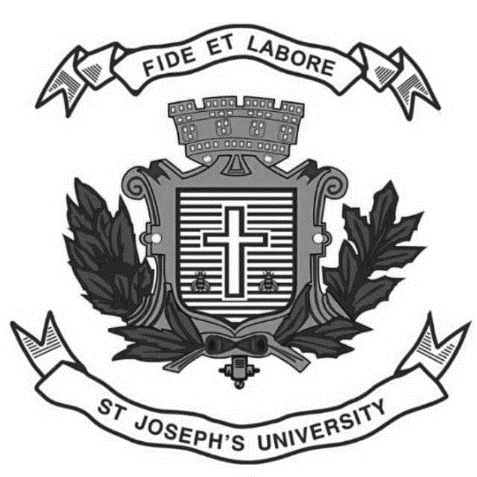
**St. Joseph’s UNIVERSITY**

**Bangalore-560027**



**Estd-1882**

**COMPUTER SCIENCE**

**PRACTICAL RECORD**

**(DESIGN AND ANALYSIS OF ALGORITHM)**

**Under the guidance of Submitted by:**

**1. Prof. Sandhya Student Name: Shashank**

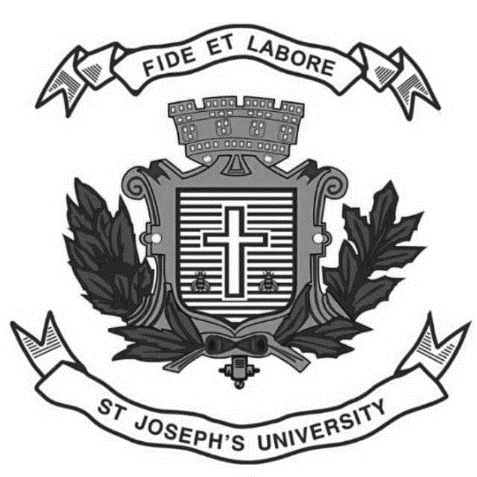
**2. Prof. Bojamma Roll No: 222CSC33**

**3. Prof. Sarakutty**

**4. Prof. Mrinmoyee**

**ST. JOSEPH’S UNIVERSITY**

**BANGALORE**



LABORATORY CERTIFICATE

This is to certify that, Sri. Shashank has satisfactorily completed the course of laboratory assignments in Design and Analysis of Algorithms prescribed by St. Joseph’s University for the First Semester Master’s degree course in Computer Science for the year 2022-23.

Name: Shashank Teacher In-Charge

Reg. No.: 222CSC33

Date of Exam: 18/11/2022 Head of the Department/PG Coordinator

***INDEX***

|  |  |  |
| --- | --- | --- |
| **S. No.** | **Program Names** | **Page No.** |
| 1. | Write a program to implement linear search for a set of given elements. | 02 |
| 2. | Write a program to implement binary search using divide and conquer technique. | 05 |
| 3. | Write a program to implement bubble sort for a set of given elements. | 08 |
| 4. | Write a program using selection sort to arrange n numbers. | 11 |
| 5. | Write a program to perform insertion sort to reorder n numbers in ascending order. | 14 |
| 6. | Write a program to perform quick sort using divide and conquer technique to arrange n numbers. | 17 |
| 7. | Write a program to perform merge sort using divide and conquer  technique to arrange n numbers. | 21 |
| 8. | Write a program to implement push and pop operations in a stack. | 26 |
| 9 | Write a program to implement insert and delete operations in a queue. | 32 |
| 10. | Write a program to find Maximum and Minimum elements using  Divide and Conquer techniques. | 38 |
| 11. | Write a program to implement 0/1 Knapsack Problem using Dynamic Programming. | 42 |
| 12. | Write a program to display the Degree of Vertices of a Edge Matrix for a particular graph G. | 46 |
| 13. | Write a program to find the shortest path for a given vertex in a  weighted connected graph using Dijkstra's Algorithm. | 49 |
| 14. | Write a program to find the minimum cost spanning tree of a  given undirected graph using Kruskal’s Algorithm. | 53 |
| 15. | Write a program to print all reachable nodes from a given starting  node in a digraph using BFS Algorithm. | 58 |
| 16. | Write a program to check whether a given graph is connected or is not  connected using DFS Algorithm. | 61 |
| 17. | Write a program to compute the transitive closure of a given directed  graph using Warshall’s Algorithm. | 64 |
| 18. | Write a program to implement all pair shortest path’s problem using Floyd’s Algorithm. | 67 |

1. **Write a program to implement linear search for a set of given elements**

package Final\_printable\_code;

import java.util.ArrayList;

import java.util.Scanner;

public class LinearSearch {

    public static void linearSearch(int element, int[] array){

        boolean flag=false;

        for(int i=0;i<array.length;i++){

            if(element==array[i]){

                flag=true;

                System.out.println("Element found in the index "+i);

                break;

            }

        }

        if(flag==false) System.out.println("element not found");

    }

    //method overload for multi-linear search

    public static void linearSearch(int element, int[] array, boolean multiple){

        if(multiple){

            ArrayList<Integer> newArr = new ArrayList<>();

            for(int i=0;i<array.length;i++){

                if(element==array[i]){

                    newArr.add(i);

                }

            }

            if(newArr.isEmpty())

                System.out.println("element not found");

            else

                System.out.println("element found in");

                System.out.println(newArr.toString());

        }

        else{

            linearSearch(element, array);

        }

    }

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        System.out.println("Enter the number of elements in the array");

        int n = sc.nextInt();

        int[] array = new int[n];

        System.out.println("Enter the elements of the array");

        for(int i =0;i<n;i++){

            array[i] = sc.nextInt();

        }

        System.out.println("Enter the element to be searched in the array");

        int element=sc.nextInt();

        linearSearch(element, array);

    }

}

1. **Write a program to implement binary search using divide and conquer technique.**

import java.util.Arrays;

import java.util.Scanner;

public class BinarySearch {

    private static int binarySearchInternal(int element, int[] array, int min, int max) {

        int mid = (min + max) / 2;

        if (max >= min) {

            if (array[mid] == element) {

                return mid;

            }

            if (array[mid] > element) {

                max = mid - 1;

                return binarySearchInternal(element, array, min, max);

            } else {

                min = mid + 1;

                return binarySearchInternal(element, array, min, max);

            }

        }

        return -1;

    }

    public static void binarySearch(int element, int[] array) {

        int min = 0;

        int max = array.length - 1;

        Arrays.sort(array);

        int pos = binarySearchInternal(element, array, min, max);

        if (pos == -1) {

            System.out.println("Element not found");

        } else {

            System.out.println("Element found in position " + pos);

        }

    }

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        System.out.println("Enter the number of elements in the array");

        int n = sc.nextInt();

        int[] array = new int[n];

        System.out.println("Enter the elements of the array");

        for (int i = 0; i < n; i++) {

            array[i] = sc.nextInt();

        }

        System.out.println("Enter the element to be searched in the array");

        int element = sc.nextInt();

        binarySearch(element, array);

    }

}

1. **Write a program to implement bubble sort for a set of given elements.**

package Final\_printable\_code;

import java.util.Arrays;

import java.util.Scanner;

public class BubbleSort {

    public static int[] bubbleSort(int[] unsorted) {

        System.out.println("Using bubble sort");

        for (int i = 0; i < unsorted.length; i++) {

            boolean swapeed = false;

            for (int j = 0; j < unsorted.length - 1 - i; j++) {

                if (unsorted[j] > unsorted[j + 1]) {

                    int t = unsorted[j];

                    unsorted[j] = unsorted[j + 1];

                    unsorted[j + 1] = t;

                    swapeed = true;

                }

            }

            if (!swapeed) {

                break;

            }

        }

        return (unsorted);

    }

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        System.out.println("Enter the number of elements in the array");

        int n = sc.nextInt();

        int[] array = new int[n];

        System.out.println("Enter the elements of the array");

        for (int i = 0; i < n; i++) {

            array[i] = sc.nextInt();

        }

        System.out.println("sorted array is " + Arrays.toString(array));

    }

}

1. **Write a program using selection sort to arrange n numbers.**

package Final\_printable\_code;

import java.util.Arrays;

import java.util.Scanner;

public class SelectionSort {

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        System.out.println("Enter the number of elements in the array");

        int n = sc.nextInt();

        int[] array = new int[n];

        System.out.println("Enter the elements of the array");

        for (int i = 0; i < n; i++) {

            array[i] = sc.nextInt();

        }

        System.out.println("sorted array is " + Arrays.toString(selectionSort(array)));

    }

    public static int[] selectionSort(int[] unsorted) {

        System.out.println("Using selection sort");

        for(int i=0;i<unsorted.length;i++){

            int min\_index=i;

            for(int j=i+1;j<unsorted.length;j++){

                if(unsorted[min\_index]>unsorted[j]){

                    min\_index=j;

                }

            }

            int t=unsorted[i];

            unsorted[i]=unsorted[min\_index];

            unsorted[min\_index]=t;

        }

        return(unsorted);

    }

}

1. **Write a program to perform insertion sort to reorder n numbers in ascending order.**

package Final\_printable\_code;

import java.util.Arrays;

import java.util.Scanner;

public class InsertionSort {

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        System.out.println("Enter the number of elements in the array");

        int n = sc.nextInt();

        int[] array = new int[n];

        System.out.println("Enter the elements of the array");

        for (int i = 0; i < n; i++) {

            array[i] = sc.nextInt();

        }

        System.out.println("sorted array is " + Arrays.toString(insertionSort(array)));

    }

    public static int[] insertionSort(int[] unsorted) {

        System.out.println("Using insertion sort to sort the array .. ");

        for(int i=1;i<unsorted.length;i++){

            int key= unsorted[i];

            int j= i-1;

            while(j>=0 && key<unsorted[j]){

                unsorted[j+1]=unsorted[j];

                j--;

            }

            unsorted[j+1]=key;

        }

        return(unsorted);

    }

}

1. **Write a program to perform quick sort using divide and conquer technique to arrange n numbers.**

package Final\_printable\_code;

import java.util.Arrays;

import java.util.Scanner;

public class QuickSort {

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        System.out.println("Enter the number of elements in the array");

        int n = sc.nextInt();

        int[] array = new int[n];

        System.out.println("Enter the elements of the array");

        for (int i = 0; i < n; i++) {

            array[i] = sc.nextInt();

        }

        System.out.println("sorted array is " + Arrays.toString(quickSort2(array,0,array.length-1)));

    }

    public static int[] quickSort2(int[] unsorted, int low, int high){

        if(low<high){

        int p= partition2(unsorted, low, high);

        quickSort2(unsorted, low, p-1);

        quickSort2(unsorted, p+1, high);

        }

        return unsorted;

    }

    private static int partition2(int[] unsorted,int low,int high){

int  i=low,j=high;

        int index = unsorted[low];

        while(i<j){

            while(unsorted[i]<index && i<high ){

                i++;

            }

            while(unsorted[j]>index && j>=low){

                j--;

            }

            if(i<j){

                int temp = unsorted[i];

                unsorted[i]=unsorted[j];

                unsorted[j]=temp;

            }

        }

            int temp2=unsorted[j];

            unsorted[j]=index;

            index=temp2;

        return j;

    }

}

1. **Write a program to perform merge sort on n number**

package Final\_printable\_code;

import java.util.Arrays;

import java.util.Scanner;

public class MergeSort {

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        System.out.println("Enter the number of elements in the array");

        int n = sc.nextInt();

        int[] array = new int[n];

        System.out.println("Enter the elements of the array");

        for (int i = 0; i < n; i++) {

            array[i] = sc.nextInt();

        }

        System.out.println("sorted array is " + Arrays.toString(mergeSort(array,0,array.length-1)));

    }

    public static int[] mergeSort(int[] unsorted, int beg, int end) {

        if(beg<end){

            int mid = (beg+end)/2;

            //Divide the entire array into 2 halves and call merge sort on them

            mergeSort(unsorted, beg, mid);

            mergeSort(unsorted, mid+1, end);

            merge(unsorted,beg,end,mid);

        }

        return unsorted;

    }

    private static void merge(int[] unsorted, int beg, int end, int mid) {

        int n1= mid- beg +1;

        int n2= end - mid;

        int i,j,k=beg;

        int[] leftArray = new int[n1];

        int[] rightArray = new int[n2];

        //Copying the half of the arrays to 2 new temporary arrays

        for( i=0;i<n1;i++)

            leftArray[i]=unsorted[beg+i];

        for( j=0;j<n2;j++)

            rightArray[j]=unsorted[mid+j+1];

        i=0;j=0;

        //comparing the left and right array and adding in order

        while(i<n1&&j<n2){

            //count++;//to keep track of each comparisions

            if(leftArray[i]<=rightArray[j]){

                unsorted[k]=leftArray[i];

                i++;

            }

            else{

                unsorted[k]=rightArray[j];

                j++;

            }

            k++;

        }

        //happens after either left or the right array runs

        //out and the remaining of the 2 are added

        while(i<n1){

            unsorted[k]=leftArray[i];

            i++; k++;

        }

        while(j<n2){

            unsorted[k]=rightArray[j];

            j++; k++;

        }

    }

}

1. **Write a program to implement push and pop operations in a stack.**

package Final\_printable\_code;

import java.util.ArrayList;

public class StackDemo {

    public ArrayList<Integer> array = new ArrayList<>();

    public int pop() throws Exception{

        if(array.isEmpty()){

            System.out.println("Array empty");

            throw new Exception("Array empty");

        }

        int ele = array.get(array.size()-1);

        array.remove(array.size()-1);

        return ele;

    }

    public void push(int ele){

        array.add(ele);

    }

    public void display(){

        System.out.println(array.toString());

    }

    public static void main(String[] args) throws Exception{

        StackDemo newStack = new StackDemo();

        newStack.push(6);

        newStack.push(5);

        newStack.pop();

        newStack.pop();

        newStack.pop();

        newStack.display();

    }

}

1. **Write a program to implement insert and delete operations in a queue.**

package Final\_printable\_code;

public class QueueDemo {

    int front = -1;

    int rear = -1;

    int[] array;

    QueueDemo(int size){

         this.array = new int[size];

    }

    QueueDemo(){

         this.array = new int[10];

    }

    public int deQueue(){

        if(isQempty()){

            System.out.println("Queue is empty and cant delete");

            return -1;

        }

        int t = array[front];

        if(front==rear){

            front=rear=-1;

            return t;

        }

        front++;

        return t;

    }

    public void enQueue(int ele){

        if(isQfull()){

            System.out.println("Queue is full");

            return;

        }

        if(front==-1){

            front=0;

        }

        rear++;

        array[rear]= ele;

    }

    public boolean isQempty(){

        if(front==-1)return true;

        return false;

    }

    public boolean isQfull(){

        if(front==0&&rear==array.length-1)

            return true;

        return false;

    }

    public int peek(){

        return array[front];

    }

    public void display(){

        int i;

        for (i = front; i != rear; i = (++i) % array.length){

            System.out.print(array[i]+" ");

        }

        System.out.print(array[i]);

        System.out.println("");

        return;

    }

    public static void main(String[] args) {

        QueueDemo newQueue = new QueueDemo(5);

        newQueue.enQueue(5);

        newQueue.enQueue(9);

        newQueue.enQueue(4);

        newQueue.enQueue(29);

        newQueue.enQueue(2);

        newQueue.enQueue(6);

        newQueue.deQueue();

        newQueue.display();

    }

}

1. **Write a program to find maximum and minimum of an array using divide and conquer algorithm.**

package Final\_printable\_code;

import java.util.Scanner;

public class MaxMinDandC {

    public static int[] minMaxSearch(int [] array, int min, int max){

        int i= min, j=max;

        //The array contains local minimum at position 0 and local maximum at position 1

        int[] minmax\_array = new int[2];

        //if its a small problem of length 1

        if(i==j){

            minmax\_array[0]=array[i];

            minmax\_array[1]=array[i];

            return minmax\_array;

        }

        //if its a small problem of length 2

        if(i==j-1){

            if(array[i]<array[j]){

                minmax\_array[0]=array[i];

                minmax\_array[1]=array[j];

            }

            else{

                minmax\_array[1]=array[i];

                minmax\_array[0]=array[j];

            }

            return minmax\_array;

        }

        //if its not a small problem then use the divide and conquor method

            int mid= (min + max)/2;

            int[] left\_minmax = minMaxSearch(array, min, mid);

            int[] right\_minmax = minMaxSearch(array, mid+1, max);

            if(left\_minmax[0]<right\_minmax[0])

                minmax\_array[0]=left\_minmax[0];

            else

                minmax\_array[0]=right\_minmax[0];

            if(left\_minmax[1]>right\_minmax[1])

                minmax\_array[1]=left\_minmax[1];

            else

                minmax\_array[1]=right\_minmax[1];

            return minmax\_array;

    }

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        System.out.println("Enter the number of elements in the array");

        int n = sc.nextInt();

        int[] array = new int[n];

        int[] mmarr = new int[2];

        mmarr = minMaxSearch(array, 0, array.length-1);

        System.out.println("Minimum of the array is "+mmarr[0]);

        System.out.println("Maximum of the array is "+mmarr[1]);

    }

}

1. **Write a program to implement 0/1 Knapsack Problem using Dynamic Programming.**
2. **Write a program to display the degree of vertices for an edge matrix for a particular graph G.**

package Final\_printable\_code;

/\*Program to find the total number of edges connected to a node  \*/

/\*Works on both directed and non directed graphs \*/

import java.util.Scanner;

public class NodeDegree {

    //private method to input edge\_matrix and output the degree array of nodes

    private static int[] getDegreeArray(int[][] edge\_matrix){

        int n=edge\_matrix.length;

        int[] degree\_array=new int[n];

        for(int i=0;i<n;i++){

            int local\_count = 0;

            for(int j=0;j<n;j++){

                if(edge\_matrix[i][j]!=999 && edge\_matrix[i][j]>=0){

                    local\_count++;

                }

            }

            degree\_array[i]=local\_count;

        }

        return degree\_array;

    }

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        System.out.println("Enter the number of vertices");

        int n=sc.nextInt();

        int[][] edge\_matrix = new int[n][n];

        System.out.println("Enter the edge adjacency matrix");

        for(int i=0;i<n;i++){

            for(int j=0;j<n;j++){

                edge\_matrix[i][j] = sc.nextInt();

                //idiot proofing to allow 0 as input in vertex to itself

                if(i==j){

                    edge\_matrix[i][j]=999;

                }

            }

        }

        int[] cost\_array = getDegreeArray(edge\_matrix);

        for(int i=0;i<n;i++){

            System.out.println("the degree of node "+i+" is "+cost\_array[i]);

        }

    }

}

1. **Write a program to find the shortest path for a given vertex in a weighted connected graph using Dijkstra’s algorithm.**

package Final\_printable\_code;

import java.util.Arrays;

import java.util.Scanner;

public class DijikstrasAlgo {

    public static void dijkstra(int[][] graph, int source) {

      int count = graph.length;

      boolean[] visitedVertex = new boolean[count];

      int[] distance = new int[count];

      Arrays.fill(visitedVertex,false);

      Arrays.fill(distance,999);

      // Distance of self loop is zero

      distance[source] = 0;

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

        // Update the distance between neighbouring vertex and source vertex

        int u = findMinDistance(distance, visitedVertex);

        visitedVertex[u] = true;

        // Update all the neighbouring vertex distances

        for (int v = 0; v < count; v++) {

          if (!visitedVertex[v] && graph[u][v] != 0 && (distance[u] + graph[u][v] < distance[v])) {

            distance[v] = distance[u] + graph[u][v];

          }

        }

      }

      for (int i = 0; i < distance.length; i++) {

        System.out.println(String.format("Distance from %s to %s is %s", source, i, distance[i]));

      }

    }

    // Finding the minimum distance

    private static int findMinDistance(int[] distance, boolean[] visitedVertex) {

      int minDistance = 999;

      int minDistanceVertex = -1;

      for (int i = 0; i < distance.length; i++) {

        if (!visitedVertex[i] && distance[i] < minDistance) {

          minDistance = distance[i];

          minDistanceVertex = i;

        }

      }

      return minDistanceVertex;

    }

    public static void main(String[] args) {

          Scanner sc = new Scanner(System.in);

          System.out.println("Enter the number of vertices");

          int n=sc.nextInt();

          int[][] cost\_matrix = new int[n][n];

          System.out.println("Enter the cost adjacency matrix");

          for(int i=0;i<n;i++){

              for(int j=0;j<n;j++){

                  cost\_matrix[i][j] = sc.nextInt();

                  //idiot proofing to allow 0 as input in vertex to itself

                  if(i==j){

                      cost\_matrix[i][j]=999;

                  }

              }

          }

      DijikstrasAlgo T = new DijikstrasAlgo();

      T.dijkstra(cost\_matrix, 0);

    }

  }

1. **Write a program to find the minimum cost spanning tree of a given undirected graph using Kruskal’s algorithm**
2. **Write a program to print all reachable nodes given a starting node in a digraph using BFS algorithm.**

package Final\_printable\_code;

import java.util.ArrayList;

import java.util.Scanner;

public class TraversalBFS {

    int n;

    boolean[] visited;

    TraversalBFS(int n) {

        visited = new boolean[n];

    }

    ArrayList<Integer> traversal\_arr = new ArrayList<>();

    int front = 0;

    void rBFS(int[][] cost\_matrix, int start\_node) {

        n = cost\_matrix.length;

        this.visited[start\_node] = true;

        if (!traversal\_arr.contains(start\_node))

            traversal\_arr.add(start\_node);

        for (int i = 0; i < n; i++) {

            if (cost\_matrix[start\_node][i] != 999 && !visited[i]) {

                if (!traversal\_arr.contains(i))

                    traversal\_arr.add(i);

            }

        }

        if (traversal\_arr.size() > front + 1)

            rBFS(cost\_matrix, traversal\_arr.get(++front));

    }

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        System.out.println("Enter the number of vertices");

        int n = sc.nextInt();

        int[][] cost\_matrix = new int[n][n];

        System.out.println("Enter the edge matrix");

        for (int i = 0; i < n; i++) {

            for (int j = 0; j < n; j++) {

                cost\_matrix[i][j] = sc.nextInt();

                if (i == j) {

                    cost\_matrix[i][j] = 999;

                }

            }

        }

        System.out.println("Enter starting node");

        int s\_node = sc.nextInt();

        TraversalBFS gs1 = new TraversalBFS(cost\_matrix.length);

        gs1.rBFS(cost\_matrix, s\_node);

        System.out.println("Traversal order in breadth first search is " + (gs1.traversal\_arr));

    }

}

1. **Write a program to check whether the given graph is connected or not using DFS algorithm.**

package Final\_printable\_code;

import java.util.ArrayList;

import java.util.Scanner;

public class TraversalDFS {

    int n;

    boolean[] visited;

    TraversalDFS(int n) {

        visited = new boolean[n];

    }

    ArrayList<Integer> traversal\_arr = new ArrayList<>();

    int front = 0;

    void DFS(int[][] cost\_matrix, int start\_node) {

        n = cost\_matrix.length;

        this.visited[start\_node] = true;

        traversal\_arr.add(start\_node);

        if (isUnvisted(visited)) {

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

                if (cost\_matrix[start\_node][i] != 999 && !visited[i]) {

                    DFS(cost\_matrix, i);

                }

            }

        }

    }

    private static boolean isUnvisted(boolean[] arr) {

        for (boolean x : arr) {

            if (x == false) {

                return true;

            }

        }

        return false;

    }

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        System.out.println("Enter the number of vertices");

        int n = sc.nextInt();

        int[][] cost\_matrix = new int[n][n];

        System.out.println("Enter the edge matrix");

        for (int i = 0; i < n; i++) {

            for (int j = 0; j < n; j++) {

                cost\_matrix[i][j] = sc.nextInt();

                if (i == j) {

                    cost\_matrix[i][j] = 999;

                }

            }

        }

        System.out.println("Enter starting node");

        int s\_node = sc.nextInt();

        TraversalDFS gs1 = new TraversalDFS(cost\_matrix.length);

        gs1.DFS(cost\_matrix, s\_node);

        System.out.println("Traversal order in breadth first search is " + (gs1.traversal\_arr));

        if (isUnvisted(gs1.visited)) {

            System.out.println("The graph is disjoint");

        } else

            System.out.println("All nodes are reachable");

    }

}

1. **Write a program to compute the transitive closure of a given directed graph using Warshall’s algorithm.**

package Final\_printable\_code;

import java.util.ArrayList;

import java.util.List;

import java.util.Scanner;

class Edge {

    int source, dest;

    public Edge(int source, int dest) {

        this.source = source;

        this.dest = dest;

    }

}

class Graph {

    List<List<Integer>> adjList = null;

    Graph(List<Edge> edges, int N) {

        adjList = new ArrayList<>(N);

        for (int i = 0; i < N; i++) {

            adjList.add(i, new ArrayList<>());

        }

        for (int i = 0; i < edges.size(); i++) {

            int src = edges.get(i).source;

            int dest = edges.get(i).dest;

            adjList.get(src).add(dest);

        }

    }

}

public class TransClosure {

    public static void DFS(Graph graph, byte[][] C, int root, int descendant) {

        for (int child : graph.adjList.get(descendant)) {

            if (C[root][child] == 0) {

                C[root][child] = 1;

                DFS(graph, C, root, child);

            }

        }

    }

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        System.out.println("Enter the number of edges initially");

        int n = sc.nextInt();

        List<Edge> edges = new ArrayList<>();

        for (int i = 0; i < n; i++) {

            System.out.println("Edge " + (i + 1));

            System.out.println("Enter source");

            int source = sc.nextInt();

            System.out.println("Enter destination");

            int destination = sc.nextInt();

            edges.add(new Edge(source, destination));

        }

        final int N = n + 1;

        Graph graph = new Graph(edges, N);

        byte[][] C = new byte[N][N];

        System.out.println("TRANSITIVE CLOSURE:-\n");

        for (int v = 0; v < N; v++) {

            C[v][v] = 1;

            DFS(graph, C, v, v);

            for (int u = 0; u < N; u++)

                System.out.print(C[v][u] + " ");

            System.out.println();

        }

    }

}

1. **Write a program to implement all pair shortest path’s problem using Floyd’s Algorithm**.

package Final\_printable\_code;

import java.util.Scanner;

class FloydWarshall {

    void floydWarshall(int graph[][], int n) {

        int dist[][] = new int[n][n];

        int i, j, k;

        for (i = 0; i < n; i++)

            for (j = 0; j < n; j++)

                dist[i][j] = graph[i][j];

        for (k = 0; k < n; k++) {

            for (i = 0; i < n; i++) {

                for (j = 0; j < n; j++) {

                    if (dist[i][k] + dist[k][j] < dist[i][j])

                        dist[i][j] = dist[i][k] + dist[k][j];

                }

            }

        }

        printSolution(dist, n);

    }

    void printSolution(int dist[][], int n) {

        System.out.println("SHORTEST PATH MATRIX:-");

        for (int i = 0; i < n; ++i) {

            for (int j = 0; j < n; ++j) {

                if (dist[i][j] == 999)

                    System.out.print("I ");

                else

                    System.out.print(dist[i][j] + " ");

            }

            System.out.println();

        }

    }

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        System.out.println("Enter the number of vertices");

        int n = sc.nextInt();

        int[][] cost\_matrix = new int[n][n];

        System.out.println("Enter the cost matrix");

        for (int i = 0; i < n; i++) {

            for (int j = 0; j < n; j++) {

                cost\_matrix[i][j] = sc.nextInt();

                if (i == j) {

                    cost\_matrix[i][j] = 0;

                }

            }

        }

        FloydWarshall a = new FloydWarshall();

        a.floydWarshall(cost\_matrix, n);

    }

}