

21AIE203
DATA

STRUCTURES & ALGORITHMS 2

Assignment 2

M.Kalyana Sundaram
CB.EN.U4AIE21120

DFS with Recursion

```
1 import java.util.Iterator;
2 import java.util.LinkedList;
3
4 /**
5  * Graph
6 */
7 public class Graph {
8
9     private int v;
10
11    private LinkedList<Integer> adj[];
12
13    Graph(int v){
14        V = v;
15        adj = new LinkedList[v];
16        for (int i = 0; i < v; ++i) {
17            adj[i] = new LinkedList();
18        }
19    }
20    void addnode(int v,int w){
21        adj[v].add(w);
22    }
23    void DFS(int v,boolean visited[]){
24        visited[v] = true;
25        System.out.print(v + " ");
26
27        Iterator<Integer> i = adj[v].listIterator();
28        while (i.hasNext()) {
29            int n = i.next();
30            if(!visited[n])
31                DFS(n, visited);
32        }
33    }
34    void DFS(int v){
35        boolean visited[] = new boolean[V];
36        DFS(v,visited);
37    }
38    public static void main(String args[])
39    {
40        Graph g=new Graph(4);
41        g.addnode(0, 1);
42        g.addnode(0, 2);
43        g.addnode(1, 2);
44        g.addnode(2, 3);
45        g.addnode(3, 3);
46        System.out.println("Following is Depth First Traversal");
47        g.DFS(0);
48    }
49 }
```



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

The terminal window shows the following command and output:

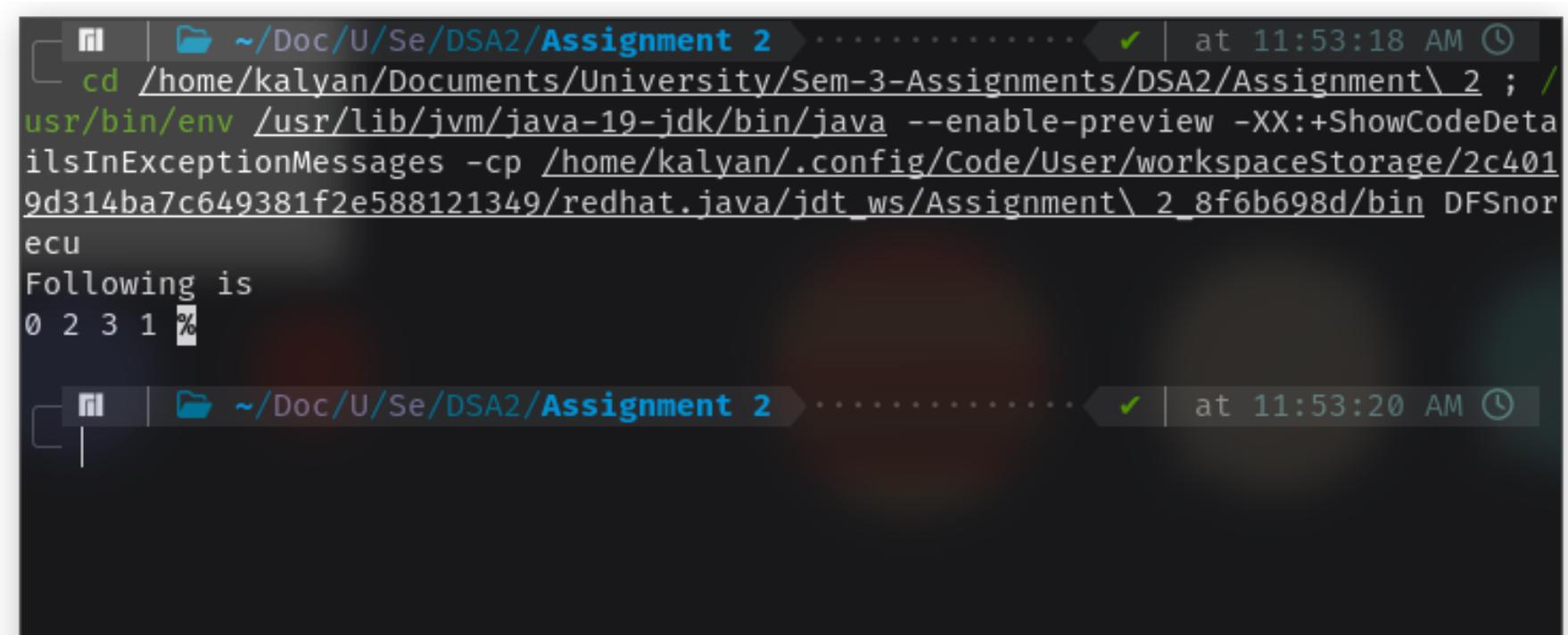
```
cd /home/kalyan/Documents/University/Sem-3-Assigments/DSA2/Assignment_2 ; /usr/bin/env /usr/lib/jvm/java-19-jdk/bin/java --enable-preview -XX:+ShowCodeDetailsInExceptionMessages -cp /home/kalyan/.config/Code/User/workspaceStorage/2c4019d314ba7c649381f2e588121349/redhat.java/jdt_ws/Assignment_2_8f6b698d/bin Graph
Following is Depth First Traversal
0 1 2 3 %
```

DFS Without Recursion

```
1 import java.util.LinkedList;
2 import java.util.Stack;
3
4 public class DFSnorecu {
5     private int V;
6
7     private LinkedList<Integer> adj[];
8
9     DFSnorecu(int v){
10         V = v;
11         adj = new LinkedList[v];
12         for (int i = 0; i < v; ++i) {
13             adj[i] = new LinkedList();
14         }
15     }
16     void addnode(int v,int w){
17         adj[v].add(w);
18     }
19     //dfs without using recursion
20     void DFS(int v){
21         boolean visited[] = new boolean[V];
22         Stack<Integer> stack = new Stack<>();
23         stack.push(v);
24         while(!stack.isEmpty()){
25             v = stack.pop();
26             if(!visited[v]){
27                 visited[v] = true;
28                 System.out.print(v + " ");
29
30                 for (Integer integer : adj[v]) {
31                     if(!visited[integer]){
32                         stack.push(integer);
33                     }
34                 }
35             }
36         }
37         public static void main(String[] args) {
38             DFSnorecu g = new DFSnorecu(4);
39             g.addnode(0, 1);
40             g.addnode(0, 2);
41             g.addnode(1, 2);
42             g.addnode(2, 3);
43             System.out.println("Following is ");
44             g.DFS(0);
45         }
46     }
47 }
```

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-Assessments/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 %

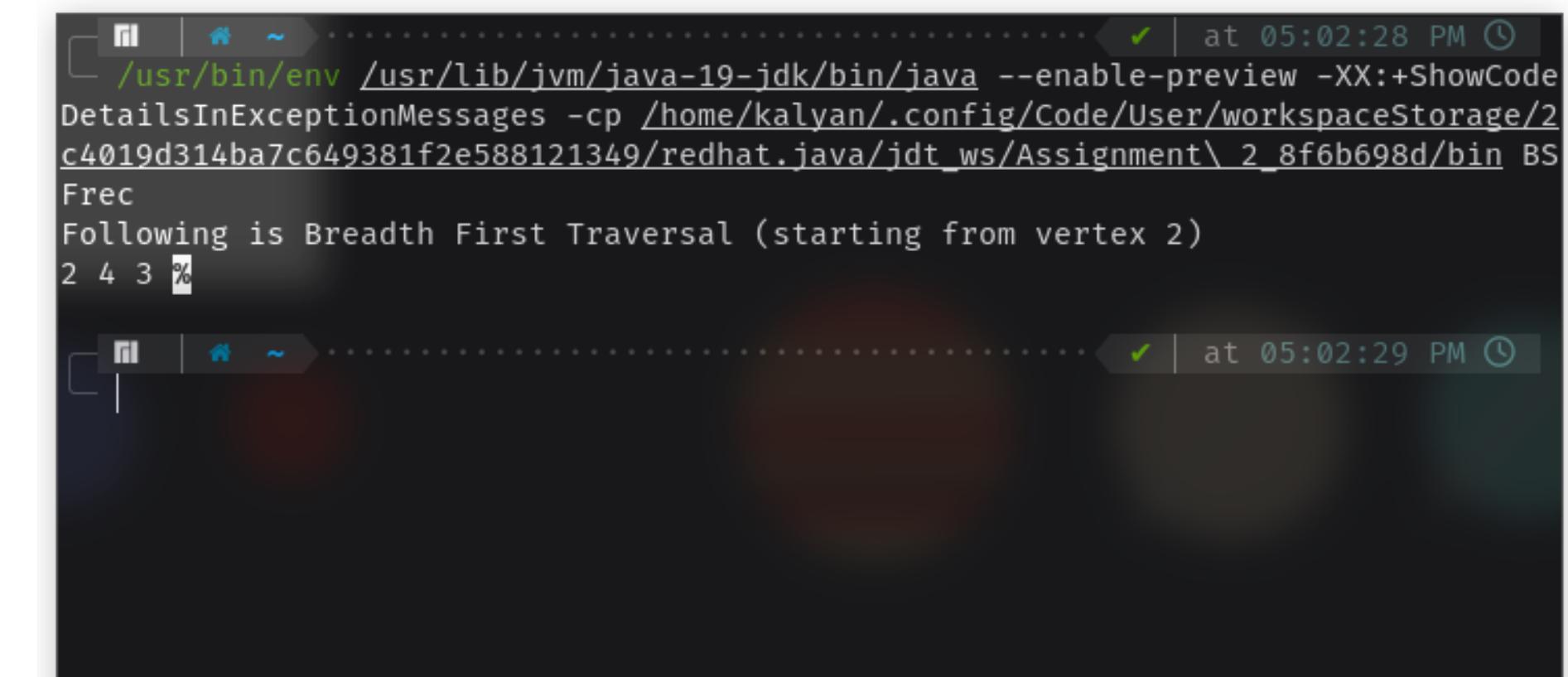
~ /Doc/U/Se/DSA2/Assignment 2 ..... at 11:53:20 AM
```

BFS Without Recursion

```
1 import java.util.LinkedList;
2
3 public class BFSnorec {
4     int A;
5
6     LinkedList<Integer> adj[];
7
8     BFSnorec(int v){
9         A = v;
10        adj = new LinkedList[v];
11        for (int i = 0; i < v; ++i) {
12            adj[i] = new LinkedList();
13        }
14    }
15    void addnode(int v,int w){
16        adj[v].add(w);
17    }
18    //dfs without using recursion using queue
19    void BFS(int v){
20        boolean visited[] = new boolean[A];
21        LinkedList<Integer> queue = new LinkedList<>();
22        queue.add(v);
23        while(!queue.isEmpty()){
24            v = queue.poll();
25            if(!visited[v]){
26                visited[v] = true;
27                System.out.print(v + " ");
28
29                for (Integer integer : adj[v]) {
30                    if(!visited[integer]){
31                        queue.add(integer);
32                    }
33                }
34            }
35        }
36    public static void main(String[] args) {
37        BFSnorec g = new BFSnorec(7);
38        g.addnode(0, 1);
39        g.addnode(0, 2);
40        g.addnode(1, 2);
41        g.addnode(0,4);
42        g.addnode(0,5);
43        g.addnode(2,4);
44        g.addnode(1,5);
45        g.addnode(2, 3);
46        g.addnode(1,6);
47        System.out.println("Following is ");
48        g.BFS(2);
49    }
50 }
```

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.

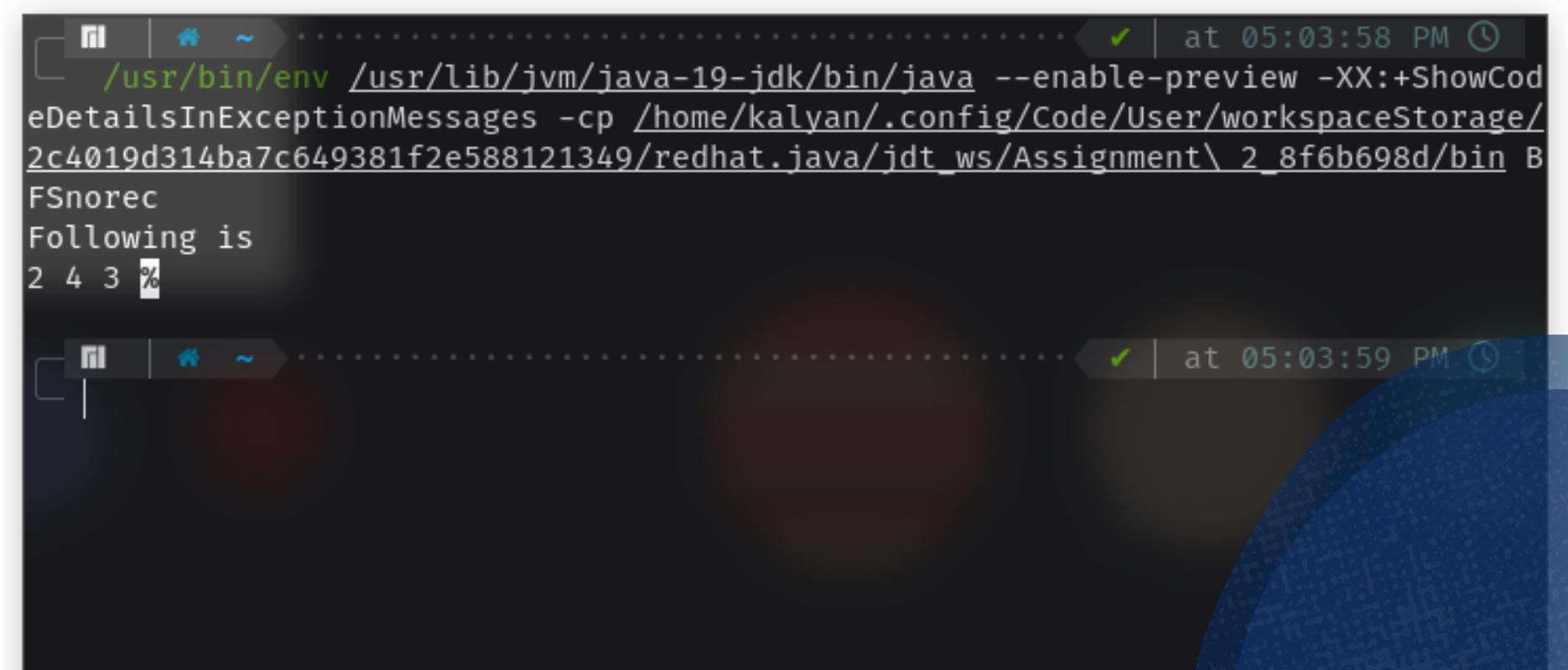


```
/usr/bin/env /usr/lib/jvm/java-19-jdk/bin/java --enable-preview -XX:+ShowCodeDetailsInExceptionMessages -cp /home/kalyan/.config/Code/User/workspaceStorage/2c4019d314ba7c649381f2e588121349/redhat.java/jdt_ws/Assignment\ 2_8f6b698d/bin BS Frec
Following is Breadth First Traversal (starting from vertex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

BFS With Recursion

```
1 import java.util.Iterator;
2 import java.util.LinkedList;
3
4 public class BSFrec {
5     int A;
6     LinkedList<Integer> adj[];
7     BSFrec(int v){
8         A = v;
9         adj = new LinkedList[v];
10        for (int i = 0; i < v; ++i) {
11            adj[i] = new LinkedList();
12        }
13    }
14    void addnode(int v,int w){
15        adj[v].add(w);
16    }
17    void BSFrecursively(boolean visited[],LinkedList<Integer> queue){
18        if(queue.size() == 0){
19            return;
20        }
21        int s = queue.poll();
22        System.out.print(s+" ");
23        Iterator<Integer> i = adj[s].listIterator();
24        while (i.hasNext()){
25            int n = i.next();
26            if(!visited[n]){
27                visited[n] = true;
28                queue.add(n);
29            }
30        }
31        BSFrecursively(visited,queue);
32    }
33    public static void main(String args[])
34    {
35        BSFrec g=new BSFrec(7);
36        g.addnode(0, 1);
37        g.addnode(0, 2);
38        g.addnode(1, 2);
39        g.addnode(0,4);
40        g.addnode(0,5);
41        g.addnode(2,4);
42        g.addnode(1,5);
43        g.addnode(2, 3);
44        g.addnode(1,6);
45        System.err.println("Following is Breadth First Traversal "+
46                            "(starting from vertex 2)");
47        boolean visited[] = new boolean[g.A];
48        LinkedList<Integer> queue = new LinkedList<Integer>();
49        visited[0] = true;
50        queue.add(2);
51        g.BSFrecursively(visited,queue);
52    }
53 }
```

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.



```
/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 BFSnrec
Following is
2 4 3 %
```

The background features several overlapping circles in shades of pink and purple, creating a soft, layered effect.

Thank You