Department of Computer Science & Engineering
Faculty of Engineering & Technology (ITER)

Assignment - 1

Q: Program for Adjacency matrix representation of an Undirected graph using JAVA.

⇒
```
import java.util.Scanner;
public class AdjacencyMatrix {
  private int [][] adjacencyMatrix;
  private int numberOfVertices;
  public AdjacencyMatrix (int numberOfVertices) {
    this.numberOfVertices = numberOfVertices;
    adjacencyMatrix = new int [numberOfVertices][numberOfVertices];
  }
  public void addEdge (int source, int destination) {
    if (source <0 || source >= numberOfVertices || destination <0 ||
                                  destination >= numberOfVertices) {
        System.out.println ("Invalid edge!");
    }else {
        adjacencyMatrix [source] [destination] = 1;
        adjacencyMatrix [destination] [source] = 1;
    }
  }

  public static void main (String [] args) {
    Scanner scanner = new Scanner (System.in);
    System.out.println ("Enter the number of vertices:");
    int vertices = scanner.nextInt();
    AdjacencyMatrix graph = new AdjacencyMatrix (vertices);
    System.out.println ("Enter the number of edges:");
    int edges = scanner.nextInt();
    System.out.println ("Enter the edges (source and destination):");
    for (int i=0; i< edges; i++) {
        int source = scanner.nextInt();
        int destination = scanner.nextInt();
        graph.addEdge (source, destination);
    }
    graph.displayMatrix();
    scanner.close();
  }
}
```

Name:_____ Regd. Number:_____

Output:

```
Enter the number of vertices: 4
Enter the number of edges: 5
Enter the edges (source and destination):
01 , 12 , 23 , 02 , 13.
Adjacency Matrix:
0 1 1 0
1 0 1 1
1 1 0 1
0 1 1 0
```

Q:- Program for Adjacency List representation of a Graph Using JAVA.

⇒

```
import java.util.LinkedList;
import java.util.Scanner;
class Graph {
    private int vertices;
    private LinkedList<Integer>[] adjacencyList;
    public Graph(int vertices){
        this.vertices = vertices;
        adjacencylist = new LinkedList[vertices];
        for (int i=0; i<vertices; i++){
            adjacencylist[i] = new LinkedList<>();
        }
    }
    public void addEdge(int source, int destination){
        adjacencyList[source].add(destination);
        adjacencyList[destination].add(source);
    }
    public void displayGraph(){
        for (int i=0; i<vertices; i++){
            System.out.print("Vertex" + i + ":");
            for (Integer neighbour : adjacencyList[i]){
                System.out.print(neighbour + " ");
            }
            System.out.println();
        }
    }
```

Department of Computer Science & Engineering
Faculty of Engineering & Technology (ITER)

```
public static void main (String [] args) {
    Scanner scanner = new Scanner (System.in);
    System.out.print ("Enter the number of vertices:");
    int vertices = scanner.nextInt();
    System.out.print("Enter the number of edges:");
    int edges = scanner.nextInt();
    Graph graph = new Graph (vertices);
    System.out.println("Enter the edges (source and destination):");
    for (int i=0; i<edges; i++){
        int source = scanner.nextInt();
        int destination = scanner.nextInt();
        graph.addEdge (source, destination);
    }
    System.out.~~println~~("\nAdjacency List:");
    graph.displayGraph();
    Scanner.close();
  }
}
```

<u>Output</u>: Enter the number of vertices : 4

Enter the number of edges : 5

Enter the edges (source and destination):

0 1 , 1 2 , 2 3 , 0 2 , 1 3.

Adjacency List:

Vertices 0 : 1 2

Vertices 1 : 0 2 3

Vertices 2 : 1 3 0

Vertices 3 : 2 1

Q.No: 2) Program for DFS (using stack) of a Graph using JAVA.

```
import java.util.Stack;
class DFS{
  public static void dfs(int startVertex, int [][] adjacencyMatrix){
    int vertices = adjacencyMatrix.length;
    boolean [] visited = new boolean [vertices];
    Stack<Integer> stack = new Stack<>();
    stack.push(startVertex);
    while (!stack.isEmpty()){
      int currentVertex = stack.pop();
      if (!visited [currentVertex]) {
        System.out.print (currentVertex + " ");
        visited [currentVertex] = true;
        for (int i= vertices -1; i>=0; i--){
          if (adjacencyMatrix [currentVertex][i]==1 &&
                                          !visited [i]) {
            stack.push(i);
          }
        }
      }
    }
  }
  public static void main (String [] args){
    int [][] adjacencyMatrix = {
      {0, 1, 1, 0, 0},
      {1, 0, 0, 1, 1},
      {1, 0, 0, 0, 0},
      {0, 1, 0, 0, 0},
      {0, 1, 0, 0, 0}
    };
    System.out.println(" DFS starting from vertex 0:");
    dfs (2, adjacencyMatrix);
  }
}
```

Output:

DFS starting from vertex 2:
2 0 1 3 4.

Department of Computer Science & Engineering
Faculty of Engineering & Technology (ITER)

Q: Program for DFS (Recursive) of A Graph using JAVA.

⇒
```
public class DFSRecursive{
    private static void dfs (int [][] graph, boolean [] visited, int node){
        visited [node] = true;
        System.out.print (node + " ");
        for (int i=0; i< graph [node].length; i++){
            if (graph [node][i] == 1 && !visited [i]){
                dfs (graph, visited, i);
            }
        }
    }
    public static void main (String [] args){
        int [][] graph = {
            {0, 1, 1, 0, 0},
            {1, 0, 0, 1, 1},
            {1, 0, 0, 0, 0},
            {0, 1, 0, 0, 0},
            {0, 1, 0, 0, 0}
        };
        int numberOfNodes = graph.length;
        boolean [] visited = new boolean [numberOfNodes];
        System.out.println ("DFS Traversal starting from node 0:");
        dfs (graph, visited, 2);
    }
}
```

Output: DFS starting from vertex 2: 2 0 1 3 4

Q: Program for BFS (Using Queue) of a Graph Using JAVA.

⇒
```
import java.util.LinkedList;
import java.util.Queue;
public class BFS{
    public static void bfs (int [][] adjacencyMatrix, int startNode){
        int numberOfNodes = adjacencyMatrix.length;
        boolean [] visited = new boolean [numberOfNodes];
        Queue <Integer> queue = new LinkedList <> ();
        visited [startNode] = true;
        queue.add (startNode);
        System.out.println ("BFS Traversal: ");
```

```
        while (!queue.isEmpty()) {
            int currentNode = queue.poll();
            System.out.print(currentNode+" ");
            for (int i=0; i< numberOfNodes; i++){
                if (adjacencyMatrix [currentNode][i] == 1 &&
                        !visited [i]{
                    visited[i] = true;
                    queue.add(i);
                }
            }
        }
    }
    public static void main (String [] args){
        int [][] adjacencyMatrix = {
            { 0, 1, 1, 0 },
            { 1, 0, 1, 1 },
            { 1, 1, 0, 1 },
            { 0, 1, 1, 0 }
        });
        bfs (adjacencyMatrix, 3);
    }
}
```

Output: BFS Traversal:

3 1 2 0.

4.2.25

Ankan Dhar
2341004233

Department of Computer Science & Engineering
Faculty of Engineering & Technology (ITER)

Assignment - 2

Q: Program to find if there is a path from vertex U to vertex V in a directed graph using JAVA.

⇒
```
import java.util.Scanner;
class DirectedGraph {
    private int vertices;
    private int [][] adjMatrix;
    public DirectedGraph (int[][] matrix){
        this.vertices = matrix.length;
        this.adjMatrix = matrix;
    }
    public boolean isPathExists(int u, int v){
        boolean [] visited = new boolean [vertices];
        return dfs(u, v, visited);
    }
    private boolean dfs (int u, int v, boolean [] ) {
        if (u==v) return true;
        visited [u] = true;
        for (int i=0; i< vertices ; i++) {
            if (adjMatrix [u][i] == 1 && !visited [i]) {
                if (dfs(i,v, visited)){
                    return true;
                }
            }
        }
        return false;
    }
    public static void main (String [] args){
        int [][] matrix = {
            {0, 1, 0, 0, 0},
            {0, 0, 1, 0, 0},
            {1, 0, 0, 1, 0},
            {0, 0, 0, 0, 1},
            {0, 0, 0, 0, 0 }
        };
        DirectedGraph graph = new DirectedGraph(matrix);
        Scanner sc = new Scanner (System.in);
        System.out.println("Enter source and destination");
        int source = sc.nextInt();
        int destination = sc.nextInt();
```

```
        if (graph.isPathExists(source, destination)) {
            System.out.println("Path exists");
        } else {
            System.out.println("No path exists");
        }
    }
}
```

Output: Enter source and destination
4 0
No path exists from 4 to 0.

Q: Program to print all paths from a given source vertex to a target in a Directed graph using JAVA.

```
public class DirectedGraph {
    private int vertices;
    private int[][] adjMatrix;
    public DirectedGraph (int[][] matrix) {
        this.vertices = matrix.length;
        this.adjMatrix = matrix;
    }
    public void printAllPaths (int source, int destination) {
        boolean[] visited = new boolean[vertices];
        int[] path = new int[vertices];
        int pathIndex = 0;
        System.out.println(" All Paths from" + source + " to " +
            destination + ":");
        dfs (source, destination, visited, path, pathIndex);
    }
    private void dfs (int u, int v, boolean[] visited, int[] path,
                      int pathIndex) {
        visited[u] = true;
        path[pathIndex] = u;
        pathIndex++;
        if (u == v) {
            printPath (path, pathIndex);
```

Department of Computer Science & Engineering
Faculty of Engineering & Technology (ITER)

```
        }else {
            for (int i=0; i< vertices ; i++){
                if (adjMatrix [u][i] == 1 && !visited [i] ){
                    dfs (i, v, visited, path, pathIndex);
                }
            }
        }
        visited [u] = false;
    }
    private void printPath (int [] path, int length){
        for (int i=0; i< length ; i++){
            System.out. print (path [i] + (i==length -1 ? " " : "->"));
        }
        System.out. println ();
    }
    public static void main (String [] args){
        int[][] matrix={
            {0,1,1,0,0},
            {0,0,1,1,0},
            {0,0,0,1,1},
            {0,0,0,0,1},
            {0,0,0,0,0}
        };
        Directed Graph graph = new Directed Graph (matrix);
        int source =0, destination = 4;
        graph.printAll paths (source, destination);
    }
}
```

<u>Outputs</u>: All Paths from 0 to 4:

0 → 1 → 2 → 3 → 4

0 → 1 → 2 → 4

0 → 1 → 3 → 4

0 → 2 → 3 → 4

0 → 2 → 4.

Q: Program to find if there is a cycle in an undirected graph using JAVA.

⇒
```
public class UndirectedGraph {
    private int vertices;
    private int[][] adjMatrix;
    public UndirectedGraph (int[][] matrix){
        this.vertices = matrix.length;
        this.adjMatrix = matrix;
    }
    public boolean hasCycle() {
        boolean[] visited = new boolean[vertices];
        for (int i=0; i<vertices; i++){
            if (!visited[i]){
                if (dfs(i, -1, visited)){
                    return true;
                }
            }
        }
        return false;
    }
    private boolean dfs(int current, int parent, boolean[] visited){
        visited[current] = true;
        for (int i=0; i<vertices; i++){
            if (adjMatrix[current][i] == 1){
                if (!visited[i]){
                    if (dfs(i, current, visited)){
                        return true;
                    }
                }else if (i != parent){
                    return true;
                }
            }
        }
        return false;
    }
    public static void main (String[] args){
        int[][] matrix = {
```

```
            {0, 1, 0, 0, 1},
            {1, 0, 1, 1, 0},
            {0, 1, 0, 1, 0},
            {0, 1, 1, 0, 1},
            {1, 0, 0, 1, 0}
        };
        UndirectedGraph graph = new UndirectedGraph(matrix);
        if (graph.hasCycle()) {
            System.out.println("The graph contains a cycle.");
        } else {
            System.out.println("The graph does not contain a cycle");
        }
    }
}
```

Output: The graph contains a cycle.

Q: Program for MST (Prim's Algorithm) of an Undirected graph Using JAVA.

⇒

```
import java.util.Arrays;
public class PrimsAlgorithm {
    public static void primMST(int[][] graph) {
        int v = graph.length;
        int [] parent = new int [v];
        int [] key = new int [v];
        boolean [] mstSet = new boolean [v];
        Arrays.fill (key, Integer.MAX_VALUE);
        Arrays.fill (mstSet, false);
        key [0] = 0;
        parent [0] = -1;
        for (int count = 0; count < v-1; count++) {
            int u = minKey (key, mstSet, v);
            mstSet [u] = true;
```

```
            for (int v=0; v<V; v++){
                if (graph[u][v] != 0 && !mstSet[v] &&
                                graph[u][v] < key[v]){
                    parent[v] = u;
                    key[v] = graph[u][v];
                }
            }
        }
        printMst(parent, graph);
    }
    private static int minKey(int[] key, boolean[] mstSet, int v){
        int min = Integer.MAX_VALUE, minIndex = -1;
        for (int v=0; v<V; v++){
            if (!mstSet[v] && key[v] < min){
                min = key[v];
                minIndex = v;
            }
        }
        return minIndex;
    }
    private static void printMST(int[] parent, int[][] graph){
        System.out.println("Edge \tWeight");
        for (int i=1; i< graph.length; i++){
            System.out.println(parent[i] + "-"+i+"\t" + graph[i]
                                        [parent[i]]);
        }
    }
    public static void main(String[] args){
        int[][] graph = {
            {0, 2, 0, 6, 0},
            {2, 0, 3, 8, 5},
            {0, 3, 0, 0, 7},
            {6, 8, 0, 0, 9},
            {0, 5, 7, 9, 0}
        };
        primMST(graph);
    }
}
```

Output:

Edge	Weight
0-1	2
1-2	3
0-3	6
1-4	5

Department of Computer Science & Engineering
Faculty of Engineering & Technology (ITER)

Q: Program for Shortest path (Dijktra's Algorithm) of an Undirected graph using JAVA.

⇒
```
import java.util.Arrays;
public class DijkstraAlgorithm {
    static int findMinDistance (int[] dist, boolean [] visited , int vertices){
        int min = Integer.MAX_VALUE , minIndex = -1;
        for (int v=0; v< vertices ; v++){
            if (!visited [v] && dist[v] < min){
                min = dist [v];
                minIndex = v;
            }
        }
        return minIndex;
    }
    static void dijkstra (int[][] graph , int source){
        int vertices = graph.length;
        int [] dist = new int [vertices];
        boolean[] visited = new boolean [vertices];
        Array.fill (dist , Integer.MAX_VALUE);
        dist [source] = 0;
        for (int i=0; i< vertices -1 ; i++){
            int u = findMinDistance (dist, visited , vertices);
            visited [u] = true;
            for (int v=0; v< vertices ; v++){
                if (!visited [v] && graph [u][v] != 0 &&
                    dist [u] != Integer.MAX_VALUE && dist [u] + graph
                                                        [u][v] < dis{
                        dist [v] = dist [u] + graph [u][v];
                }
            }
        }
        printSolution(dist);
    }
```

```
    static void printSolution (int[] dist){
        System.out.println(" Vertex \t Distance from Source");
        for (int i=0; i< dist.length; i++){
            System.out.println(i + "\t" + dist[i]);
        }
    }
    public static void main (String[] args) {
        int [][] graph = {
                {0, 7, 9, 0, 0, 14},
                {7, 0, 10, 15, 0, 0},
                {9, 10, 0, 11, 0, 2},
                {0, 15, 11, 0, 6, 0},
                {0, 0, 0, 6, 0, 9},
                {14, 0, 2, 0, 9, 0},
        };
        int source = 0;
        dijkstra(graph, source);
    }
}
```

Output:

Vertex	Distance from Source.
0	0
1	7
2	9
3	20
4	20
5	11

4.3.23

Aukan Dhar
2341004233

Department of Computer Science & Engineering
Faculty of Engineering & Technology (ITER)

Assignment - 3

Q: Implement LCS algorithm.

⇒
```
public class LCS{
  static int lcs (String S1, String S2){
      int n=S1.length();
      int m = S2.length();
      int [][]c = new int [n+1][m+1];
      for (int i=1; i<=m; i++){
         for (int j=1; j<=m; j++){
            if (S1.charAt(i-1) == S2.charAt(j-1))
               c[i][j] = c[i-1][j-1]+1;
            else
               c[i][j] = Math.max(c[i][j-1], c[i-1][j]);
         }
      }
      return c[n][m]
  }
  public static void main (String [] args){
      String S1 = "bcdaacd";
      String S2 = "acbdac";
      System.out.println("LCS(longest common subsequence) length :"+
                                                lcs(S1,S2));
  }
}
```

Output:

LCS (longest common subsequence) length : 4.

Q: Implement matrix chain multiplication.

⇒
```
public class MatrixChainMult{
  Static int matrixchainorder (int[]P, int n){
      int [][]m = new int [n][m];
      for (int i=1; 1<n; i++){
          m[1][i] = 0;
      }
      for (int l=2; l<n; l++){
         for (int i=1; i<n-l+1; i++){
            int J = i+L-1;
            m[i][J] = Integer.MaxVALUE;
```

```
            for (int k = i ; k < j ; k++){
                int cost = m[i][k] + m[k+1][j] + L[i-1] * C[k] * P[j];
                m[i][j] = Math.min (m[i][j], cost);
            }
        }
    }
    return m[1][n-1];
}
public static void main (String [] args){
    int [] arr = {1, 2, 3};
    int n = arr. length;
    System. out. println ("Minimum multiplication cost : " +
                                matrixchainorder(arr, n));
    }
}
```

Output:

Minimum multiplication cost : 6.

18-3-25

Ankan Dhar
2341004233

Department of Computer Science & Engineering
Faculty of Engineering & Technology (ITER)

Assignment - 4

1.

⇒
```
public class HashTableP {
    private static final int EMPTY_VALUE = -1;
    private static final int DELETED_VALUE = -2;
    private static final int FILLED_VALUE = 0;
    private int tableSize;
    private int[] key;
    private int[] value;
    private int[] flag;
    public HashTableLP (int size) {
        this.tableSize = size;
        Key = new int [tableSize];
        Value = new int [tableSize];
        Flag = new int [tableSize];
        for (int i=0; i < tableSize; i++) {
            Flag [i] = EMPTY_VALUE;
        }
    }
    private int computeHash (int key) {
        return key % tableSize;
    }
    private int resolverFun (int index) {
        return (index + 1) % tableSize;
    }
public boolean add (int key, int value) {
    int index = computeHash (key);
    for (int i=0; i < tableSize; i++) {
        if (Flag [index] != FILLED_VALUE) {
            Key [index] = key;
            Value [index] = value;
            Flag [index] = FILLED_VALUE;
            return true;
        } else if (key [index] == key) {
            Value [index] = value;
            return true;
        }
        index = resolverFun (index);
    }
    System.out.println ("Hash table is full !");
    return false;
}
```

```
public boolean find (int key){
    int index = computeHash (key);
    for (int i=0; i< tableSize && Flag [index] != EMPTY_VALUE ; i++){
        if (Flag [index] == FILLED_VALUE && Key [index] == key ){
            return true;
        }
        index = resolverFun (index);
    }
    return false;
}
public int get (int key){
    int index = computeHash (key);
    for (int i=0; i< tableSize && Flag [index] != EMPTY_VALUE ; i++){
        if (Flag [index] == FILLED_VALUE && key [index] == key){
            return value [index];
        }
        index = resolverFun (index);
    }
    throw new IllegalArgumentException ("Key not found");
}
public boolean remove (int key){
    int index = computeHash (key);
    for (int i=0; i< tableSize && Flag [index] != EMPTY_VALUE; i++){
        if (Flag [index] == FILLED_VALUE && KEY [index] == key){
            Flag [index] = DELETED_VALUE;
            return true;
        }
        index = resolverFun (index);
    }
    return false
}
public void print(){
    for (int i=0; i < tableSize ; i++){
        if (Flag [i] == FILLED_VALUE){
            System.out.println ("Index "+ i + ": Key = "+ Key [i] + ", value
                                = "+ Value [i] );
        } else if ( Flag [i] == DELETED_VALUE){
            System.out.println ("Index " + i + ": Deleted");
        } else {
            System.out.println ("Index" + i + ": Empty");
        }
    }
}
```

```
public static void main (String [] args){
    HashTableLP ht = new HashTableLP(5);
    ht.add (1, 10);
    ht.add (2, 20);
    ht.add (3, 30);
    ht.add (6, 60);
    ht.print();
    System.out.println ("Find key 2:" + ht.find (2));
    System.out.println (" Get key 3:" + ht.get (3));
    ht.remove (2);
    System.out.println ("After removing key 2:");
    ht.print();
  }
}
```

Output:

Index 1 : Key = 1 , Value = 10
Index 2 : Key = 2 , Value = 20
Index 3 : Key = 3 , Value = 30
Index 4 : Key = 6 , Value : 60
Find key 2 : true
Get key 3 : 30
After removing key 2 :
Index 0 : Empty
Index 1 : Key 1, Value = 10.
Index 2 : Deleted .
Index 3 : Key = 3 , Value = 30 .
Index 4 : Key = 6 , Value = 60 .

2.

⇒

```
import java.util.HashMap;
import java.util.Map;
public class DictionaryExample {
    HashMap<String, String> dictionary = new HashMap<>();
    dictionary.put ("Java", "A high-level programing language.");
    dictionary.put ("Python", "A widely used interpreted language.");
    dictionary.put ("HashMap", "A data structure to store key-value
                                pairs.");
```

```
        System.out.println("Definition of Java:" + dictionary.get("Java"));
        System.out.println("Definition of Python:" + dictionary.get("Python"));
        if (dictionary.containsKey("C++")) {
            System.out.println("Definition of C++:" + dictionary.get("C++"));
        } else {
            System.out.println("C++ is not in the dictionary.");
        }
        dictionary.remove("HashMap");
        System.out.println("\nDictionary Contents:");
        for (Map.Entry<String, String> entry : dictionary.entrySet()) {
            System.out.println(entry.getKey() + ":" + entry.getValue());
        }
    }
}
```

Output:

Definition of Java: A high-level programming language.
Definition of Python: A widely used interpreted language.
C++ is not in the dictionary.

Dictionary Contents:
Java: A high-level programming language.
Python: A widely used interpreted language.

25.3.25

Ankan Dhar
2341004233

Assignment - 5

1. Write a program to implement naive string matching algorithm in java.

⇒
```
public class NaiveStringMatching {
    public static void naiveStringMatcher (String text, String pattern) {
        int n = text.length();
        int m = pattern.length();
        for (int s = 0; s<=n-m ; s++){
            int i;
            for (i=0; i<m ; i++) {
                if (text.charAt (s+i) != pattern.charAt (i)) {
                    break;
                }
            }
            if (i==m) {
                System.out.println ("Pattern found at index" + s);
            }
        }
    }
    public static void main (String[] args) {
        String text = "CCACCAAEDBA";
        String Pattern = "DBA";
        naiveStringMatcher (text, pattern);
    }
}
```

Output: Pattern found at index 8.

2. Write a program to implement Rabin-Karp string matching algorithm in Java.

⇒
```
public class RabinKarp {
    public static void rabinKarp (String text, String pattern, int base){
        int n = text.length();
        int m = pattern.length();
        long patternHash = calculateHash (pattern, m, base);
        long textHash = calculateHash (text, m, base);
        long h = calculatePower (base, m-1);

        for (int i=0; i <=n-m ; i++) {
            if (patternHash == textHash) {
                if (pattern.equals (text.substring (i, i+m))) {
```

```
                System.out.println("Pattern found at index" + 1);
            }
        }
        if (i<n-m){
            textHash = (textHash - text.charAt(i) * h);
            textHash = (textHash * base + text.charAt(i+m));
        }
    }
}
private static long calculateHash (String str, int length, int base){
    long hash = 0;
    for (int i=0; i< length ; i++){
        hash = hash * base + str.charAt(i);
    }
    return hash;
}
private static long calculatePower (int base, int exponent){
    long result =1;
    for (int i=0; i< exponent ; i++){
        result = result * base;
    }
    return result;
}
public static void main (String [] args){
    String text = "CCACCAAEDBA";
    String pattern = "EDB";
    int base = 256 ;
    rabinKarp (text, pattern, base);
}
}
```

Output: Pattern found at index 7.

Dhanda
1.4.25

Ankan Dhar
2341004233

Department of Computer Science & Engineering
Faculty of Engineering & Technology (ITER)

Assignment - 06

1. Symbol Table Using String:

⇒
```
public class StringTree {
    Node root = null;
    class Node {
        String value;
        int count;
        Node lchild;
        Node rchild;
    }
    public void printTree() {
        printTree(root);
    }
    public void printTree (Node curr) {
        if (curr != null) {
            System.out.print("value is : : "+ curr.value);
            System.out.println("\tcount is : : "+ curr.value);
            printTree (curr.lchild);
            printTree (curr.rchild);
        }
    }
    public void add (String value) {
        root = add (value, root);
    }
    Node add (String value, Node curr) {
        if (curr == null) {
            curr = new Node();
            curr.value = value;
            curr.lchild = curr.rchild = null;
            curr.count = 1;
        } else {
            int compare = curr.value.compareTo (value);
            if (compare == 0)
                curr.count ++;
            else if (compare == 1)
                curr.lchild = add (value, curr.lchild);
            else
                curr.rchild = add (value, curr.rchild);
        }
        return curr;
    }
    boolean find (String value) {
        boolean ret = find (root, value);
        return ret;
    }
```

```
boolean find (Node curr, String value) {
    if (curr = null)
        return false;
    int compare = curr.value.compareTo(value);
    if (compare = = 0)
        return true;
    else {
        if (compare = = 1)
            return find (curr.lchild, value);
        else
            return find (curr.rchild, value);
    }
}
int frequency (String value) {
    return frequency (root, value);
}
int frequency (Node curr, String value) {
    if (curr = = null)
        return 0;
    int compare = curr.value.compareTo (value);
    if (compare = = 0)
        return curr.count;
    else {
        if (compare > 0)
            return frequency (curr.lchild, value);
        else
            return frequency (curr.rchild, value);
    }
}
void freeTree () { root = null;
}
public static void main (String [] args) {
    StringTree tt = new StringTree ();
    tt.add ("mango");
    tt.add ("banana");
    tt.add ("apple");
    tt.add ("mango");
    System.out.println ("Apple Found : "+ tt.find ("apple"));
    System.out.println ("Banana Found : "+ tt.find ("banana"));
    System.out.println ("Grapes Found : "+ tt.find ("grapes"));
```

```
        System.out.println("\n Symbol Table is: ");
        tt.printTree();
    }
}
```

Output: Apple Found : true.
Banana Found : true.
Grapes found : false.
Sysbol Table is
Value is :: mango count is :: 2
Value is :: banana count is :: 1
Value is :: apple count is :: 1.

2. i) Interval Scheduling

⇒
```
import java.util.*;
class Interval implements Comparable<Interval>{
    int start, end;
    public Interval (int start, int end){
        this.start = start;
        this.end = end;
    }
    @Override
    public int compareTo (Int start, int end){
        this.start = start;
        this.end = end;
    }
    public class IntervalScheduling {
        public static List<Interval> intervalScheduling (List<Interval>){
            Collections.sort(intervals);
            List<Interval> selectedIntervals = new ArrayList<>();
            int lastEndTime = Integer.MIN_VALUE.
            for (Interval interval : intervals){
                if (interval.start >= lastEndTime){
                    selectedIntervals.add(interval);
                    lastEndTime = interval.end;
                }
            }
            return selectedIntervals;
    }
```

```
public static void main (String [] args) {
    List < Interval> intervals = Arrays . asList(
        new Interval (1, 3),
        new Interval (2, 4),
        new Interval (3,5),
        new (Interval (0, 6),
        new (Interval (5,7),
        new  Interval (8,9),
        new  Interval (5,9));
    List <Interval > result = intervalScheduling (intervals);
    System.out. println ("Selected Intervals: ");
    for (Interval interval : result){
        System.out. println("[" + interval. start + " , " + interval.end)
    }
  }
}
```

Output: Selected Intervals:

[1, 3]
[3, 5]
[5, 7]
[8, 9]

ii) Fractional Knapsack.

⇒

```
public class FractionalKnapsack{
    public static double fractionalKnapsack( Item [] items, int capacity) {
        int n = items. length;
        double [] valuePerWeight = new double [n];
        int [] index = new int [n];
        for (int i = 0; i<n; i++) {
            valuePerWeight [i] = (double) items[i].value / items[i].weight;
            index [i] = i;
        }
        for (int i=0; i<n-1 ; i++) {
            int maxIdx = i;
            for (int j = i+ 1; j<n ; j++ ){
                if (valuePerWeight [j] > valuePerWeight (maxIdx ] {
                    maxIdx = j;
                }
            }
```

Department of Computer Science & Engineering
Faculty of Engineering & Technology (ITER)

```
            double tempRatio = valuePerWeight [i];
            valuePerWeight [i] = valuePerWeight [maxIdx];
            valuePerWeight [maxIdx] = tempRatio;
            int tempIndex = index [i];
            index [i] = index [maxIdx];
            index [maxIdx] = tempIndex;
        }
    double totalValue = 0.0;
    for (int i = 0; i < n; i++) {
        int currIndex = index [i];
        if (items [currIndex].weight <= capacity) {
            capacity -= items [currIndex].weight;
            totalValue += items [currIndex].value;
        } else {
            totalValue += items [currIndex].value * ((double) capacity/
                                items[currIndex].weight);
            break;
        }
    }
    return totalValue;
}
public static void main (String[] args) {
    Item [] items = {
        new Item (10, 60),
        new Item (20, 100),
        new Item (30, 120), };
    int capacity = 50;
    double maxValue = fractionalKnapsack (items, capacity);
    System.out.println ("Maximum value in Knapsack = " + maxValue);
    }
}
```

Output:

Maximum value ~~of~~ in Knapsack = 240.0.

iii) Huffman Coding

=>
```
class Node {
    char character;
    int frequency;
    Node left, right;
    public Node (char character, int frequency) {
        this.character = character;
        this.frequency = frequency;
        this.left = null;
        this.right = null;
    }
}
public class HuffmanCoding {
    private static int findMinIndex (Node[] nodes, int size) {
        int minIndex = 0;
        for (int i = 1; i<size; i++) {
            if (nodes[i].frequency < nodes[minIndex].frequency) {
                mnixIndex = i;
            }
        }
        return minIndex;
    }
    public static Node buildHuffmanTree (char[] chars, int[] freqs) {
        int n = chars.length;
        Node [] nodes = new Node [n];
        for (int i=0; i<n; i++) {
            nodes [i] = new Node (chars[i], freqs (i));
        }
        int size = n;
        while (size>1) {
            int min1 = findMinIndex (nodes, size);
            Node left = nodes [min1];
            nodes [min1] = nodes [size -1];
            size -- ;
            Node merged = new Node ('-', left.frequency + right.frequency);
            merged.left = left;
            merged.right = right;
            nodes [size] = merged;
            size++;
        }
```

Department of Computer Science & Engineering
Faculty of Engineering & Technology (ITER)

```
        return nodes[0];
    }
    public static void printCodes(Node root, String code){
        if (root == null) return;
        if (root.character != '-') {
            System.out.println(root.character + ":" + code);
        }
        printCodes(root.left, code + "0");
        printCodes(root.right, code + "1");
    }
    public static void main(String[] args){
        char[] chars = {'a', 'b', 'c', 'd', 'e', 'f'};
        int[] freqs = {5, 9, 12, 13, 16, 45};
        Node root = buildHuffmanTree(chars, freqs);
        System.out.println("Huffman Codes:");
        printCodes(root, "");
    }
}
```

<u>Output</u>: Huffman Codes:

f: 0

c: 100

d: 101

a: 1100

b: 1101

e: 111

Ankan Dhal
2341004233