**QPREP5-Message passing**

**Module Introduction**

Write a program to find if the message from a friend standing at one end of the spiral tunnel can reach another friend at the other end.

#### Objective

Two friends stand at the 2 ends of a spiral matrix, with additional people in some cells in the middle. Each person can speak with a specific strength which indicates the maximum number of cells their voice can reach. Confirm if the friend at the beginning can pass a message to his friend at the end. The others in the middle who hear the message can repeat the message.

The message can be heard by multiple people in the middle if they are within reach of the previous person’s strength. More than one person can be in the same cell in which case you can pick the person with maximum reach.

Note that voice can only travel in the sequence of the spiral matrix.

#### Examples

**Example 1**

Input Format

First line contains N, S and M, which are the size of the square maze(N\*N), strength of the first friend’s voice and the number of people standing in the maze. Next M lines contain the position of each person in the maze and their voice strength.

1 ≤ N ≤ 103

1 ≤ M ≤ 105

Output Format

Print 'Yes' if the message could reach from the friend at the beginning to the friend at the end, else print 'No' (case-sensitive).

Input 1

4 5 4 --> Size is 4 \* 4, first friend’s strength is 5, number of people is 4

0 3 2 --> First person’s position is 0,3 in the matrix. Their strength is 2

2 3 5

3 1 4

1 1 5

Output 1

Yes

Explanation 1

Y is the friend at the beginning. Z is the friend at the end. P are the other people.

0 1 2 3

0 Y(5) X X P1(3)

1 X P4(5) X X

2 X Z X P2(5)

3 X P3(4) X X

When Y speaks, voice will travel 5 cells through the following path and reach person1 and person2.

(0, 0)Y -> (0, 1) -> (0, 2) -> (0, 3)P1 -> (1, 3)

(2, 3)P2 -> (3, 3) -> (3, 2) -> (3, 1)P3 -> . . .

Following the spiral path, the message will be passed from P3 to P4 and then finally to Z.

***SOLUTION STEPS FROM NEXT PAGE:***

**Write down at least 3 examples in the following format. Kindly stick to the format.**

**Suggestion:**

EXAMPLE#1

INPUT:

3 4 3 --> Size is 3 \* 3, first friend’s strength is 4, number of people is 3

0 1 2 --> First person’s position is 0,1 in the matrix. Their strength is 2

2 2 2

2 0 2

OUTPUT:

Yes

EXAMPLE#2

INPUT:

3 4 3 --> Size is 3 \* 3, first friend’s strength is 4, number of people is 3

0 1 2 --> First person’s position is 0,1 in the matrix. Their strength is 2

2 2 1

2 0 2

OUTPUT:

No

EXAMPLE#3

INPUT:

4 3 4 --> Size is 4 \* 4, first friend’s strength is 3, number of people is 4

0 1 2 --> First person’s position is 0,1 in the matrix. Their strength is 2

0 2 8

1 2 3

3 0 5

OUTPUT:

Yes

**Detail your problem understanding here**

**Suggestion:**

The matrix needs to be treated as a spiral. Create a single dimensional array with the unraveled spiral. Once the array is created we can traverse through it to find which people are reachable and if the those people in turn can reach other people and so on till the end.

**Does this problem follow a known algorithmic pattern or standard application of a data structure? If there are multiple approaches, which one would you choose and why? Write down your chosen approach in 2-3 sentences like you would explain to a 10 year old.**

Problem is a combination of the known spiral matrix problem and adding some logic on top of it.

With spiral matrix solution, we can reduce the given input to a linear array. Searching the array to see if the last index can be reached from the first index using the intermediate indices can be solved using Backtracking or Dynamic programing or Greedy approach. Greedy approach is pretty intuitive and works well in this case.

**Write the pseudocode here in plain English**

Read the inputs.

Form a matrix where each node represents the maximum voice strength of the person available there or has 0 otherwise

peopleArray = spiralMatrix(inputPeopleMatrix)

This array (peopleArray) will contain the strength of the person at that cell if there is a person in that cell, else you can assume the person has 0 voice strength.

So the example problem

0 1 2 3

0 5 0 0 3

1 0 5 0 0

2 0 Z 0 5

3 0 4 0 0

results in the following peopleArray

5 0 0 3 0 5 0 0 4 0 0 0 5 0 0 Z

currentVoiceStrength = strength of first friend = 5

Loop through peopleArray till currentVoiceStrength > 0 {

currentVoiceStrength--;

Pick the maximum voice available at this position maxVoice(currentVoiceStrength, voiceStrengthOfCurrentPerson)

}

If we are able to reach Z, return Yes, else return No

**Can you specify a few boundary or edge cases here?**

**Edge cases**

EXAMPLE#1

INPUT:

2 4 0 --> Size is 2 \* 2, first friend’s strength is 4, number of people is 0

OUTPUT:

Yes

EXAMPLE#2

INPUT:

2 2 0 --> Size is 2 \* 2, first friend’s strength is 2, number of people is 0

OUTPUT:

No

EXAMPLE#3 - 2 people in same position, pick one with higher strength

INPUT:

3 3 2 --> Size is 3 \* 3, first friend’s strength is 3, number of people is 2

1 2 4 --> First person’s position is 1,2 in the matrix. Their strength is 4

1 2 5

OUTPUT:

Yes

**Write the functions you would create here**

createMatrixWithVoiceStrength() - Helps convert the given input into a voice strength matrix.

spiralMatrix() from previous module.

**Summary**

Starting with a brief explanation of the problem statement followed by pseudocode and then implementing the solution helps you approach the problem in a systematic way. This methodology helps with easy as well as hard problems.

**Time Complexity: O(N \* N) or O(N \* N)^2**

Where the matrix is of size N \* N. We need to traverse through every cell in the spiral which is O(N\*N).

Once we have the traversal array, we need to confirm if each of the following people in the array are reachable. This is O (N*N)^2. This can be reduced into O(N*N) if we use the greedy method with one pass.

**Space Complexity: O(N \* N)**

Additional array is used to store the unraveled spiral, so it can be searched easily.

**Concepts**

Concepts covered in this Module

* 2D Array
* Matrix
* Backtracking or Greedy method

Similar problems

* <https://leetcode.com/problems/jump-game/>

References

* <https://www.geeksforgeeks.org/solve-dynamic-programming-problem/>
* <https://www.geeksforgeeks.org/greedy-algorithms/>

**Good habits**

Think about these for your solution:

* Comments - have you used comments in a way that others can understand this code?
* Test Cases - Are most of the scenarios/corner cases/boundary conditions handled in the solution?
* Naming Convention - Are the variables and functions named sensibly and with uniform convention?
* Modular Functions - Has the solution been addressed using concise functions? Will these functions work without any changes if they are to be used in another problem?
* Optimization - Analyze the Time Complexity and Space Complexity for your solution. Has the solution been optimized or did it use the brute force method? Is further optimization desirable/possible?
* Data Structures - Has the optimal/appropriate data structure been used?

#### How hard was this problem?

If you had attempted solving this problem directly, there is a good chance that you would have struggled badly. Why? As the complexity of an interview problem increases, cognitive load increases as well. At some point, it becomes overwhelming and you stop operating at maximum efficiency. Note that this threshold may be different for different people.

Real world problems, unlike interview problems are much larger in scope and require days of effort to solve. And it is close to impossible to approach them like a 1 hr interview problem where you jump into coding. You need to divide them into multiple sub-problems, solve each part individually before assembling them back together. This is exactly what we have done here!

Consciously dividing a problem however simple into smaller parts till it becomes brain-dead and then solving them is extremely effective during interviews. Common practice among junior developers is to skip this step and jump directly into coding. How many times have you started writing down a for loop without even thinking what the problem demands? You might sometimes solve problems a tad faster, but you will definitely lack consistency and miss your estimates far too often.

Also when you approach a problem very structurally, the interviewer will clearly see that you can break down a problem of any size into smaller parts and then come up with a solution.

SOLUTION:

APPROACH 1:

import java.io.\*;

import java.util.\*;

class MessagePassing {

//spiral matrix

public int[] spiralOrder(int[][] matrix) {

//List<Integer> lst = new ArrayList<Integer>();

int[] result = new int[(matrix.length)\*(matrix.length)];

int i;

int rowStart = 0;

int colStart = 0;

int rowEnd = matrix.length;

int colEnd = matrix[0].length;

int index = 0;

while (rowStart < rowEnd && colStart < colEnd) {

// Print the first row from the remaining rows

for (i = colStart; i < colEnd; ++i) {

result[index]=matrix[rowStart][i];

index++;

}

rowStart++;

// Print the last column from the remaining columns

for (i = rowStart; i < rowEnd; ++i) {

result[index]=matrix[i][colEnd - 1];

index++;

}

colEnd--;

// Print the last row from the remaining rows \*/

if (rowStart < rowEnd) {

for (i = colEnd - 1; i >= colStart; --i) {

result[index]=matrix[rowEnd - 1][i];

index++;

}

rowEnd--;

}

// Print the first column from the remaining columns \*/

if (colStart < colEnd) {

for (i = rowEnd - 1; i >= rowStart; --i) {

result[index]=matrix[i][colStart];

index++;

}

colStart++;

}

}

return result;

}

enum Index {

GOOD, BAD, UNKNOWN

}

// Complete the below function implementation

// Print "Yes" for success and "No" for Failure

public void messagePassTest(int n, int s, int m, int[][] matrix) {

matrix[0][0]=s;

int[] result = spiralOrder(matrix);

boolean answer = canJump(result);

if(answer==true)

System.out.println("Yes");

else

System.out.println("No");

}

public boolean canJump(int[] nums) {

Index[] memo = new Index[nums.length];

for (int i = 0; i < memo.length; i++) {

memo[i] = Index.UNKNOWN;

}

memo[memo.length - 1] = Index.GOOD;

for (int i = nums.length - 2; i >= 0; i--) {

int furthestJump = Math.min(i + nums[i], nums.length - 1);

for (int j = i + 1; j <= furthestJump; j++) {

if (memo[j] == Index.GOOD) {

memo[i] = Index.GOOD;

break;

}

}

}

return memo[0] == Index.GOOD;

}

public static void main(String args[]) {

Scanner scanner = new Scanner(System.in);

int n = scanner.nextInt();

int s = scanner.nextInt();

int m = scanner.nextInt();

int[][] matrix = new int[n][n];

for(int i = 0 ; i < m ; ++i) {

int x , y , p;

x = scanner.nextInt();

y = scanner.nextInt();

p = scanner.nextInt();

matrix[x][y] = Math.max(p, matrix[x][y]);

}

scanner.close();

new MessagePassing().messagePassTest(n,s,m,matrix);

}

}

**Complexity Analysis:**

* **Time Complexity:**
* **Space Complexity:**