Assignment 2

21_AIE_203

Data Structure and Algorithms – SEM-III

Professor – Dr. Sachin Sir

Submitted By: Vikhyat Bansal [CB.EN.U4AIE21076]

Q1. Implement Breadth-First Search using Recursion.

Code:

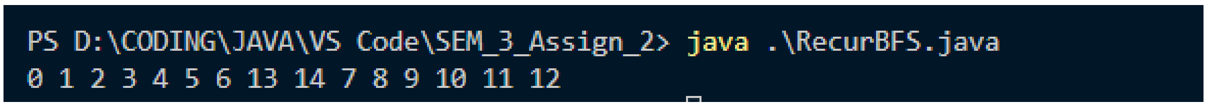
```
package SEM_3_Assign_2;
import java.util.*;

// breadth first search in graph using adjacency list WITH
RECURSION

import java.util.*;

class RecurBFS
{
    // Perform BFS recursively on the graph
    public static void recursiveBFS(Graph graph, Queue<Integer> q,
                                    boolean[] discovered)
    {
        if (q.isEmpty()) {
            return;
        }

        // dequeue front node and print it
        int v = q.poll();
        System.out.print(v + " ");

        // do for every edge (v, u)
        for (int u: graph.adjList.get(v))
        {
            if (!discovered[u])
            {
                // mark it as discovered and enqueue it
                discovered[u] = true;
                q.add(u);
            }
        }

        recursiveBFS(graph, q, discovered);
    }

    public static void main(String[] args)
    {
        // List of graph edges as per the above diagram
        List<Edge> edges = Arrays.asList(
                new Edge(1, 2), new Edge(1, 3), new Edge(1, 4), new
Edge(2, 5),
                new Edge(2, 6), new Edge(5, 9), new Edge(5, 10), new
Edge(4, 7),
                new Edge(4, 8), new Edge(7, 11), new Edge(7, 12),new
Edge
                (3,13),new Edge(3,14)
        );

        // total number of nodes in the graph (labelled from 0 to
14)
        int n = 15;

        // build a graph from the given edges
        Graph graph = new Graph(edges, n);

        // to keep track of whether a vertex is discovered or not
        boolean[] discovered = new boolean[n];

        // create a queue for doing BFS
        Queue<Integer> q = new ArrayDeque<>();

        // Perform BFS traversal from all undiscovered nodes to
        // cover all connected components of a graph
        for (int i = 0; i < n; i++)
        {
            if (discovered[i] == false)
            {
                // mark the source vertex as discovered
                discovered[i] = true;

                // enqueue source vertex
                q.add(i);

                // start BFS traversal from vertex `i`
                recursiveBFS(graph, q, discovered);
            }
        }
    }
}
```

```java
// A class to store a graph edge
class Edge
{
    int source, destin;

    public Edge(int source, int destin)
    {
        this.source = source;
        this.destin = destin;
    }
}

// A class to represent a graph object
class Graph
{
    // A list of lists to represent an adjacency list
    List<List<Integer>> adjList = null;

    // Constructor
    Graph(List<Edge> edges, int n)
    {
        adjList = new ArrayList<>();

        for (int i = 0; i < n; i++) {
            adjList.add(new ArrayList<>());
        }

        // add edges to the undirected graph
        for (Edge edge: edges)
        {
            int src = edge.source;
            int destin = edge.destin;

            adjList.get(src).add(destin);
            adjList.get(destin).add(src);
        }
    }
}
```

Explanation:

- We are creating two classes which will help us in creating Graph data structure namely Edge and Graph.
- Now, we are going to create a class for Recursive BFS, which will contain a function for recursiveBFS.
- In method recursiveBFS, it will check whether the queue is empty or not.
- Then, accordingly it will visit each node and then after visiting each node it will mark it as discovered and then enqueue it in a queue where it will get marked and printed.
- This method will recursively call itself until and unless the queue is empty and as soon as queue is empty, it will stop the method.
- Now, we will create the MAIN method where we will create an adjacency list and then call the function/method which will mention all the nodes after traversing.

OUTPUT:



```
PS D:\CODING\JAVA\VS Code\SEM_3_Assign_2> java .\RecurBFS.java
0 1 2 3 4 5 6 13 14 7 8 9 10 11 12
```

Q2. Implement Breadth-First Search without using Recursion.

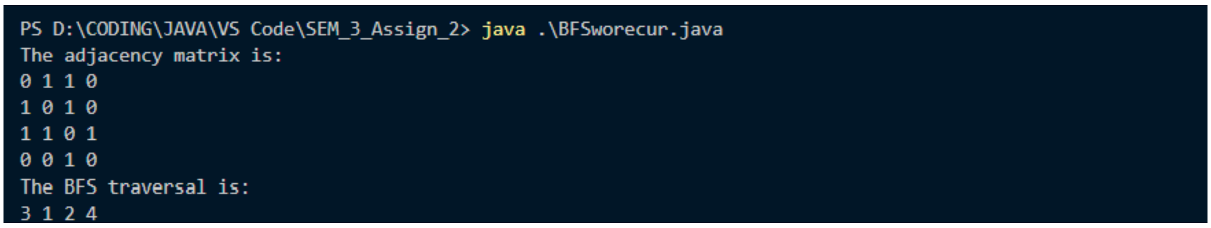
Code:

```java
package SEM_3_Assign_2;

// breadth first search in graph using adjacency matrix WITHOUT
RECURSION

import java.util.*;

public class BFSworecur {
    static int[][] mat = new int[4][4];
    static int[] visited = new int[4];
    static int[] queue = new int[4];
    static int front = 0;
    static int rear = 0;
    static int count = 0;

    static void insert(int x){
        queue[rear] = x;
        rear++;
        count++;
    }

    static int delete(){
        int x = queue[front];
        front++;
        count--;
        return x;
    }

    static void bfs(int v){
        int i;
        visited[v] = 1;
        insert(v);
        while(count!=0){
            v = delete();
            System.out.print((v+1) + " ");
            for (i = 0;i<4;i++){
                if (mat[v][i] == 1 && visited[i] == 0){
                    insert(i);
                    visited[i] = 1;
                }
            }
        }
    }

    public static void main(String[] args) {
        mat[0][1] = 1;
        mat[0][2] = 1;
        mat[1][0] = 1;
        mat[1][2] = 1;
        mat[2][0] = 1;
        mat[2][1] = 1;
        mat[2][3] = 1;
        mat[3][2] = 1;


        System.out.println("The adjacency matrix is: ");
        for (int i = 0;i<4;i++){
            for (int j = 0;j<4;j++){
                System.out.print(mat[i][j] + " ");
            }
            System.out.println();
        }
        System.out.println("The BFS traversal is: ");
        bfs(2);
}
}
```

Explanation:

- We have created a public class for breadth first without recursion.
- We have initialized 1-D array which will act as QUEUE and 2-D array which will act as adjacency matrix for representing GRAPH. Taking some variables that will do working of QUEUE functions.
- Finally, a method for BFS is created which will visit the node connected to the vertex and then we will insert the values in QUEUE which were never visited and then after visiting once those once visited vertex will get dequeued.
- Finally, MAIN is created where we will call the methods.

OUTPUT:



```
PS D:\CODING\JAVA\VS Code\SEM_3_Assign_2> java .\BFSworecur.java
The adjacency matrix is:
0 1 1 0
1 0 1 0
1 1 0 1
0 0 1 0
The BFS traversal is:
3 1 2 4
```

Q3. Implement Depth-First Search using Recursion.

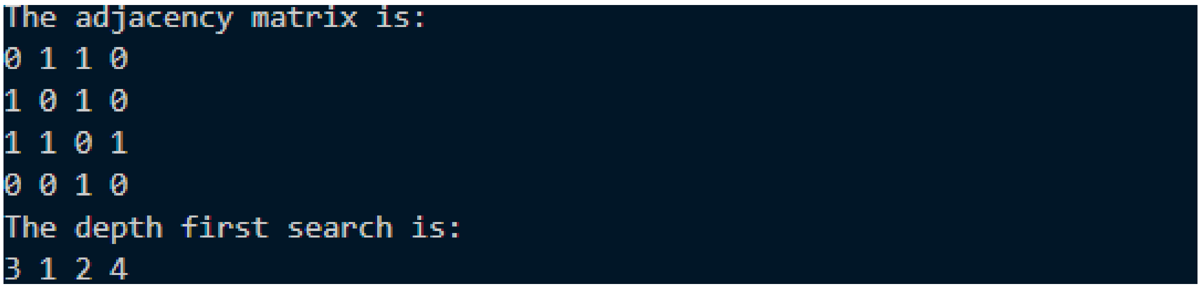
CODE:

```
package SEM_3_Assign_2;
// depth first search in graph using adjacency matrix WITH RECURSION
public class RecurDFS {
    static int[][] mat = new int[4][4];
    static int[] visited = new int[4];

    static void dfs(int v){
        int i;
        visited[v] = 1;
        System.out.print((v+1) + " ");
        for (i = 0;i<4;i++){
            if (mat[v][i] == 1 && visited[i] == 0){
                dfs(i);
            }
        }
    }
    public static void main(String[] args) {
        mat[0][1] = 1;
        mat[0][2] = 1;
        mat[1][0] = 1;
        mat[1][2] = 1;
        mat[2][0] = 1;
        mat[2][1] = 1;
        mat[2][3] = 1;
        mat[3][2] = 1;

        System.out.println("The adjacency matrix is: ");
        for (int i = 0;i<4;i++){
            for (int j = 0;j<4;j++){
                System.out.print(mat[i][j] + " ");
            }
            System.out.println();
        }

        System.out.println("The depth first search is: ");
        dfs(2);
    }
}
```

Explanation:

- Creating a class RecurDFS which will initialize a matrix and an array which will store all visited nodes.
- A method named dfs is created which will take argument as int v that is the node number.
- After each node is being visited, it is stored in the array and then printed.
- Finally, MAIN class is created which will call the dfs method for traversal.

OUTPUT:



```
The adjacency matrix is:
0 1 1 0
1 0 1 0
1 1 0 1
0 0 1 0
The depth first search is:
3 1 2 4
```

Q4. Implement Depth-First Search without Recursion.

CODE:

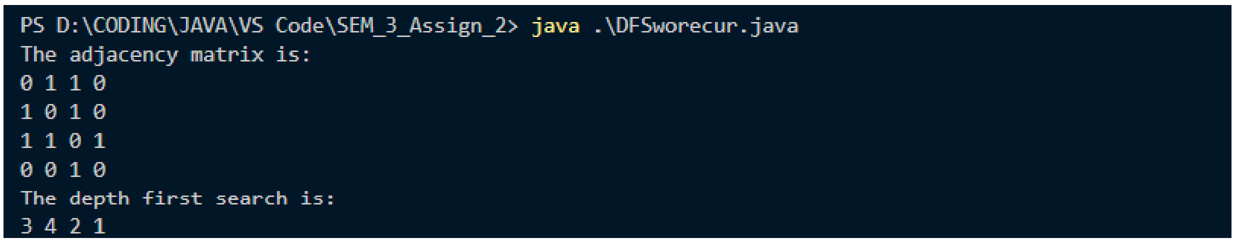
```
package SEM_3_Assign_2;

// depth first search in graph using adjacency matrix WITHOUT
RECURSION
public class DFSworecur {

    static int[][] mat = new int[4][4];
    static int[] visited = new int[4];
    static int[] stack = new int[4];
    static int top = -1;
    static int count = 0;

    static void push(int x){
        stack[++top] = x;
        count++;
    }

    static int pop(){
        int x = stack[top--];
        count--;
        return x;
    }

    static void dfs(int v){
        int i;
        visited[v] = 1;
        push(v);
        while(count!=0){
            v = pop();
            System.out.print((v+1) + " ");
            for (i = 0;i<4;i++){
                if (mat[v][i] == 1 && visited[i] == 0){
                    push(i);
                    visited[i] = 1;
                }
            }
        }
    }
```

```java
    public static void main(String[] args) {
        mat[0][1] = 1;
        mat[0][2] = 1;
        mat[1][0] = 1;
        mat[1][2] = 1;
        mat[2][0] = 1;
        mat[2][1] = 1;
        mat[2][3] = 1;
        mat[3][2] = 1;

        System.out.println("The adjacency matrix is: ");
        for (int i = 0;i<4;i++){
            for (int j = 0;j<4;j++){
                System.out.print(mat[i][j] + " ");
            }
            System.out.println();
        }

        System.out.println("The depth first search is: ");
        dfs(2);
    }

}
```

Explanation:

- We have created a public class for depth first without recursion.
- We have initialized 1-D array which will act as STACK and 2-D array which will act as adjacency matrix for representing GRAPH. Taking some variables that will do working of STACK functions.
- Finally, a method for DFS is created which will visit the node connected to the vertex and then we will insert the values in STACK which were never visited and then after visiting once those once visited vertex will get popped.
- Finally, MAIN is created where we will call the methods.

OUTPUT:



```
PS D:\CODING\JAVA\VS Code\SEM_3_Assign_2> java .\DFSworecur.java
The adjacency matrix is:
0 1 1 0
1 0 1 0
1 1 0 1
0 0 1 0
The depth first search is:
3 4 2 1
```

THANK YOU