21AIE203

DATA STRUCTURES & ALGORITHMS 2

Assignment 2

M.Kalyana Sundaram
CB.EN.U4AIE21120

DFS with Recursion

```java
import java.util.Iterator;
import java.util.LinkedList;

/**
 * Graph
 */
public class Graph {

    private int V;

    private LinkedList<Integer> adj[];

    Graph(int v){
        V = v;
        adj = new LinkedList[v];
        for (int i = 0; i < v; ++i) {
            adj[i] = new LinkedList();
        }
    }
    void addnode(int v,int w){
        adj[v].add(w);
    }
    void DFS(int v,boolean visited[]){
        visited[v] = true;
        System.out.print(v + " ");

        Iterator<Integer> i = adj[v].listIterator();
        while (i.hasNext()) {
            int n = i.next();
            if(!visited[n])
            DFS(n, visited);
        }
}
void DFS(int v){
    boolean visited[] = new boolean[V];
    DFS(v,visited);
}
public static void main(String args[])
    {
        Graph g=new Graph(4);
        g.addnode(0, 1);
        g.addnode(0, 2);
        g.addnode(1, 2);
        g.addnode(2, 3);
        g.addnode(3, 3);
        System.out.println("Following is Depth First Traversal");
        g.DFS(0);
    }
}
```

1. *We create a graph with n vertices and no edges.*
2. *We add an edge to the graph by appending w to the list of v.*
3. *We can iterate over the list of v by using the following code:*

```
for (int w : G.adj(v)) {
   // do something with w
}
```

1. *Mark the current node as visited and print it*
2. *Recur for all the vertices adjacent to this vertex*
3. *If a adjacent has not been visited, then recur on that adjacent*

```
 ~                                                                    at 11:51:45 AM
 cd /home/kalyan/Documents/University/Sem-3-Assignments/DSA2/Assignment\ 2 ; /
usr/bin/env /usr/lib/jvm/java-19-jdk/bin/java --enable-preview -XX:+ShowCodeDeta
ilsInExceptionMessages -cp /home/kalyan/.config/Code/User/workspaceStorage/2c401
9d314ba7c649381f2e588121349/redhat.java/jdt_ws/Assignment\ 2_8f6b698d/bin Graph
Following is Depth First Traversal
0 1 2 3 %

 ~/Doc/U/Se/DSA2/Assignment 2                                         at 11:51:58 AM
```

DFS Without Recursion

```java
import java.util.LinkedList;
import java.util.Stack;

public class DFSnorecu {
    private int V;

    private LinkedList<Integer> adj[];

    DFSnorecu(int v){
        V = v;
        adj = new LinkedList[v];
        for (int i = 0; i < v; ++i) {
            adj[i] = new LinkedList();
        }
    }
    void addnode(int v,int w){
        adj[v].add(w);
    }
    //dfs without using recussion
    void DFS(int v){
        boolean visited[] = new boolean[V];
        Stack<Integer> stack = new Stack<>();
        stack.push(v);
        while(!stack.isEmpty()){
            v = stack.pop();
            if(!visited[v]){
                visited[v] = true;
                System.out.print(v + " ");
            }
            for (Integer integer : adj[v]) {
                if(!visited[integer]){
                    stack.push(integer);
                }
            }
        }
    }
    public static void main(String[] args) {
        DFSnorecu g = new DFSnorecu(4);
        g.addnode(0, 1);
        g.addnode(0, 2);
        g.addnode(1, 2);
        g.addnode(2, 3);
        System.out.println("Following is ");
        g.DFS(0);
    }
}

```

1. V is the number of vertices in the graph. 2. Adj is the linked list of each vertex, which contains nodes that are adjacent to the vertex. 3. The addnode function is used to add a node to the adjacency list of the vertex.

1. Create a stack and push the root node in it. Also, create a boolean array to mark the visited nodes.
2. Pop the top node from the stack and print it.
3. Push all the adjacent nodes of the popped node into the stack, if they are not visited already.
4. Repeat steps 2 and 3 until the stack is empty.

```
~/Doc/U/Se/DSA2/Assignment 2 ✔ at 11:53:18 AM
cd /home/kalyan/Documents/University/Sem-3-Assignments/DSA2/Assignment\ 2 ; /usr/bin/env /usr/lib/jvm/java-19-jdk/bin/java --enable-preview -XX:+ShowCodeDetailsInExceptionMessages -cp /home/kalyan/.config/Code/User/workspaceStorage/2c4019d314ba7c649381f2e588121349/redhat.java/jdt_ws/Assignment\ 2_8f6b698d/bin DFSnorecu
Following is
0 2 3 1 %

~/Doc/U/Se/DSA2/Assignment 2 ✔ at 11:53:20 AM
```

BFS Without Recursion

```java
import java.util.LinkedList;

public class BFSnorec {
    int A;

    LinkedList<Integer> adj[];

    BFSnorec(int v){
        A = v;
        adj = new LinkedList[v];
        for (int i = 0; i < v; ++i) {
            adj[i] = new LinkedList();
        }
    }
    void addnode(int v,int w){
        adj[v].add(w);
    }
    //dfs without using recussion using queue
    void BFS(int v){
        boolean visited[] = new boolean[A];
        LinkedList<Integer> queue = new LinkedList<>();
        queue.add(v);
        while(!queue.isEmpty()){
            v = queue.poll();
            if(!visited[v]){
                visited[v] = true;
                System.out.print(v + " ");
            }
            for (Integer integer : adj[v]) {
                if(!visited[integer]){
                    queue.add(integer);
                }
            }
        }
    }
    public static void main(String[] args) {
        BFSnorec g = new BFSnorec(7);
        g.addnode(0, 1);
        g.addnode(0, 2);
        g.addnode(1, 2);
        g.addnode(0,4);
        g.addnode(0,5);
        g.addnode(2,4);
        g.addnode(1,5);
        g.addnode(2, 3);
        g.addnode(1,6);
        System.out.println("Following is ");
        g.BFS(2);
    }
}
```

1. The function BFSnorec is the constructor of the class, which is used to initialize the graph.
2. The function addnode is used to add the edge between two nodes.
3. The function BFSnorec is used to implement the BFS algorithm.

1. We create a boolean array visited[] and initialize all values as false. This array is used to keep track of visited nodes.
2. We create a queue data structure and add the source node to it. The source node is the first node that is visited and hence added to the queue.
3. We loop until the queue is empty.
4. In the loop, we deque a vertex from queue and print it.
5. We then enqueue all adjacent vertices of the dequeued vertex s. If a adjacent has not been visited, then we mark it visited and enqueue it.

```
~ ✔ at 05:02:28 PM
/usr/bin/env /usr/lib/jvm/java-19-jdk/bin/java --enable-preview -XX:+ShowCode
DetailsInExceptionMessages -cp /home/kalyan/.config/Code/User/workspaceStorage/2
c4019d314ba7c649381f2e588121349/redhat.java/jdt_ws/Assignment\ 2_8f6b698d/bin BS
Frec
Following is Breadth First Traversal (starting from vertex 2)
2 4 3 %
~ ✔ at 05:02:29 PM
```

BFS With Recursion

```java
import java.util.Iterator;
import java.util.LinkedList;

public class BSFrec {
    int A;
    LinkedList<Integer> adj[];
    BSFrec(int v){
        A = v;
        adj = new LinkedList[v];
        for (int i = 0; i < v; ++i) {
            adj[i] = new LinkedList();
        }
    }
    void addnode(int v,int w){
        adj[v].add(w);
    }
    void BSFrecursivly(boolean visited[],LinkedList<Integer> queue){
        if(queue.size() == 0){
            return;
        }
        int s = queue.poll();
        System.out.print(s+" ");
        Iterator<Integer> i = adj[s].listIterator();
        while (i.hasNext()){
            int n = i.next();
            if(!visited[n]){
                visited[n] = true;
                queue.add(n);
            }
        }
        BSFrecursivly(visited,queue);
    }
    public static void main(String args[])
    {
        BSFrec g=new BSFrec(7);
        g.addnode(0, 1);
        g.addnode(0, 2);
        g.addnode(1, 2);
        g.addnode(0,4);
        g.addnode(0,5);
        g.addnode(2,4);
        g.addnode(1,5);
        g.addnode(2, 3);
        g.addnode(1,6);
        System.err.println("Following is Breadth First Traversal "+
                "(starting from vertex 2)");
        boolean visited[] = new boolean[g.A];
        LinkedList<Integer> queue = new LinkedList<Integer>();
        visited[0] = true;
        queue.add(2);
        g.BSFrecursivly(visited,queue);
    }
}
```

1. A is the number of vertices
2. adj is a linked list array of size A
3. addnode() function adds an edge to an undirected graph

1. If the queue is empty, we return.
2. We poll the first element from the queue, and print it.
3. We iterate through the list of the element we polled.
4. If the element we iterate through is not visited, we mark it as visited, and add it to the queue.
5. We call the function again, and it will go on until the queue is empty.

```
~ ✔ at 05:03:58 PM
/usr/bin/env /usr/lib/jvm/java-19-jdk/bin/java --enable-preview -XX:+ShowCod
eDetailsInExceptionMessages -cp /home/kalyan/.config/Code/User/workspaceStorage/
2c4019d314ba7c649381f2e588121349/redhat.java/jdt_ws/Assignment\ 2_8f6b698d/bin B
FSnorec
Following is
2 4 3 %

~ ✔ at 05:03:59 PM
```

Thank You