**Breadth First Search**

**Approach:**

1. For Graph as well we will use the Queue for per­form­ing the BFS.
2. We will use the **boolean[]** to keep a track of the nodes because unlike tree dur­ing tra­ver­sal we might keep mov­ing into the cir­cles by vis­it­ing same nodes repeatedly.
3. In our exam­ple we are using adja­cency List for the Graph Rep­re­sen­ta­tion.



Breadth-First Search (Tra­ver­sal)  in a Graph is quite sim­i­lar to Binary Tree.

…………………………………………………………………………………………………..

 import java.util.LinkedList;

 import java.util.Queue;

 public class GraphBFS {

 public static void main(String args[]) {

 Graph g = new Graph(6);

 g.addEdge(0, 1);

 g.addEdge(0, 2);

 g.addEdge(1, 2);

 g.addEdge(1, 3);

 g.addEdge(3, 4);

 g.addEdge(2, 3);

 g.addEdge(4, 0);

 g.addEdge(4, 1);

 g.addEdge(4, 5);

 g.BFS(0);

 }

 }

 class Node {

 int dest;

 Node next;

 public Node(int d) {

 dest = d;

 next = null;

 }

 }

 class adjList {

 Node head;

 }

 class Graph {

 int V;

 adjList[] array;

 public Graph(int V) {

 this.V = V;

 array = new adjList[V]; // linked lists = number of Nodes in Graph

 for (int i = 0; i < V; i++) {

 array[i] = new adjList();

 array[i].head = null;

 }

 }

 public void addEdge(int src, int dest) {

 Node n = new Node(dest);

 n.next = array[src].head;

 array[src].head = n;

 }

 public void BFS(int startVertex) {

 boolean[] visited = new boolean[V];

 Queue<Integer> s = new LinkedList<Integer>();

 s.add(startVertex);

 while (s.isEmpty() == false) {

 int n = s.poll();

 System.out.print(" " + n);

 visited[n] = true;

 Node head = array[n].head;

 while (head != null) {

 if (visited[head.dest] == false) {

 s.add(head.dest);

 visited[head.dest] = true;

 }

 head = head.next;

 }

 }

 }

 }