Sukkur IBA University
Department of Computer Science

DATA STRUCTURES
Lab01 – Arrays, LinkesLists

Instructor: Saif Hassan

READ IT FIRST

Prior to start solving the problems in this assignments, please give full concentration on following points.

1. WORKING – This is individual lab. If you are stuck in a problem contact your teacher, but, in mean time start doing next question (don't waste time).
2. DEADLINE – 11th March, 2022
3. SUBMISSION – This assignment needs to be submitted in a soft copy.
4. WHERE TO SUBMIT – Please visit your LMS.
5. WHAT TO SUBMIT – Submit this docx and pdf file.

KEEP IT WITH YOU!

1. Indent your code inside the classes and functions. It's a good practice!
2. It is not bad if you keep your code indented inside the loops, if and else blocks as well.
3. Comment your code, where it is necessary.
4. Read the entire question. Don't jump to the formula directly.

I, **Amjad Ali** with student ID _**191-21-0001**_

Section _**A**_hereby declare that I do understand the instructions above and follow them. This is my own work.

Exercises

Task1 Description

Task 01: (Reverse linked list)

You have worked on all types of linkedlist, now design a method for single linked list that will reverse the linked list. Same linked list will be in reverse direction. You are given head of linked list and just have to change next pointer of nodes so that list may be reversed.

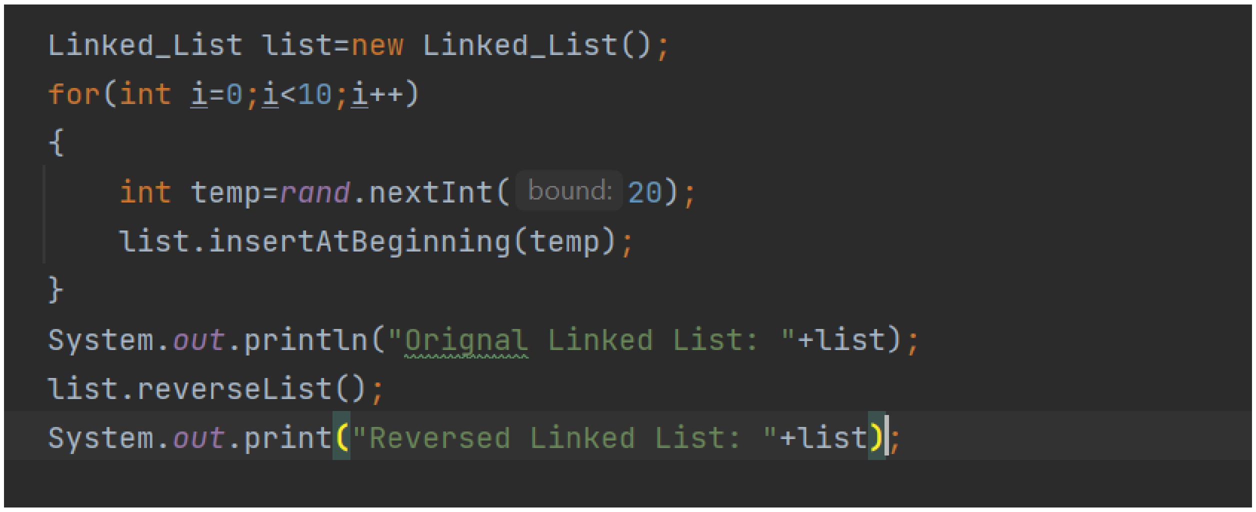
Public Node makeReverse(Node head)

Solution:

Code

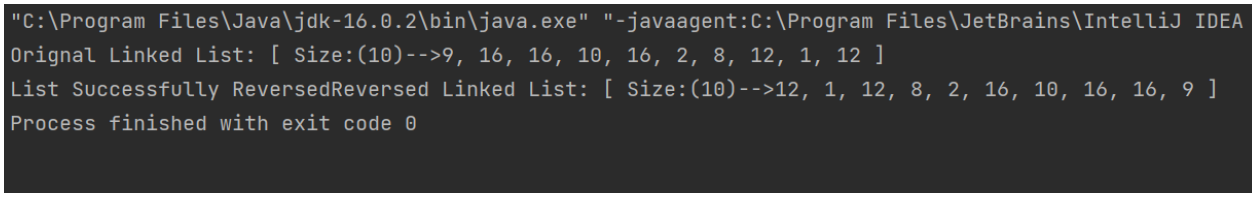
```java
public void reverseList(){
        if(Head==null)
            System.out.println("List is Empty");
        else{
            Node current=Head;
            Node previous=null;
            while(current.next!=null)
            {
                Node temp=current.next;
                  current.next=previous;
                  previous=current;
                  current=temp;
              }
              current.next=previous;
              Head=current;
              System.out.print("List Successfully Reversed");


          }

    }
```

<u>**Sample Input:**</u>



```
Linked_List list=new Linked_List();
for(int i=0;i<10;i++)
{
    int temp=rand.nextInt( bound: 20);
    list.insertAtBeginning(temp);
}
System.out.println("Orignal Linked List: "+list);
list.reverseList();
System.out.print("Reversed Linked List: "+list);
```

<u>**Sample Output**</u>



```
"C:\Program Files\Java\jdk-16.0.2\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA
Orignal Linked List: [ Size:(10)-->9, 16, 16, 10, 16, 2, 8, 12, 1, 12 ]
List Successfully ReversedReversed Linked List: [ Size:(10)-->12, 1, 12, 8, 2, 16, 10, 16, 16, 9 ]
Process finished with exit code 0
```

Task2 Description

<u>Task 02: (Print in reverse order)</u>

You are asked to design a method in linked list to print data in reverse order. You don’t need to reverse linkedlist permanently.

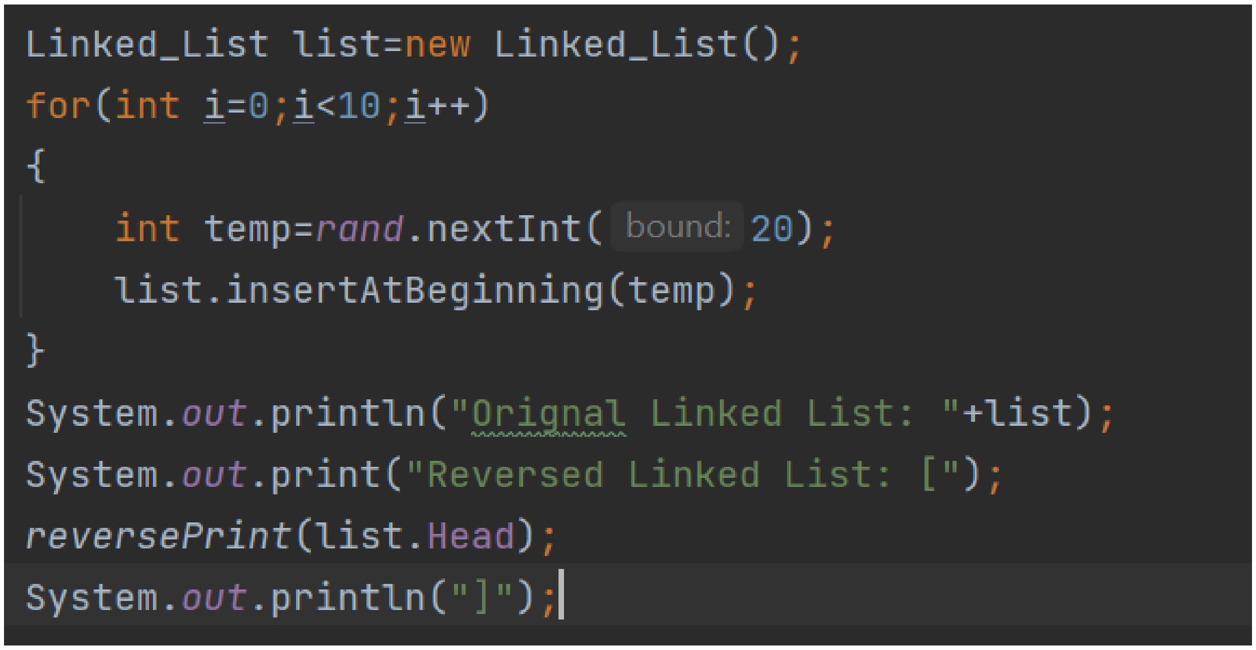
`Public void printReverse()`

Solution:

Code

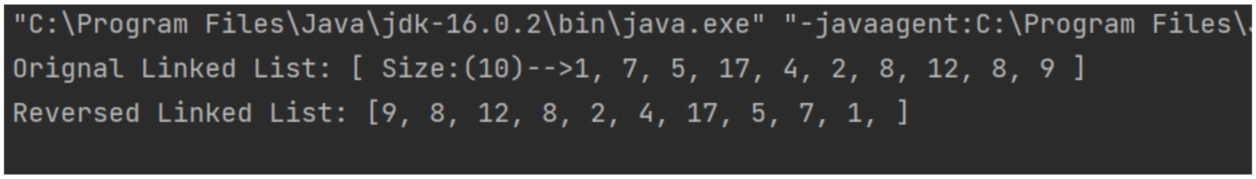
```java
public static void reversePrint(Node current)
    {
        if(current.next!=null)
        {
            reversePrint(current.next);
        }
        System.out.print(current.data+", ");
    }
```

<u>Sample Input:</u>



```java
Linked_List list=new Linked_List();
for(int i=0;i<10;i++)
{
    int temp=rand.nextInt( bound: 20);
    list.insertAtBeginning(temp);
}
System.out.println("Orignal Linked List: "+list);
System.out.print("Reversed Linked List: [");
reversePrint(list.Head);
System.out.println("]");
```

<u>**Sample Output**</u>



```
"C:\Program Files\Java\jdk-16.0.2\bin\java.exe" "-javaagent:C:\Program Files\
Orignal Linked List: [ Size:(10)-->1, 7, 5, 17, 4, 2, 8, 12, 8, 9 ]
Reversed Linked List: [9, 8, 12, 8, 2, 4, 17, 5, 7, 1, ]
```

Task3 Description

Task 03: (Cycle Detection):

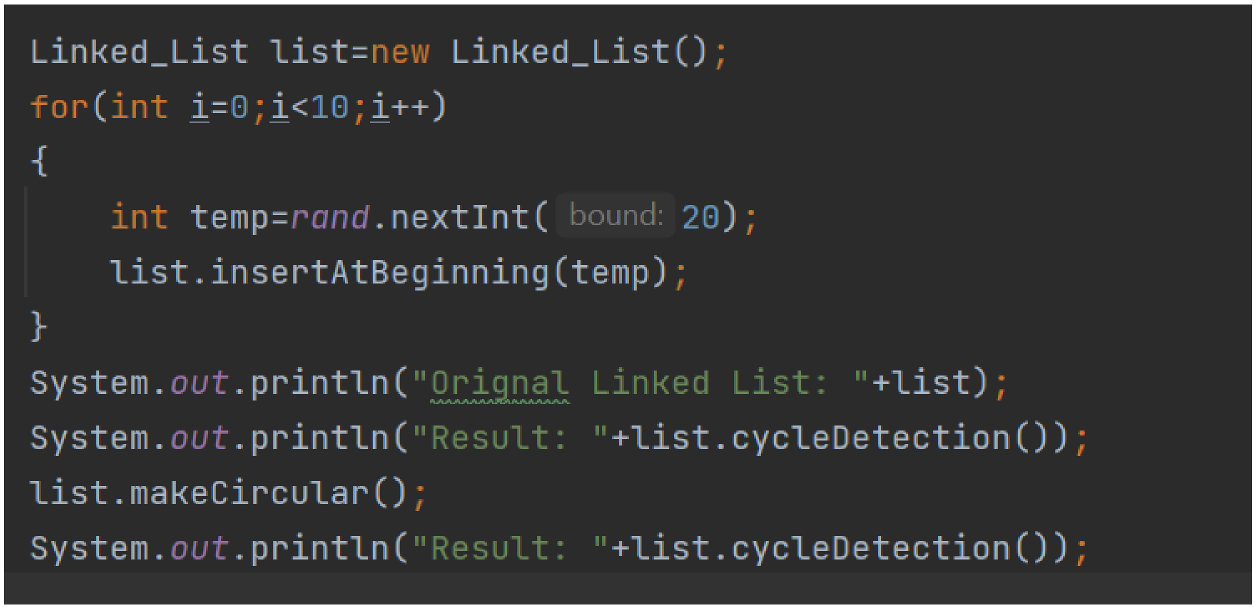
Write a method in linkedlist class that will detect cycle in list?

Solution:

Code

```java
public boolean cycleDetection()
    {
        Set<Node> set=new HashSet<>();
        int size=size();
        Node current=Head;
        while(current!=null)
        {
            if(!set.add(current))
            {
                 return true;
             }
             current=current.next;
         }

         return false;
     }
```

Sample Input:



```
Linked_List list=new Linked_List();
for(int i=0;i<10;i++)
{
    int temp=rand.nextInt( bound: 20);
    list.insertAtBeginning(temp);
}
System.out.println("Orignal Linked List: "+list);
System.out.println("Result: "+list.cycleDetection());
list.makeCircular();
System.out.println("Result: "+list.cycleDetection());
```

Sample Output

Task4 Description

Task 04: (Balanced Brackets)

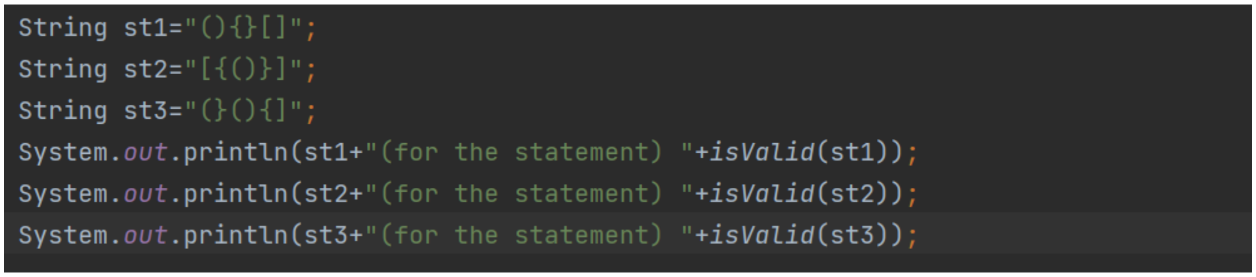
We have discussed in class about Balanced brackets problem using Stack. Take user string input and check whether it's balanced or not. Use stack functions. Input may contain any of the bracket among {, [, (and any number and letters like: ({[a+b]+c}-1) and so on.

Solution:

Code

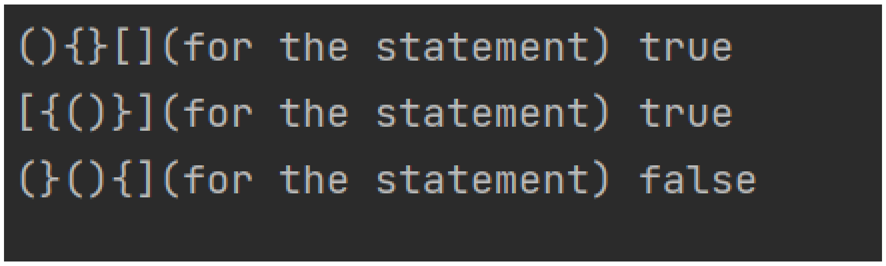
```java
public boolean isValid(String s) {
        Stack<Character> arr=new Stack<>();
        int length=s.length();
        for(int i=0;i<length;i++)
        {
            if(s.charAt(i)=='[' || s.charAt(i)=='('
||s.charAt(i)=='{'
||s.charAt(i)=='<')
                arr.push(s.charAt(i));
            if(s.charAt(i)==']')
            {
                 if(arr.size()==0 || arr.pop()!='[')
                     return false;
              }
              if(s.charAt(i)==')' )
              {
                 if(arr.size()==0 || arr.pop()!='(')
                     return false;

              }
              if(s.charAt(i)=='}')
              {
                 if(arr.size()==0 ||arr.pop()!='{')
                     return false;
              }
              if(s.charAt(i)=='>')
              {
                 if(arr.size()==0 ||arr.pop()!='<')
                     return false;
              }


          }
          if(arr.size()==0)
              return true;
          else
              return false;

      }
```

Sample Input:



```
String st1="(){}[]";
String st2="[{()}]";
String st3="(}(){]";
System.out.println(st1+"(for the statement) "+isValid(st1));
System.out.println(st2+"(for the statement) "+isValid(st2));
System.out.println(st3+"(for the statement) "+isValid(st3));
```

Sample Output



```
(){}[](for the statement) true
[{()}](for the statement) true
(}(){](for the statement) false
```

Task5 Description

Task05: (FirstSingleLetter)

Create the non-recursive function char firstSingleLetter (const char text [], const int n) which finds and returns the first letter of text that occurs only once. n is the number of characters in the text.

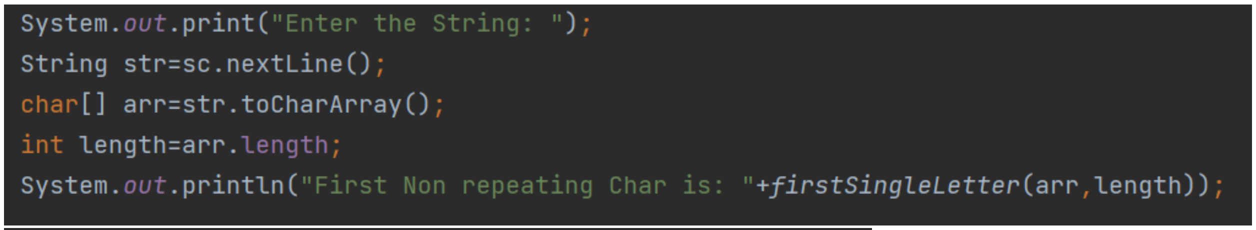
Here in task 5, it is not allowed to use any help structures - like for example. array, stack, queue or list. The function will therefore be as O (n^2). Text contains only the letters A-Z (only uppercase letters, and no spaces).

Solution:

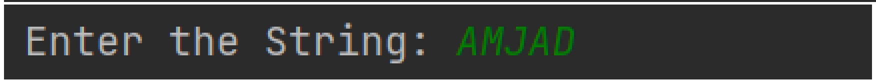
Code:

```
public static char firstSingleLetter(char text[], int n)
{
        boolean cond = true;
        for (int i = 0; i < n; i++) {
            for (int j = i + 1; j < n; j++) {
                if (text[i] == text[j]) {
                    cond = false;
                }
            }

              if (cond) {
                  return text[i];
              }
              cond = true;
          }
       System.out.println("No such Character in Text");
         return '1';
     }
```

Sample Input:

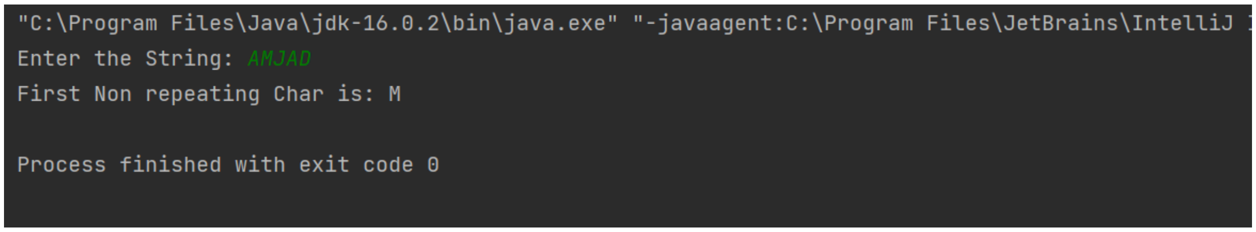


```
System.out.print("Enter the String: ");
String str=sc.nextLine();
char[] arr=str.toCharArray();
int length=arr.length;
System.out.println("First Non repeating Char is: "+firstSingleLetter(arr,length));
```



```
Enter the String: AMJAD
```

Sample Output



```
"C:\Program Files\Java\jdk-16.0.2\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ
Enter the String: AMJAD
First Non repeating Char is: M

Process finished with exit code 0
```

Task06 Description

<u>Task06: Apply Greedy approach</u>

0	-1	-1	1	1	1	0
-1	2	5	4	10	3	-1
3	2	-1	-1	0	3	8
7	-1	10	2	-1	-1	17
4	3	9	-1	-1	8	33
17	-1	-1	1	0	44	100

Given a matrix above, you must start from YELLOW Box and reach to the goal which is Green Box by traversing each cell with maximum value of neighbors. -1 in any cell means BLOCK, you cannot visit that cell. You can move any cell in neighbors ONLY.

In case all the neighbors have -1 value then show message, NO POSSIBLE PATH due to BLOCKs on every side.

START: [0, 0] → YELLOW

GOAL: [n-1, n-1] → GREEN

You have to follow Greedy Approach, Greedy approach means selecting with immediate high reward (short term). In this problem, reward is your neighbors value.

Example of selecting path is given in above matrix, highlighted with ORANGE color

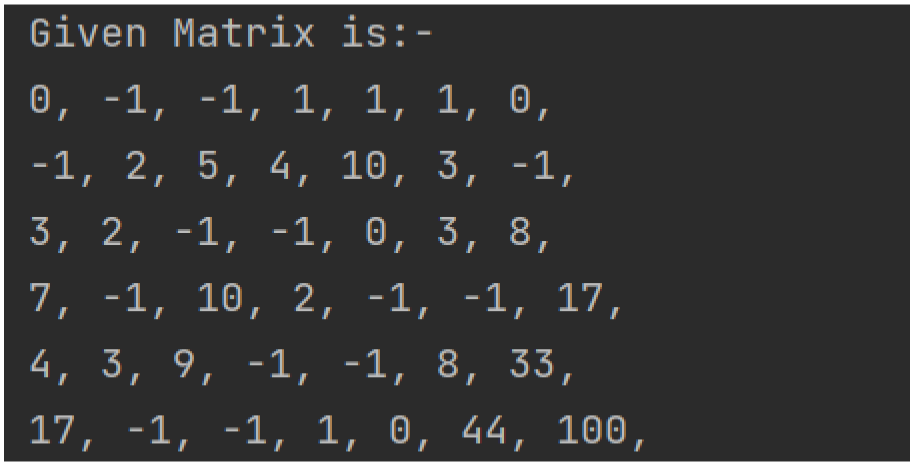
Solution:

Code:

```java
package com.company;

import java.util.*;
import java.util.List;

public class Greedy {
    static Random rand = new Random();

    public static int[][] makeMatrix(int row, int col) {
        int[][] matrix = new int[row][col];
        for (int i = 0; i < row; i++) {
            for (int j = 0; j < col; j++) {
                int random = rand.nextInt(90);
                if (i == 0 && j == 0) {
                    matrix[i][j] = 0;
                } else if (i == (row - 1) && j == (col - 1)) {
                    matrix[i][j] = 100;
                } else {
                    matrix[i][j] = random;
                }
            }
        }

        for (int i = 0; i < 18; i++) {
            int r = rand.nextInt(row);
            int c = rand.nextInt(col);
            if ((r != 0 && r != (row - 1)) && (c != 0 && c != (col - 1))) {
                matrix[r][c] = -1;
            }
        }

        return matrix;
    }

    public static void printMatrix(int[][] matrix) {
        for (int i = 0; i < matrix.length; i++) {
            for (int j = 0; j < matrix[0].length; j++) {
                System.out.print(matrix[i][j] + ", ");
            }
            System.out.println();
        }

    }

    public static int[] findGreatest(int[][] matrix, int[] ind) {
        int[][] point = new int[8][3];
        for (int i = 0; i < 8; i++)
            point[i][0] = -1;

        try {
            //1st
            point[0][0] = matrix[ind[0] + 1][ind[1] + 1];
            point[0][1] = ind[0] + 1;
            point[0][2] = ind[1] + 1;
        } catch (Exception e) {

        }
        try {
            //2nd
            point[1][0] = matrix[ind[0]][ind[1] + 1];
            point[1][1] = ind[0];
            point[1][2] = ind[1] + 1;
        } catch (Exception e) {

        }
        try {
            //3rd
            point[2][0] = matrix[ind[0] + 1][ind[1]];
            point[2][1] = ind[0] + 1;
            point[2][2] = ind[1];
        } catch (Exception e) {

        }
        try {
            //4th
            point[3][0] = matrix[ind[0] + 1][ind[1] - 1];
            point[3][1] = ind[0] + 1;
            point[3][2] = ind[1] - 1;
        } catch (Exception e) {

        }
        try {
            //5th
            point[4][0] = matrix[ind[0] - 1][ind[1] + 1];
            point[4][1] = ind[0] - 1;
            point[4][2] = ind[1] + 1;
        } catch (Exception e) {

        }
        try {
            //6th
            point[5][0] = matrix[ind[0] - 1][ind[1]];
            point[5][1] = ind[0] - 1;
            point[5][2] = ind[1];
        } catch (Exception e) {

        }
        try {
            //7th
            point[6][0] = matrix[ind[0]][ind[1] - 1];
            point[6][1] = ind[0];
            point[6][2] = ind[1] - 1;
        } catch (Exception e) {

        }
        try {
            //8th
            point[7][0] = matrix[ind[0] - 1][ind[1] - 1];
            point[7][1] = ind[0] - 1;
            point[7][2] = ind[1] - 1;
        } catch (Exception e) {

        }

        int bigind = 0;
        int biggest = point[0][0];
        for (int i = 0; i < 8; i++) {

            if (point[i][0] > biggest) {
                bigind = i;
                biggest = point[i][0];

            }

        }
        matrix[point[bigind][1]][point[bigind][2]] = -1;
        int[] arr = point[bigind];
        return arr;

    }

    public static List<int[]> findPath(int[][] matrix) {

        int result = 0;
        List<int[]> path = new ArrayList<>();
        int[] ind = {0, 0};
        while (result != 100) {
            int[] values = findGreatest(matrix, ind);
            result = values[0];
            ind[0] = values[1];
            ind[1] = values[2];
            if (result == -1) {

                break;
            }
```

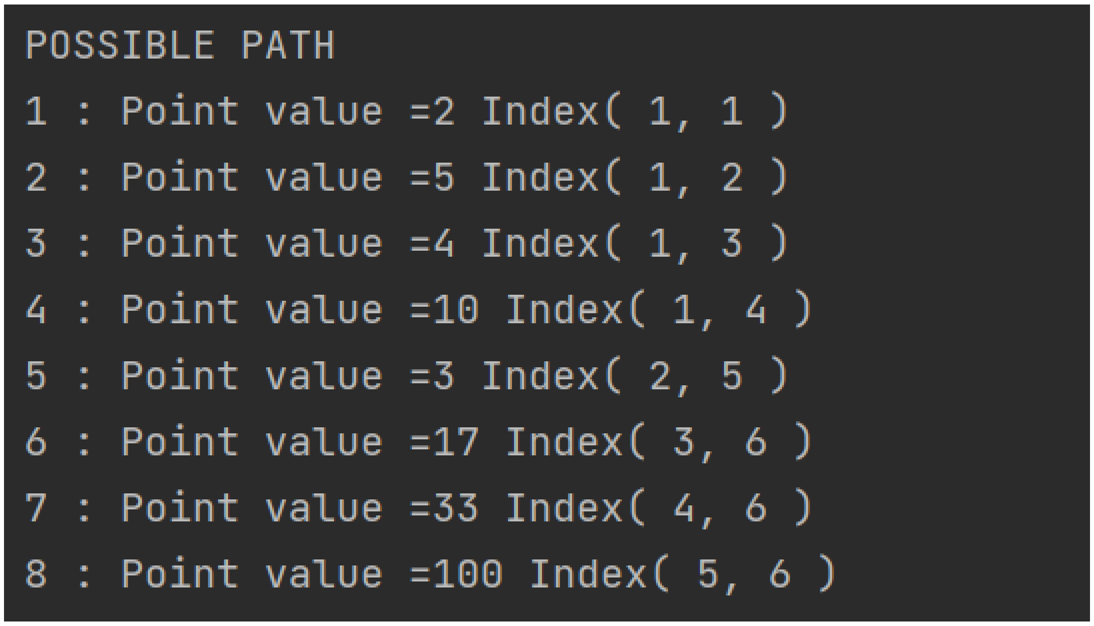
```java

            path.add(values);

        }

        return path;
    }

    public static boolean pathcheck(List<int[]> list) {
        int size = list.size();
        int[] check = list.get((size - 1));
        if (check[0] != 100)
            return false;
        else
            return true;
    }

    public static void printPath(int[][] matrix) {
        List<int[]> path = findPath(matrix);
        boolean result = pathcheck(path);
        if (result) {
            int size = path.size();
            System.out.println("POSSIBLE PATH");
            for (int i = 0; i < size; i++) {
                int arr[] = path.get(i);
                System.out.println((i + 1) + " : Point value =" + arr[0] + " Index( " + arr[1] + ", " + arr[2] + " )");
            }

        } else {
            System.out.println(" NO POSSIBLE PATH due to BLOCKs");
        }


    }

    public static void main(String[] args) {
//          Scanner sc=new Scanner(System.in);
//          System.out.print("Enter the rows: ");
//         int row=sc.nextInt();
//         System.out.print("Enter the colomns: ");
//         int col=sc.nextInt();
//          int[][] matrix=makeMatrix(row,col);
        int[][] matrix = {{0, -1, -1, 1, 1, 1, 0},
                {-1, 2, 5, 4, 10, 3, -1},
                {3, 2, -1, -1, 0, 3, 8},
                {7, -1, 10, 2, -1, -1, 17},
                {4, 3, 9, -1, -1, 8, 33},
                {17, -1, -1, 1, 0, 44, 100}};
        System.out.println("Given Matrix is:-");
        printMatrix(matrix);
        printPath(matrix);


    }
}
```

Sample Input:



```
Given Matrix is:-
0, -1, -1, 1, 1, 1, 0,
-1, 2, 5, 4, 10, 3, -1,
3, 2, -1, -1, 0, 3, 8,
7, -1, 10, 2, -1, -1, 17,
4, 3, 9, -1, -1, 8, 33,
17, -1, -1, 1, 0, 44, 100,
```

Sample Output



```
POSSIBLE PATH
1 : Point value =2 Index( 1, 1 )
2 : Point value =5 Index( 1, 2 )
3 : Point value =4 Index( 1, 3 )
4 : Point value =10 Index( 1, 4 )
5 : Point value =3 Index( 2, 5 )
6 : Point value =17 Index( 3, 6 )
7 : Point value =33 Index( 4, 6 )
8 : Point value =100 Index( 5, 6 )
```