Long Contest Editorial

November 12, 2015

Moscow International Workshop ACM ICPC, MIPT, 2015

A. Too Rich

The problem regarded representing P dollars given a certain amount of 1, 2, 5, 10, 20, 50, 100, 200, 500, 1000 and 2000-dollar coins (or banknotes), while maximizing total number of coins used.

A. Too Rich

The problem regarded representing P dollars given a certain amount of 1, 2, 5, 10, 20, 50, 100, 200, 500, 1000 and 2000-dollar coins (or banknotes), while maximizing total number of coins used.

Observation

Denote T total amount of dollars we have. Obtaining P dollars using the most number of coins is the same as taking $T - P$ using the least number of coins and leaving them out. In the following we discuss the problem of representing S using minimal amount of coins.

A. Too Rich

Example (A simpler case)

Consider a set of denominations $d_1 < \ldots < d_k$ such that every denomination divides the previous one: $d_{i+1} \vdots d_i$ for all $i \in [1; k-1]$. Can we come up with an easy solution for the same problem?

A. Too Rich

Example (A simpler case)

Consider a set of denominations $d_1 < \ldots < d_k$ such that every denomination divides the previous one: $d_{i+1} \vdots d_i$ for all $i \in [1; k-1]$. Can we come up with an easy solution for the same problem?

Greedy algorithm for the simpler case

In this case a greedy algorithm works: take maximal amount of d_k-dollar coins such that the sum does not exceed S, then take maximal amount of d_{k-1}-dollar coins, and so on. If the total amount of money taken this way is S, then the representation is minimal, otherwise no representation is possible.

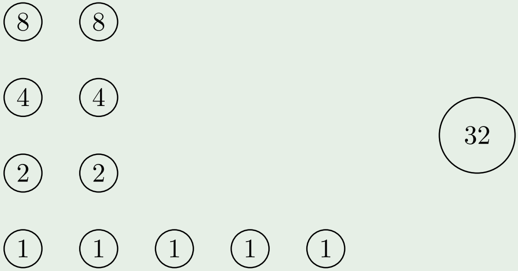
A. Too Rich

Proof for the greedy algorithm

Suppose that $c_1 d_1 + \ldots + c_j d_j \geqslant d_{j+1}$ for some integer non-negative c_j. Then we can choose integer non-negative c'_j such that $c'_j \leqslant c_j$ and $c'_1 d_1 + \ldots + c'_j d_j = d_{j+1}$. This can be done by induction: take maximal possible amount of d_j-dollar coins, and represent the rest using first $j - 1$ denominations (the rest amount is divisible by d_j).
Now, consider any representation of $P = c_1 d_1 + \ldots + c_k d_k$. If c_k is not maximal possible, choose a subset of smaller coins with sum d_k and replace them with a single coin; repeat until the sum of smaller coins becomes less than d_k. So on for smaller coins.

A. Too Rich

Example (Choose a subset with sum of exactly 32)



A. Too Rich

Example (Choose a subset with sum of exactly 32)

4 4

2 2

1 1 1 1 1

$$32 = 8 + 8 + 8 + 8$$

A. Too Rich

Example (Choose a subset with sum of exactly 32)

2 2

1 1 1 1 1

$$32 = 8 + 8 + 4 + 4 + 4 + 4$$

A. Too Rich

Example (Choose a subset with sum of exactly 32)

1 1 1 1 1

$$32 = 8 + 8 + 4 + 4 + 2 + 2 + 2 + 2$$

A. Too Rich

Example (Choose a subset with sum of exactly 32)

1

$$32 = 8 + 8 + 4 + 4 + 2 + 2 + 1 + 1 + 1 + 1$$

A. Too Rich

In the actual problem the divisibility condition does not hold: 20 does not divide 50, and 200 does not divide 500.

A. Too Rich

In the actual problem the divisibility condition does not hold: 20 does not divide 50, and 200 does not divide 500.

However, the greedy approach can be slightly modified to work here. Suppose that on some step of the greedy algorithm the maximal number of d_j-dollar coins that we can take is X.

A. Too Rich

In the actual problem the divisibility condition does not hold: 20 does not divide 50, and 200 does not divide 500.
However, the greedy approach can be slightly modified to work here. Suppose that on some step of the greedy algorithm the maximal number of d_j-dollar coins that we can take is X. Then, there is a minimal representation such that the number X' of d_j-dollar coins we take is at least $X - 1$, because if X' is at most $X - 2$, we can always replace a subset of smaller coins with a total amount of $2d_j$ with two d_j-dollar coins; the existence of this subset if proved similarily to greedy solution analysis.

A. Too Rich

In the actual problem the divisibility condition does not hold: 20 does not divide 50, and 200 does not divide 500.
However, the greedy approach can be slightly modified to work here. Suppose that on some step of the greedy algorithm the maximal number of d_j-dollar coins that we can take is X. Then, there is a minimal representation such that the number X' of d_j-dollar coins we take is at least $X - 1$, because if X' is at most $X - 2$, we can always replace a subset of smaller coins with a total amount of $2d_j$ with two d_j-dollar coins; the existence of this subset if proved similarily to greedy solution analysis.
Thus, the algorithm that recursively tries X and $X - 1$ for the number of largest coins will always give an optimal answer. Without any optimizations this performs $\sim 2^9$ operations per test, which works fast enough.

B. Count $a \times b$

Let $f(n)$ be the number of pairs $0 \leqslant a, b < n$ such ab is not divisible by n, and $g(n) = \sum_{d|n} f(d)$. Find $g(n)$.

B. Count $a \times b$

Let's start with $f(n)$. $f(n) = n^2 - h(n)$, where $h(n)$ is the number of pairs $0 \leqslant a, b < n$ such that $ab \vdots n$.

B. Count $a \times b$

Let's start with $f(n)$. $f(n) = n^2 - h(n)$, where $h(n)$ is the number of pairs $0 \leqslant a, b < n$ such that $ab \vdots n$.

Factorize n: $n = p_1^{\alpha_1} \dots p_k^{\alpha_k}$. Chinese remainder theorem implies that $h(n)$ is multiplicative: $h(n) = h(p_1^{\alpha_1}) \dots h(p_k^{\alpha_k})$.

B. Count $a \times b$

Let's start with $f(n)$. $f(n) = n^2 - h(n)$, where $h(n)$ is the number of pairs $0 \leqslant a, b < n$ such that $ab \vdots n$.
Factorize n: $n = p_1^{\alpha_1} \dots p_k^{\alpha_k}$. Chinese remainder theorem implies that $h(n)$ is multiplicative: $h(n) = h(p_1^{\alpha_1}) \dots h(p_k^{\alpha_k})$.
Find $h(p^\alpha)$. For $0 \leqslant a < p^\alpha$ let $d(a)$ be the maximal power of p dividing a (set $d(0) = \alpha$ by definition). For any given a and b:

$$ab \vdots p^\alpha \iff d(a) + d(b) \geqslant \alpha.$$

B. Count $a \times b$

Let's start with $f(n)$. $f(n) = n^2 - h(n)$, where $h(n)$ is the number of pairs $0 \leqslant a, b < n$ such that $ab \vdots n$.

Factorize n: $n = p_1^{\alpha_1} \dots p_k^{\alpha_k}$. Chinese remainder theorem implies that $h(n)$ is multiplicative: $h(n) = h(p_1^{\alpha_1}) \dots h(p_k^{\alpha_k})$.

Find $h(p^\alpha)$. For $0 \leqslant a < p^\alpha$ let $d(a)$ be the maximal power of p dividing a (set $d(0) = \alpha$ by definition). For any given a and b:

$$ab \vdots p^\alpha \iff d(a) + d(b) \geqslant \alpha.$$

For any $0 \leqslant k \leqslant \alpha$, the number of a's such that $d(a) \geqslant k$ is exactly $p^{\alpha-k}$. Thus, we obtain the formula:

$$h(p^\alpha) = \sum_{k=0}^{\alpha-1}((p^{\alpha-k} - p^{\alpha-k-1})p^k) + p^\alpha = \alpha p^\alpha - (\alpha - 1)p^{\alpha-1}.$$

B. Count $a \times b$

Now, $g(n) = \sum_{d|n} f(n) = \sum_{d|n} d^2 - \sum_{d|n} h(d) = s_2(n) - H(n)$.

B. Count $a \times b$

Now, $g(n) = \sum_{d|n} f(n) = \sum_{d|n} d^2 - \sum_{d|n} h(d) = s_2(n) - H(n)$.

Let $n = p_1^{\alpha_1} \dots p_k^{\alpha_k}$. Then

$$s_2(n) = \prod_{i=1}^{k} \sum_{j=0}^{\alpha_i} p_i^{2j}.$$

B. Count $a \times b$

Now, $g(n) = \sum_{d|n} f(n) = \sum_{d|n} d^2 - \sum_{d|n} h(d) = s_2(n) - H(n)$.

Let $n = p_1^{\alpha_1} \dots p_k^{\alpha_k}$. Then

$$s_2(n) = \prod_{i=1}^{k} \sum_{j=0}^{\alpha_i} p_i^{2j}.$$

Since $h(n)$ is multiplicative, $H(n)$ is multiplicative too:

$$H(n) = \prod_{i=1}^{k} \sum_{j=0}^{\alpha_i} h(p_i^j) = \prod_{i=1}^{k} \alpha_i p_i^{\alpha_i} = n \prod_{i=1}^{k} \alpha_i.$$

B. Count $a \times b$

Now, $g(n) = \sum_{d|n} f(n) = \sum_{d|n} d^2 - \sum_{d|n} h(d) = s_2(n) - H(n)$.
Let $n = p_1^{\alpha_1} \dots p_k^{\alpha_k}$. Then

$$s_2(n) = \prod_{i=1}^{k} \sum_{j=0}^{\alpha_i} p_i^{2j}.$$

Since $h(n)$ is multiplicative, $H(n)$ is multiplicative too:

$$H(n) = \prod_{i=1}^{k} \sum_{j=0}^{\alpha_i} h(p_i^j) = \prod_{i=1}^{k} \alpha_i p_i^{\alpha_i} = n \prod_{i=1}^{k} \alpha_i.$$

Both $s_2(n)$ and $H(n)$ can be computed easily given factorization of n. It can be found straghtforwardly in $O(\sqrt{n})$, with posslble speed-up to $O(\sqrt{n}/\log n)$ using precomputed prime tables up to $\sqrt{n}$.

C. Play a game

We are given a string s and a set of forbidden strings A. Two players play a game: if at the beginning of one's turn the current string is empty or belongs to A, the player loses immediately, otherwise, he can erase a symbol either from the beginning of from the end of the string. Find winning player for several substrings of s as starting strings.

C. Play a game

If the string s were small, the following simple $O(n^2)$ solution would suffice.

C. Play a game

If the string s were small, the following simple $O(n^2)$ solution would suffice. Denote $w_{l,r}$ the outcome of the game if the game is played on the substring with positions $s[l;r)$. Values of $w_{l,r}$ are deteremined easily using DP:

C. Play a game

If the string s were small, the following simple $O(n^2)$ solution would suffice. Denote $w_{l,r}$ the outcome of the game if the game is played on the substring with positions $s[l;r)$. Values of $w_{l,r}$ are deteremined easily using DP:

- if $l = r$ (empty substring), or substring $s[l;r)$ belongs to A, then $w_{l,r} = L$ (forced lose)

C. Play a game

If the string s were small, the following simple $O(n^2)$ solution would suffice. Denote $w_{l,r}$ the outcome of the game if the game is played on the substring with positions $s[l;r)$. Values of $w_{l,r}$ are deteremined easily using DP:

- if $l = r$ (empty substring), or substring $s[l;r)$ belongs to A, then $w_{l,r} = L$ (forced lose)
- otherwise, $w_{l,r} = W$ if one of $w_{l+1,r}$ or $w_{l,r-1}$ is L, otherwise, $w_{l,r} = L$.

C. Play a game

Example

Let $s = abacaba$, $A = \{b, bac, cab\}$

The table of $w_{l,r}$ looks as follows:

$l \backslash r$	0	1	2	3	4	5	6	7
0	L							
1		L						
2			L					
3				L				
4					L			
5						L		
6							L	
7								L

C. Play a game

Example

Let $s = abacaba$, $A = \{b, bac, cab\}$

The table of $w_{l,r}$ looks as follows:

$l \backslash r$	0	1	2	3	4	5	6	7
0	L	W						
1		L	**L**					
2			L	W				
3				L	W			
4					L	W		
5						L	**L**	
6							L	W
7								L

C. Play a game

Example

Let $s = abacaba$, $A = \{b, bac, cab\}$

The table of $w_{l,r}$ looks as follows:

$l \backslash r$	0	1	2	3	4	5	6	7
0	L	W	W					
1		L	**L**	W				
2			L	W	L			
3				L	W	L		
4					L	W	W	
5						L	**L**	W
6							L	W
7								L

C. Play a game

Example

Let $s = abacaba$, $A = \{b, bac, cab\}$

The table of $w_{l,r}$ looks as follows:

$l \backslash r$	0	1	2	3	4	5	6	7
0	L	W	W	L				
1		L	**L**	W	**L**			
2			L	W	L	W		
3				L	W	L	**L**	
4					L	W	W	L
5						L	**L**	W
6							L	W
7								L

C. Play a game

Example

Let $s = abacaba$, $A = \{b, bac, cab\}$

The table of $w_{l,r}$ looks as follows:

$l \backslash r$	0	1	2	3	4	5	6	7
0	L	W	W	L	W			
1		L	**L**	W	**L**	W		
2			L	W	L	W	W	
3				L	W	L	**L**	W
4					L	W	W	L
5						L	**L**	W
6							L	W
7								L

C. Play a game

Example

Let $s = abacaba$, $A = \{b, bac, cab\}$

The table of $w_{l,r}$ looks as follows:

$l \backslash r$	0	1	2	3	4	5	6	7
0	L	W	W	L	W	L		
1		L	**L**	W	**L**	W	*L*	
2			L	W	L	W	W	L
3				L	W	L	**L**	W
4					L	W	W	L
5						L	**L**	W
6							L	W
7								L

C. Play a game

Example

Let $s = abacaba$, $A = \{b, bac, cab\}$

The table of $w_{l,r}$ looks as follows:

$l \backslash r$	0	1	2	3	4	5	6	7
0	L	W	W	L	W	L	W	
1		L	**L**	W	**L**	W	*L*	W
2			L	W	L	W	W	L
3				L	W	L	**L**	W
4					L	W	W	L
5						L	**L**	W
6							L	W
7								L

C. Play a game

Example

Let $s = abacaba$, $A = \{b, bac, cab\}$

The table of $w_{l,r}$ looks as follows:

$l \backslash r$	0	1	2	3	4	5	6	7
0	L	W	W	L	W	L	W	L
1		L	**L**	W	**L**	W	*L*	W
2			L	W	L	W	W	L
3				L	W	L	**L**	W
4					L	W	W	L
5						L	**L**	W
6							L	W
7								L

C. Play a game

$l \backslash r$	0	1	2	3	4	5	6	7
0	L	W	W	L	W	L	W	L
1		L	**L**	W	**L**	W	*L*	W
2			L	W	L	W	W	L
3				L	W	L	**L**	W
4					L	W	W	L
5						L	**L**	W
6							L	W
7								L

We can notice that $w_{l,r}$ is almost always equal to $w_{l+1,r-1}$. The exceptions are:

C. Play a game

$l \backslash r$	0	1	2	3	4	5	6	7
0	L	W	W	L	W	L	W	L
1		L	**L**	W	**L**	W	*L*	W
2			L	W	L	W	W	L
3				L	W	L	**L**	W
4					L	W	W	L
5						L	**L**	W
6							L	W
7								L

We can notice that $w_{l,r}$ is almost always equal to $w_{l+1,r-1}$. The exceptions are:

1. Forced lose because the substring is forbidden (e.g. $[1;2)$, $[3;6)$)

C. Play a game

$l \backslash r$	0	1	2	3	4	5	6	7
0	L	W	W	L	W	L	W	L
1		L	**L**	W	**L**	W	*L*	W
2			L	W	L	W	W	L
3				L	W	L	**L**	W
4					L	W	W	L
5						L	**L**	W
6							L	W
7								L

We can notice that $w_{l,r}$ is almost always equal to $w_{l+1,r-1}$. The exceptions are:

1. Forced lose because the substring is forbidden (e.g. $[1;2)$, $[3;6)$)
2. Win because $w_{l+1,r}$ or $w_{l,r-1}$ is a forced lose (e.g. $[1;3)$, $[1,5)$)

C. Play a game

$l \backslash r$	0	1	2	3	4	5	6	7
0	L	W	W	L	W	L	W	L
1		L	**L**	W	**L**	W	*L*	W
2			L	W	L	W	W	L
3				L	W	L	**L**	W
4					L	W	W	L
5						L	**L**	W
6							L	W
7								L

We can notice that $w_{l,r}$ is almost always equal to $w_{l+1,r-1}$. The exceptions are:

1. Forced lose because the substring is forbidden (e.g. $[1;2)$, $[3;6)$)
2. Win because $w_{l+1,r}$ or $w_{l,r-1}$ is a forced lose (e.g. $[1;3)$, $[1,5)$)
3. Lose because $w_{l+2,r}$ and $w_{l,r-2}$ are loses (e.g. $[1;6)$)

C. Play a game

$l\backslash r$	0	1	2	3	4	5	6	7
0	L	W	W	L	W	L	W	L
1		L	**L**	W	**L**	W	*L*	W
2			L	W	L	W	W	L
3				L	W	L	**L**	W
4					L	W	W	L
5						L	**L**	W
6							L	W
7								L

We can notice that $w_{l,r}$ is almost always equal to $w_{l+1,r-1}$. The exceptions are:

1. Forced lose because the substring is forbidden (e.g. $[1;2)$, $[3;6)$)
2. Win because $w_{l+1,r}$ or $w_{l,r-1}$ is a forced lose (e.g. $[1;3)$, $[1,5)$)
3. Lose because $w_{l+2,r}$ and $w_{l,r-2}$ are loses (e.g. $[1;6)$)

It can easily be shown that in all other cases $w_{l,r}$ is indeed equal to $w_{l+1,r-1}$.

C. Play a game

Knowing that, we will do the following: store two adjacent diagonals of the table, and gradually move them to the up and to the right while processing all the cases the elements change and answering queries off-line. We assume that we know all occurences of elements of A as substrings of s.

C. Play a game

Knowing that, we will do the following: store two adjacent diagonals of the table, and gradually move them to the up and to the right while processing all the cases the elements change and answering queries off-line. We assume that we know all occurences of elements of A as substrings of s.
For each query and each occurence store the index of diagonal it is concerned.

C. Play a game

Knowing that, we will do the following: store two adjacent diagonals of the table, and gradually move them to the up and to the right while processing all the cases the elements change and answering queries off-line. We assume that we know all occurences of elements of A as substrings of s.
For each query and each occurence store the index of diagonal it is concerned.

- When answering a query, simply access the diagonal's element (we assume that is has been maintained correctly)
- When processing an occurence, change the element of the diagonal to L, and the elements immediately to the up and to the right to W

C. Play a game

It suffices to maintain the third condition ($w_{l+2,r} = w_{l,r-2} = L$).
We cannot scan the diagonals and find these situations explicitly.

C. Play a game

It suffices to maintain the third condition ($w_{l+2,r} = w_{l,r-2} = L$). We cannot scan the diagonals and find these situations explicitly. However, we can notice that these situations arise only as a result of a forced lose in the diagonal which is filled with W by default.

C. Play a game

It suffices to maintain the third condition ($w_{l+2,r} = w_{l,r-2} = L$). We cannot scan the diagonals and find these situations explicitly. However, we can notice that these situations arise only as a result of a forced lose in the diagonal which is filled with W by default. Call a position $w_{l,r}$ *interesting* if $w_{l+2,r}$ or $w_{l,r-2}$ was forcedly changed from W to L. We will maintain a list of interesting positions in each diagonal.

C. Play a game

It suffices to maintain the third condition ($w_{l+2,r} = w_{l,r-2} = L$). We cannot scan the diagonals and find these situations explicitly. However, we can notice that these situations arise only as a result of a forced lose in the diagonal which is filled with W by default. Call a position $w_{l,r}$ *interesting* if $w_{l+2,r}$ or $w_{l,r-2}$ was forcedly changed from W to L. We will maintain a list of interesting positions in each diagonal.

- Every time we make a forced lose (including the condition 3), we put $w_{l-2,r}$ and $w_{l,r+2}$ into the list of interesting positions.

C. Play a game

It suffices to maintain the third condition ($w_{l+2,r} = w_{l,r-2} = L$). We cannot scan the diagonals and find these situations explicitly. However, we can notice that these situations arise only as a result of a forced lose in the diagonal which is filled with W by default. Call a position $w_{l,r}$ *interesting* if $w_{l+2,r}$ or $w_{l,r-2}$ was forcedly changed from W to L. We will maintain a list of interesting positions in each diagonal.

- Every time we make a forced lose (including the condition 3), we put $w_{l-2,r}$ and $w_{l,r+2}$ into the list of interesting positions.
- Before answering queries for the current diagonal, we scan the list of interesting positions and make forced loses if needed.

C. Play a game

It suffices to maintain the third condition ($w_{l+2,r} = w_{l,r-2} = L$). We cannot scan the diagonals and find these situations explicitly. However, we can notice that these situations arise only as a result of a forced lose in the diagonal which is filled with W by default. Call a position $w_{l,r}$ *interesting* if $w_{l+2,r}$ or $w_{l,r-2}$ was forcedly changed from W to L. We will maintain a list of interesting positions in each diagonal.

- Every time we make a forced lose (including the condition 3), we put $w_{l-2,r}$ and $w_{l,r+2}$ into the list of interesting positions.
- Before answering queries for the current diagonal, we scan the list of interesting positions and make forced loses if needed.

Let T be the total number of occurences of elements of A as substrings of s. It can be shown that the number of times situation 3 arises is $O(T)$.

C. Play a game

It suffices to maintain the third condition ($w_{l+2,r} = w_{l,r-2} = L$). We cannot scan the diagonals and find these situations explicitly. However, we can notice that these situations arise only as a result of a forced lose in the diagonal which is filled with W by default. Call a position $w_{l,r}$ *interesting* if $w_{l+2,r}$ or $w_{l,r-2}$ was forcedly changed from W to L. We will maintain a list of interesting positions in each diagonal.

- Every time we make a forced lose (including the condition 3), we put $w_{l-2,r}$ and $w_{l,r+2}$ into the list of interesting positions.
- Before answering queries for the current diagonal, we scan the list of interesting positions and make forced loses if needed.

Let T be the total number of occurences of elements of A as substrings of s. It can be shown that the number of times situation 3 arises is $O(T)$. Therefore, the total number of events occuring during the "sweep-line diagonal" process is $O(T)$, and its time complexity is $O(T)$.

C. Play a game

Let all elements of A be unique. Denote $L = \sum_{a_i \in A} |a_i|$.
We use Aho-Corasick to find all occurences in time $O(n + L + T)$.

C. Play a game

Let all elements of A be unique. Denote $L = \sum_{a_i \in A} |a_i|$.
We use Aho-Corasick to find all occurences in time $O(n + L + T)$.
Therefore, it suffices to estimate T.

C. Play a game

Let all elements of A be unique. Denote $L = \sum_{a_i \in A} |a_i|$.
We use Aho-Corasick to find all occurences in time $O(n + L + T)$.
Therefore, it suffices to estimate T.

Statement

$T = O(n\sqrt{L})$.

Proof

C. Play a game

Let all elements of A be unique. Denote $L = \sum_{a_i \in A} |a_i|$.
We use Aho-Corasick to find all occurences in time $O(n + L + T)$.
Therefore, it suffices to estimate T.

Statement

$T = O(n\sqrt{L})$.

Proof

The total number of occurences of strings of length l does not exceed n. Thus, T does not exceed $n\times$(number of different lengths of elements of A).

C. Play a game

Let all elements of A be unique. Denote $L = \sum_{a_i \in A} |a_i|$.
We use Aho-Corasick to find all occurences in time $O(n + L + T)$.
Therefore, it suffices to estimate T.

Statement

$T = O(n\sqrt{L})$.

Proof

The total number of occurences of strings of length l does not exceed n. Thus, T does not exceed $n\times$(number of different lengths of elements of A).
The number of different lengths is maximal if lengths are 1, 2, ..., and is $O(\sqrt{L})$, which implies the statement.

C. Play a game

Let all elements of A be unique. Denote $L = \sum_{a_i \in A} |a_i|$.
We use Aho-Corasick to find all occurences in time $O(n + L + T)$.
Therefore, it suffices to estimate T.

Statement

$T = O(n\sqrt{L})$.

Proof

The total number of occurences of strings of length l does not exceed n. Thus, T does not exceed $n \times$(number of different lengths of elements of A).
The number of different lengths is maximal if lengths are 1, 2, ..., and is $O(\sqrt{L})$, which implies the statement.

This concludes the analysis of the problem. The resulting solution has complexity $O(L + n\sqrt{L})$.

D. Pipes selection

We are given an array of non-negative integers with sum s. Let k_x be total number of segments with sum x. For every x from 1 to s find $\lfloor \frac{k_x+1}{2} \rfloor$-th lexicographically smallest segment with sum x (segments are ordered by left end, then by right end).

D. Pipes selection

First of all, how do we find all k_x efficiently?

D. Pipes selection

First of all, how do we find all k_x efficiently? This is a standard application of fast polynomial multiplication using FFT (fast Fourier transform).

D. Pipes selection

First of all, how do we find all k_x efficiently? This is a standard application of fast polynomial multiplication using FFT (fast Fourier transform).

Let p_n be the sum of first n elements, and let q_x be the number of such n that $p_n = x$. Construct polynomials $A(x) = \sum_{i=0}^{s} q_i x^i$ and $B(x) = \sum_{i=0}^{s} q_{s-i} x^i$. Define $C(x) = A(x)B(x) = \sum_{i=0}^{2s} c_i x^i$.

D. Pipes selection

First of all, how do we find all k_x efficiently? This is a standard application of fast polynomial multiplication using FFT (fast Fourier transform).

Let p_n be the sum of first n elements, and let q_x be the number of such n that $p_n = x$. Construct polynomials $A(x) = \sum_{i=0}^{s} q_i x^i$ and $B(x) = \sum_{i=0}^{s} q_{s-i} x^i$. Define $C(x) = A(x)B(x) = \sum_{i=0}^{2s} c_i x^i$. It is evident from multiplication definition that $k_x = c_{s+x}$.

D. Pipes selection

Ok, but how do we find a segment with given sum and lexicographical position? Iterate over all possible beginning takes too long ($O(ns)$ time in the worst case).

D. Pipes selection

Ok, but how do we find a segment with given sum and lexicographical position? Iterate over all possible beginning takes too long ($O(ns)$ time in the worst case).

Sqrt-decomposition helps. Let's divide the array into t blocks of equal size.

D. Pipes selection

Ok, but how do we find a segment with given sum and lexicographical position? Iterate over all possible beginning takes too long ($O(ns)$ time in the worst case).

Sqrt-decomposition helps. Let's divide the array into t blocks of equal size.

For j-th block with beginning l_j and end r_j, construct polynomial $B_j(x) = \sum_{i=l_j}^{r_j} q_{s-i}x^i$, and define $C_j(x) = A(x)B_j(x) = \sum_{i=0}^{2s} c_{j_i}x^i$.

The number of segments with sum x which left end lies in segment $[l_j; r_j]$ is exactly $c_{j_{s+x}}$.

D. Pipes selection

Given $C(x)$ and all $C_j(x)$, we can find every answer in $O(n/t + t)$.

D. Pipes selection

Given $C(x)$ and all $C_j(x)$, we can find every answer in $O(n/t + t)$.

- First, find the block which contains the beginning of the sought segment by simply iterating the blocks from left to right (this requires comparisons of lex number with $c_{j_{s+x}}$.

D. Pipes selection

Given $C(x)$ and all $C_j(x)$, we can find every answer in $O(n/t + t)$.

- First, find the block which contains the beginning of the sought segment by simply iterating the blocks from left to right (this requires comparisons of lex number with $c_{j_{s+x}}$.
- Then, iterate over elements inside the block to find the actual segment (this doesn't require any knowledge about $C_j(x)$)

D. Pipes selection

The complexity is $O(ts \log s + s(n/t + t))$.

D. Pipes selection

The complexity is $O(ts \log s + s(n/t + t))$.
To achieve optimum choose $t \sim \sqrt{n/\log s}$, for a total complexity of $O(s\sqrt{n \log s})$.

D. Pipes selection

The complexity is $O(ts \log s + s(n/t + t))$.
To achieve optimum choose $t \sim \sqrt{n/\log s}$, for a total complexity of $O(s\sqrt{n \log s})$.
In practice, FFT has significantly higher intrinsic constant factor, which means that in order to balance it out, t should be slightly lower than the theoretic optimum.

E. Rebuild

Given a sequence of points in the plane, build circles centered at each point such that circles which are centered at consecutive points are tangent, and also the first and the last circles are tangent. Minimize total area of circles.

E. Rebuild

Given a sequence of points in the plane, build circles centered at each point such that circles which are centered at consecutive points are tangent, and also the first and the last circles are tangent. Minimize total area of circles.
Let d_i be the distance between i-th point and $(i+1)$-th point, and d_n be the distance between the first and the last point. Restate the problem: we have to choose radii x_i such that $x_1 + x_2 = d_1, \ldots,$ $x_n + x_1 = d_1$, while minimizing $\sum x_i^2$.

E. Rebuild

Consider cases when n is even or odd.

E. Rebuild

Consider cases when n is even or odd.

- n is odd.

E. Rebuild

Consider cases when n is even or odd.

- n is odd. We can find $S = \sum x_i$ as $\sum d_i/2$, and then express all x_i explicitly, which means that there is unique solution to the system. It suffices to check that all x_i are non-negative.

E. Rebuild

Consider cases when n is even or odd.

- n is odd. We can find $S = \sum x_i$ as $\sum d_i/2$, and then express all x_i explicitly, which means that there is unique solution to the system. It suffices to check that all x_i are non-negative.
- n is even.

E. Rebuild

Consider cases when n is even or odd.

- n is odd. We can find $S = \sum x_i$ as $\sum d_i/2$, and then express all x_i explicitly, which means that there is unique solution to the system. It suffices to check that all x_i are non-negative.
- n is even. Now $S = \sum_{i \text{ is odd}} d_i = \sum_{i \text{ is even}} d_i$ must hold, else no solution exists.

E. Rebuild

Consider cases when n is even or odd.

- n is odd. We can find $S = \sum x_i$ as $\sum d_i/2$, and then express all x_i explicitly, which means that there is unique solution to the system. It suffices to check that all x_i are non-negative.
- n is even. Now $S = \sum_{i \text{ is odd}} d_i = \sum_{i \text{ is even}} d_i$ must hold, else no solution exists. Fix x_1, then $x_2 = d_1 - x_1$, $x_3 = d_2 - d_1 + x_1$, and so on. Every x_i should be non-negative, which implies inequalities in terms of x_1. If these inequalities contradict, there is no solution either, otherwise x_1 is forced to belong to a segment $L \leqslant x_1 \leqslant R$.

E. Rebuild

Consider cases when n is even or odd.

- n is odd. We can find $S = \sum x_i$ as $\sum d_i/2$, and then express all x_i explicitly, which means that there is unique solution to the system. It suffices to check that all x_i are non-negative.
- n is even. Now $S = \sum_{i \text{ is odd}} d_i = \sum_{i \text{ is even}} d_i$ must hold, else no solution exists. Fix x_1, then $x_2 = d_1 - x_1$, $x_3 = d_2 - d_1 + x_1$, and so on. Every x_i should be non-negative, which implies inequalities in terms of x_1. If these inequalities contradict, there is no solution either, otherwise x_1 is forced to belong to a segment $L \leqslant x_1 \leqslant R$.
  Finally, substituting expressions for x_i, obtain $\sum x_i^2 = ax_1^2 + bx_1 + c$ for some real a, b, c.

E. Rebuild

Consider cases when n is even or odd.

- n is odd. We can find $S = \sum x_i$ as $\sum d_i/2$, and then express all x_i explicitly, which means that there is unique solution to the system. It suffices to check that all x_i are non-negative.
- n is even. Now $S = \sum_{i \text{ is odd}} d_i = \sum_{i \text{ is even}} d_i$ must hold, else no solution exists. Fix x_1, then $x_2 = d_1 - x_1$, $x_3 = d_2 - d_1 + x_1$, and so on. Every x_i should be non-negative, which implies inequalities in terms of x_1. If these inequalities contradict, there is no solution either, otherwise x_1 is forced to belong to a segment $L \leqslant x_1 \leqslant R$.
  Finally, substituting expressions for x_i, obtain $\sum x_i^2 = ax_1^2 + bx_1 + c$ for some real a, b, c.
  Minimizing a quadratic function on a segment is trivial: if global minimum $x_0 = -\frac{b}{2a}$ belongs to $[L; R]$, then x_0 is the answer, otherwise one of the segment ends L, R is the answer.

F. Almost sorted array

Given an array of numbers, we have to erase at most one element so that to make the array becomes either non-decreasing or non-increasing.

F. Almost sorted array

Given an array of numbers, we have to erase at most one element so that to make the array becomes either non-decreasing or non-increasing.
Let us try to make array non-decreasing, and then repeat the procedure for the reversed array.

F. Almost sorted array

Given an array of numbers, we have to erase at most one element so that to make the array becomes either non-decreasing or non-increasing.
Let us try to make array non-decreasing, and then repeat the procedure for the reversed array.
Suppose that we have erased a_i. The resulting array is non-decreasing if first $i - 1$ elements are sorted, last $n - i$ elements are sorted, and $a_{i-1} \leqslant a_{i+1}$ (if $i = 1$ or $i = n$, this condition is redundant). Find the longest sorted prefix and suffix, then try to erase each element. This makes for a simple $O(n)$ solution.

G. Dancing Stars on Me

Given a set of points with integer coordinates, determine if it coincides with the set of vertices of a regular polygon.

G. Dancing Stars on Me

Given a set of points with integer coordinates, determine if it coincides with the set of vertices of a regular polygon.

Fact

The only possible regular polygon with all vertices at integer points is a square.

G. Dancing Stars on Me

Given a set of points with integer coordinates, determine if it coincides with the set of vertices of a regular polygon.

Fact

The only possible regular polygon with all vertices at integer points is a square.

To show this, consider three consecutive vertices A, B, C of the regular n-gon. Observe that vector $\overline{BC}$ is the vector $\overline{AB}$ rotated by $2\pi/n$. Since both vectors have integer coordinates, we conclude that $\cos(2\pi/n)$ and $\sin(2\pi/n)$ are both rational. The only $n \geqslant 3$ satisfying this is $n = 4$.

G. Dancing Stars on Me

Given a set of points with integer coordinates, determine if it coincides with the set of vertices of a regular polygon.

Fact

The only possible regular polygon with all vertices at integer points is a square.

To show this, consider three consecutive vertices A, B, C of the regular n-gon. Observe that vector $\overline{BC}$ is the vector $\overline{AB}$ rotated by $2\pi/n$. Since both vectors have integer coordinates, we conclude that $\cos(2\pi/n)$ and $\sin(2\pi/n)$ are both rational. The only $n \geqslant 3$ satisfying this is $n = 4$.
Checking that four given points are at vertices of a square is trivial.

H. Partial Tree

We want to build a tree on n vertices. For a vertex of degree i we get score d_i. Maximize total score (over all vertices).

H. Partial Tree

Which degree sequences $d_1, \ldots, d_n$ correspond to trees on n vertices? Trivial necessary coniditions are $d_i \geqslant 1$ and $\sum d_i = 2(n-1)$.

H. Partial Tree

Which degree sequences $d_1, \ldots, d_n$ correspond to trees on n vertices? Trivial necessary coniditions are $d_i \geqslant 1$ and $\sum d_i = 2(n-1)$.
These are actually sufficient. Construct a tree as follows: connect all vertices with $d_i > 1$ in a chain, then connect enough leaves to every vertex of the chain to obtain needed degrees.

H. Partial Tree

Which degree sequences $d_1, \ldots, d_n$ correspond to trees on n vertices? Trivial necessary coniditions are $d_i \geqslant 1$ and $\sum d_i = 2(n-1)$.

These are actually sufficient. Construct a tree as follows: connect all vertices with $d_i > 1$ in a chain, then connect enough leaves to every vertex of the chain to obtain needed degrees.

Thus we have to solve a variety of the backpack problem: given cost for an item of every weight from 1 to $n-1$, choose n items with total weight of $2(n-1)$ and maximal possible cost. For convenience, we substract 1 from all weights, so the total weight becomes $n-2$.

H. Partial Tree

Which degree sequences $d_1, \ldots, d_n$ correspond to trees on n vertices? Trivial necessary coniditions are $d_i \geqslant 1$ and $\sum d_i = 2(n-1)$.

These are actually sufficient. Construct a tree as follows: connect all vertices with $d_i > 1$ in a chain, then connect enough leaves to every vertex of the chain to obtain needed degrees.

Thus we have to solve a variety of the backpack problem: given cost for an item of every weight from 1 to $n-1$, choose n items with total weight of $2(n-1)$ and maximal possible cost. For convenience, we substract 1 from all weights, so the total weight becomes $n-2$.

Start from the set of n items of weight 0, then replace them with heavier items one by one. Denote dp_w the maximal cost of a set with total weight w obtained this way. By definition, $dp_0 = nf(0)$, $dp_w = \max_{k=1}^{w} dp_{w-k} + f(k) - f(0)$. The answer is dp_{n-2}. This yields an $O(n^2)$ solution.

I. Chess puzzle

We are given a rectangular board, where some cells have fixed colors (black or white), while some haven't. We have to color all non-colored cells. We get 1 point for every pair of cells (x_1, y_1) and (x_2, y_2) if:

- $|x_1 - x_2| = a$, $|y_1 - y_2| = b$ $(a, b > 0)$
- cells (x_1, y_1) and (x_2, y_2) are of different colors

Find a coloring that maximizes total score, if there are several colorings, choose lexicographically minimal.

I. Chess puzzle

Let's ignore lex-min requirement for now. How to build any optimal coloring?

I. Chess puzzle

Let's ignore lex-min requirement for now. How to build any optimal coloring?

If we add edges between pairs of cells with $|x_1 - x_2| = a$, $|y_1 - y_2| = b$, we obtain a bipartite graph. We can construct a convenient partition: first a rows are in the first part, next a rows are in the second part and so on.

I. Chess puzzle

Let's ignore lex-min requirement for now. How to build any optimal coloring?

If we add edges between pairs of cells with $|x_1 - x_2| = a$, $|y_1 - y_2| = b$, we obtain a bipartite graph. We can construct a convenient partition: first a rows are in the first part, next a rows are in the second part and so on.

If $n = m = 5$, $a = 2$, the partition looks as follows:

1	1	1	1	1
1	1	1	1	1
2	2	2	2	2
2	2	2	2	2
1	1	1	1	1

Flip the colors of all cells in the second part. Now we have to maximize number of adjacent pairs with the *same* color.

I. Chess puzzle

This can be done using min-cut:

I. Chess puzzle

This can be done using min-cut:

- add source and sink

I. Chess puzzle

This can be done using min-cut:

- add source and sink
- add edges with capacity 1 between adjacent pairs of cells (in both directions)

I. Chess puzzle

This can be done using min-cut:

- add source and sink
- add edges with capacity 1 between adjacent pairs of cells (in both directions)
- add edges with capacity ∞ from source to cells which are initially colored black (accounting for flipping colors of the second part)

I. Chess puzzle

This can be done using min-cut:

- add source and sink
- add edges with capacity 1 between adjacent pairs of cells (in both directions)
- add edges with capacity ∞ from source to cells which are initially colored black (accounting for flipping colors of the second part)
- add edges with capacity ∞ from cells which are initially colored white to sink

I. Chess puzzle

This can be done using min-cut:

- add source and sink
- add edges with capacity 1 between adjacent pairs of cells (in both directions)
- add edges with capacity ∞ from source to cells which are initially colored black (accounting for flipping colors of the second part)
- add edges with capacity ∞ from cells which are initially colored white to sink

Any $S - T$ cut corresponds to coloring (S's part — black, T's part — white), and its capacity is exactly the number of points of different color. Minimal $S - T$ cut corresponds to coloring with maximal number of same-colored adjacent pairs.

I. Chess puzzle

This can be done using min-cut:

- add source and sink
- add edges with capacity 1 between adjacent pairs of cells (in both directions)
- add edges with capacity ∞ from source to cells which are initially colored black (accounting for flipping colors of the second part)
- add edges with capacity ∞ from cells which are initially colored white to sink

Any $S - T$ cut corresponds to coloring (S's part — black, T's part — white), and its capacity is exactly the number of points of different color. Minimal $S - T$ cut corresponds to coloring with maximal number of same-colored adjacent pairs.
Any sufficiently fast algorithm for max-flow (e.g. Dinic) will allow us to build some minimal cut.

I. Chess puzzle

How to build lexicographically minimal coloring?

I. Chess puzzle

How to build lexicographically minimal coloring?
First build any optimal coloring.

I. Chess puzzle

How to build lexicographically minimal coloring?
First build any optimal coloring. Then, consider cells in the lexicographical order. If actual color (that is, without flipping) of the current cell is `B`, then we can't minimize it further. Add the edge between the cell and source/sink (depending on the part of the cell), as if the color were fixed from the beginning.

I. Chess puzzle

How to build lexicographically minimal coloring?
First build any optimal coloring. Then, consider cells in the lexicographical order. If actual color (that is, without flipping) of the current cell is `B`, then we can't minimize it further. Add the edge between the cell and source/sink (depending on the part of the cell), as if the color were fixed from the beginning.
If color of the current cell is `W`, then we should try to change it to B. Assume that colors of all earlier cells are fixed by adding edges from source/to sink.

I. Chess puzzle

How to build lexicographically minimal coloring?

First build any optimal coloring. Then, consider cells in the lexicographical order. If actual color (that is, without flipping) of the current cell is `B`, then we can't minimize it further. Add the edge between the cell and source/sink (depending on the part of the cell), as if the color were fixed from the beginning.

If color of the current cell is `W`, then we should try to change it to B. Assume that colors of all earlier cells are fixed by adding edges from source/to sink.

Suppose that changing is possible. Add edge between the cell and source/sink (depending on whether the cell's color was flipped or not).

I. Chess puzzle

How to build lexicographically minimal coloring?
First build any optimal coloring. Then, consider cells in the lexicographical order. If actual color (that is, without flipping) of the current cell is `B`, then we can't minimize it further. Add the edge between the cell and source/sink (depending on the part of the cell), as if the color were fixed from the beginning.
If color of the current cell is `W`, then we should try to change it to B. Assume that colors of all earlier cells are fixed by adding edges from source/to sink.
Suppose that changing is possible. Add edge between the cell and source/sink (depending on whether the cell's color was flipped or not). The value of min-cut should not increase; equivalently, it should be impossible to push one unit of flow after adding the edge.

I. Chess puzzle

How to build lexicographically minimal coloring?

First build any optimal coloring. Then, consider cells in the lexicographical order. If actual color (that is, without flipping) of the current cell is `B`, then we can't minimize it further. Add the edge between the cell and source/sink (depending on the part of the cell), as if the color were fixed from the beginning.

If color of the current cell is `W`, then we should try to change it to B. Assume that colors of all earlier cells are fixed by adding edges from source/to sink.

Suppose that changing is possible. Add edge between the cell and source/sink (depending on whether the cell's color was flipped or not). The value of min-cut should not increase; equivalently, it should be impossible to push one unit of flow after adding the edge. It means that the cell and sink/source should not be connected in the residual network.

I. Chess puzzle

To sum up, we have to add edges to the network and for each vertex remember whether it is reachable from the source, and whether sink is reachable from it using edges of residual network.

I. Chess puzzle

To sum up, we have to add edges to the network and for each vertex remember whether it is reachable from the source, and whether sink is reachable from it using edges of residual network. To do this, after adding every edge run DFS from the cell adjacent to it; visibility markings should not be cleared between runs.

I. Chess puzzle

To sum up, we have to add edges to the network and for each vertex remember whether it is reachable from the source, and whether sink is reachable from it using edges of residual network. To do this, after adding every edge run DFS from the cell adjacent to it; visibility markings should not be cleared between runs. This process takes $O(n^2)$ total time, which means complexity depends entirely on the max-flow algorithm used.

J. Chip Factory

Given an array s_i, find

$$\max_{i,j,k \text{ — distinct indices}} (s_i + s_j) \oplus s_k$$

.

J. Chip Factory

Suppose that we have fixed i and j in the $(s_i + s_j) \oplus s_k$ expression.

J. Chip Factory

Suppose that we have fixed i and j in the $(s_i + s_j) \oplus s_k$ expression. Let l be the maximal position of 1 in a binary representation of any s_k.

J. Chip Factory

Suppose that we have fixed i and j in the $(s_i + s_j) \oplus s_k$ expression.
Let l be the maximal position of 1 in a binary representation of any s_k.
In order to maximize the above expression, the l-th bit of s_k should differ from the l-th bit of $s_i + s_j$ if that is possible

J. Chip Factory

Suppose that we have fixed i and j in the $(s_i + s_j) \oplus s_k$ expression. Let l be the maximal position of 1 in a binary representation of any s_k.

In order to maximize the above expression, the l-th bit of s_k should differ from the l-th bit of $s_i + s_j$ if that is possible (that is, if we can choose k (different from i and j) such that l-th bit of s_k satisfies us).

J. Chip Factory

Suppose that we have fixed i and j in the $(s_i + s_j) \oplus s_k$ expression. Let l be the maximal position of 1 in a binary representation of any s_k.

In order to maximize the above expression, the l-th bit of s_k should differ from the l-th bit of $s_i + s_j$ if that is possible (that is, if we can choose k (different from i and j) such that l-th bit of s_k satisfies us).

Once we have fixed the l-th bit, we choose $(l-1)$-th bit according to the same reasoning, and so on.

J. Chip Factory

Suppose that we have fixed i and j in the $(s_i + s_j) \oplus s_k$ expression. Let l be the maximal position of 1 in a binary representation of any s_k.

In order to maximize the above expression, the l-th bit of s_k should differ from the l-th bit of $s_i + s_j$ if that is possible (that is, if we can choose k (different from i and j) such that l-th bit of s_k satisfies us).

Once we have fixed the l-th bit, we choose $(l-1)$-th bit according to the same reasoning, and so on.

At every given moment, several greatest bits of s_k are fixed. Next bit of s_k depends on whether we can choose k (different from i and j) so that a prefix of s_k matches our preference.

J. Chip Factory

s: $i = 1, j = 3$

00100_2

10110_2

00101_2

00111_2

01011_2

$s_i + s_j = 11101_2$

$s_k = ?????$

J. Chip Factory

s: $i = 1, j = 3$

00100_2

10110_2

00101_2

00111_2

01011_2

$$s_i + s_j = 11101_2$$

$$s_k = 0????_2$$

3 possible s_k match prefix.

J. Chip Factory

s: $i = 1, j = 3$

00100_2

10110_2

00101_2

00111_2

01011_2

$s_i + s_j = 11101_2$

$s_k = 00???_2$

2 possible s_k match prefix.

J. Chip Factory

s: $i = 1, j = 3$

00100_2

10110_2

00101_2

00111_2

01011_2

$s_i + s_j = 11101_2$

$s_k = 000??_2$

No possible s_k match prefix, have to choose 2-nd bit otherwise.

J. Chip Factory

s: $i = 1, j = 3$

00100_2

10110_2

00101_2

00111_2

01011_2

$s_i + s_j = 11101_2$

$s_k = 001??_2$

J. Chip Factory

s: $i = 1, j = 3$

00100_2

10110_2

00101_2

00111_2

01011_2

$s_i + s_j = 11101_2$

$s_k = 0011?_2$

No possible s_k match prefix (note that s_3 matches but k has to be different from i and j).

J. Chip Factory

s: $i = 1, j = 3$

00100_2

10110_2

00101_2

00111_2

01011_2

$$s_i + s_j = 11101_2$$

$$s_k = 0010?_2$$

J. Chip Factory

s: $\quad i = 1, j = 3$

00100_2

10110_2

00101_2

00111_2

01011_2

$$s_i + s_j = 11101_2$$

$$s_k = 00100_2$$

Maximal possible $(s_i + s_j) \oplus s_k$ is $11101_2 \oplus 00100_2 = 11001_2 = 25$.

J. Chip Factory

To determine existence of s_k with given prefix, store binary representations in a trie, with greater bits being first characters.

J. Chip Factory

To determine existence of s_k with given prefix, store binary representations in a trie, with greater bits being first characters. Also, for every reachable prefix store a list of s_k that begin with this prefix.

J. Chip Factory

To determine existence of s_k with given prefix, store binary representations in a trie, with greater bits being first characters. Also, for every reachable prefix store a list of s_k that begin with this prefix.

While building a prefix of optimal s_k, keep the position in the trie corresponding to current prefix. Use the list of s_k for the prefix when deciding the next symbol of s_k.

J. Chip Factory

To determine existence of s_k with given prefix, store binary representations in a trie, with greater bits being first characters. Also, for every reachable prefix store a list of s_k that begin with this prefix.

While building a prefix of optimal s_k, keep the position in the trie corresponding to current prefix. Use the list of s_k for the prefix when deciding the next symbol of s_k.

Total working time of this solution is $O(n^2 l)$.

K. Maximum spanning forest

Consider a region of rectangular grid. We have to answer n queries "add edges with weight c between all pairs of adjacent points inside a rectangle", and find the weight of maximal spanning forest built on the points inside a region.

K. Maximum spanning forest

We will solve the problem off-line. Represent a query rectangle as $[x_l; x_r) \times [y_l; y_r)$.

K. Maximum spanning forest

We will solve the problem off-line. Represent a query rectangle as $[x_l; x_r) \times [y_l; y_r)$.
Compress the coordinates: let x_i be the sorted sequence of different x's appearing as a border coordinate of a query rectangle; similarly, construct y_i.

K. Maximum spanning forest

We will solve the problem off-line. Represent a query rectangle as $[x_l; x_r) \times [y_l; y_r)$.

Compress the coordinates: let x_i be the sorted sequence of different x's appearing as a border coordinate of a query rectangle; similarly, construct y_i.

Call a rectangle *elementary* if its x-borders are adjacent elements of x_i, and same for y-borders.

K. Maximum spanning forest

We will solve the problem off-line. Represent a query rectangle as $[x_l; x_r) \times [y_l; y_r)$.

Compress the coordinates: let x_i be the sorted sequence of different x's appearing as a border coordinate of a query rectangle; similarily, construct y_i.

Call a rectangle *elementary* if its x-borders are adjacent elements of x_i, and same for y-borders. Denote $R_{i,j}$ $[x_i; x_{i+1}) \times [y_j; y_{j+1})$.

Clearly, there are $O(n^2)$ elementary rectangles.

K. Maximum spanning forest

We will solve the problem off-line. Represent a query rectangle as $[x_l; x_r) \times [y_l; y_r)$.

Compress the coordinates: let x_i be the sorted sequence of different x's appearing as a border coordinate of a query rectangle; similarily, construct y_i.

Call a rectangle *elementary* if its x-borders are adjacent elements of x_i, and same for y-borders. Denote $R_{i,j}$ $[x_i; x_{i+1}) \times [y_j; y_{j+1})$.

Clearly, there are $O(n^2)$ elementary rectangles.

All edges of the grid lie either inside of an elementary rectangle, or connect two points of adjacent elementary rectangles.

K. Maximum spanning forest

Observation

At any moment, all edges inside an elementary rectangle have the same weight (if we consider only the heaviest of multiple edges), and all edges between points of two adjacent rectangles have the same weight.

K. Maximum spanning forest

Observation

At any moment, all edges inside an elementary rectangle have the same weight (if we consider only the heaviest of multiple edges), and all edges between points of two adjacent rectangles have the same weight.
Moreover, let a and b be the weights of edges inside two adjacent rectangles, and c be the weight of edges between these rectangles. Then, $c \leqslant \min(a, b)$, since in-between edges can lie inside a query rectangle only if both adjacent rectangles do.

K. Maximum spanning forest

Observation

At any moment, all edges inside an elementary rectangle have the same weight (if we consider only the heaviest of multiple edges), and all edges between points of two adjacent rectangles have the same weight.
Moreover, let a and b be the weights of edges inside two adjacent rectangles, and c be the weight of edges between these rectangles. Then, $c \leqslant \min(a, b)$, since in-between edges can lie inside a query rectangle only if both adjacent rectangles do.

Introduce the following arrays:

- $w_{i,j}$ — the weight of edges inside the elementary rectangle $R_{i,j}$
- $h_{i,j}$ — the weight of edges between $R_{i,j}$ and $R_{i+1,j}$
- $v_{i,j}$ — the weight of edges between $R_{i,j}$ and $R_{i,j+1}$

K. Maximum spanning forest

Observation

At any moment, all edges inside an elementary rectangle have the same weight (if we consider only the heaviest of multiple edges), and all edges between points of two adjacent rectangles have the same weight.
Moreover, let a and b be the weights of edges inside two adjacent rectangles, and c be the weight of edges between these rectangles. Then, $c \leqslant \min(a, b)$, since in-between edges can lie inside a query rectangle only if both adjacent rectangles do.

Introduce the following arrays:

- $w_{i,j}$ — the weight of edges inside the elementary rectangle $R_{i,j}$
- $h_{i,j}$ — the weight of edges between $R_{i,j}$ and $R_{i+1,j}$
- $v_{i,j}$ — the weight of edges between $R_{i,j}$ and $R_{i,j+1}$

For each query update entries of arrays which correspond to edges lying inside the query rectangle. Each update takes $O(n^2)$ time.

K. Maximum spanning forest

Imagine that we're running Kruskal's algorithm on the graph: look through edges by decreasing of weight, add an edge if it doesn't create a cycle.

K. Maximum spanning forest

Imagine that we're running Kruskal's algorithm on the graph: look through edges by decreasing of weight, add an edge if it doesn't create a cycle.
It follows from the observation (the $c \leqslant min(a, b)$ part) that we can consider all the edges inside elementary rectangles first, and then consider in-between edges.

K. Maximum spanning forest

Imagine that we're running Kruskal's algorithm on the graph: look through edges by decreasing of weight, add an edge if it doesn't create a cycle.
It follows from the observation (the $c \leqslant min(a, b)$ part) that we can consider all the edges inside elementary rectangles first, and then consider in-between edges.
In every elementary rectangle all points become merged into a single component, for a total weight of (number of points $-$ 1) $\cdot\ w_{i,j}$.

K. Maximum spanning forest

Imagine that we're running Kruskal's algorithm on the graph: look through edges by decreasing of weight, add an edge if it doesn't create a cycle.

It follows from the observation (the $c \leqslant min(a, b)$ part) that we can consider all the edges inside elementary rectangles first, and then consider in-between edges.

In every elementary rectangle all points become merged into a single component, for a total weight of (number of points $-$ 1) $\cdot w_{i,j}$.

After that, we can consider each elementary rectangle a single vertex. Thus, adding in-between edges is reduced to building MST of a simple graph with $O(n^2)$ vertices and edges.

K. Maximum spanning forest

Imagine that we're running Kruskal's algorithm on the graph: look through edges by decreasing of weight, add an edge if it doesn't create a cycle.
It follows from the observation (the $c \leqslant min(a, b)$ part) that we can consider all the edges inside elementary rectangles first, and then consider in-between edges.
In every elementary rectangle all points become merged into a single component, for a total weight of (number of points $- 1) \cdot w_{i,j}$.
After that, we can consider each elementary rectangle a single vertex. Thus, adding in-between edges is reduced to building MST of a simple graph with $O(n^2)$ vertices and edges. That is, every query can be answered in $O(n^2 \log n)$ time, for a total $O(n^3 \log n)$ complexity solution.

L. House Building

Given a set of $1 \times 1 \times 1$ cubes on rectangular grid lying on the ground in several towers, determine the outer area of the construction.

L. House Building

Possible external faces are as follows:

L. House Building

Possible external faces are as follows:

- Top of a tower — the number of such faces is the number of towers with non-zero height.

L. House Building

Possible external faces are as follows:

- Top of a tower — the number of such faces is the number of towers with non-zero height.
- Side of a tower. Consider adjacent towers of heights a and b. The number of side faces lying in their common border plane is $|a - b|$.

L. House Building

Possible external faces are as follows:

- Top of a tower — the number of such faces is the number of towers with non-zero height.
- Side of a tower. Consider adjacent towers of heights a and b. The number of side faces lying in their common border plane is $|a - b|$. It's convenient to think that the construction is surrounded by towers of height 0.

L. House Building

Possible external faces are as follows:

- Top of a tower — the number of such faces is the number of towers with non-zero height.
- Side of a tower. Consider adjacent towers of heights a and b. The number of side faces lying in their common border plane is $|a - b|$. It's convenient to think that the construction is surrounded by towers of height 0.

Thus, just iterate over all adjacent pairs of towers, there is only linear ($O(nm)$) number of them. This solution is $O(nm)$.

M. Security Corporations

We are given a set of lines in the plane, no three of them share a point. Choose minimal number c and assign an index from $[1; c]$ to every intersection of two lines in such a way that every neighbouring intersections on the same line have different indices.

M. Security Corporations

First of all, how large c can be?

M. Security Corporations

First of all, how large c can be?

If no three lines form a triangle, then there are only two classes of parallel lines. The intersection graph in this case is essentially a grid (or a single point), so $c \leqslant 2$, and it is fairly easy to assign indices.

M. Security Corporations

First of all, how large c can be?

If no three lines form a triangle, then there are only two classes of parallel lines. The intersection graph in this case is essentially a grid (or a single point), so $c \leqslant 2$, and it is fairly easy to assign indices.

If there are three lines forming a triangle, then it's easy to show that some intersections form a triangle as well, so $c \geqslant 3$.

M. Security Corporations

Actually, $c = 3$ suffices in all cases. Build a 3-coloring of the intersection graph constructively:

M. Security Corporations

Actually, $c = 3$ suffices in all cases. Build a 3-coloring of the intersection graph constructively:

- Find all intersection points and sort them lexicographically, that is, by increasing of x, and in the case of equal x's, by increasing of y.

M. Security Corporations

Actually, $c = 3$ suffices in all cases. Build a 3-coloring of the intersection graph constructively:

- Find all intersection points and sort them lexicographically, that is, by increasing of x, and in the case of equal x's, by increasing of y. All the points on every line are thus ordered from one end to another.

M. Security Corporations

Actually, $c = 3$ suffices in all cases. Build a 3-coloring of the intersection graph constructively:

- Find all intersection points and sort them lexicographically, that is, by increasing of x, and in the case of equal x's, by increasing of y. All the points on every line are thus ordered from one end to another.
- Consider points one by one in sorted order. For the current point, find the lines it belongs to, and choose a different color from previous points on these lines.

M. Security Corporations

Actually, $c = 3$ suffices in all cases. Build a 3-coloring of the intersection graph constructively:

- Find all intersection points and sort them lexicographically, that is, by increasing of x, and in the case of equal x's, by increasing of y. All the points on every line are thus ordered from one end to another.
- Consider points one by one in sorted order. For the current point, find the lines it belongs to, and choose a different color from previous points on these lines.

It's easy to see that the algorithm constructs a correct 3-coloring. Moreover, it uses minimal number of colors in cases when $c < 3$.

M. Security Corporations

Actually, $c = 3$ suffices in all cases. Build a 3-coloring of the intersection graph constructively:

- Find all intersection points and sort them lexicographically, that is, by increasing of x, and in the case of equal x's, by increasing of y. All the points on every line are thus ordered from one end to another.
- Consider points one by one in sorted order. For the current point, find the lines it belongs to, and choose a different color from previous points on these lines.

It's easy to see that the algorithm constructs a correct 3-coloring. Moreover, it uses minimal number of colors in cases when $c < 3$. Its complexity is $O(n^2 \log n)$, the hardest part being sorting of points.