**Binary Maze**

Given an R x C matrix where each element can either be 0 or 1, find the shortest path between a given source node to a destination node. The path can only be created out of a node if its value is 1.

**Input Format**

The first line consists of three integers R, C, and T. R and C denote the number of rows and columns of the matrix. T denotes the number of test cases for this maze.

The next R lines contain the rows and columns of the binary maze. The remaining lines contain the test case data. The first pair of integers represent the source node; the second pair represent the destination node.

**Output Format**

Print a single integer for each test case denoting the shortest path to the destination, or -1 if no path exists.

**Note:** A breadth-first search (BFS) of an "unweighted" graph (a graph where there is no cost to travel from one vertex to another) will return the shortest path, due to the expanding radial nature of the search. However, if the graph was "weighted" (there *was* a cost to travel from one node to another), a BFS would not (necessarily) find the shortest path. Finding the shortest path of a weighted graph requires ***Dijkstra's algorithm,*** which preferentially chooses edges with lower costs. You'll learn about Dijkstra's algorithm soon!

**Hints:**

* Due to the simple nature of this problem, the graph can be easily modeled with an adjacency matrix of integers. You *could* represent it with an adjacency list... but why?
* It may help to make a Location class, such that the queue will store Location objects
* You need some way of tracking which nodes have been visited, such that you don't process nodes more than once.
* The number of steps from one node to the next node is one more than the number of steps it took to reach the previous node.

**SAMPLE OUTPUT**

11

8

4

-1

**SAMPLE OUTPUT**

9 10 4  
**1** 0 **1** **1** **1** 1 0 1 1 1   
**1** 0 **1** 0 **1** 1 1 1 0 1  
**1** **1** **1** 0 **1** 1 0 1 1 1  
0 0 0 0 **1** 0 0 0 0 1   
1 1 1 0 1 1 1 0 1 0   
1 0 1 1 1 1 0 1 0 0   
1 0 0 0 0 1 0 0 0 1  
1 0 1 1 1 1 0 1 1 1   
1 1 1 0 0 0 1 0 0 1  
0 0 3 4  
4 6 8 2  
0 9 1 6  
8 9 8 6

//the shortest path is bolded above for the first test case, purely for demonstration purposes