```
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.util.Scanner;
import java.util.ArrayList;

class Assignment7{
    public static void main(String[] args) {
        //using https://www.cs.usfca.edu/~galles/visualization/DFS.html the print_Graph() function
prints what they call an adjacency matrix and not the adjacency list
        // Both adjacency list and adjacency matrix representationsare printed to be safe
        // in This file the only change made to Adj_List_Graph is to make it a static class
        /*the adjacency matrix of the example in Programming Task
        Should Display
        0 1 1 0
        0 0 1 1
        0 0 0 1
        0 0 0 0
         */
        //example given in Programming Task
        Adj_List_Graph Example_1 = create_Graph(new int[][]{
            {0,1,0,0},
            {0,0,1,0},
            {0,0,0,1},
            {0,0,0,0}
        });

        Adj_List_Graph input7_1 = inputFile("input-7-1.txt");
        Adj_List_Graph input7_2 = inputFile("input-7-2.txt");

        System.err.print("\nExample 1: ");
            Example_1.printGraph();
            Example_1.printList();
        System.err.print("\nComputed Adjacency Matrix of Example 1: ");
            Compute_AdjacencyList(Example_1).printGraph();

        System.err.print("\nComputed Adjacency List of Example 1: ");
            Compute_AdjacencyList(Example_1).printList();


        System.err.print("\n\ninput-7-1.txt: ");
          input7_1.printGraph();
          input7_1.printList();
        System.err.print("\nComputed Adjacency Matrix of input-7-1.txt: ");
            Compute_AdjacencyList(input7_1).printGraph();
        System.err.print("\nComputed Adjacency List of input-7-1.txt: ");
            Compute_AdjacencyList(input7_1).printList();

        System.err.print("\n\ninput-7-2.txt: ");
            input7_2.printGraph();
            input7_2.printList();
        System.err.print("\nComputed Adjacency Matrix of input-7-2.txt: ");
            Compute_AdjacencyList(input7_2).printGraph();
        System.err.print("\nComputed Adjacency List of input-7-2.txt: ");
            Compute_AdjacencyList(input7_2).printList();
    }

    /** reads the file and converts it to an {@link Adj_List_Graph} */
    public static Adj_List_Graph inputFile(String filename) {
        try (Scanner console = new Scanner(new FileReader(filename))) {
            //node count
            int N = console.nextInt();

            //birth of graph of size N
            final Adj_List_Graph GRAPH = new Adj_List_Graph(N);
```

```
            //add edges
            for (int u = 0 ; u < N;u++)
                for (int v = 0; v < N; v++)
                    GRAPH.addEdge(u, console.nextInt()); //(u,v) u is the head , v is a single link

            return GRAPH;
        }catch(FileNotFoundException e) {
            System.err.println("File not found: '" + filename + "'");
        }

        return null;
    }

    /** computes the adjacency list of a <b>directed</b> graph
     * @return G2
     */
    public static Adj_List_Graph Compute_AdjacencyList(final Adj_List_Graph G) {
        final int N = G.n;

        final Adj_List_Graph G2 = new Adj_List_Graph(N);

        //copying G
        for (int u = 0; u < N; u++)
            for (int v : G.adj.get(u))
                G2.addEdge(u, v);

        //calculating G2
        int i,j;
        for (int k = 0; k < N*N; k++) {
            i = k / N; //calc row [i][?]
            j = k % N; //calc column [?][j]

            if (G.adj.get(i).get(j) == 1) {  // If there's an edge at [i][j]
                for (int v = 0; v < N; v++)
                    if (G.adj.get(j).get(v) == 1 && G.adj.get(i).get(v) == 0)  //if there's an edge
at [j][v] but not at [i][v] then the path length is atleast 1
                        G2.adj.get(i).set(v,1);  //replaces [i][v] with 1
            }
        }
       return G2;
    }


    /** exists for manual testing */

    public static Adj_List_Graph create_Graph(final int[][] MATRIX){
        Adj_List_Graph GRAPH = new Adj_List_Graph(MATRIX.length);

        final int N = MATRIX.length;

        for (int u = 0; u< N; u++)
            for (int v = 0; v < N; v++)
                GRAPH.addEdge(u, MATRIX[u][v]);//(u,v) u is the head , v is a single link

        return GRAPH;
    }



    public static class Adj_List_Graph{
        int n;  // no of nodes
        ArrayList<ArrayList<Integer> > adj;

        //constructor taking as the single parameter the number of nodes
```

```
        Adj_List_Graph(int no_nodes) {
          n = no_nodes;
          adj =  new ArrayList<ArrayList<Integer> >(n);
          for (int i = 0; i < n; i++)
            adj.add(new ArrayList<Integer>());

        }


        // A utility function to add an edge in an
        // undirected graph; for directed graph remove the second line
        public void addEdge(int u, int v)
        {
          adj.get(u).add(v);
          // adj.get(v).add(u);  //this line should be un-commented, if graph is undirected
        }

        /** A utility function to print the adjacency <strike>list</strike> <b>Matrix</b>
representation of graph */
        //this function was not changed
        public void printGraph()
        {
          for (int i = 0; i < n; i++) {
            System.out.println("\nAdjacency matrix of vertex" + i);
            System.out.print("head");
            for (int j = 0; j < adj.get(i).size(); j++) {
              System.out.print(" -> "+adj.get(i).get(j));
            }
            System.out.println();
          }
        }

        /** A utility function to print the adjacency <b>list</b> representation of graph */
        //this function was added
        public void printList(){

          int vertex = 0;

          for (ArrayList <Integer> u : this.adj) {
              System.out.printf("\n%d: ", vertex);

              for (int v = 0; v < u.size(); v++){
                  if (u.get(v) != 0)
                      System.out.print(" -> " + v);

              }
              System.out.println();

              vertex++;
          }
        }

      }

}
```