CEPC 2010
problemset discussion

CERC 2010
problemset discussion

...but first a few statistics for your amusement!

The total size of all submits was 2004450. Only 457380 of this code was bug-free.

First correct submission was at 09:40:00 (pretty quick, eh?). And the last correct submission was at 14:25:04.

The longest series of unsuccessful submits with a happy end (i.e., getting AC) was 11.

We got submissions in c, cpp, Java, and Pascal. The percentage of accepted submissions in each language was:

c	0.145455
cpp	0.274306
Java	0.0652174
Pascal	0.25

...but first a few statistics for your amusement!

The total size of all submits was 2004450. Only 457380 of this code was bug-free.

First correct submission was at 09:40:00 (pretty quick, eh?). And the last correct submission was at 14:25:04.

The longest series of unsuccessful submits with a happy end (i.e., getting AC) was 11.

We got submissions in c, cpp, Java, and Pascal. The percentage of accepted submissions in each language was:

c	0.145455
cpp	0.274306
Java	0.0652174
Pascal	0.25

...but first a few statistics for your amusement!

The total size of all submits was 2004450. Only 457380 of this code was bug-free.

First correct submission was at 09:40:00 (pretty quick, eh?). And the last correct submission was at 14:25:04.

The longest series of unsuccessful submits with a happy end (i.e., getting AC) was 11.

We got submissions in c, cpp, Java, and Pascal. The percentage of accepted submissions in each language was:

c	0.145455
cpp	0.274306
Java	0.0652174
Pascal	0.25

...but first a few statistics for your amusement!

The total size of all submits was 2004450. Only 457380 of this code was bug-free.

First correct submission was at 09:40:00 (pretty quick, eh?). And the last correct submission was at 14:25:04.

The longest series of unsuccessful submits with a happy end (i.e., getting AC) was 11.

We got submissions in c, cpp, Java, and Pascal. The percentage of accepted submissions in each language was:

c	0.145455
cpp	0.274306
Java	0.0652174
Pascal	0.25

...but first a few statistics for your amusement!

The total size of all submits was 2004450. Only 457380 of this code was bug-free.

First correct submission was at 09:40:00 (pretty quick, eh?). And the last correct submission was at 14:25:04.

The longest series of unsuccessful submits with a happy end (i.e., getting AC) was 11.

We got submissions in c, cpp, Java, and Pascal. The percentage of accepted submissions in each language was:

c	0.145455
cpp	0.274306
Java	0.0652174
Pascal	0.25

G - Game

RTE	WA	TLE	AC
1	86	18	81

..nothing to see here, really!

I - Insults

RTE	WA	TLE	AC
13	45	14	30

You are given a well-bracketed word $w[1..n]$ and asked to find the lexicographically next one (or detect there is none).
Check the prefixes of w one-by-one, starting with the longest one. Try to extend $w[1..k-1]$ by one letter (lexicographically) bigger than $w[k]$. When is it possible?

- If $w[k]$ is a closing bracket, there must be a corresponding open bracket.
- It must be possible to close all f open brackets using the $n-k$ free places. If there are more than $n-k$ open brackets, it is clearly impossible. Otherwise it is possible, and the lexicographically best way of doing that is to first append $a^{\frac{n-k-f}{2}} e^{\frac{n-k-f}{2}}$ and then close all open brackets.

I - Insults

RTE	WA	TLE	AC
13	45	14	30

You are given a well-bracketed word $w[1..n]$ and asked to find the lexicographically next one (or detect there is none).
Check the prefixes of w one-by-one, starting with the longest one. Try to extend $w[1..k-1]$ by one letter (lexicographically) bigger than $w[k]$.
When is it possible?

- If $w[k]$ is a closing bracket, there must be a corresponding open bracket.
- It must be possible to close all f open brackets using the $n-k$ free places. If there are more than $n-k$ open brackets, it is clearly impossible. Otherwise it is possible, and the lexicographically best way of doing that is to first append $a^{\frac{n-k-f}{2}}e^{\frac{n-k-f}{2}}$ and then close all open brackets.

E - Enter The Dragon

RTE	WA	TLE	AC
17	88	51	53

We are given a forecast $t[1], t[2], \ldots, t[n]$. Each $t[i]$ is

- 0, meaning that the day is non-rainy, and we can use it to empty one lake.
- nonzero, meaning that the corresponding lake should be empty.

How to decide which lake should be made empty on each non-rainy day? Go through the sequence from left to right, for each nonzero $t[i]$ choose $j < i$ such that $t[j] = 0$ and:

- j is as small as possible? WRONG
- j is as small as possible, but greater than the previous occurrence of $t[i]$.

You can use set for selecting j for $\mathcal{O}(n \log n)$ running time. Or you can use union-find to get $\mathcal{O}(n \log^* n)$. Or...

E - Enter The Dragon

RTE	WA	TLE	AC
17	88	51	53

We are given a forecast $t[1], t[2], \ldots, t[n]$. Each $t[i]$ is

- 0, meaning that the day is non-rainy, and we can use it to empty one lake.
- nonzero, meaning that the corresponding lake should be empty.

How to decide which lake should be made empty on each non-rainy day? Go through the sequence from left to right, for each nonzero $t[i]$ choose $j < i$ such that $t[j] = 0$ and:

- j is as small as possible? WRONG
- j is as small as possible, but greater than the previous occurrence of $t[i]$.

You can use `set` for selecting j for $\mathcal{O}(n \log n)$ running time. Or you can use union-find to get $\mathcal{O}(n \log^* n)$. Or...

D - Defense Lines

RTE	WA	TLE	AC
29	65	77	45

We have a sequence of n numbers $t[1], t[2], \ldots, t[n]$. We can remove any of its consecutive fragments. Then we compute the length of the longest consecutive increasing subsequence. What is the maximum possible length we can get?

First for any i compute two values:

- $starting[i]$, the longest consecutive increasing subsequence starting at the i-th position,
- $ending[i]$, the longest consecutive increasing subsequence ending at the i-th position.

Then the problem reduces to maximizing $ending[i] + starting[j]$ for $i < j$ such that $t[i] < t[j]$. How to do that?

D - Defense Lines

RTE	WA	TLE	AC
29	65	77	45

We have a sequence of n numbers $t[1], t[2], \ldots, t[n]$. We can remove any of its consecutive fragments. Then we compute the length of the longest consecutive increasing subsequence. What is the maximum possible length we can get?

First for any i compute two values:

- $starting[i]$, the longest consecutive increasing subsequence starting at the i-th position,
- $ending[i]$, the longest consecutive increasing subsequence ending at the i-th position.

Then the problem reduces to maximizing $ending[i] + starting[j]$ for $i < j$ such that $t[i] < t[j]$. How to do that?

Go through the sequence from left to right. For each j try to find the best i, i.e., maximize $ending[i]$ among all already processed i such that $t[i] < t[j]$. Plenty of ways to do that.

- Use a static (or balanced) binary search tree.
- Define $best[k]$ equal to the smallest possible value of $t[i]$ such that $ending[i] = k$. Then observe that:
  - all $best[k]$ are nondecreasing.
  - adding new $t[i]$ requires updating at most one $best[k]$.
  - having the values of $best[k]$ allows computing the maximum possible value of $ending[i]$ with just one binary search.

Go through the sequence from left to right. For each j try to find the best i, i.e., maximize $ending[i]$ among all already processed i such that $t[i] < t[j]$. Plenty of ways to do that.

- Use a static (or balanced) binary search tree.
- Define $best[k]$ equal to the smallest possible value of $t[i]$ such that $ending[i] = k$. Then observe that:
    - all $best[k]$ are nondecreasing.
    - adding new $t[i]$ requires updating at most one $best[k]$.
    - having the values of $best[k]$ allows computing the maximum possible value of $ending[i]$ with just one binary search.

C - Casting Spells

RTE	WA	TLE	AC
19	63	12	23

We are asked to find the longest subword of the form ww^Rww^R, or, in other words, the longest even palindrome composed of two smaller even palindromes.

First we would like to construct a succint description of all palindromic subwords of the input word s. More specifically, for each i we would like to compute the largest k such that $s[i] = s[i-1]$, $s[i+1] = s[i-2]$, ..., $s[i+k] = s[i-k-1]$.

This $k = r(i)$ is called the palindromic radius at i. There are quite a few ways to compute it.

1. use the Manacher's algorithm which computes all $r(i)$ in $\mathcal{O}(n)$ time.
2. use a separate binary search at each position i. For this to work we need an efficient way of checking if two fragments of s are equal, this can be done using hashing. Total time is $\mathcal{O}(n \log n)$.

C - Casting Spells

RTE	WA	TLE	AC
19	63	12	23

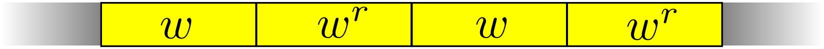
We are asked to find the longest subword of the form ww^Rww^R, or, in other words, the longest even palindrome composed of two smaller even palindromes.

First we would like to construct a succint description of all palindromic subwords of the input word s. More specifically, for each i we would like to compute the largest k such that $s[i] = s[i-1]$, $s[i+1] = s[i-2]$, ..., $s[i+k] = s[i-k-1]$.

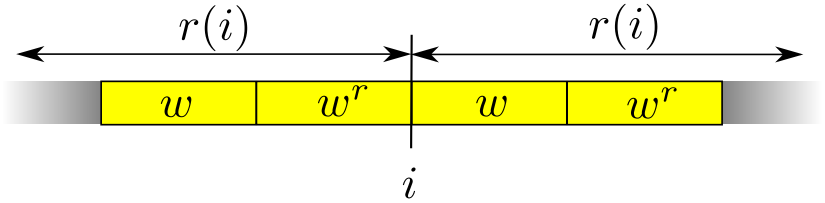
This $k = r(i)$ is called the palindromic radius at i. There are quite a few ways to compute it.

1. use the Manacher's algorithm which computes all $r(i)$ in $\mathcal{O}(n)$ time.
2. use a separate binary search at each position i. For this to work we need an efficient way of checking if two fragments of s are equal, this can be done using hashing. Total time is $\mathcal{O}(n \log n)$.

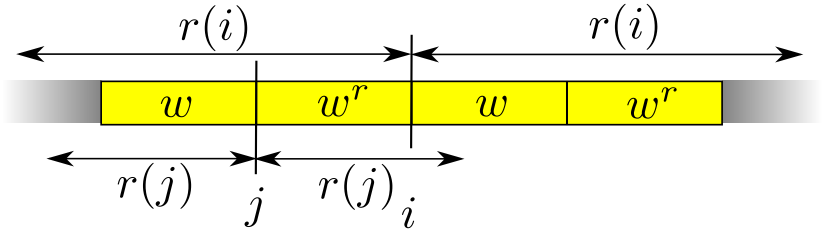
Now we are left with the fun part of the problem.



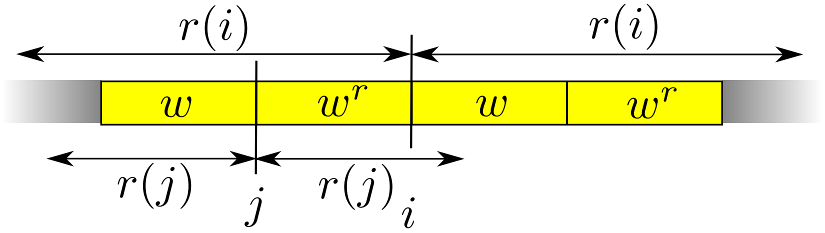
Now we are left with the fun part of the problem.



Now we are left with the fun part of the problem.



Now we are left with the fun part of the problem.



So for a given center i we are looking for the leftmost j such that:

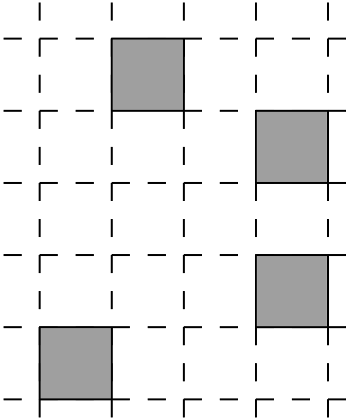
- $j + r(j) \geq i$,
- $j \geq i - \frac{r(i)}{2}$.

Finding best possible values of j for all i can be done in a single left to right sweep. We store the possible candidates in a structure allowing inserting, erasing, and finding successor. Can be done (again) using `set`.

F - Fields and Farmers

RTE	WA	TLE	AC
1	39	2	11

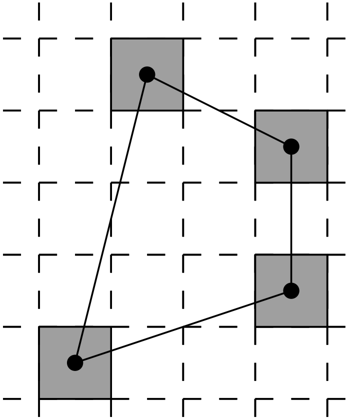
The problem statement describes a process of adding fields to already existing ones. We will call the final result the *strange hull* of the given initial set of fields.



F - Fields and Farmers

RTE	WA	TLE	AC
1	39	2	11

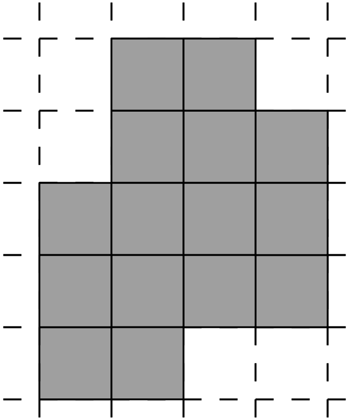
The problem statement describes a process of adding fields to already existing ones. We will call the final result the *strange hull* of the given initial set of fields.



F - Fields and Farmers

RTE	WA	TLE	AC
1	39	2	11

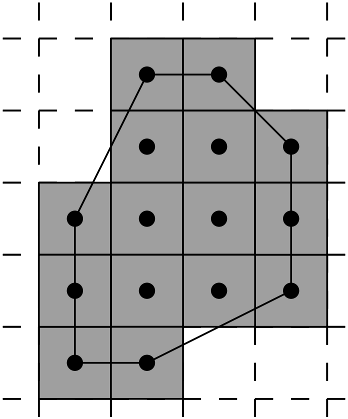
The problem statement describes a process of adding fields to already existing ones. We will call the final result the *strange hull* of the given initial set of fields.



F - Fields and Farmers

RTE	WA	TLE	AC
1	39	2	11

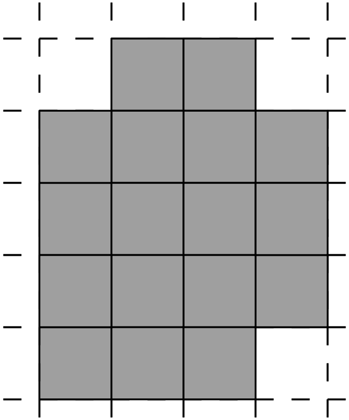
The problem statement describes a process of adding fields to already existing ones. We will call the final result the *strange hull* of the given initial set of fields.



F - Fields and Farmers

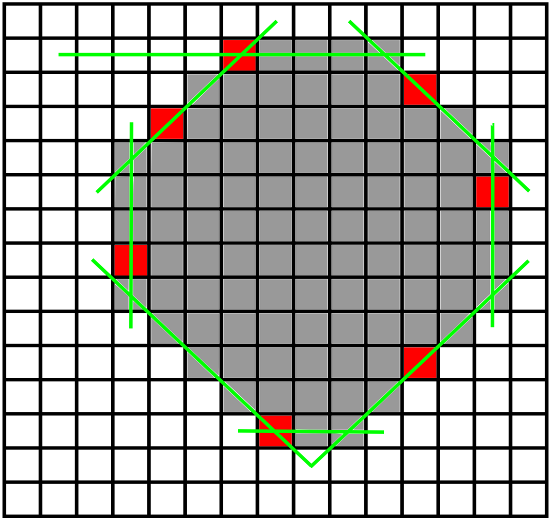
RTE	WA	TLE	AC
1	39	2	11

The problem statement describes a process of adding fields to already existing ones. We will call the final result the *strange hull* of the given initial set of fields.

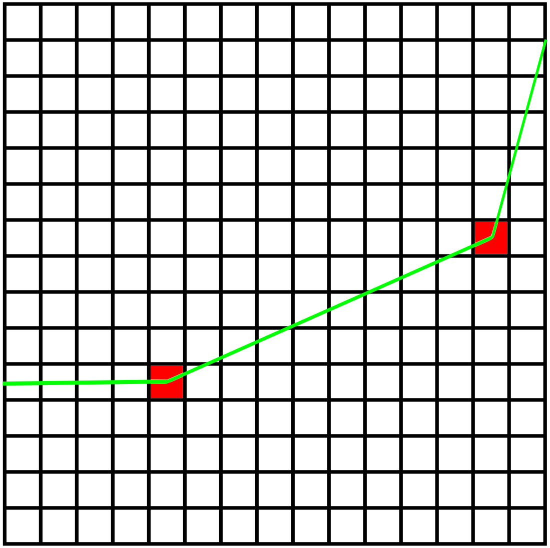


The problem asks you to count the subsets of the original set of fields with the same strange hull. First lets try to find its efficient characterization. First note that if there are no initial fields on the left of some line $x = a$, there are no such fields in the hull as well. The same applies to lines $y = a$, $y = x + a$, $y = -x + a$ but is not necessarily true for other lines! We claim that in fact this characterizes all fields in the strange hull. In other words, it contains all fields (x, y) adding which does not change the minimum/maximum values of $x, y, x - y, x + y$.

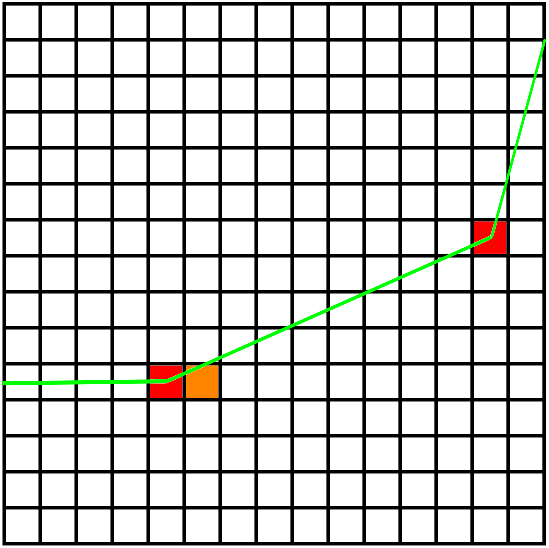
The problem asks you to count the subsets of the original set of fields with the same strange hull. First lets try to find its efficient characterization. First note that if there are no initial fields on the left of some line $x = a$, there are no such fields in the hull as well. The same applies to lines $y = a$, $y = x + a$, $y = -x + a$ but is not necessarily true for other lines! We claim that in fact this characterizes all fields in the strange hull. In other words, it contains all fields (x, y) adding which does not change the minimum/maximum values of $x, y, x - y, x + y$.



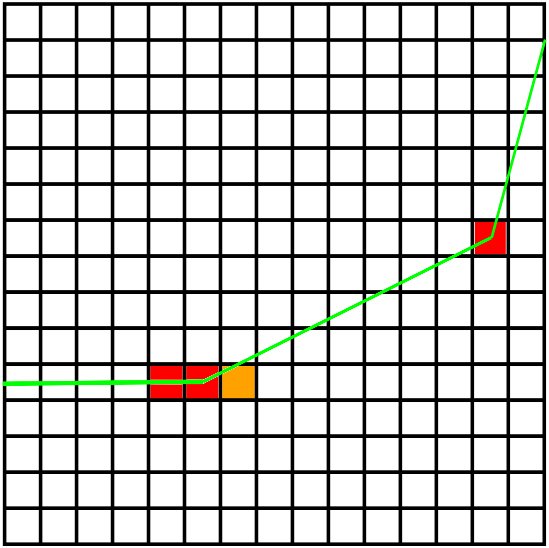
How to prove such claim? Consider the situation on two neighbouring segments of the boundary.



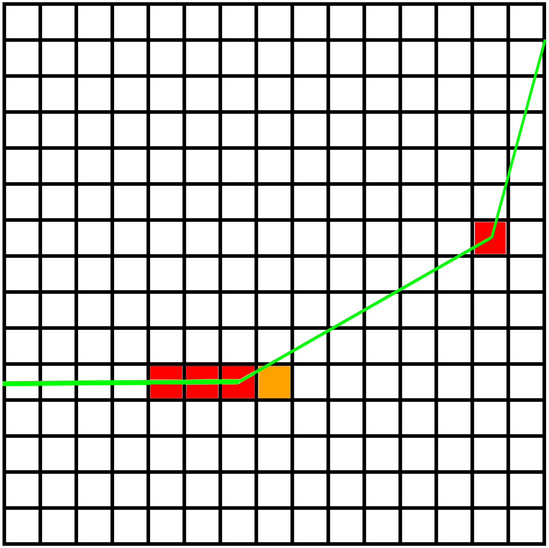
How to prove such claim? Consider the situation on two neighbouring segments of the boundary.



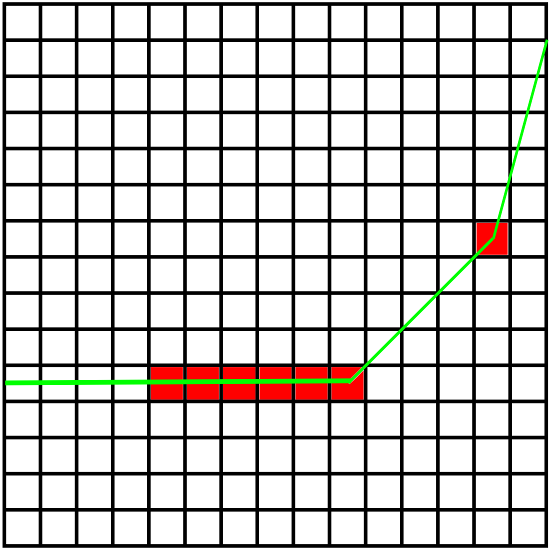
How to prove such claim? Consider the situation on two neighbouring segments of the boundary.



How to prove such claim? Consider the situation on two neighbouring segments of the boundary.



How to prove such claim? Consider the situation on two neighbouring segments of the boundary.



We still haven't solved the original problem, but now we know that it reduces to something like: given a collection of pairs (x_i, y_i) count its subsets with the same values of minimum/maximum x, y, $x+y$, $x-y$. In other words, we must take at least one pair with the minimum/maximum value of x, y, $x+y$, $x-y$. But... there can be so many choices here...

For each pair construct an small set $B_i \subseteq \{1, 2, \ldots, 8\}$ describing which minimum/maximum values it achieves. Then we should count ways to choose a subset which sums up to the whole $\{1, 2, 3, \ldots, 8\}$.

This can be done using the inclusion-exclusion principle: we can easily count the ways to choose a subset which sums up to something contained in some specified X, lets call this quantity $f(X)$. Then reconstruct the answer for all such partial data:

$$\sum_{X \subseteq \{1,2,3,\ldots,8\}} (-1)^{|X|} f(X)$$

We still haven't solved the original problem, but now we know that it reduces to something like: given a collection of pairs (x_i, y_i) count its subsets with the same values of minimum/maximum x, y, $x + y$, $x - y$. In other words, we must take at least one pair with the minimum/maximum value of x, y, $x + y$, $x - y$. But... there can be so many choices here...
For each pair construct an small set $B_i \subseteq \{1, 2, \ldots, 8\}$ describing which minimum/maximum values it achieves. Then we should count ways to choose a subset which sums up to the whole $\{1, 2, 3, \ldots, 8\}$.
This can be done using the inclusion-exclusion principle: we can easily count the ways to choose a subset which sums up to something contained in some specified X, lets call this quantity $f(X)$. Then reconstruct the answer for all such partial data:

$$\sum_{X \subseteq \{1,2,3,\ldots,8\}} (-1)^{|X|} f(X)$$

We still haven't solved the original problem, but now we know that it reduces to something like: given a collection of pairs (x_i, y_i) count its subsets with the same values of minimum/maximum x, y, $x+y$, $x-y$. In other words, we must take at least one pair with the minimum/maximum value of x, y, $x+y$, $x-y$. But... there can be so many choices here...
For each pair construct an small set $B_i \subseteq \{1, 2, \ldots, 8\}$ describing which minimum/maximum values it achieves. Then we should count ways to choose a subset which sums up to the whole $\{1, 2, 3, \ldots, 8\}$.
This can be done using the inclusion-exclusion principle: we can easily count the ways to choose a subset which sums up to something contained in some specified X, lets call this quantity $f(X)$. Then reconstruct the answer for all such partial data:

$$\sum_{X \subseteq \{1,2,3,\ldots,8\}} (-1)^{|X|} f(X)$$

J - Justice for all

RTE	WA	TLE	AC
0	6	0	3

You are asked to construct a (not too big) simple bipartite graph with exactly C perfect matchings, with $C \leq 10^6$.

Easy to see that for any C there exists a graph with exactly C perfect matchings. But the problem is to keep is small...

First idea: if you can construct a small G_1 with exactly C_1 matchings, and a small G_2 with exactly C_2 matchings, you can simply take their disjoint union, which has $C_1 C_2$ matchings.

Maybe you could somehow combine them to get something with $C_1 + C_2$ matchings?

Maybe...

Try to use a stronger assumption on the structure of G_1 and G_2 (a little bit like making the induction hypothesis stronger to make the proof easier).

J - Justice for all

RTE	WA	TLE	AC
0	6	0	3

You are asked to construct a (not too big) simple bipartite graph with exactly C perfect matchings, with $C \leq 10^6$.
Easy to see that for any C there exists a graph with exactly C perfect matchings. But the problem is to keep is small...
First idea: if you can construct a small G_1 with exactly C_1 matchings, and a small G_2 with exactly C_2 matchings, you can simply take their disjoint union, which has $C_1 C_2$ matchings.
Maybe you could somehow combine them to get something with $C_1 + C_2$ matchings?
Maybe...

Try to use a stronger assumption on the structure of G_1 and G_2 (a little bit like making the induction hypothesis stronger to make the proof easier).

J - Justice for all

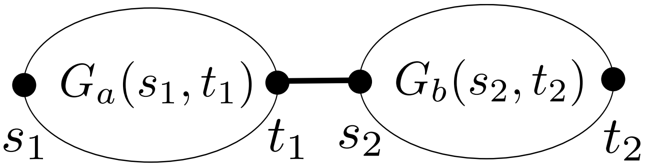
RTE	WA	TLE	AC
0	6	0	3

You are asked to construct a (not too big) simple bipartite graph with exactly C perfect matchings, with $C \leq 10^6$.
Easy to see that for any C there exists a graph with exactly C perfect matchings. But the problem is to keep is small...
First idea: if you can construct a small G_1 with exactly C_1 matchings, and a small G_2 with exactly C_2 matchings, you can simply take their disjoint union, which has $C_1 C_2$ matchings.
Maybe you could somehow combine them to get something with $C_1 + C_2$ matchings?
Maybe...

Try to use a stronger assumption on the structure of G_1 and G_2 (a little bit like making the induction hypothesis stronger to make the proof easier).

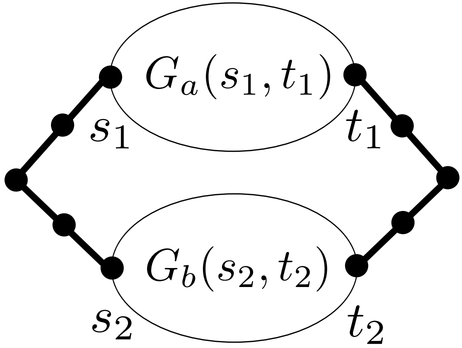
Construct a sequence of graphs $G_C(s,t)$ with two distinguished vertices s (source) and t (sink), with the property that after adding $\{s,t\}$ to the set of edges, there are exactly $C+1$ perfect matchings. More specifically, there are C matchings on $G_C(s,t)$, and exactly 1 on $G_C(s,t)$ with both s and t removed.
$G_1(s,t)$ is just a single edge.

Construct a sequence of graphs $G_C(s,t)$ with two distinguished vertices s (source) and t (sink), with the property that after adding $\{s,t\}$ to the set of edges, there are exactly $C+1$ perfect matchings. More specifically, there are C matchings on $G_C(s,t)$, and exactly 1 on $G_C(s,t)$ with both s and t removed.
$G_1(s,t)$ is just a single edge.



Construct a sequence of graphs $G_C(s,t)$ with two distinguished vertices s (source) and t (sink), with the property that after adding $\{s,t\}$ to the set of edges, there are exactly $C+1$ perfect matchings. More specifically, there are C matchings on $G_C(s,t)$, and exactly 1 on $G_C(s,t)$ with both s and t removed.

$G_1(s,t)$ is just a single edge.



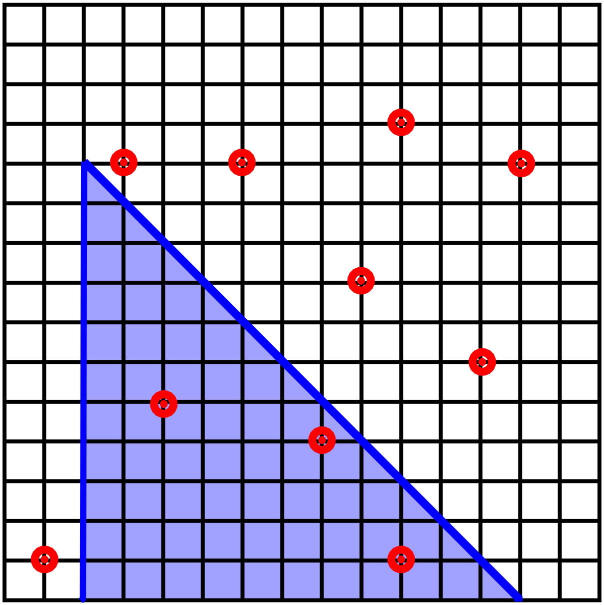
H - Hanging Hats

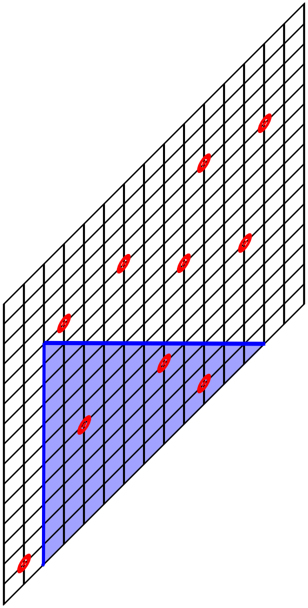
RTE	WA	TLE	AC
2	14	4	3

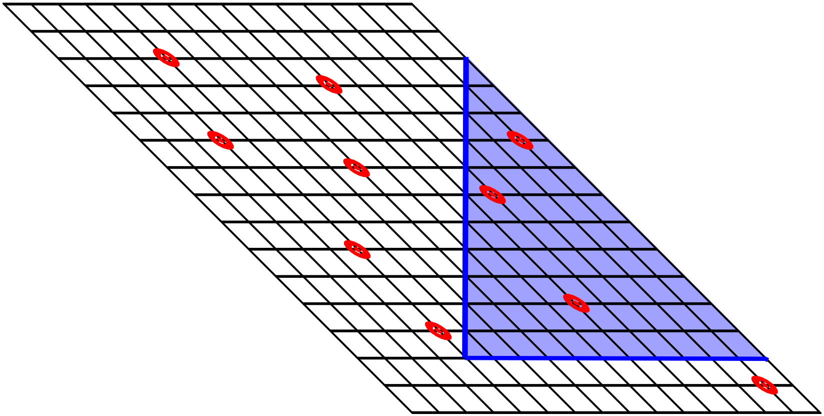
The problem reduces to storing a dynamic set of 2D points in a structure allowing efficient reporting of all points inside regions of the form:

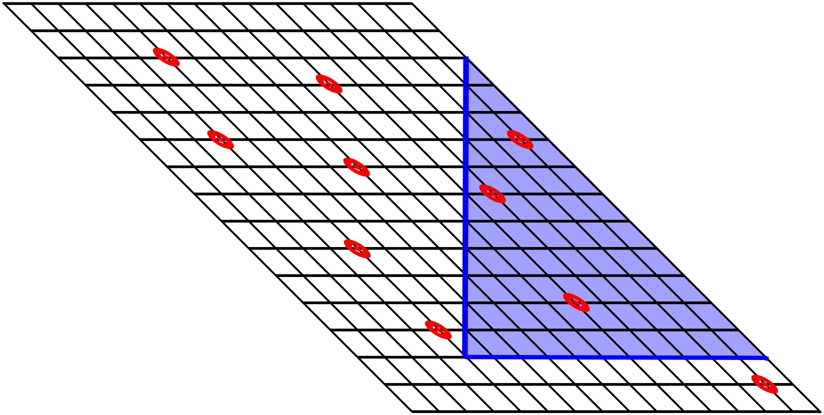
1. $y \leq \min(x + y_0 - x_0, -x + x_0 + y_0)$ for some (x_0, y_0),
2. $y \leq \min(2x + y_0 - 2x_0, -2x + 2x_0 + y_0)$ for some (x_0, y_0).

For starters lets assume that we care only about the first type regions. Each such region can be split into two parts along the $x = x_0$ line so we can focus on reporting the points inside a region bounded by $x = x_0$ and $y = x + y_0 - x_0$.









So this is actually the *dominance reporting* problem: store a dynamic collection of points so that all (x, y) with $x \geq x_0$, $y \geq y_0$ can be reported quickly. Any type of regions can be reduced to this problem by an affine tranformation.

There are quite a few classic solution to this problem.

1. range search trees, reporting in $\mathcal{O}(\log^2 n + k)$
2. priority search trees, reporting in $\mathcal{O}(\log n + k)$

(k is the number of reported points)
The structures are usually described as static, but if we know all queries in advance, it is not difficult to modify them so that points can be inserted/erased.
But does it really help? In our original problem we had two types of regions, and we split each type into two subtypes...
Build a separate structure for each type! And remember to ignore all but the first time the point is reported.
The structure is quite specific here, and the range search tree can be speed up to $\mathcal{O}(\log n + k)$. The total complexity is $\mathcal{O}(n \log n)$ then.

There are quite a few classic solution to this problem.

1. range search trees, reporting in $\mathcal{O}(\log^2 n + k)$
2. priority search trees, reporting in $\mathcal{O}(\log n + k)$

(k is the number of reported points)
The structures are usually described as static, but if we know all queries in advance, it is not difficult to modify them so that points can be inserted/erased.
But does it really help? In our original problem we had two types of regions, and we split each type into two subtypes...
Build a separate structure for each type! And remember to ignore all but the first time the point is reported.
The structure is quite specific here, and the range search tree can be speed up to $\mathcal{O}(\log n + k)$. The total complexity is $\mathcal{O}(n \log n)$ then.

B - Beasts

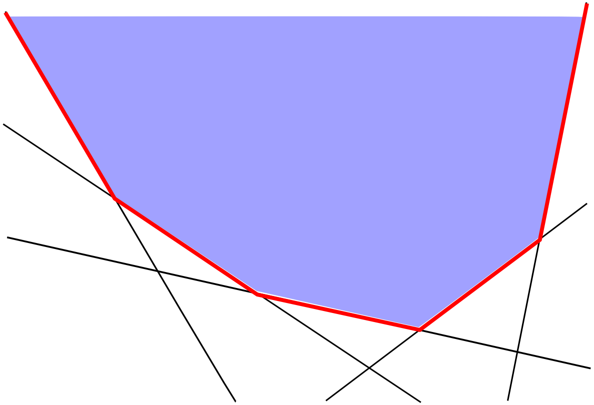
RTE	WA	TLE	AC
0	8	1	0

First we should somehow determine the boundary of the lower and the upper region. The situation is completely symmetric here so lets focus on the former.

B - Beasts

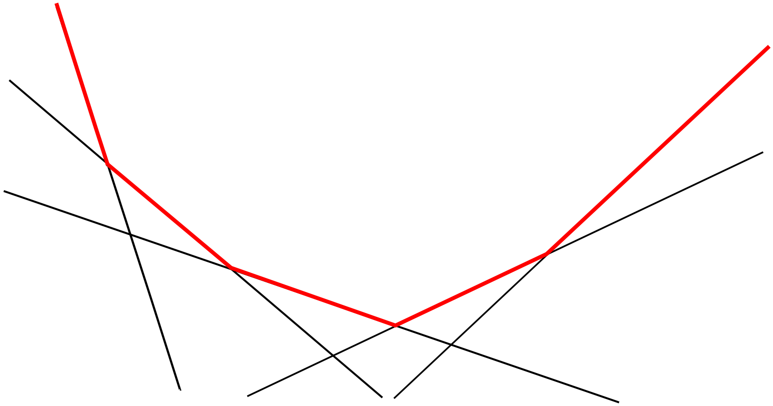
RTE	WA	TLE	AC
0	8	1	0

First we should somehow determine the boundary of the lower and the upper region. The situation is completely symmetric here so lets focus on the former.

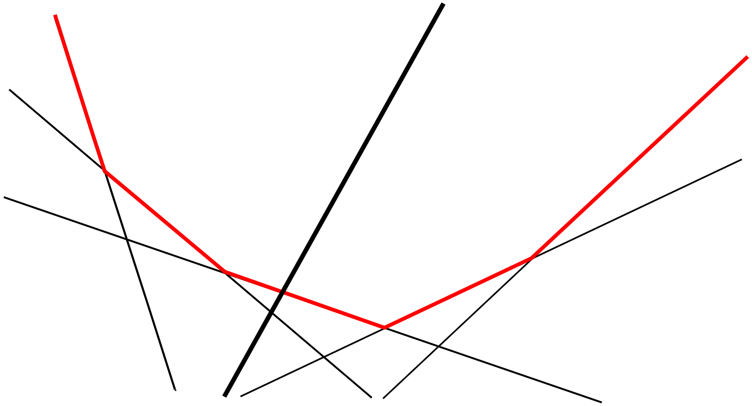


The boundary read from left to right consists of segments belonging to lines $y = ax + b$ with increasing values of a. This suggest that we should look at how the boundary changes after adding a new line with bigger value of a.

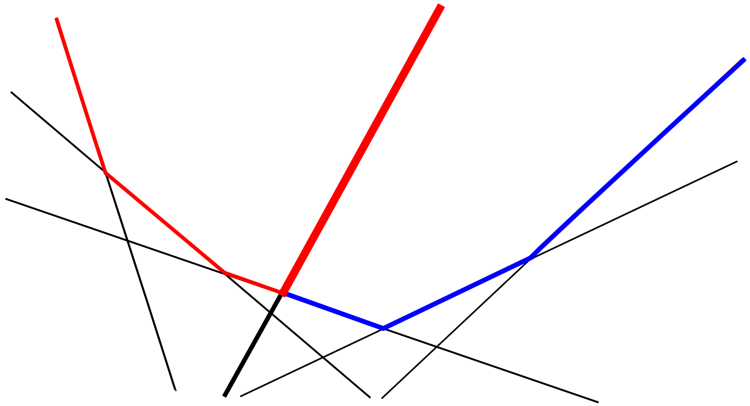
The boundary read from left to right consists of segments belonging to lines $y = ax + b$ with increasing values of a. This suggest that we should look at how the boundary changes after adding a new line with bigger value of a.



The boundary read from left to right consists of segments belonging to lines $y = ax + b$ with increasing values of a. This suggest that we should look at how the boundary changes after adding a new line with bigger value of a.



The boundary read from left to right consists of segments belonging to lines $y = ax + b$ with increasing values of a. This suggest that we should look at how the boundary changes after adding a new line with bigger value of a.



Consider the lines $y = ax + b$ in order of increasing a. Store the current boundary on a stack.

1. while the last segment on the stack is completely on the right of the current $y = ax + b$, pop it,
2. split the last segment on the stack,
3. push a new segment on the stack.

Total time is $\mathcal{O}(n \log n)$, and we got the boundary as a sequence of segments.
(the whole procedure is very similar to computing the convex hull, do you know why?)

Having the description of both boundaries, we still need to compute their distance.

If you have a generic two convex polygons distance procedure, you are good to go already. But for the rest of us mere mortals...

The optimal distance is achieved for two points p, q such that at least one of them is the endpoints of some segment. We can afford to check all endpoints one-by-one.

For each endpoint we have to compute its distance to a convex polygon (quite special convex polygon, actually).

1. This can be using binary search.
2. Or, even better, you can observe that when you move with this endpoint from left to right, the corresponding best point on the second boundary moves from left to right as well (because of the way the bounadries are constructed).

This gives a linear time solution for the second part of the problem.

Having the description of both boundaries, we still need to compute their distance.

If you have a generic two convex polygons distance procedure, you are good to go already. But for the rest of us mere mortals...

The optimal distance is achieved for two points p, q such that at least one of them is the endpoints of some segment. We can afford to check all endpoints one-by-one.

For each endpoint we have to compute its distance to a convex polygon (quite special convex polygon, actually).

1. This can be using binary search.
2. Or, even better, you can observe that when you move with this endpoint from left to right, the corresponding best point on the second boundary moves from left to right as well (because of the way the bounadries are constructed).

This gives a linear time solution for the second part of the problem.

Having the description of both boundaries, we still need to compute their distance.
If you have a generic two convex polygons distance procedure, you are good to go already. But for the rest of us mere mortals...
The optimal distance is achieved for two points p, q such that at least one of them is the endpoints of some segment. We can afford to check all endpoints one-by-one.
For each endpoint we have to compute its distance to a convex polygon (quite special convex polygon, actually).

1. This can be using binary search.
2. Or, even better, you can observe that when you move with this endpoint from left to right, the corresponding best point on the second boundary moves from left to right as well (because of the way the bounadries are constructed).

This gives a linear time solution for the second part of the problem.

A - Ardenia

RTE	WA	TLE	AC
3	33	9	0

Calculate the (squared) distance between two segments in 3D. Your answer should be given as a irreducible fraction $\frac{p}{q}$.

First idea: two nested binary searches. Would work if we need the answer up to a few decimal places, which is not the case here.

Second idea: be more analytical.

Minimize $f(a, b) = Aa^2 + Bb^2 + Cab + Da + Eb + F$ over $0 \leq a, b \leq 1$.

Two cases to consider:

- the minimum is achieved for a boundary value of a or b, i.e., 0 or 1. Check all such possibilities, for each of them we get a univariate quadratic equation, which is simple to minimize.
- $a, b \in (0, 1)$. Then the minimum is achieved when $\frac{\partial f(a,b)}{\partial a} = 0$ and $\frac{\partial f(a,b)}{\partial b} = 0$. Solving this gives 2 linear equations in two variables.

A - Ardenia

RTE	WA	TLE	AC
3	33	9	0

Calculate the (squared) distance between two segments in 3D. Your answer should be given as a irreducible fraction $\frac{p}{q}$.
First idea: two nested binary searches. Would work if we need the answer up to a few decimal places, which is not the case here.
Second idea: be more analytical.
Minimize $f(a, b) = Aa^2 + Bb^2 + Cab + Da + Eb + F$ over $0 \leq a, b \leq 1$.
Two cases to consider:

- the minimum is achieved for a boundary value of a or b, i.e., 0 or 1. Check all such possibilities, for each of them we get a univariate quadratic equation, which is simple to minimize.
- $a, b \in (0, 1)$. Then the minimum is achieved when $\frac{\partial f(a,b)}{\partial a} = 0$ and $\frac{\partial f(a,b)}{\partial b} = 0$. Solving this gives 2 linear equations in two variables.

THE END