Codeforces Round #577 (Div. 2)

A. Important Exam

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

A class of students wrote a multiple-choice test.

There are n students in the class. The test had m questions, each of them had 5 possible answers (A, B, C, D or E). There is exactly one correct answer for each question. The correct answer for question i worth a_i points. Incorrect answers are graded with zero points.

The students remember what answers they gave on the exam, but they don't know what are the correct answers. They are very optimistic, so they want to know what is the maximum possible total score of all students in the class.

Input

The first line contains integers n and m ($1 \le n, m \le 1000$) — the number of students in the class and the number of questions in the test.

Each of the next n lines contains string s_i ($|s_i| = m$), describing an answer of the i-th student. The j-th character represents the student answer (A, B, C, D or E) on the j-th question.

The last line contains m integers $a_1, a_2, \ldots, a_m$ ($1 \le a_i \le 1000$) — the number of points for the correct answer for every question.

Output

Print a single integer — the maximum possible total score of the class.

Examples

input
2 4 ABCD ABCE 1 2 3 4
output
16

input
3 3 ABC BCD CDE 5 4 12
output
21

Note

In the first example, one of the most optimal test answers is "ABCD", this way the total number of points will be 16.

In the second example, one of the most optimal test answers is "CCC", this way each question will be answered by exactly one student and the total number of points is $5 + 4 + 12 = 21$.

B. Zero Array

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

You are given an array $a_1, a_2, \ldots, a_n$.

In one operation you can choose two elements a_i and a_j ($i \neq j$) and decrease each of them by one.

You need to check whether it is possible to make all the elements equal to zero or not.

Input

The first line contains a single integer n ($2 \le n \le 10^5$) — the size of the array.

The second line contains n integers $a_1, a_2, \ldots, a_n$ ($1 \le a_i \le 10^9$) — the elements of the array.

Output
Print "YES" if it is possible to make all elements zero, otherwise print "NO".

Examples

input
4 1 1 2 2
output
YES

input
6 1 2 3 4 5 6
output
NO

Note
In the first example, you can make all elements equal to zero in 3 operations:

- Decrease a_1 and a_2,
- Decrease a_3 and a_4,
- Decrease a_3 and a_4

In the second example, one can show that it is impossible to make all elements equal to zero.

C. Maximum Median

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

You are given an array a of n integers, where n is odd. You can make the following operation with it:

- Choose one of the elements of the array (for example a_i) and increase it by 1 (that is, replace it with $a_i + 1$).

You want to make the median of the array the largest possible using at most k operations.

The median of the odd-sized array is the middle element after the array is sorted in non-decreasing order. For example, the median of the array $[1, 5, 2, 3, 5]$ is 3.

Input
The first line contains two integers n and k ($1 \le n \le 2 \cdot 10^5$, n is odd, $1 \le k \le 10^9$) — the number of elements in the array and the largest number of operations you can make.

The second line contains n integers $a_1, a_2, \ldots, a_n$ ($1 \le a_i \le 10^9$).

Output
Print a single integer — the maximum possible median after the operations.

Examples

input
3 2 1 3 5
output
5

input
5 5 1 2 1 1 1
output
3

input
7 7 4 1 2 4 3 4 4
output
5

Note
In the first example, you can increase the second element twice. Than array will be $[1, 5, 5]$ and it's median is 5.

In the second example, it is optimal to increase the second number and than increase third and fifth. This way the answer is 3.

In the third example, you can make four operations: increase first, fourth, sixth, seventh element. This way the array will be $[5, 1, 2, 5, 3, 5, 5]$ and the median will be 5.

D. Treasure Hunting

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

You are on the island which can be represented as a $n \times m$ table. The rows are numbered from 1 to n and the columns are numbered from 1 to m. There are k treasures on the island, the i-th of them is located at the position (r_i, c_i).

Initially you stand at the lower left corner of the island, at the position $(1, 1)$. If at any moment you are at the cell with a treasure, you can pick it up without any extra time. In one move you can move up (from (r, c) to $(r + 1, c)$), left (from (r, c) to $(r, c - 1)$), or right (from position (r, c) to $(r, c + 1)$). Because of the traps, you can't move down.

However, moving up is also risky. You can move up only if you are in a safe column. There are q safe columns: $b_1, b_2, \ldots, b_q$. You want to collect all the treasures as fast as possible. Count the minimum number of moves required to collect all the treasures.

Input
The first line contains integers n, m, k and q ($2 \leq n, m, k, q \leq 2 \cdot 10^5$, $q \leq m$) — the number of rows, the number of columns, the number of treasures in the island and the number of safe columns.

Each of the next k lines contains two integers r_i, c_i, ($1 \leq r_i \leq n$, $1 \leq c_i \leq m$) — the coordinates of the cell with a treasure. All treasures are located in distinct cells.

The last line contains q distinct integers $b_1, b_2, \ldots, b_q$ ($1 \leq b_i \leq m$) — the indices of safe columns.

Output
Print the minimum number of moves required to collect all the treasures.

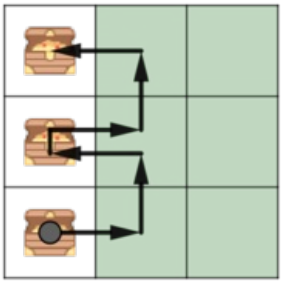
Examples

input
3 3 3 2 1 1 2 1 3 1 2 3
output
6

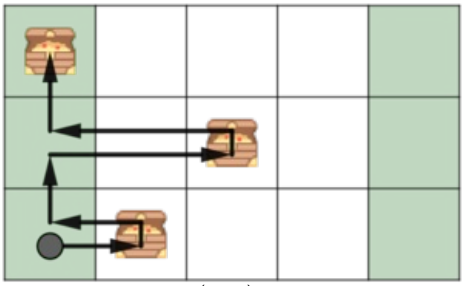
input
3 5 3 2 1 2 2 3 3 1 1 5
output
8

input
3 6 3 2 1 6 2 2 3 4 1 6
output
15

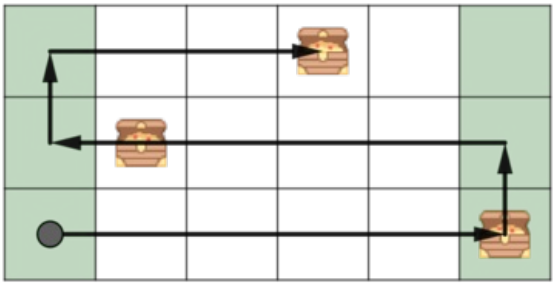
Note
In the first example you should use the second column to go up, collecting in each row treasures from the first column.



In the second example, it is optimal to use the first column to go up.



In the third example, it is optimal to collect the treasure at cell $(1;6)$, go up to row 2 at column 6, then collect the treasure at cell $(2;2)$, go up to the top row at column 1 and collect the last treasure at cell $(3;4)$. That's a total of 15 moves.



E1. Knightmare (easy)

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

This problem only differs from the next problem in constraints.

This is an interactive problem.

Alice and Bob are playing a game on the chessboard of size $n \times m$ where n and m are **even**. The rows are numbered from 1 to n and the columns are numbered from 1 to m. There are two knights on the chessboard. A white one initially is on the position (x_1, y_1), while the black one is on the position (x_2, y_2). Alice will choose one of the knights to play with, and Bob will use the other one.

The Alice and Bob will play in turns and whoever controls **the white** knight starts the game. During a turn, the player must move their knight adhering the chess rules. That is, if the knight is currently on the position (x, y), it can be moved to any of those positions (as long as they are inside the chessboard):

$$(x+1, y+2), (x+1, y-2), (x-1, y+2), (x-1, y-2),$$
$$(x+2, y+1), (x+2, y-1), (x-2, y+1), (x-2, y-1).$$

We all know that knights are strongest in the middle of the board. Both knight have a single position they want to reach:

- the owner of the white knight wins if it captures the black knight or if the white knight is at $(n/2, m/2)$ and this position is not under attack of the black knight at this moment;
- The owner of the black knight wins if it captures the white knight or if the black knight is at $(n/2+1, m/2)$ and this position is not under attack of the white knight at this moment.

Formally, the player who captures the other knight wins. The player who is at its target square ($(n/2, m/2)$ for white, $(n/2+1, m/2)$ for black) and this position is not under opponent's attack, also wins.

A position is under attack of a knight if it can move into this position. Capturing a knight means that a player moves their knight to the cell where the opponent's knight is.

If Alice made 350 moves and nobody won, the game is a draw.

Alice is unsure in her chess skills, so she asks you for a help. Choose a knight and win the game for her. It can be shown, that Alice always has a winning strategy.

Interaction

The interaction starts with two integers n and m ($6 \le n, m \le 40$, n and m are even) — the dimensions of the chessboard.

The second line contains four integers x_1, y_1, x_2, y_2 ($1 \le x_1, x_2 \le n$, $1 \le y_1, y_2 \le m$) — the positions of the white and the black knight. It is guaranteed that the two knights have different starting positions. It is also guaranteed that none of the knights are in their own target square in the beginning of the game (however, they can be on the opponent's target position).

Your program should reply with either "`WHITE`" or "`BLACK`", depending on the knight you want to play with. In case you select the white knight, you start the game.

During every your turn, you need to print two integers: x and y, the position to move the knight. If you won the game by this turn, you must terminate your program immediately.

After every turn of the opponent, you will receive two integers: x and y, the position where Bob moved his knight.

If your last move was illegal or you lost the game after jury's turn, or you made 350 moves, and haven't won, you will receive "`-1 -1`". In such cases, you should terminate your program and then you will get a `Wrong Answer` verdict.

After printing anything, do not forget to output the end of line and flush the output. Otherwise, you might get `Idleness limit exceeded`. To do this, use:

- `fflush(stdout)` or `cout.flush()` in C++;
- `System.out.flush()` in Java;
- `flush(output)` in Pascal;
- `stdout.flush()` in Python;
- see documentation for other languages.

Hacks are disabled for this problem.

Jury's program is *adaptive*: the moves of jury may depend on the moves made by your program.

Examples

input
8 8 2 3 1 8
output
WHITE 4 4

input
6 6 4 4 2 2 6 3
output
BLACK 4 3

Note

In the first example, the white knight can reach it's target square in one move.

In the second example black knight wins, no matter what white knight moves.

E2. Knightmare (hard)

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

This is an interactive problem.

Alice and Bob are playing a game on the chessboard of size $n \times m$ where n and m are **even**. The rows are numbered from 1 to n and the columns are numbered from 1 to m. There are two knights on the chessboard. A white one initially is on the position (x_1, y_1), while the black one is on the position (x_2, y_2). Alice will choose one of the knights to play with, and Bob will use the other one.

The Alice and Bob will play in turns and whoever controls **the white** knight starts the game. During a turn, the player must move their knight adhering the chess rules. That is, if the knight is currently on the position (x, y), it can be moved to any of those positions (as long as they are inside the chessboard):

$$(x+1, y+2), (x+1, y-2), (x-1, y+2), (x-1, y-2),$$
$$(x+2, y+1), (x+2, y-1), (x-2, y+1), (x-2, y-1).$$

We all know that knights are strongest in the middle of the board. Both knight have a single position they want to reach:

- the owner of the white knight wins if it captures the black knight or if the white knight is at $(n/2, m/2)$ and this position is not under attack of the black knight at this moment;
- The owner of the black knight wins if it captures the white knight or if the black knight is at $(n/2+1, m/2)$ and this position is not under attack of the white knight at this moment.

Formally, the player who captures the other knight wins. The player who is at its target square ($(n/2, m/2)$ for white, $(n/2+1, m/2)$ for black) and this position is not under opponent's attack, also wins.

A position is under attack of a knight if it can move into this position. Capturing a knight means that a player moves their knight to the cell where the opponent's knight is.

If Alice made 350 moves and nobody won, the game is a draw.

Alice is unsure in her chess skills, so she asks you for a help. Choose a knight and win the game for her. It can be shown, that Alice always has a winning strategy.

Interaction

The interaction starts with two integers n and m ($6 \le n, m \le 1000$, n and m are even) — the dimensions of the chessboard.

The second line contains four integers x_1, y_1, x_2, y_2 ($1 \le x_1, x_2 \le n$, $1 \le y_1, y_2 \le m$) — the positions of the white and the black knight. It is guaranteed that the two knights have different starting positions. It is also guaranteed that none of the knights are in their own target square in the beginning of the game (however, they can be on the opponent's target position).

Your program should reply with either "`WHITE`" or "`BLACK`", depending on the knight you want to play with. In case you select the white knight, you start the game.

During every your turn, you need to print two integers: x and y, the position to move the knight. If you won the game by this turn, you must terminate your program immediately.

After every turn of the opponent, you will receive two integers: x and y, the position where Bob moved his knight.

If your last move was illegal or you lost the game after jury's turn, or you made 350 moves, and haven't won, you will receive "`-1 -1`". In such cases, you should terminate your program and then you will get a `Wrong Answer` verdict.

After printing anything, do not forget to output the end of line and flush the output. Otherwise, you might get `Idleness limit exceeded`. To do this, use:

- `fflush(stdout)` or `cout.flush()` in C++;
- `System.out.flush()` in Java;
- `flush(output)` in Pascal;
- `stdout.flush()` in Python;
- see documentation for other languages.

Hacks are disabled for this problem.

Jury's program is *adaptive*: the moves of jury may depend on the moves made by your program.

Examples

input
8 8 2 3 1 8
output
WHITE 4 4

input
6 6 4 4 2 2 6 3
output
BLACK 4 3

Note

In the first example, the white knight can reach it's target square in one move.

In the second example black knight wins, no matter what white knight moves.