4. Grid Teleportation

A grid of size *n x m* is to be traversed starting from the top-left cell to the bottom-right cell. In a single step, it is possible to move either to the cell to the right of the current cell or to the down of the current cell i.e. from the cell *(i, j)*, it is possible to move to the cell *(i + 1, j)* or *(i, j + 1)*.

The grid has *k* teleporters represented by the 2-d array *teleporters* of size *k x 5* where the i^{th} teleporter allows teleportation from the cell *(teleporters[i][0], teleporters[i][1])* to *(teleporters[i][2], teleporters[i][3])* but can be used only if the parity of the number of steps traveled so far is the same as that of *teleporters[i][4]*. More formally, *teleporter[i]* can only be used if *teleporters[i][4]* is 0 and the number of steps traveled so far is even or *teleporters[i][4]* is 1 and the number of steps is odd. Note that using a teleporter also counts as a step.

Given *n, m,* and *teleporters,* find the minimum number of steps required to travel from the top-left cell to the bottom-right cell i.e. from (0, 0) to *(n - 1, m - 1).*

Example
Suppose *n* = 5, *m* = 5, *k* = 2 and *teleporters* = [[0, 0, 4, 4, 1], [0, 0, 3, 3, 0]]

The parity of *teleporters[0] = teleporters[0][4] = 1,* and of *teleporters[1] = 0.* Both have their origins at position (0, 0). After 0 moves, the parity of the moves is 0 so *teleporters[1]* can be used to travel to (3, 3) in 1 move. The cell at (4, 4) is reached in 2 more moves, for example, (3, 3) -> (3, 4) -> (4, 4).

Return the minimum number of total steps, 3.

Function Description
Complete the function *getMinSteps* in the editor below.

getMinSteps has the following parameters:

int n: the number of rows in the grid
int m: the number of columns in the grid
int teleporters[k][5]: the teleporters present in the grid

Returns
int: the minimum number of steps required to start from the top-left cell to reach the bottom-right cell

Constraints

- $2 \le n \times m \le 10^5$
- $1 \le k \le 10^5$
- $0 \le$ ***teleporters[i][0], teleporters[i][2] < n, where 0 ≤ i < k***
- $0 \le$ ***teleporters[i][1], teleporters[i][3] < m, where 0 ≤ i < k***
- ***teleporters[i][4] = 0 or teleporters[i][4] = 1***

▾ Input Format For Custom Testing

The first line contains an integer, *n*, which denotes the number of rows in the grid.
The second line contains an integer, *m*, which denotes the number of columns in the grid.
The third line contains an integer, *k*, which denotes the number of *teleporters*.
The fourth line always contains an integer, *5*, which denotes the number of elements in *teleporters[i]*.
Each line *i* of the *k* subsequent lines (where *0 ≤ i < k*) contains five space-separated integers that describe a teleporter.

▾ Sample Case 0

Sample Input For Custom Testing

```
STDIN          FUNCTION
-----          --------
5         →    n = 5
5         →    m = 5
2         →    teleporters[] size k = 2
5         →    teleporters[][] size const = 5
0 0 2 2 0 →    teleporters = [[0, 0, 2, 2, 0], [2, 2, 4, 4, 1]]
2 2 4 4 1
```

Sample Output

```
2
```

Explanation
The first teleporter can be used to reach (2, 2) in 1 step. Since the number of steps so far is odd and the position is (2, 2), the second teleporter can be used to reach the destination cell in another step.

▾ Sample Case 1

Sample Input For Custom Testing

```
STDIN          FUNCTION
-----          --------
5         →    n = 5
5         →    m = 5
2         →    teleporters[] size k = 2
5         →    teleporters[][] size const = 5
0 0 2 2 0 →    teleporters = [[0, 0, 2, 2, 0], [1, 1, 4, 4, 0]]
1 1 4 4 0
```

Sample Output

```
3
```

Explanation
The optimal strategy is to go to (1, 0) and then to (1, 1) in 2 steps. Now the second teleporter can be used to reach the destination cell in another step.