INDEX

1.AIM: Write a java program to implement Quick sort algorithm for sorting a list of integers in ascending order

SOURCE CODE:

```
import java.util.Arrays;

public class QuickSortDemo{

   public static void main(String args[]) {

      // unsorted integer array
      int[] unsorted = {6, 5, 3, 1, 8, 7, 2, 4};
      System.out.println("Unsorted array :" + Arrays.toString(unsorted));

      QuickSort algorithm = new QuickSort();

      // sorting integer array using quicksort algorithm
      algorithm.sort(unsorted);

      // printing sorted array
      System.out.println("Sorted array :" + Arrays.toString(unsorted));

   }

}

/**
 * Java Program sort numbers using QuickSort Algorithm. QuickSort is a divide
 * and conquer algorithm, which divides the original list, sort it and then
 * merge it to create sorted output.

 */
class QuickSort {

   private int input[];
   private int length;

   public void sort(int[] numbers) {

      if (numbers == null || numbers.length == 0) {
         return;
      }
      this.input = numbers;
      length = numbers.length;
      quickSort(0, length - 1);
   }
```

```
    /*
     * This method implements in-place quicksort algorithm recursively.
     */
    private void quickSort(int low, int high) {
        int i = low;
        int j = high;

        // pivot is middle index
        int pivot = input[low + (high - low) / 2];

        // Divide into two arrays
        while (i <= j) {
            /**
             * As shown in above image, In each iteration, we will identify a
             * number from left side which is greater then the pivot value, and
             * a number from right side which is less then the pivot value. Once
             * search is complete, we can swap both numbers.
             */
            while (input[i] < pivot) {
                i++;
            }
            while (input[j] > pivot) {
                j--;
            }
            if (i <= j) {
                swap(i, j);
                // move index to next position on both sides
                i++;
                j--;
            }
        }
        // calls quickSort() method recursively
        if (low < j) {
            quickSort(low, j);
        }
        if (i < high) {
            quickSort(i, high);
        }
    }
    private void swap(int i, int j) {
        int temp = input[i];
        input[i] = input[j];
        input[j] = temp;
    }
}
```

Output :
Unsorted array :[6, 5, 3, 1, 8, 7, 2, 4]
Sorted array :[1, 2, 3, 4, 5, 6, 7, 8]

2. AIM: Write a java program to implement Merge sort algorithm for sorting a list of integers in ascending order

SOURCE CODE:

```
import java.util.Scanner;

/* Class MergeSort */
public class MergeSort
{
    /* Merge Sort function */
    public static void sort(int[] a, int low, int high)
    {
        int N = high - low;
        if (N <= 1)
            return;
        int mid = low + N/2;
        // recursively sort
        sort(a, low, mid);
        sort(a, mid, high);
        // merge two sorted subarrays
        int[] temp = new int[N];
        int i = low, j = mid;
        for (int k = 0; k < N; k++)
        {
            if (i == mid)
                temp[k] = a[j++];
            else if (j == high)
                temp[k] = a[i++];
            else if (a[j]<a[i])
                temp[k] = a[j++];
            else
                temp[k] = a[i++];
        }
        for (int k = 0; k < N; k++)
            a[low + k] = temp[k];
    }
    /* Main method */
    public static void main(String[] args)
    {
        Scanner scan = new Scanner( System.in );
        System.out.println("Merge Sort Test\n");
        int n, i;
        /* Accept number of elements */
        System.out.println("Enter number of integer elements");
        n = scan.nextInt();
        /* Create array of n elements */
        int arr[] = new int[ n ];
```

```
        /* Accept elements */
        System.out.println("\nEnter "+ n +" integer elements");
        for (i = 0; i < n; i++)
            arr[i] = scan.nextInt();
        /* Call method sort */
        sort(arr, 0, n);
        /* Print sorted Array */
        System.out.println("\nElements after sorting ");
        for (i = 0; i < n; i++)
            System.out.print(arr[i]+" ");
        System.out.println();
    }
}
```

OUTPUT:

Merge Sort Test

Enter number of integer elements
20

Enter 20 integer elements
415 440 845 535 420 555 984 509 11 561 900 413 195 963 566 305 2 169 547 607

Elements after sorting
2 11 169 195 305 413 415 420 440 509 535 547 555 561 566 607 845 900 963 98

3. AIM: Write a java program to implement the DFS algorithm for a graph.

SOURCE CODE:

```
import java.util.InputMismatchException;
    import java.util.Scanner;
    import java.util.Stack;

public class DFS
    {
      private Stack<Integer> stack;

      public DFS()
      {
        stack = new Stack<Integer>();
     }

   public void dfs(int adjacency_matrix[][], int source)
   {
     int number_of_nodes = adjacency_matrix[source].length - 1;

     int visited[] = new int[number_of_nodes + 1];
          int element = source;
     int i = source;
          System.out.print(element + "\t");
          visited[source] = 1;
     stack.push(source);

     while (!stack.isEmpty())
     {
       element = stack.peek();
       i = element;
      while (i <= number_of_nodes)
               {
        if (adjacency_matrix[element][i] == 1 && visited[i] == 0)
        {
             stack.push(i);
             visited[i] = 1;
             element = i;
             i = 1;
             System.out.print(element + "\t");
           continue;
         }
         i++;
      }
        stack.pop();
     }
   }

   public static void main(String...arg)
```

```
    {
      int number_of_nodes, source;
      Scanner scanner = null;
     try
      {
       System.out.println("Enter the number of nodes in the graph");
        scanner = new Scanner(System.in);
        number_of_nodes = scanner.nextInt();

       int adjacency_matrix[][] = new int[number_of_nodes + 1][number_of_nodes + 1];
       System.out.println("Enter the adjacency matrix");
       for (int i = 1; i <= number_of_nodes; i++)
           for (int j = 1; j <= number_of_nodes; j++)
                 adjacency_matrix[i][j] = scanner.nextInt();

       System.out.println("Enter the source for the graph");
          source = scanner.nextInt();

          System.out.println("The DFS Traversal for the graph is given by ");
          DFS dfs = new DFS();
          dfs.dfs(adjacency_matrix, source);
       }catch(InputMismatchException inputMismatch)
       {
          System.out.println("Wrong Input format");
       }
       scanner.close();
     }
   }
```

```
$javac DFS.java
$java DFS
Enter the number of nodes in the graph
4
Enter the adjacency matrix
0 1 0 1
0 0 1 0
0 1 0 1
0 0 0 1
Enter the source for the graph
1
The DFS Traversal for the graph is given by
1       2       3       4
```

4. AIM: Write a java program to implement the BFS algorithm for a graph.

SOURCE CODE:

```
import java.util.InputMismatchException;
import java.util.LinkedList;
import java.util.Queue;
import java.util.Scanner;
 public class BFS
{
    private Queue<Integer> queue;

   public BFS()
   {
      queue = new LinkedList<Integer>();
   }

   public void bfs(int adjacency_matrix[][], int source)
   {
      int number_of_nodes = adjacency_matrix[source].length - 1;

      int[] visited = new int[number_of_nodes + 1];
      int i, element;

      visited[source] = 1;
      queue.add(source);

      while (!queue.isEmpty())
      {
         element = queue.remove();
         i = element;
         System.out.print(i + "\t");
         while (i <= number_of_nodes)
         {
            if (adjacency_matrix[element][i] == 1 && visited[i] == 0)
            {
               queue.add(i);
               visited[i] = 1;
            }
            i++;
         }
      }
   }
    public static void main(String... arg)
```

```
  {
     int number_no_nodes, source;
     Scanner scanner = null;
      try
     {
        System.out.println("Enter the number of nodes in the graph");
        scanner = new Scanner(System.in);
        number_no_nodes = scanner.nextInt();
         int adjacency_matrix[][] = new int[number_no_nodes + 1][number_no_nodes + 1];
        System.out.println("Enter the adjacency matrix");
        for (int i = 1; i <= number_no_nodes; i++)
           for (int j = 1; j <= number_no_nodes; j++)
              adjacency_matrix[i][j] = scanner.nextInt();

        System.out.println("Enter the source for the graph");
        source = scanner.nextInt();

        System.out.println("The BFS traversal of the graph is ");
        BFS bfs = new BFS();
        bfs.bfs(adjacency_matrix, source);

     } catch (InputMismatchException inputMismatch)
     {
        System.out.println("Wrong Input Format");
     }
     scanner.close();
  }
}
```

```
$javac BFS.java
$java BFS
Enter the number of nodes in the graph
4
Enter the adjacency matrix
0 1 0 1
0 0 1 0
0 1 0 1
0 0 0 1
Enter the source for the graph
1
The BFS traversal of the graph is
1       2       4       3
```

5. AIM: Write a java programs to implement backtracking algorithm for the N-queens problem.

SOURCE CODE:

```
import java.io.*;
class operation
{
int x[]=new int[20];
int count=0;
public boolean place(int row,int column)
{
int i;
for(i=1;i<=row-1;i++)
{ //checking for column and diagonal conflicts
if(x[i] == column)
return false;
else
if(Math.abs(x[i] – column) == Math.abs(i – row))
return false;
}
return true;
}
public void Queen(int row,int n)
{
int column;
for(column=1;column<=n;column++)
{
if(place(row,column))
{
x[row] = column;
if(row==n)
print_board(n);//printing the board configuration
else //try next queen with next position
Queen(row+1,n);
}
}
}


public void print_board(int n)
{
int i;
System.out.println(“\n\nSolution :”+(++count));
for(i=1;i<=n;i++)
{
System.out.print(” “+i);
}
for(i=1;i<=n;i++)
{
System.out.print(“\n\n”+i);
for(int j=1;j<=n;j++)// for nXn board
```

```
{
if(x[i]==j)
System.out.print(" Q");
else
System.out.print(" -");
}
}
}
}
class BacktrackDemo
{
public static void main (String args[] )throws IOException
{
DataInputStream in=new DataInputStream(System.in);
System.out.println("Enter no Of Queens");
int n=Integer.parseInt(in.readLine());
operation op=new operation();
op.Queen(1,n);
}
}
```

OUTPUT:

```
Enter no Of Queens
4
Solution :1
1 2 3 4

1 – Q – –

2 – – – Q

3 Q – – –

4 – – Q –
```

6. AIM: Write a java program to implement the backtracking algorithm for the sum of subsets problem.

SOURCE CODE:

```
import java.util.*;
class SumOfSubsets
{
	static int w[],x[],i,j,s=0,k=1,r=0,m,n;
	void read()
	{
		Scanner s=new Scanner(System.in);
		System.out.println("Plaese enter the number of values");
		n=s.nextInt();
		w=new int[n+1];
		x=new int[n+1];
		System.out.println("Plaese enter the corresponding weights");
		for(i=1;i<=n;i++)
		{
			System.out.println("Please enter the weight");
			w[i]=s.nextInt();
			r+=w[i];
			x[i]=0;
		}
		System.out.println("Please enter the total sum");
		m=s.nextInt();
	}
	void sumOfSubset(int s,int k, int r)
	{
		if(s+w[k]==m)
			{
			x[i]=1;
			System.out.println("Possible Solutions are");
			for(i=1;i<=n;i++)
				System.out.print(" "+x[i]);
			System.out.println();
			}
		else if(s+w[k]+w[k+1]<=m)
			{
				x[k]=1;
				sumOfSubset(s+w[k],k+1,r-w[k]);
			}
		if((s+r-w[k]>=m)&&(s+w[k+1]<=m))
			{
				x[k]=0;
				sumOfSubset(s,k+1,r-w[k]);
			}
	}
	public static void main(String args[])
	{
```

```
                SumOfSubsets ss=new SumOfSubsets();
                ss.read();
                ss.sumOfSubset(s,k,r);
        }
}
```

Output:

```
Please enter the number of values
4
Please enter the corresponding weights
Please enter the weight
7
Please enter the weight
11
Please enter the weight
13
Please enter the weight
24
Please enter the total sum
31
Possible Solutions are
 1 1 1 0
Possible Solutions are
 1 0 0 1
```

7. AIM: Write a java program to implement the backtracking algorithm for the Hamiltonian Circuits problem

SOURCE CODE:

```
import java.util.Scanner;
import java.util.Arrays;

/** Class HamiltonianCycle **/
public class HamiltonianCycle
{
    private int V, pathCount;
    private int[] path;
    private int[][] graph;

    /** Function to find cycle **/
    public void findHamiltonianCycle(int[][] g)
    {
        V = g.length;
        path = new int[V];

        Arrays.fill(path, -1);
        graph = g;
        try
        {
            path[0] = 0;
            pathCount = 1;
            solve(0);
            System.out.println("No solution");
        }
        catch (Exception e)
        {
            System.out.println(e.getMessage());
            display();
        }
    }
    /** function to find paths recursively **/
    public void solve(int vertex) throws Exception
    {
        /** solution **/
        if (graph[vertex][0] == 1 && pathCount == V)
            throw new Exception("Solution found");
        /** all vertices selected but last vertex not linked to 0 **/
        if (pathCount == V)
            return;

        for (int v = 0; v < V; v++)
        {
            /** if connected **/
            if (graph[vertex][v] == 1 )
```

```
        {
            /** add to path **/
            path[pathCount++] = v;
            /** remove connection **/
            graph[vertex][v] = 0;
            graph[v][vertex] = 0;

            /** if vertex not already selected  solve recursively **/
            if (!isPresent(v))
                solve(v);

            /** restore connection **/
            graph[vertex][v] = 1;
            graph[v][vertex] = 1;
            /** remove path **/
            path[--pathCount] = -1;
        }
    }
}
/** function to check if path is already selected **/
public boolean isPresent(int v)
{
    for (int i = 0; i < pathCount - 1; i++)
        if (path[i] == v)
            return true;
    return false;
}
/** display solution **/
public void display()
{
    System.out.print("\nPath : ");
    for (int i = 0; i <= V; i++)
        System.out.print(path[i % V] +" ");
    System.out.println();
}
/** Main function **/
public static void main (String[] args)
{
    Scanner scan = new Scanner(System.in);
    System.out.println("HamiltonianCycle Algorithm Test\n");
    /** Make an object of HamiltonianCycle class **/
    HamiltonianCycle hc = new HamiltonianCycle();

    /** Accept number of vertices **/
    System.out.println("Enter number of vertices\n");
    int V = scan.nextInt();

    /** get graph **/
    System.out.println("\nEnter matrix\n");
    int[][] graph = new int[V][V];
```

```
        for (int i = 0; i < V; i++)
            for (int j = 0; j < V; j++)
                graph[i][j] = scan.nextInt();

        hc.findHamiltonianCycle(graph);
    }
}
```

HamiltonianCycle Algorithm Test

Enter number of vertices

8

Enter matrix

0 1 0 1 1 0 0 0
1 0 1 0 0 1 0 0
0 1 0 1 0 0 1 0
1 0 1 0 0 0 0 1
1 0 0 0 0 1 0 1
0 1 0 0 1 0 1 0
0 0 1 0 0 1 0 1
0 0 0 1 1 0 1 0
Solution found

Path : 0 1 2 3 7 6 5 4 0

8. AIM: Write a java program to implement greedy algorithm for job sequencing with deadlines

SOURCE CODE

```
import java.util.*;
class job
{
int p;    //.............for profit of a job
int d;    //.............for deadline of a job
int v;    //.............for checking if that job has been selected
/**************default constructor***************/
job()
{
p=0;
d=0;
v=0;
}
job(int x,int y,int z) // parameterised constructor
{
p=x;
d=y;
v=z;
}
}
class js
{
static int n;
static int out(job jb[],int x)
{
for(int i=0;i<n;++i)
if(jb[i].p==x)
return i;
return 0;
}
public static void main(String args[])
{
Scanner scr=new Scanner(System.in);
System.out.println("Enter the number of jobs");
n=scr.nextInt();
int max=0;   // this is to find the maximum deadline
job jb[]=new job[n];
/************************Accepting job from user************************/
for(int i=0;i<n;++i)
{
System.out.println("Enter profit and deadline(p d)");
int p=scr.nextInt();
int d=scr.nextInt();
if(max<d)
max=d;     // assign maximum value of deadline to "max" variable
jb[i]=new job(p,d,0);  //zero as third parameter to mark that initially it is unvisited
```

```
}
//accepted jobs from user

/*************************Sorting in increasing order of
deadlines************************/
for(int i=0;i<=n-2;++i)
{
 for(int j=i;j<=n-1;++j)
 {
   if(jb[i].d>jb[j].d)
   {
     job temp=jb[i];
     jb[i]=jb[j];
     jb[j]=temp;
   }
 }
}
// sorting process ends

/*******************************Displaying the jobs to the user***********************/
System.out.println("The jobs are as follows ");
for(int i=0;i<n;++i)
System.out.println("Job "+i+" Profit = "+jb[i].p+" Deadline = "+jb[i].d);
// jobs displayed to the user

int count;
int hold[]=new int[max];
for(int i=0;i<max;++i)
hold[i]=0;
/*******************************Process of job sequencing begins************************/
for(int i=0;i<n;++i)
{
count=0;
  for(int j=0;j<n;++j)
  {

   if(count<jb[j].d && jb[j].v==0 && count<max && jb[j].p>hold[count])
   {
     int ch=0;

  if(hold[count]!=0)
     {
  ch=out(jb,hold[count]);
     jb[ch].v=0;
  }

  hold[count]=jb[j].p;
  jb[j].v=1;
     ++count;
```

```
  } // end of if

  } //end of inner for
}// end of outer for
/*******************************job sequencing process
ends*******************************/

/*******************************calculating max
profit*********************************/
int profit=0;
for(int i=0;i<max;++i)
profit+=hold[i];
System.out.println("The maximum profit is "+profit);

}//end main method
}//end class
```

```
Enter the number of jobs
4
Enter profit and deadline(p d)
70 2
Enter profit and deadline(p d)
12 1
Enter profit and deadline(p d)
18 2
Enter profit and deadline(p d)
35 1
The jobs are as follows
Job 0 Profit = 12 Deadline = 1
Job 1 Profit = 35 Deadline = 1
Job 2 Profit = 18 Deadline = 2
Job 3 Profit = 70 Deadline = 2

The maximum profit is 105
```

9. AIM: Write a java program to implement Dijkstra's algorithm for the Single source shortest path problem

SOURCE CODE

```
import java.util.*;
public class Dijkstra
{
 public  int distance[] = new int[10];
 public  int cost[][] = new int[10][10];

 public void calc(int n,int s)
 {
  int flag[] = new int[n+1];
  int i,minpos=1,k,c,minimum;

  for(i=1;i<=n;i++)
  {
          flag[i]=0;
     this.distance[i]=this.cost[s][i];
    }
    c=2;
  while(c<=n)
  {
   minimum=99;
   for(k=1;k<=n;k++)
   {
        if(this.distance[k]<minimum && flag[k]!=1)
      {
       minimum=this.distance[i];
       minpos=k;
      }
     }
          flag[minpos]=1;
     c++;
     for(k=1;k<=n;k++)
{
       if(this.distance[minpos]+this.cost[minpos][k] <  this.distance[k] && flag[k]!=1 )
   this.distance[k]=this.distance[minpos]+this.cost[minpos][k];
}
 }

}
public static void main(String args[])
{
 int nodes,source,i,j;
 Scanner in = new Scanner(System.in);
 System.out.println("Enter the Number of Nodes \n");
 nodes = in.nextInt();
 Dijkstra d = new Dijkstra();
```

```
 System.out.println("Enter the Cost Matrix Weights: \n");
      for(i=1;i<=nodes;i++)
        for(j=1;j<=nodes;j++)
  {
         d.cost[i][j]=in.nextInt();
         if(d.cost[i][j]==0)
           d.cost[i][j]=999;
        }
 System.out.println("Enter the Source Vertex :\n");
 source=in.nextInt();

   d.calc(nodes,source);
 System.out.println("The Shortest Path from Source \t"+source+"\t to all other vertices are : \n");
      for(i=1;i<=nodes;i++)
        if(i!=source)
System.out.println("source :"+source+"\t destination :"+i+"\t MinCost is :"+d.distance[i]+"\t");



 }
}
```

OUTPUT

```
javac Dijkstra.java
java Dijkstra

Enter the Number of Nodes :4
Enter the Cost Matrix Weights:
0     3   999   7
3     0    4     2
999   4    0     5
7     2    5     0
Enter the Source Vertex :
1
The Shortest Path from Source  Vertex  1  to all other vertices are :

source :1       destination :2   MinCost is :3
source :1       destination :3   MinCost is :7
source :1       destination :4   MinCost is :5
```

10. AIM: Write a java program to implement Prim's algorithm to generate minimum cost spanning tree.

SOURCE CODE

```
import java.util.InputMismatchException;
import java.util.Scanner;

public class Prims
{
   private boolean unsettled[];
   private boolean settled[];
   private int numberofvertices;
   private int adjacencyMatrix[][];
   private int key[];
   public static final int INFINITE = 999;
   private int parent[];

   public Prims(int numberofvertices)
   {
      this.numberofvertices = numberofvertices;
      unsettled = new boolean[numberofvertices + 1];
      settled = new boolean[numberofvertices + 1];
      adjacencyMatrix = new int[numberofvertices + 1][numberofvertices + 1];
      key = new int[numberofvertices + 1];
      parent = new int[numberofvertices + 1];
   }

   public int getUnsettledCount(boolean unsettled[])
   {
      int count = 0;
      for (int index = 0; index < unsettled.length; index++)
      {
         if (unsettled[index])
         {
            count++;
         }
      }
      return count;
   }

   public void primsAlgorithm(int adjacencyMatrix[][])
   {
      int evaluationVertex;
      for (int source = 1; source <= numberofvertices; source++)
      {
         for (int destination = 1; destination <= numberofvertices; destination++)
         {
            this.adjacencyMatrix[source][destination] = adjacencyMatrix[source][destination];
         }
```

```
    }

    for (int index = 1; index <= numberofvertices; index++)
    {
       key[index] = INFINITE;
    }
    key[1] = 0;
    unsettled[1] = true;
    parent[1] = 1;

    while (getUnsettledCount(unsettled) != 0)
    {
       evaluationVertex = getMimumKeyVertexFromUnsettled(unsettled);
       unsettled[evaluationVertex] = false;
       settled[evaluationVertex] = true;
       evaluateNeighbours(evaluationVertex);
    }
}

private int getMimumKeyVertexFromUnsettled(boolean[] unsettled2)
{
    int min = Integer.MAX_VALUE;
    int node = 0;
    for (int vertex = 1; vertex <= numberofvertices; vertex++)
    {
       if (unsettled[vertex] == true && key[vertex] < min)
       {
          node = vertex;
          min = key[vertex];
       }
    }
    return node;
}

public void evaluateNeighbours(int evaluationVertex)
{

    for (int destinationvertex = 1; destinationvertex <= numberofvertices; destinationvertex++)
    {
       if (settled[destinationvertex] == false)
       {
          if (adjacencyMatrix[evaluationVertex][destinationvertex] != INFINITE)
          {
             if (adjacencyMatrix[evaluationVertex][destinationvertex] < key[destinationvertex])
             {
                key[destinationvertex] = adjacencyMatrix[evaluationVertex][destinationvertex];
                parent[destinationvertex] = evaluationVertex;
             }
             unsettled[destinationvertex] = true;
          }
```

```
            }
        }
    }

    public void printMST()
    {
        System.out.println("SOURCE  : DESTINATION = WEIGHT");
        for (int vertex = 2; vertex <= numberofvertices; vertex++)
        {
            System.out.println(parent[vertex] + "\t:\t" + vertex +"\t=\t"+
adjacencyMatrix[parent[vertex]][vertex]);
        }
    }

    public static void main(String... arg)
    {
        int adjacency_matrix[][];
        int number_of_vertices;
        Scanner scan = new Scanner(System.in);

        try
        {
            System.out.println("Enter the number of vertices");
            number_of_vertices = scan.nextInt();
            adjacency_matrix = new int[number_of_vertices + 1][number_of_vertices + 1];

            System.out.println("Enter the Weighted Matrix for the graph");
            for (int i = 1; i <= number_of_vertices; i++)
            {
                for (int j = 1; j <= number_of_vertices; j++)
                {
                    adjacency_matrix[i][j] = scan.nextInt();
                    if (i == j)
                    {
                        adjacency_matrix[i][j] = 0;
                        continue;
                    }
                    if (adjacency_matrix[i][j] == 0)
                    {
                        adjacency_matrix[i][j] = INFINITE;
                    }
                }
            }

            Prims prims = new Prims(number_of_vertices);
            prims.primsAlgorithm(adjacency_matrix);
            prims.printMST();

        } catch (InputMismatchException inputMismatch)
        {
```

```
            System.out.println("Wrong Input Format");
        }
        scan.close();
    }
}
```

```
$javac Prims.java
$java Prims

Enter the number of vertices
5
Enter the Weighted Matrix for the graph
0 4 0 0 5
4 0 3 6 1
0 3 0 6 2
0 6 6 0 7
5 1 2 7 0
SOURCE  : DESTINATION   =    WEIGHT
1       :    2          =       4
5       :    3          =       2
2       :    4          =       6
2       :    5          =       1
```

11. AIM: Write a java program to implement Kruskal's algorithm to generate minimum cost spanning tree.

SOURCE CODE

```
import java.util.Collections;
import java.util.Comparator;
import java.util.LinkedList;
import java.util.List;
import java.util.Scanner;
import java.util.Stack;

public class KruskalAlgorithm
{
   private List<Edge> edges;
   private int numberOfVertices;
   public static final int MAX_VALUE = 999;
   private int visited[];
   private int spanning_tree[][];

   public KruskalAlgorithm(int numberOfVertices)
   {
      this.numberOfVertices = numberOfVertices;
      edges = new LinkedList<Edge>();
      visited = new int[this.numberOfVertices + 1];
      spanning_tree = new int[numberOfVertices + 1][numberOfVertices + 1];
   }

   public void kruskalAlgorithm(int adjacencyMatrix[][])
   {
      boolean finished = false;
      for (int source = 1; source <= numberOfVertices; source++)
      {
         for (int destination = 1; destination <= numberOfVertices; destination++)
         {
            if (adjacencyMatrix[source][destination] != MAX_VALUE && source != destination)
            {
               Edge edge = new Edge();
               edge.sourcevertex = source;
               edge.destinationvertex = destination;
               edge.weight = adjacencyMatrix[source][destination];
               adjacencyMatrix[destination][source] = MAX_VALUE;
               edges.add(edge);
            }
         }
      }
      Collections.sort(edges, new EdgeComparator());
      CheckCycle checkCycle = new CheckCycle();
      for (Edge edge : edges)
      {
```

```
            spanning_tree[edge.sourcevertex][edge.destinationvertex] = edge.weight;
            spanning_tree[edge.destinationvertex][edge.sourcevertex] = edge.weight;
            if (checkCycle.checkCycle(spanning_tree, edge.sourcevertex))
            {
               spanning_tree[edge.sourcevertex][edge.destinationvertex] = 0;
               spanning_tree[edge.destinationvertex][edge.sourcevertex] = 0;
               edge.weight = -1;
               continue;
            }
            visited[edge.sourcevertex] = 1;
            visited[edge.destinationvertex] = 1;
            for (int i = 0; i < visited.length; i++)
            {
               if (visited[i] == 0)
               {
                  finished = false;
                  break;
               } else
               {
                  finished = true;
               }
            }
            if (finished)
               break;
         }
         System.out.println("The spanning tree is ");
         for (int i = 1; i <= numberOfVertices; i++)
            System.out.print("\t" + i);
         System.out.println();
         for (int source = 1; source <= numberOfVertices; source++)
         {
            System.out.print(source + "\t");
            for (int destination = 1; destination <= numberOfVertices; destination++)
            {
               System.out.print(spanning_tree[source][destination] + "\t");
            }
            System.out.println();
         }
      }

      public static void main(String... arg)
      {
         int adjacency_matrix[][];
         int number_of_vertices;

         Scanner scan = new Scanner(System.in);
         System.out.println("Enter the number of vertices");
         number_of_vertices = scan.nextInt();
         adjacency_matrix = new int[number_of_vertices + 1][number_of_vertices + 1];
```

```
        System.out.println("Enter the Weighted Matrix for the graph");
        for (int i = 1; i <= number_of_vertices; i++)
        {
            for (int j = 1; j <= number_of_vertices; j++)
            {
                adjacency_matrix[i][j] = scan.nextInt();
                if (i == j)
                {
                    adjacency_matrix[i][j] = 0;
                    continue;
                }
                if (adjacency_matrix[i][j] == 0)
                {
                    adjacency_matrix[i][j] = MAX_VALUE;
                }
            }
        }
        KruskalAlgorithm kruskalAlgorithm = new KruskalAlgorithm(number_of_vertices);
        kruskalAlgorithm.kruskalAlgorithm(adjacency_matrix);
        scan.close();
    }
}

class Edge
{
    int sourcevertex;
    int destinationvertex;
    int weight;
}

class EdgeComparator implements Comparator<Edge>
{
    @Override
    public int compare(Edge edge1, Edge edge2)
    {
        if (edge1.weight < edge2.weight)
            return -1;
        if (edge1.weight > edge2.weight)
            return 1;
        return 0;
    }
}

class CheckCycle
{
    private Stack<Integer> stack;
    private int adjacencyMatrix[][];

    public CheckCycle()
    {
```

```
    stack = new Stack<Integer>();
  }

  public boolean checkCycle(int adjacency_matrix[][], int source)
  {
    boolean cyclepresent = false;
    int number_of_nodes = adjacency_matrix[source].length - 1;

    adjacencyMatrix = new int[number_of_nodes + 1][number_of_nodes + 1];
    for (int sourcevertex = 1; sourcevertex <= number_of_nodes; sourcevertex++)
    {
      for (int destinationvertex = 1; destinationvertex <= number_of_nodes; destinationvertex++)
      {
        adjacencyMatrix[sourcevertex][destinationvertex] =
adjacency_matrix[sourcevertex[destinationvertex];
      }
    }

    int visited[] = new int[number_of_nodes + 1];
    int element = source;
    int i = source;
    visited[source] = 1;
    stack.push(source);

    while (!stack.isEmpty())
    {
      element = stack.peek();
      i = element;
      while (i <= number_of_nodes)
      {
        if (adjacencyMatrix[element][i] >= 1 && visited[i] == 1)
        {
          if (stack.contains(i))
          {
            cyclepresent = true;
            return cyclepresent;
          }
        }
        if (adjacencyMatrix[element][i] >= 1 && visited[i] == 0)
        {
          stack.push(i);
          visited[i] = 1;
          adjacencyMatrix[element][i] = 0;// mark as labelled;
          adjacencyMatrix[i][element] = 0;
          element = i;
          i = 1;
          continue;
        }
        i++;
      }
```

```
            stack.pop();
        }
        return cyclepresent;
    }
}
```

```
$javac KruskalAlgorithm.java
$java KruskalAlgorithm
Enter the number of vertices
6
Enter the Weighted Matrix for the graph
0 6 8 6 0 0
6 0 0 5 10 0
8 0 0 7 5 3
6 5 7 0 0 0
0 10 5 0 0 3
0 0 3 0 3 0
The spanning tree is
```

	1	2	3	4	5	6
1	0	6	0	0	0	0
2	6	0	0	5	0	0
3	0	0	0	7	0	3
4	0	5	7	0	0	0
5	0	0	0	0	0	3
6	0	0	3	0	3	0

12. AIM: Write a java program to implement Floyd's algorithm for the all pairs shortest path problem

SOURCE CODE

```
import java.util.Scanner;

public class FloydWarshall
{
   private int distancematrix[][];
   private int numberofvertices;
   public static final int INFINITY = 999;

   public FloydWarshall(int numberofvertices)
   {
      distancematrix = new int[numberofvertices + 1][numberofvertices + 1];
      this.numberofvertices = numberofvertices;
   }

   public void floydwarshall(int adjacencymatrix[][])
   {
      for (int source = 1; source <= numberofvertices; source++)
      {
         for (int destination = 1; destination <= numberofvertices; destination++)
         {
            distancematrix[source][destination] = adjacencymatrix[source][destination];
         }
      }

      for (int intermediate = 1; intermediate <= numberofvertices; intermediate++)
      {
         for (int source = 1; source <= numberofvertices; source++)
         {
            for (int destination = 1; destination <= numberofvertices; destination++)
            {
               if (distancematrix[source][intermediate] + distancematrix[intermediate][destination]
                  < distancematrix[source][destination])
                  distancematrix[source][destination] = distancematrix[source][intermediate]
                     + distancematrix[intermediate][destination];
            }
         }
      }

      for (int source = 1; source <= numberofvertices; source++)
         System.out.print("\t" + source);

      System.out.println();
      for (int source = 1; source <= numberofvertices; source++)
      {
         System.out.print(source + "\t");
         for (int destination = 1; destination <= numberofvertices; destination++)
```

```
            {
                System.out.print(distancematrix[source][destination] + "\t");
            }
            System.out.println();
        }
    }

    public static void main(String... arg)
    {
        int adjacency_matrix[][];
        int numberofvertices;

        Scanner scan = new Scanner(System.in);
        System.out.println("Enter the number of vertices");
        numberofvertices = scan.nextInt();

        adjacency_matrix = new int[numberofvertices + 1][numberofvertices + 1];
        System.out.println("Enter the Weighted Matrix for the graph");
        for (int source = 1; source <= numberofvertices; source++)
        {
            for (int destination = 1; destination <= numberofvertices; destination++)
            {
                adjacency_matrix[source][destination] = scan.nextInt();
                if (source == destination)
                {
                    adjacency_matrix[source][destination] = 0;
                    continue;
                }
                if (adjacency_matrix[source][destination] == 0)
                {
                    adjacency_matrix[source][destination] = INFINITY;
                }
            }
        }

        System.out.println("The Transitive Closure of the Graph");
        FloydWarshall floydwarshall = new FloydWarshall(numberofvertices);
        floydwarshall.floydwarshall(adjacency_matrix);
        scan.close();
    }
}
$javac FloydWarshall.java
$java FloydWarshall

Enter the number of vertices
4

Enter the Weighted Matrix for the graph
0 0 3 0
2 0 0 0
```

0 7 0 1
6 0 0 0

The Transitive Closure of the Graph

	1	2	3	4
1	0	10	3	4
2	2	0	5	6
3	7	7	0	1
4	6	16	9	0

13. AIM: Write a java program to implement Dynamic Programming algorithm for the 0/1 Knapsack problem.

SOURCE CODE

```
import java.util.Scanner;

public class Knapsack_DP
{
   static int max(int a, int b)
   {
      return (a > b)? a : b;
   }
   static int knapSack(int W, int wt[], int val[], int n)
   {
      int i, w;
      int [][]K = new int[n+1][W+1];

            // Build table K[][] in bottom up manner
      for (i = 0; i <= n; i++)
      {
         for (w = 0; w <= W; w++)
         {
            if (i==0 || w==0)
               K[i][w] = 0;
            else if (wt[i-1] <= w)
               K[i][w] = max(val[i-1] + K[i-1][w-wt[i-1]],  K[i-1][w]);
            else
               K[i][w] = K[i-1][w];
         }
      }

      return K[n][W];
   }

   public static void main(String args[])
   {
      Scanner sc = new Scanner(System.in);
      System.out.println("Enter the number of items: ");
      int n = sc.nextInt();
      System.out.println("Enter the items weights: ");
      int []wt = new int[n];
      for(int i=0; i<n; i++)
         wt[i] = sc.nextInt();

      System.out.println("Enter the items values: ");
      int []val = new int[n];
      for(int i=0; i<n; i++)
         val[i] = sc.nextInt();
```

```
        System.out.println("Enter the maximum capacity: ");
        int W = sc.nextInt();

        System.out.println("The maximum value that can be put in a knapsack of capacity W is: " +
knapSack(W, wt, val, n));
        sc.close();
    }
}
```

Output:

```
$ javac Knapsack_DP.java
$ java Knapsack_DP

Enter the number of items:
5
Enter the items weights:
01 56 42 78 12
Enter the items values:
50 30 20 10 50
Enter the maximum capacity:
150
The maximum value that can be put in a knapsack of capacity W is: 150
```

14. AIM: Write a java program to implement Dynamic Programming algorithm for the Optimal Binary Search Tree Problem.

SOURCE CODE

```
import java.util.Scanner;

 /* Class BSTNode */
 class BSTNode
 {
    BSTNode left, right;
    int data;

    /* Constructor */
    public BSTNode()
    {
       left = null;
       right = null;
       data = 0;
    }
    /* Constructor */
    public BSTNode(int n)
    {
       left = null;
       right = null;
       data = n;
    }
    /* Function to set left node */
    public void setLeft(BSTNode n)
    {
       left = n;
    }
    /* Function to set right node */
    public void setRight(BSTNode n)
    {
       right = n;
    }
    /* Function to get left node */
    public BSTNode getLeft()
    {
       return left;
    }
    /* Function to get right node */
    public BSTNode getRight()
    {
       return right;
    }
    /* Function to set data to node */
    public void setData(int d)
    {
```

```
        data = d;
    }
    /* Function to get data from node */
    public int getData()
    {
        return data;
    }
}

/* Class BST */
class BST
{
    private BSTNode root;

    /* Constructor */
    public BST()
    {
        root = null;
    }
    /* Function to check if tree is empty */
    public boolean isEmpty()
    {
        return root == null;
    }
    /* Functions to insert data */
    public void insert(int data)
    {
        root = insert(root, data);
    }
    /* Function to insert data recursively */
    private BSTNode insert(BSTNode node, int data)
    {
        if (node == null)
            node = new BSTNode(data);
        else
        {
            if (data <= node.getData())
                node.left = insert(node.left, data);
            else
                node.right = insert(node.right, data);
        }
        return node;
    }
    /* Functions to delete data */
    public void delete(int k)
    {
        if (isEmpty())
            System.out.println("Tree Empty");
        else if (search(k) == false)
            System.out.println("Sorry "+ k +" is not present");
```

```
    else
    {
       root = delete(root, k);
       System.out.println(k+ " deleted from the tree");
    }
}
private BSTNode delete(BSTNode root, int k)
{
    BSTNode p, p2, n;
    if (root.getData() == k)
    {
       BSTNode lt, rt;
       lt = root.getLeft();
       rt = root.getRight();
       if (lt == null && rt == null)
          return null;
       else if (lt == null)
       {
          p = rt;
          return p;
       }
       else if (rt == null)
       {
          p = lt;
          return p;
       }
       else
       {
          p2 = rt;
          p = rt;
          while (p.getLeft() != null)
             p = p.getLeft();
          p.setLeft(lt);
          return p2;
       }
    }
    if (k < root.getData())
    {
       n = delete(root.getLeft(), k);
       root.setLeft(n);
    }
    else
    {
       n = delete(root.getRight(), k);
       root.setRight(n);
    }
    return root;
}
/* Functions to count number of nodes */
public int countNodes()
```

```
{
   return countNodes(root);
}
/* Function to count number of nodes recursively */
private int countNodes(BSTNode r)
{
   if (r == null)
      return 0;
   else
   {
      int l = 1;
      l += countNodes(r.getLeft());
      l += countNodes(r.getRight());
      return l;
   }
}
/* Functions to search for an element */
public boolean search(int val)
{
   return search(root, val);
}
/* Function to search for an element recursively */
private boolean search(BSTNode r, int val)
{
   boolean found = false;
   while ((r != null) && !found)
   {
      int rval = r.getData();
      if (val < rval)
         r = r.getLeft();
      else if (val > rval)
         r = r.getRight();
      else
      {
         found = true;
         break;
      }
      found = search(r, val);
   }
   return found;
}
/* Function for inorder traversal */
public void inorder()
{
   inorder(root);
}
private void inorder(BSTNode r)
{
   if (r != null)
   {
```

```
            inorder(r.getLeft());
            System.out.print(r.getData() +" ");
            inorder(r.getRight());
        }
    }
    /* Function for preorder traversal */
    public void preorder()
    {
        preorder(root);
    }
    private void preorder(BSTNode r)
    {
        if (r != null)
        {
            System.out.print(r.getData() +" ");
            preorder(r.getLeft());
            preorder(r.getRight());
        }
    }
    /* Function for postorder traversal */
    public void postorder()
    {
        postorder(root);
    }
    private void postorder(BSTNode r)
    {
        if (r != null)
        {
            postorder(r.getLeft());
            postorder(r.getRight());
            System.out.print(r.getData() +" ");
        }
    }
}

/* Class BinarySearchTree */
public class BinarySearchTree
{
    public static void main(String[] args)
    {
        Scanner scan = new Scanner(System.in);
        /* Creating object of BST */
        BST bst = new BST();
        System.out.println("Binary Search Tree Test\n");
        char ch;
        /*  Perform tree operations  */
        do
        {
            System.out.println("\nBinary Search Tree Operations\n");
            System.out.println("1. insert ");
```

```
            System.out.println("2. delete");
            System.out.println("3. search");
            System.out.println("4. count nodes");
            System.out.println("5. check empty");

            int choice = scan.nextInt();
            switch (choice)
            {
            case 1 :
               System.out.println("Enter integer element to insert");
               bst.insert( scan.nextInt() );
               break;
            case 2 :
               System.out.println("Enter integer element to delete");
               bst.delete( scan.nextInt() );
               break;
            case 3 :
               System.out.println("Enter integer element to search");
               System.out.println("Search result : "+ bst.search( scan.nextInt() ));
               break;
            case 4 :
               System.out.println("Nodes = "+ bst.countNodes());
               break;
            case 5 :
               System.out.println("Empty status = "+ bst.isEmpty());
               break;
            default :
               System.out.println("Wrong Entry \n ");
               break;
            }
            /*  Display tree  */
            System.out.print("\nPost order : ");
            bst.postorder();
            System.out.print("\nPre order : ");
            bst.preorder();
            System.out.print("\nIn order : ");
            bst.inorder();

            System.out.println("\nDo you want to continue (Type y or n) \n");
            ch = scan.next().charAt(0);
        } while (ch == 'Y'|| ch == 'y');
    }
 }
```

Binary Search Tree Test

Binary Search Tree Operations
1. insert
2. delete
3. search

```
4. count nodes
5. check empty
5
Empty status = true
 Post order :
Pre order :
In order :
Do you want to continue (Type y or n)

y
 Binary Search Tree Operations
 1. insert
2. delete
3. search
4. count nodes
5. check empty
1
Enter integer element to insert
8
 Post order : 8
Pre order : 8
In order : 8
Do you want to continue (Type y or n)
```