Program and Output

1. **Breadth First Search (BFS)**

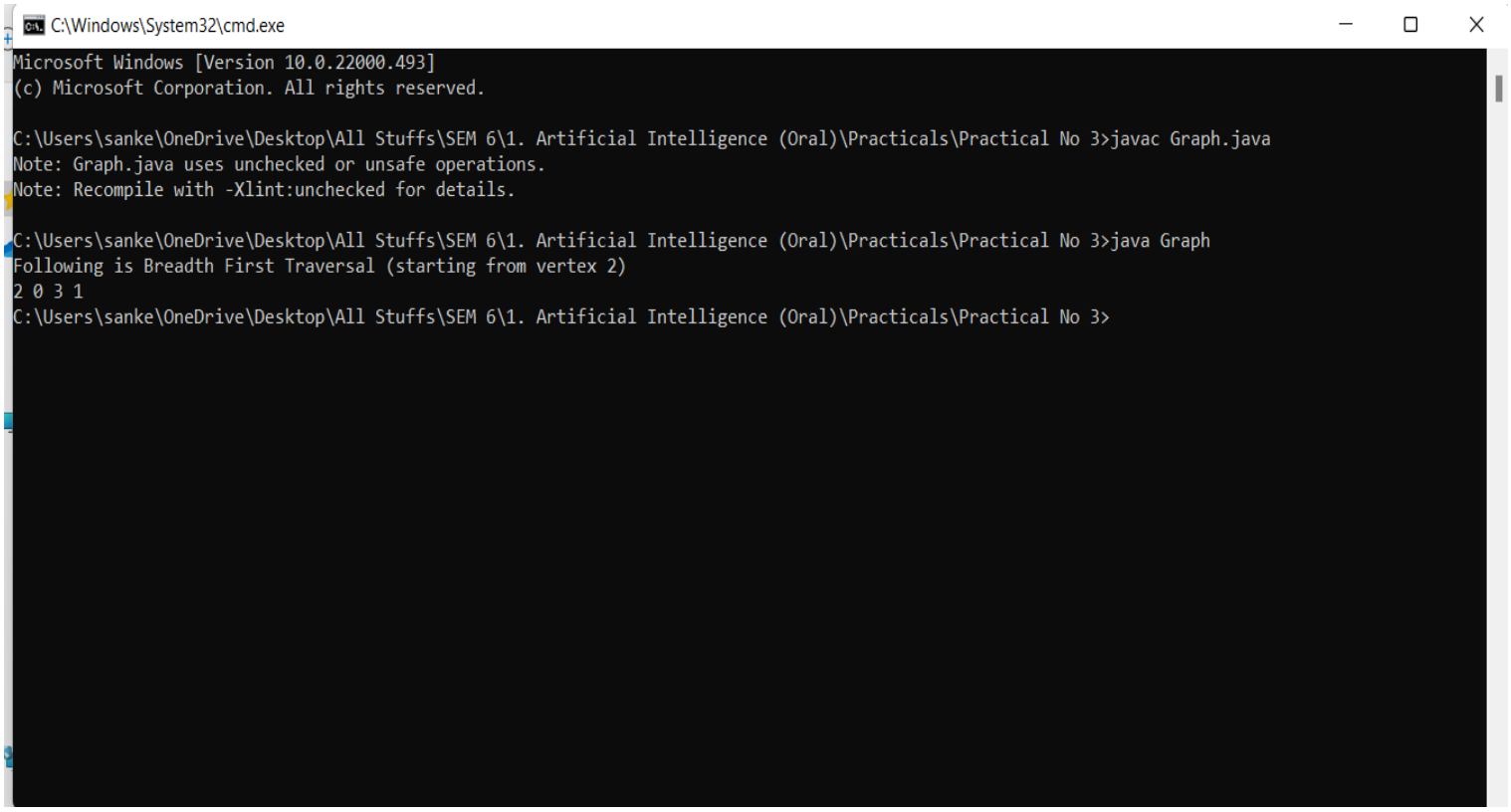
Program :

```
// Java program to print BFS traversal from a given source vertex.
// BFS(int s) traverses vertices reachable from s.

import java.io.*;
import java.util.*;

// This class represents a directed graph using adjacency list
// representation

public class Graph
{
   private int V;   // No. of vertices
   private LinkedList<Integer> adj[]; //Adjacency Lists

   // Constructor
   Graph(int v)
   {
      V = v;
      adj = new LinkedList[v];
      for (int i=0; i<v; ++i)
         adj[i] = new LinkedList();
   }



// Function to add an edge into the graph

   void addEdge(int v,int w)
   {
      adj[v].add(w);
   }

// prints BFS traversal from a given source s
```

```
void BFS(int s)
{
   // Mark all the vertices as not visited(By default
   // set as false)

   boolean visited[] = new boolean[V];

   // Create a queue for BFS
   LinkedList<Integer> queue = new LinkedList<Integer>();

   // Mark the current node as visited and enqueue it

   visited[s]=true;
   queue.add(s);
   while (queue.size() != 0)
   {
      // Dequeue a vertex from queue and print it
      s = queue.poll();
      System.out.print(s+" ");
      // Get all adjacent vertices of the dequeued vertex s
      // If a adjacent has not been visited, then mark it
      // visited and enqueue it

      Iterator<Integer> i = adj[s].listIterator();
      while (i.hasNext())
      {
         int n = i.next();
         if (!visited[n])
         {
            visited[n] = true;
            queue.add(n);
         }
      }

   }


}

// Driver method to
```

```
public static void main(String args[])
{
   Graph g = new Graph(4);
   g.addEdge(0, 1);
   g.addEdge(0, 2);
   g.addEdge(1, 2);
   g.addEdge(2, 0);
   g.addEdge(2, 3);
   g.addEdge(3, 3);

   System.out.println("Following is Breadth First Traversal "+
                      "(starting from vertex 2)");
   g.BFS(2);
}
}
```

Output :



2. Depth First Search (DFS)

Program :

```
// Java program to print DFS
// mtraversal from a given given
// graph
import java.io.*;
import java.util.*;

// This class represents a
// directed graph using adjacency
// list representation
public class DGraph
{
   private int V; // No. of vertices

   // Array  of lists for
   // Adjacency List Representation
   private LinkedList<Integer> adj[];

   // Constructor
   @SuppressWarnings("unchecked") DGraph(int v)
   {
      V = v;
      adj = new LinkedList[v];
      for (int i = 0; i < v; ++i)
         adj[i] = new LinkedList();
   }

   // Function to add an edge into the graph
   void addEdge(int v, int w)
   {
      adj[v].add(w); // Add w to v's list.
   }

   // A function used by DFS
   void DFSUtil(int v, boolean visited[])
   {
      // Mark the current node as visited and print it
```

```
    visited[v] = true;
    System.out.print(v + " ");

    // Recur for all the vertices adjacent to this
    // vertex
    Iterator<Integer> i = adj[v].listIterator();
    while (i.hasNext()) {
        int n = i.next();
        if (!visited[n])
            DFSUtil(n, visited);
    }
}

// The function to do DFS traversal.
// It uses recursive
// DFSUtil()
void DFS(int v)
{
    // Mark all the vertices as
    // not visited(set as
    // false by default in java)
    boolean visited[] = new boolean[V];

    // Call the recursive helper
    // function to print DFS
    // traversal
    DFSUtil(v, visited);
}

// Driver Code
public static void main(String args[])
{
    DGraph g = new DGraph(4);

    g.addEdge(0, 1);
    g.addEdge(0, 2);
    g.addEdge(1, 2);
    g.addEdge(2, 0);
    g.addEdge(2, 3);
    g.addEdge(3, 3);
```

```
        System.out.println(
            "Following is Depth First Traversal "
            + "(starting from vertex 2) : ");

        g.DFS(2);
    }
}
```

Output: