Problem Statement:-

Path Finder is a program which takes a matrix as an input, and displays the shortest path distance possible between first element and last element of this matrix under some specific conditions.

A path’s distance is calculated as sum of elements falling under its path. The shortest distance builds the output off this program.

Conditions followed by program :

1. Path is not possible if an element of negative value occurs.
2. Cursor can only move either right or down i.e. [i+1][j] or [i][j+1] increments.

Example:-

Here 9 to 12 is valid jump, but 12 to -55 is not a valid jump.

|  |  |  |
| --- | --- | --- |
| 9 | 41 | 23 |
| 12 | 0 | -2 |
| -55 | 44 | 12 |
| 43 | 76 | 1 |

**Approach:-**

Let us consider a simple 3x3 matrix having elements as shown.

**Table- Possible Paths:-**

|  |  |  |
| --- | --- | --- |
| **1** | **2** | **4** |
| **3** | **5** | **7** |
| **6** | **8** | **9** |

Let us compare 7+9 and 8+9. Larger one eliminates its path where there are **choices**, in this case **7< 8.**

Similarly now comparing **children of ’2’,** 4+7+9 < 5+7+9. So path 5+7+9 gets removed.

Then comparing **children of ‘3’,** 5+7+9 < 6+8+9. So, path 6+8+9 is removed.

Now comparing **children of ‘1’**. 2+4+7+9 < 3+5+7+9.

Produced path distance will be 1+2+4+7+9. This is the smallest distance.

**Possible Applications:**

Game development: When an in game object is to be moved robotically with the shortest path, considering obstacles to be negative values on the matrix.

Maps: To find shortest/fastest routes where elements of the matrix may represent actual distance or probable time and negative values represent path blocked or jammed.

**Concepts used in code**

2D Arrays: A **2D array** is organized as a matrix with a number of rows and columns  collecting data cells, all of the same type, which can be given a single name.

Recursion**:** **Recursion** is a method of solving a problem where the solution depends on solutions to smaller instances of the same problem. Here we used recursion of functions to break the problem into smaller levels and then adding up to form complete answer.

**Imported class**

Java.util.Scanner: This class allows us to use its scanning properties for taking input from user within the terminal window.

import java.util.Scanner;

/\*\*

\*

\* @author manan

\*/

class inp{

int sum=0,m=0,n=0;

int[][] arr;

Scanner s= new Scanner(System.in);

int small,max;

//String smp="",ps="", sp="";

public int[][] getarr(){

//this function is used to input the contents in array

System.out.println("enter no. of rows and columns");

m=s.nextInt();

n=s.nextInt();

arr= new int[m][n];

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

for(int j=0; j<n; j++)

{

arr[i][j] = s.nextInt();

}

}

return arr;

}

public int path(int[][] arr,int sum,int a,int b){

//this function returns the value of smallest path sum

if(arr[a][b]>=0){

if((a>=m-1)&&(b>=n-1))

{

//successfully reached end

if(sum<small)

{ small=sum;

}

return small;

}

else if((a>=m-1)&&(b<n-1)){

//only down possible

return path(arr,sum+ arr[a][b+1],a,b+1);

}

else if((b>=n-1)&&(a<m-1)){

// only right possible

return path(arr,sum+ arr[a+1][b],a+1,b);

}

else{

// both right and down possible

//System.out.println("path divided into 2");

int right= path(arr,sum +arr[a+1][b],a+1,b);

int down= path(arr,sum+ arr[a][b+1],a,b+1);

if(right<down)

{return right;}

else

{return down;}

}

}

else{

//System.out.println("this path is not possible");

return max+1;

}

}

public int patha(int[][] arr,int sum,int a,int b){

// this function takes out a max value(no path length can exceed this value)

max=0;

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

for(int j=0; j<n; j++)

{

if(arr[i][j] > 0)

{

max=max+arr[i][j];

}

}

}

small=max;

small= path(arr,sum,a,b);

return small;

}

}

public class Pathfinder {

/\*\*

\* @param args the command line arguments

\*/

public static void main(String[] args) {

// TODO code application logic here

int[][] arr= null;

inp o= new inp();

arr=o.getarr();

System.out.print("Shortest path sum = "+o.patha(arr,0,0,0));

}

}