Theory DA1

Name: Sankalp Mukim

Registration Number: 20BDS0128

Slots: A1+TA1

# Important Note:

I have attached 2 versions of codes and outputs in the below report.

1. The first one is the direct solution that finds the required element.
2. The second one *demonstrates* **entire traversal** with the help of println statements

# Question:

Diagram

Description automatically generated

Diagram

Description automatically generated

A picture containing text

Description automatically generated

# Direct Solution:

## Code:

import java.util.HashMap;

import java.util.Map;

class Node {

    public int key;

    public Node[] adjacentsNodes;

    private int idx;

    public Node(int k) {

        this.key = k;

        this.idx = -1;

        adjacentsNodes = new Node[0];

    }

    public Node() {

        this.idx = -1;

    }

    public void setKey(int key) {

        this.key = key;

    }

    public void addAdjacent(Node mNode) {

        Node[] newArr = new Node[adjacentsNodes.length + 1];

        int i;

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

            newArr[i] = adjacentsNodes[i];

        }

        newArr[i] = mNode;

        this.adjacentsNodes = newArr;

        // System.out.println("Getting added idx =" + (i) + " value = " + mNode.key);

    }

    public Node getCurrAdjNode() {

        this.idx++;

        // System.out.println("this.idx" + this.idx);

        // System.out.println("this.adjacentNodes.length" + this.adjacentsNodes.length);

        if (this.idx >= adjacentsNodes.length) {

            return null;

        } else {

            return this.adjacentsNodes[this.idx];

        }

    }

}

class StackElement {

    public Node n;

    public StackElement next;

    public StackElement(Node n) {

        this.n = n;

        next = null;

    }

}

class Stack {

    public StackElement top;

    public Stack() {

        top = null;

    }

    public void addElement(Node n) {

        if (top == null) {

            StackElement ele = new StackElement(n);

            top = ele;

        } else {

            StackElement x = new StackElement(n);

            x.next = top;

            top = x;

        }

    }

    public StackElement popElement() {

        if (top != null) {

            StackElement x = top;

            top = top.next;

            return x;

        } else {

            return null;

        }

    }

}

class Assessment {

    public static void main(String[] args) {

        int[][] matrix = {

                { 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

                { 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

                { 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0 },

                { 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

                { 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

                { 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

                { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

                { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

                { 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0 },

                { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0 },

                { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

                { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0 },

                { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0 },

                { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0 },

                { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0 },

                { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

                { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1 },

                { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 }

        };

        int elementToFind = 16;

        Map<Integer, Boolean> visited = new HashMap<Integer, Boolean>();

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

            visited.put(i + 1, false);

        }

        int starterElementIdx = 0;

        Node x = new Node(starterElementIdx + 1);

        Stack s = new Stack();

        s.addElement(x);

        while (s.top != null) {

            Node currNode = s.top.n;

            // System.out.println(I + ": " + currNode.key);

            // marking visited node to avoid deadlock

            // checking if current node is the elementToFind

            if (currNode.key == elementToFind) {

                System.out.println("Element to find is found = " + currNode.key);

                break;

            }

            if (visited.get(currNode.key) == false) {

                // System.out.println("Finding adjacent for " + currNode.key);

                // finding all adjacent nodes to current node using adjacency matrix

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

                    if (matrix[currNode.key - 1][i] == 1) {

                        currNode.addAdjacent(new Node(i + 1));

                    }

                }

            }

            // traversing and adding all adjacent nodes of current node to stack

            Node adjNode = currNode.getCurrAdjNode();

            if (adjNode == null || visited.get(adjNode.key)) {

                // System.out.println("Popped?");

                s.popElement();

            } else {

                // System.out.println("Added");

                s.addElement(adjNode);

            }

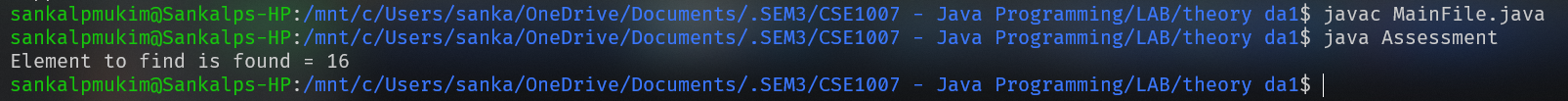
            visited.put(currNode.key, true);

        }

    }

}

## Output:



# Full demonstration:

Because of the long code and output, I recommend running the code locally rather than looking at screenshot to check for correctness.

## Code:

import java.util.HashMap;

import java.util.Map;

class Node {

    public int key;

    public Node[] adjacentsNodes;

    private int idx;

    public Node(int k) {

        this.key = k;

        this.idx = -1;

        adjacentsNodes = new Node[0];

    }

    public Node() {

        this.idx = -1;

    }

    public void setKey(int key) {

        this.key = key;

    }

    public void addAdjacent(Node mNode) {

        Node[] newArr = new Node[adjacentsNodes.length + 1];

        int i;

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

            newArr[i] = adjacentsNodes[i];

        }

        newArr[i] = mNode;

        this.adjacentsNodes = newArr;

        System.out.println("Getting added idx =" + (i) + " value = " + mNode.key);

    }

    public Node getCurrAdjNode() {

        this.idx++;

        System.out.println("this.idx" + this.idx);

        System.out.println("this.adjacentNodes.length" + this.adjacentsNodes.length);

        if (this.idx >= adjacentsNodes.length) {

            return null;

        } else {

            return this.adjacentsNodes[this.idx];

        }

    }

}

class StackElement {

    public Node n;

    public StackElement next;

    public StackElement(Node n) {

        this.n = n;

        next = null;

    }

}

class Stack {

    public StackElement top;

    public Stack() {

        top = null;

    }

    public void addElement(Node n) {

        if (top == null) {

            StackElement ele = new StackElement(n);

            top = ele;

        } else {

            StackElement x = new StackElement(n);

            x.next = top;

            top = x;

        }

    }

    public StackElement popElement() {

        if (top != null) {

            StackElement x = top;

            top = top.next;

            return x;

        } else {

            return null;

        }

    }

}

class Assessment {

    public static void main(String[] args) {

        int[][] matrix = {

                { 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

                { 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

                { 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0 },

                { 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

                { 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

                { 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

                { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

                { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

                { 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0 },

                { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0 },

                { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

                { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0 },

                { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0 },

                { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0 },

                { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0 },

                { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

                { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1 },

                { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 }

        };

        int elementToFind = 16;

        Map<Integer, Boolean> visited = new HashMap<Integer, Boolean>();

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

            visited.put(i + 1, false);

        }

        int starterElementIdx = 0;

        Node x = new Node(starterElementIdx + 1);

        Stack s = new Stack();

        s.addElement(x);

        int I = 0;

        while (s.top != null) {

            I++;

            Node currNode = s.top.n;

            System.out.println(I + ": " + currNode.key);

            // marking visited node to avoid deadlock

            // checking if current node is the elementToFind

            if (currNode.key == elementToFind) {

                System.out.println("Element to find is found = " + currNode.key);

            }

            if (visited.get(currNode.key) == false) {

                System.out.println("Finding adjacent for " + currNode.key);

                // finding all adjacent nodes to current node using adjacency matrix

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

                    if (matrix[currNode.key - 1][i] == 1) {

                        currNode.addAdjacent(new Node(i + 1));

                    }

                }

            }

            // traversing and adding all adjacent nodes of current node to stack

            Node adjNode = currNode.getCurrAdjNode();

            if (adjNode == null || visited.get(adjNode.key)) {

                System.out.println("Popped?");

                s.popElement();

            } else {

                System.out.println("Added");

                s.addElement(adjNode);

            }

            visited.put(currNode.key, true);

        }

    }

}

## Output:

Graphical user interface, text

Description automatically generated

Text

Description automatically generatedText

Description automatically generated

Text

Description automatically generatedA picture containing background pattern

Description automatically generated