**Practical No:1**

**Name :Mansi Kirloskar**

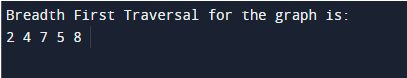
**Roll No:38**

**BFS(Breadth first serch)**

**Code:**

1. **import** java.io.\*;
2. **import** java.util.\*;
3. **public** **class** BFSTraversal
4. {
5. **private** **int** vertex;       /\* total number number of vertices in the graph \*/
6. **private** LinkedList<Integer> adj[];      /\* adjacency list \*/
7. **private** Queue<Integer> que;           /\* maintaining a queue \*/
8. BFSTraversal(**int** v)
9. {
10. vertex = v;
11. adj = **new** LinkedList[vertex];
12. **for** (**int** i=0; i<v; i++)
13. {
14. adj[i] = **new** LinkedList<>();
15. }
16. que = **new** LinkedList<Integer>();
17. }
18. **void** insertEdge(**int** v,**int** w)
19. {
20. adj[v].add(w);      /\* adding an edge to the adjacency list (edges are bidirectional in this example) \*/
21. }
22. **void** BFS(**int** n)
23. {
24. **boolean** nodes[] = **new** **boolean**[vertex];       /\* initialize boolean array for holding the data \*/
25. **int** a = 0;
26. nodes[n]=**true**;
27. que.add(n);       /\* root node is added to the top of the queue \*/
28. **while** (que.size() != 0)
29. {
30. n = que.poll();        /\* remove the top element of the queue \*/
31. System.out.print(n+" ");    /\* print the top element of the queue \*/
32. **for** (**int** i = 0; i < adj[n].size(); i++)  /\* iterate through the linked list and push all neighbors into queue \*/
33. {
34. a = adj[n].get(i);
35. **if** (!nodes[a])      /\* only insert nodes into queue if they have not been explored already \*/
36. {
37. nodes[a] = **true**;
38. que.add(a);
39. }
40. }
41. }
42. }
43. **public** **static** **void** main(String args[])
44. {
45. BFSTraversal graph = **new** BFSTraversal(10);
46. graph.insertEdge(0, 1);
47. graph.insertEdge(0, 2);
48. graph.insertEdge(0, 3);
49. graph.insertEdge(1, 3);
50. graph.insertEdge(2, 4);
51. graph.insertEdge(3, 5);
52. graph.insertEdge(3, 6);
53. graph.insertEdge(4, 7);
54. graph.insertEdge(4, 5);
55. graph.insertEdge(5, 2);
56. graph.insertEdge(6, 5);
57. graph.insertEdge(7, 5);
58. graph.insertEdge(7, 8);
59. System.out.println("Breadth First Traversal for the graph is:");
60. graph.BFS(2);
61. }
62. }

**Output:**



**DFS(Depth first search)**

**Code:**

1. **import** java.util.\*;
2. **class** DFSTraversal {
3. **private** LinkedList<Integer> adj[]; /\*adjacency list representation\*/
4. **private** **boolean** visited[];
5. /\* Creation of the graph \*/
6. DFSTraversal(**int** V) /\*'V' is the number of vertices in the graph\*/
7. {
8. adj = **new** LinkedList[V];
9. visited = **new** **boolean**[V];
10. **for** (**int** i = 0; i < V; i++)
11. adj[i] = **new** LinkedList<Integer>();
12. }
13. /\* Adding an edge to the graph \*/
14. **void** insertEdge(**int** src, **int** dest) {
15. adj[src].add(dest);
16. }
17. **void** DFS(**int** vertex) {
18. visited[vertex] = **true**; /\*Mark the current node as visited\*/
19. System.out.print(vertex + " ");
20. Iterator<Integer> it = adj[vertex].listIterator();
21. **while** (it.hasNext()) {
22. **int** n = it.next();
23. **if** (!visited[n])
24. DFS(n);
25. }
26. }
27. **public** **static** **void** main(String args[]) {
28. DFSTraversal graph = **new** DFSTraversal(8);
29. graph.insertEdge(0, 1);
30. graph.insertEdge(0, 2);
31. graph.insertEdge(0, 3);
32. graph.insertEdge(1, 3);
33. graph.insertEdge(2, 4);
34. graph.insertEdge(3, 5);
35. graph.insertEdge(3, 6);
36. graph.insertEdge(4, 7);
37. graph.insertEdge(4, 5);
38. graph.insertEdge(5, 2);
39. System.out.println("Depth First Traversal for the graph is:");
40. graph.DFS(0);
41. }
42. }

Output:

