contiguous $k \times k$ submatrices of the given $n \times m$ matrix B.

L. Laura's Function

How can we find a single value of $f(A)$ faster than in $O(n^4)$ time?

L. Laura's Function

How can we find a single value of $f(A)$ faster than in $O(n^4)$ time?

Let us sort all elements of A, denote them $a_1, \ldots, a_{n^2}$ in this order. The sum if now equal to $\sum_{1 \leqslant i,j \leqslant n^2} |a_i - a_j|$.

L. Laura's Function

How can we find a single value of $f(A)$ faster than in $O(n^4)$ time?

Let us sort all elements of A, denote them $a_1, \ldots, a_{n^2}$ in this order. The sum if now equal to $\sum_{1 \leqslant i,j \leqslant n^2} |a_i - a_j|$.

Since a_i are ordered, we can expand the brackets in each absolute value with a certain sign. After summing similar terms, we have that the sum is equal to $-\sum_{i=1}^{n^2} 2(n^2 - 2i - 1)a_i$.

L. Laura's Function

We can apply this idea directly to the original problem to obtain an $O(nm(k^2 + \log(nm)))$ solution.

L. Laura's Function

We can apply this idea directly to the original problem to obtain an $O(nm(k^2 + \log(nm)))$ solution.

Sort all elements of B. We will process them one by one, and basically do the former solution for all $k \times k$ submatrices in parallel.

L. Laura's Function

We can apply this idea directly to the original problem to obtain an $O(nm(k^2 + \log(nm)))$ solution.

Sort all elements of B. We will process them one by one, and basically do the former solution for all $k \times k$ submatrices in parallel.

For each submatrix let us maintain a counter of how many processed elements belonged to the submatrix (we will refer to each $k \times k$ submatrix by the position of its top left corner).

L. Laura's Function

We can apply this idea directly to the original problem to obtain an $O(nm(k^2 + \log(nm)))$ solution.

Sort all elements of B. We will process them one by one, and basically do the former solution for all $k \times k$ submatrices in parallel.

For each submatrix let us maintain a counter of how many processed elements belonged to the submatrix (we will refer to each $k \times k$ submatrix by the position of its top left corner).

For the current element x, iterate over all submatrices it belongs to. For a particular submatrix, we increase its counter $cnt_{i,j}$ by one. At this point, we know that x contributes $-2x(k^2 - 2cnt_{i,j} - 1)$ to the total.

L. Laura's Function

We can apply this idea directly to the original problem to obtain an $O(nm(k^2 + \log(nm)))$ solution.

Sort all elements of B. We will process them one by one, and basically do the former solution for all $k \times k$ submatrices in parallel.

For each submatrix let us maintain a counter of how many processed elements belonged to the submatrix (we will refer to each $k \times k$ submatrix by the position of its top left corner).

For the current element x, iterate over all submatrices it belongs to. For a particular submatrix, we increase its counter $cnt_{i,j}$ by one. At this point, we know that x contributes $-2x(k^2 - 2cnt_{i,j} - 1)$ to the total.

Alternatively, we can keep the value of $w_{i,j} = -(k^2 - 2cnt_{i,j} - 1)$ for each submatrix, then each subsequent element increases $w_{i,j}$ by 2

L. Laura's Function

Let's optimize this solution. We can see that the problem is reduced to 2D range queries of two types:

- take the sum of $w_{i,j}$ over a subrectangle;
- increase all $w_{i,j}$ within a subrectangle by 2.

L. Laura's Function

Let's optimize this solution. We can see that the problem is reduced to 2D range queries of two types:

- take the sum of $w_{i,j}$ over a subrectangle;
- increase all $w_{i,j}$ within a subrectangle by 2.

A data structure that allows to perform each of these operations in $O(\log n \log m)$ is 2D binary indexed tree (aka BIT, Fenwick tree) with range addition. The rest of the discussion is a description of this structure.

L. Laura's Function

Let's start with the one-dimensional case. For the purpose of our discussion, BIT is an abstract RSQ data structure that allows changing a single element of an array and computing range sums (clearly, prefix sums always suffice).

L. Laura's Function

Let's start with the one-dimensional case. For the purpose of our discussion, BIT is an abstract RSQ data structure that allows changing a single element of an array and computing range sums (clearly, prefix sums always suffice).

A range query "add x to $a_l, \ldots, a_r$" can be interchanged with two prefix queries "add x to $a_1, \ldots, a_r$" and "add $-x$ to $a_1, \ldots, a_{l-1}$". We only consider prefix additions from now on.

L. Laura's Function

Let's start with the one-dimensional case. For the purpose of our discussion, BIT is an abstract RSQ data structure that allows changing a single element of an array and computing range sums (clearly, prefix sums always suffice).

A range query "add x to $a_l, \ldots, a_r$" can be interchanged with two prefix queries "add x to $a_1, \ldots, a_r$" and "add $-x$ to $a_1, \ldots, a_{l-1}$". We only consider prefix additions from now on.

Let's see how adding x to first k elements changes sums on every prefix. If we denote $s_i = a_1 + \ldots + a_i$, we have that s_i increases by $x \cdot i$ when $i \leqslant k$, and by $x \cdot k$ when $i > k$.

L. Laura's Function

Let's start with the one-dimensional case. For the purpose of our discussion, BIT is an abstract RSQ data structure that allows changing a single element of an array and computing range sums (clearly, prefix sums always suffice).

A range query "add x to $a_l, \ldots, a_r$" can be interchanged with two prefix queries "add x to $a_1, \ldots, a_r$" and "add $-x$ to $a_1, \ldots, a_{l-1}$". We only consider prefix additions from now on.

Let's see how adding x to first k elements changes sums on every prefix. If we denote $s_i = a_1 + \ldots + a_i$, we have that s_i increases by $x \cdot i$ when $i \leqslant k$, and by $x \cdot k$ when $i > k$.

To allow for quick changes, we introduce two auxiliary arrays p_i and q_i. At any point we want to maintain the condition

$$s_i = i \cdot (p_1 + \ldots + p_i) + (q_1 + \ldots + q_i).$$

L. Laura's Function

One can check that after performing operations:

- p_1 += x, p_{k+1} −= x,
- q_{k+1} += $x \cdot k$,

all values of s_i computed from p_i and q_i are changed exactly as we want them to.

L. Laura's Function

One can check that after performing operations:

- p_1 += x, p_{k+1} −= x,
- q_{k+1} += $x \cdot k$,

all values of s_i computed from p_i and q_i are changed exactly as we want them to.

Since we have reduced to $O(1)$ single-element updates and/or prefix sums queries for p_i and q_i per each query of the original problem, storing p_i and q_i in BITs clearly does the trick.

L. Laura's Function

The same idea can be carried over to 2D (and, in fact, any dimension) range queries. Let $s_{i,j}$ be the two-dimensional "prefix sums":

$$s_{i,j} = \sum_{x=1}^{i} \sum_{y=1}^{j} a_{x,y}.$$

L. Laura's Function

The same idea can be carried over to 2D (and, in fact, any dimension) range queries. Let $s_{i,j}$ be the two-dimensional "prefix sums":

$$s_{i,j} = \sum_{x=1}^{i} \sum_{y=1}^{j} a_{x,y}.$$

The key to efficient updates is the representation

$$s_{i,j} = \sum_{x=1}^{i} \sum_{y=1}^{j} p_{x,y}^{00} + i \sum_{x=1}^{i} \sum_{y=1}^{j} p_{x,y}^{10} + j \sum_{x=1}^{i} \sum_{y=1}^{j} p_{x,y}^{01} + ij \sum_{x=1}^{i} \sum_{y=1}^{j} p_{x,y}^{11}.$$

L. Laura's Function

The same idea can be carried over to 2D (and, in fact, any dimension) range queries. Let $s_{i,j}$ be the two-dimensional "prefix sums":

$$s_{i,j} = \sum_{x=1}^{i} \sum_{y=1}^{j} a_{x,y}.$$

The key to efficient updates is the representation

$$s_{i,j} = \sum_{x=1}^{i} \sum_{y=1}^{j} p_{x,y}^{00} + i \sum_{x=1}^{i} \sum_{y=1}^{j} p_{x,y}^{10} + j \sum_{x=1}^{i} \sum_{y=1}^{j} p_{x,y}^{01} + ij \sum_{x=1}^{i} \sum_{y=1}^{j} p_{x,y}^{11}.$$

Now we can reduce to $O(1)$ single-element updates and rectangle sum queries in a similar fashion to the previous discussion.

L. Laura's Function

The same idea can be carried over to 2D (and, in fact, any dimension) range queries. Let $s_{i,j}$ be the two-dimensional "prefix sums":

$$s_{i,j} = \sum_{x=1}^{i} \sum_{y=1}^{j} a_{x,y}.$$

The key to efficient updates is the representation

$$s_{i,j} = \sum_{x=1}^{i} \sum_{y=1}^{j} p^{00}_{x,y} + i \sum_{x=1}^{i} \sum_{y=1}^{j} p^{10}_{x,y} + j \sum_{x=1}^{i} \sum_{y=1}^{j} p^{01}_{x,y} + ij \sum_{x=1}^{i} \sum_{y=1}^{j} p^{11}_{x,y}.$$

Now we can reduce to $O(1)$ single-element updates and rectangle sum queries in a similar fashion to the previous discussion.

2D BIT allows to perform each single-element update and sum query in $O(\log n \log m)$ time, which finishes the story.