Educational Codeforces Round 44 (Rated for Div. 2)

A. Chess Placing

time limit per test: 1 second
memory limit per test: 256 megabytes
input: standard input
output: standard output

You are given a chessboard of size $1 \times n$. It is guaranteed that n **is even**. The chessboard is painted like this: "BWBW...BW".

Some cells of the board are occupied by the chess pieces. Each cell contains no more than one chess piece. It is known that the total number of pieces equals to $\frac{n}{2}$.

In one step you can move one of the pieces one cell to the left or to the right. You cannot move pieces beyond the borders of the board. You also cannot move pieces to the cells that are already occupied.

Your task is to place all the pieces in the cells of the same color **using the minimum number of moves** (all the pieces must occupy only the black cells or only the white cells after all the moves are made).

Input

The first line of the input contains one integer n ($2 \le n \le 100$, n **is even**) — the size of the chessboard.

The second line of the input contains $\frac{n}{2}$ integer numbers $p_1, p_2, \ldots, p_{\frac{n}{2}}$ ($1 \le p_i \le n$) — initial positions of the pieces. It is guaranteed that all the positions are distinct.

Output

Print one integer — **the minimum number of moves** you have to make to place all the pieces in the cells of the same color.

Examples

input
6 1 2 6
output
2

input
10 1 2 3 4 5
output
10

Note

In the first example the only possible strategy is to move the piece at the position 6 to the position 5 and move the piece at the position 2 to the position 3. Notice that if you decide to place the pieces in the white cells the minimum number of moves will be 3.

In the second example the possible strategy is to move $5 \to 9$ in 4 moves, then $4 \to 7$ in 3 moves, $3 \to 5$ in 2 moves and $2 \to 3$ in 1 move.

B. Switches and Lamps

time limit per test: 3 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

You are given n switches and m lamps. The i-th switch turns on some subset of the lamps. This information is given as the matrix a consisting of n rows and m columns where $a_{i,j} = 1$ if the i-th switch turns on the j-th lamp and $a_{i,j} = 0$ if the i-th switch is not connected to the j-th lamp.

Initially all m lamps are turned off.

Switches change state only from "off" to "on". It means that if you press two or more switches connected to the same lamp then the lamp will be turned on after any of this switches is pressed and will remain its state even if any switch connected to this lamp is pressed afterwards.

It is guaranteed that if you push all n switches then **all m lamps will be turned on**.

Your think that you have too many switches and you would like to ignore one of them.

Your task is to say if there exists such a switch that if you will ignore (not use) it but press all the other n - 1 switches then all the m lamps will be turned on.

Input
The first line of the input contains two integers n and m ($1 \le n, m \le 2000$) — the number of the switches and the number of the lamps.

The following n lines contain m characters each. The character $a_{i,j}$ is equal to '1' if the i-th switch turns on the j-th lamp and '0' otherwise.

It is guaranteed that if you press all n switches **all m lamps will be turned on**.

Output
Print "YES" if there is a switch that if you will ignore it and press all the other n - 1 switches then all m lamps will be turned on. Print "NO" if there is no such switch.

Examples

input
4 5 10101 01000 00111 10000
output
YES

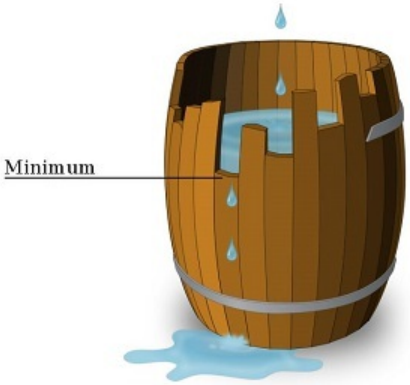
input
4 5 10100 01000 00110 00101
output
NO

C. Liebig's Barrels

time limit per test: 2 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

You have $m = n \cdot k$ wooden staves. The i-th stave has length a_i. You have to assemble n barrels consisting of k staves each, you can use any k staves to construct a barrel. Each stave must belong to exactly one barrel.

Let volume v_j of barrel j be equal to the length of the **minimal** stave in it.



You want to assemble exactly n barrels with the maximal total sum of volumes. But you have to make them *equal enough*, so a difference between volumes of any pair of the resulting barrels must not exceed l, i.e. $|v_x - v_y| \le l$ for any $1 \le x \le n$ and $1 \le y \le n$.

Print maximal total sum of volumes of *equal enough* barrels or 0 if it's impossible to satisfy the condition above.

Input
The first line contains three space-separated integers n, k and l ($1 \le n, k \le 10^5$, $1 \le n \cdot k \le 10^5$, $0 \le l \le 10^9$).

The second line contains $m = n \cdot k$ space-separated integers $a_1, a_2, \ldots, a_m$ ($1 \le a_i \le 10^9$) — lengths of staves.

Output
Print single integer — maximal total sum of the volumes of barrels or 0 if it's impossible to construct exactly n barrels satisfying the condition $|v_x - v_y| \le l$ for any $1 \le x \le n$ and $1 \le y \le n$.

Examples

input
4 2 1 2 2 1 2 3 2 2 3
output
7

input
2 1 0 10 10
output
20

input
1 2 1 5 2
output
2

input
3 2 1 1 2 3 4 5 6
output
0

Note

In the first example you can form the following barrels: $[1, 2]$, $[2, 2]$, $[2, 3]$, $[2, 3]$.

In the second example you can form the following barrels: $[10]$, $[10]$.

In the third example you can form the following barrels: $[2, 5]$.

In the fourth example difference between volumes of barrels in any partition is at least 2 so it is impossible to make barrels equal enough.

D. Sand Fortress

time limit per test: 2 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

You are going to the beach with the idea to build the greatest sand castle ever in your head! The beach is not as three-dimensional as you could have imagined, it can be decribed as a line of spots to pile up sand pillars. Spots are numbered 1 through infinity from left to right.

Obviously, there is not enough sand on the beach, so you brought n packs of sand with you. Let height h_i of the sand pillar on some spot i be the number of sand packs you spent on it. **You can't split a sand pack to multiple pillars, all the sand from it should go to a single one.** There is a fence of height equal to the height of pillar with H sand packs to the left of the first spot and you should prevent sand from going over it.

Finally you ended up with the following conditions to building the castle:

- $h_1 \le H$: no sand from the leftmost spot should go over the fence;
- For any $i \in [1; \infty) |h_i - h_{i+1}| \le 1$: large difference in heights of two neighboring pillars can lead sand to fall down from the higher one to the lower, you really don't want this to happen;
- $\sum_{i=1}^{\infty} h_i = n$: you want to spend all the sand you brought with you.

As you have infinite spots to build, it is always possible to come up with some valid castle structure. Though you want the castle to be as compact as possible.

Your task is to calculate the minimum number of spots you can occupy so that all the aforementioned conditions hold.

Input

The only line contains two integer numbers n and H ($1 \le n, H \le 10^{18}$) — the number of sand packs you have and the height of the fence, respectively.

Output

Print the minimum number of spots you can occupy so the all the castle building conditions hold.

Examples

input
5 2
output
3

input
6 8
output
3

Note

Here are the heights of some valid castles:

- $n = 5, H = 2$, [2, 2, 1, 0, ...], [2, 1, 1, 1, 0, ...], [1, 0, 1, 2, 1, 0, ...]
- $n = 6, H = 8$, [3, 2, 1, 0, ...], [2, 2, 1, 1, 0, ...], [0, 1, 0, 1, 2, 1, 1, 0...] (this one has 5 spots occupied)

The first list for both cases is the optimal answer, 3 spots are occupied in them.

And here are some invalid ones:

- $n = 5, H = 2$, [3, 2, 0, ...], [2, 3, 0, ...], [1, 0, 2, 2, ...]
- $n = 6, H = 8$, [2, 2, 2, 0, ...], [6, 0, ...], [1, 4, 1, 0...], [2, 2, 1, 0, ...]

E. Pencils and Boxes

time limit per test: 2 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

Mishka received a gift of multicolored pencils for his birthday! Unfortunately he lives in a monochrome world, where everything is of the same color and only saturation differs. This pack can be represented as a sequence $a_1, a_2, \ldots, a_n$ of n integer numbers — saturation of the color of each pencil. Now Mishka wants to put all the mess in the pack in order. He has an infinite number of empty boxes to do this. He would like to fill some boxes in such a way that:

- Each pencil belongs to **exactly** one box;
- Each non-empty box has at least k pencils in it;
- If pencils i and j belong to the same box, then $|a_i - a_j| \le d$, where $|x|$ means absolute value of x. Note that the opposite is optional, there can be pencils i and j such that $|a_i - a_j| \le d$ and they belong to different boxes.

Help Mishka to determine if it's possible to distribute all the pencils into boxes. Print "YES" if there exists such a distribution. Otherwise print "NO".

Input

The first line contains three integer numbers n, k and d ($1 \le k \le n \le 5 \cdot 10^5$, $0 \le d \le 10^9$) — the number of pencils, minimal size of any non-empty box and maximal difference in saturation between any pair of pencils in the same box, respectively.

The second line contains n integer numbers $a_1, a_2, \ldots, a_n$ ($1 \le a_i \le 10^9$) — saturation of color of each pencil.

Output

Print "YES" if it's possible to distribute all the pencils into boxes and satisfy all the conditions. Otherwise print "NO".

Examples

input
6 3 10 7 2 7 7 4 2
output
YES

input
6 2 3 4 5 3 13 4 10
output
YES

input
3 2 5 10 16 22
output

```
NO
```

Note

In the first example it is possible to distribute pencils into 2 boxes with 3 pencils in each with any distribution. And you also can put all the pencils into the same box, difference of any pair in it won't exceed 10.

In the second example you can split pencils of saturations $[4, 5, 3, 4]$ into 2 boxes of size 2 and put the remaining ones into another box.

F. Isomorphic Strings

time limit per test: 3 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

You are given a string s of length n consisting of lowercase English letters.

For two given strings s and t, say S is the set of distinct characters of s and T is the set of distinct characters of t. The strings s and t are *isomorphic* if their lengths are equal and there is a one-to-one mapping (bijection) f between S and T for which $f(s_i) = t_i$. Formally:

1. $f(s_i) = t_i$ for any index i,
2. for any character $x \in S$ there is exactly one character $y \in T$ that $f(x) = y$,
3. for any character $y \in T$ there is exactly one character $x \in S$ that $f(x) = y$.

For example, the strings "`aababc`" and "`bbcbcz`" are isomorphic. Also the strings "`aaaww`" and "`wwwaa`" are isomorphic. The following pairs of strings are not isomorphic: "`aab`" and "`bbb`", "`test`" and "`best`".

You have to handle m queries characterized by three integers x, y, len ($1 \le x, y \le n - len + 1$). For each query check if two substrings $s[x \dots x + len - 1]$ and $s[y \dots y + len - 1]$ are isomorphic.

Input

The first line contains two space-separated integers n and m ($1 \le n \le 2 \cdot 10^5$, $1 \le m \le 2 \cdot 10^5$) — the length of the string s and the number of queries.

The second line contains string s consisting of n lowercase English letters.

The following m lines contain a single query on each line: x_i, y_i and len_i ($1 \le x_i, y_i \le n$, $1 \le len_i \le n - max(x_i, y_i) + 1$) — the description of the pair of the substrings to check.

Output

For each query in a separate line print "`YES`" if substrings $s[x_i \dots x_i + len_i - 1]$ and $s[y_i \dots y_i + len_i - 1]$ are isomorphic and "`NO`" otherwise.

Example

input
7 4 abacaba 1 1 1 1 4 2 2 1 3 2 4 3
output
YES YES NO YES

Note

The queries in the example are following:

1. substrings "`a`" and "`a`" are isomorphic: $f(a) = a$;
2. substrings "`ab`" and "`ca`" are isomorphic: $f(a) = c, f(b) = a$;
3. substrings "`bac`" and "`aba`" are not isomorphic since $f(b)$ and $f(c)$ must be equal to a at same time;
4. substrings "`bac`" and "`cab`" are isomorphic: $f(b) = c, f(a) = a, f(c) = b$.

G. Team Players

time limit per test: 2 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

There are $$$n$$$ players numbered from $$$0$$$ to $$$n-1$$$ with ranks. The $$$i$$$-th player has rank $$$i$$$.

Players can form teams: the team should consist of three players and **no pair** of players in the team should have a conflict. The rank of the team is calculated using the following algorithm: let $$$i$$$, $$$j$$$, $$$k$$$ be the ranks of players in the team and $$$i < j < k$$$, then the rank of the team is equal to $$$A \cdot i + B \cdot j + C \cdot k$$$.

You are given information about the pairs of players who **have** a conflict. Calculate the total sum of ranks over all possible valid teams modulo $$$2^{64}$$$.

Input

The first line contains two space-separated integers $$$n$$$ and $$$m$$$ ($$$3 \le n \le 2 \cdot 10^5$$$, $$$0 \le m \le 2 \cdot 10^5$$$) — the number of players and the number of conflicting pairs.

The second line contains three space-separated integers $$$A$$$, $$$B$$$ and $$$C$$$ ($$$1 \le A, B, C \le 10^6$$$) — coefficients for team rank calculation.

Each of the next $$$m$$$ lines contains two space-separated integers $$$u_i$$$ and $$$v_i$$$ ($$$0 \le u_i, v_i < n, u_i \neq v_i$$$) — pair of conflicting players.

It's guaranteed that each unordered pair of players appears in the input file no more than once.

Output

Print single integer — the total sum of ranks over all possible teams modulo $$$2^{64}$$$.

Examples

input
4 0 2 3 4
output
64

input
4 1 2 3 4 1 0
output
38

input
6 4 1 5 3 0 3 3 5 5 4 4 3
output
164

Note

In the first example all $$$4$$$ teams are valid, i.e. triples: {0, 1, 2}, {0, 1, 3}, {0, 2, 3} {1, 2, 3}.

In the second example teams are following: {0, 2, 3}, {1, 2, 3}.

In the third example teams are following: {0, 1, 2}, {0, 1, 4}, {0, 1, 5}, {0, 2, 4}, {0, 2, 5}, {1, 2, 3}, {1, 2, 4}, {1, 2, 5}.