Assignment 3

COSC2007 – Data Structure 2
Jeel Tikiwala
239659420

Exercise No: 1
Question: Implement the table as an ADT by following methods: 1- Array-based implementation 2- Reference-based implementation 3- Binary search tree-based implementation. Implement the following TABLE ADT operations: a) Insert a new item into a table b) Delete the item with a given search key from a table c) Retrieve the item with a given search key from a table.

Algorithm/Pseudocode:

Array-Based Implementation Pseudocode:

```
Class City:
   Initialize(name, country, population)

Class CityTableArray:
   Initialize():
      cities = empty array
      size = 0

   Insert(city):
      If size equals capacity of cities array:
         Double the size of cities array
      Add city to cities array
      Increment size

   Delete(name):
      For each city in cities array:
         If city name equals name:
            Replace this city with the last city in the array
            Remove the last element from the array
            Decrement size
            Break the loop

   Retrieve(name):
      For each city in cities array:
         If city name equals name:
            Return city
      Return null
```

Reference-Based Implementation (Linked List) Pseudocode:

```
Class CityNode:
    Initialize(city):
        this.city = city
        next = null

Class CityTableLinkedList:
    Initialize():
        head = null

    Insert(city):
        newNode = new CityNode(city)
        If head is null:
            head = newNode
        Else:
            current = head
            While current.next is not null:
                Move to next node
            current.next = newNode

    Delete(name):
        If head is null:
            return
        If head.city.name equals name:
            head = head.next
            return
        current = head
        While current.next is not null:
            If current.next.city.name equals name:
                current.next = current.next.next
                return

    Retrieve(name):
        current = head
        While current is not null:
            If current.city.name equals name:
                Return current.city
            Move to next node
        Return null
```

Binary Search Tree-Based Implementation Pseudocode:

```
Class CityNodeBST:
    Initialize(city):
        this.city = city
        left = null
        right = null

Class CityTableBST:
    Initialize():
        root = null

    Insert(city):
        root = InsertRec(root, city)

    InsertRec(node, city):
        If node is null:
            Return new CityNodeBST(city)
        If city name is less than node.city name:
            node.left = InsertRec(node.left, city)
        Else if city name is greater than node.city name:
            node.right = InsertRec(node.right, city)
        Return node

    Delete(name):
        root = DeleteRec(root, name)

    DeleteRec(node, name):
        If node is null:
            Return node
        If name is less than node.city.name:
            node.left = DeleteRec(node.left, name)
        Else if name is greater than node.city.name:
            node.right = DeleteRec(node.right, name)
        Else:
            If node.left is null:
                Return node.right
            Else if node.right is null:
                Return node.left
            node.city = MinValue(node.right)
            node.right = DeleteRec(node.right, node.city.name)
        Return node

    MinValue(node):
        minv = node.city
        While node.left is not null:
            minv = node.left.city
            node = node.left
        Return minv
```

Retrieve(name):
Return RetrieveRec(root, name)

RetrieveRec(node, name):
If node is null or node.city.name equals name:
Return node
If node.city.name is greater than name:
Return RetrieveRec(node.left, name)
Return RetrieveRec(node.right, name)

Code:

City.java

```java
package TableItem;
//common for all implementations
/**
Name: Jeel Tikiwala
Student ID: 239659420
Assignment 3
Any and all work in this file is my own.*/
public class City {
    String name;
    String country;
    int population;

    public City(String name, String country, int
population) {
        this.name = name;
        this.country = country;
        this.population = population;
    }

    // Getter methods
    public String getName() {
        return name;
    }

    public String getCountry() {
        return country;
    }

    public int getPopulation() {
        return population;
    }
```

```
    @Override
    public String toString() {
        return name + ", " + country + ", " +
population;
    }
}
```

CityTableArray.java

```
package TableItem;
/**
Name: Jeel Tikiwala
Student ID: 239659420
Assignment 3
Any and all work in this file is my own.*/
import java.util.Arrays;

public class CityTableArray {
    private City[] table;
    private int size;

    public CityTableArray() {
        table = new City[10]; // Initial capacity, if
we want we can increase as well
        size = 0;
    }

    public void insert(City city) {
        if (size == table.length) {
            table = Arrays.copyOf(table, size * 2);
        }
        table[size++] = city;
    }

    public void delete(String name) {
        for (int i = 0; i < size; i++) {
            if (table[i].getName().equals(name)) {
                table[i] = table[size - 1];
                table[size - 1] = null;
                size--;
                break;
            }
        }
```

```
    }

    public City retrieve(String name) {
        for (int i = 0; i < size; i++) {
            if (table[i].getName().equals(name)) {
                return table[i];
            }
        }
        return null;
    }

    // Utility method to print all cities
    public void printCities() {
        for (int i = 0; i < size; i++) {
            System.out.println(table[i]);
        }
    }
}
```

CityNode.java

```
package TableItem;
//Reference-Based Implementation (Linked List)
/**
Name: Jeel Tikiwala
Student ID: 239659420
Assignment 3
Any and all work in this file is my own.*/
public class CityNode {
    City city;
    CityNode next;

    public CityNode(City city) {
        this.city = city;
        this.next = null;
    }
}
```

CityTableLinkedList.java

```
package TableItem;
//Reference-Based Implementation (Linked List)
/**
Name: Jeel Tikiwala
Student ID: 239659420
```

```java
Assignment 3
Any and all work in this file is my own.*/
public class CityTableLinkedList {
    private CityNode head;

    public void insert(City city) {
        CityNode newNode = new CityNode(city);
        if (head == null) {
            head = newNode;
        } else {
            CityNode current = head;
            while (current.next != null) {
                current = current.next;
            }
            current.next = newNode;
        }
    }

    public void delete(String name) {
        if (head == null) return;
        if (head.city.getName().equals(name)) {
            head = head.next;
            return;
        }
        CityNode current = head;
        while (current.next != null) {
            if
(current.next.city.getName().equals(name)) {
                current.next = current.next.next;
                return;
            }
            current = current.next;
        }
    }

    public City retrieve(String name) {
        CityNode current = head;
        while (current != null) {
            if (current.city.getName().equals(name))
{
                return current.city;
            }
```

```
            current = current.next;
        }
        return null;
    }

    // Utility method to print all cities
    public void printCities() {
        CityNode current = head;
        while (current != null) {
            System.out.println(current.city);
            current = current.next;
        }
    }
}
```

CityNodeBST.java

```
package TableItem;
//Binary Search Tree-Based Implementation
/**
Name: Jeel Tikiwala
Student ID: 239659420
Assignment 3
Any and all work in this file is my own.*/
public class CityNodeBST {
    City city;
    CityNodeBST left, right;

    public CityNodeBST(City city) {
        this.city = city;
        this.left = this.right = null;
    }
}
```

CityTableBST.java

```
package TableItem;
//Binary Search Tree-Based Implementation
/**
Name: Jeel Tikiwala
Student ID: 239659420
Assignment 3
Any and all work in this file is my own.*/
public class CityTableBST {
    private CityNodeBST root;
```

```
    public CityTableBST() {
        this.root = null;
    }

    public void insert(City city) {
        root = insertRec(root, city);
    }

    private CityNodeBST insertRec(CityNodeBST root,
City city) {
        if (root == null) {
            root = new CityNodeBST(city);
            return root;
        }
        if
(city.getName().compareTo(root.city.getName()) < 0) {
            root.left = insertRec(root.left, city);
        } else if
(city.getName().compareTo(root.city.getName()) > 0) {
            root.right = insertRec(root.right, city);
        }
        return root;
    }

    public void delete(String name) {
        root = deleteRec(root, name);
    }

    private CityNodeBST deleteRec(CityNodeBST root,
String name) {
        if (root == null) return root;
        if (name.compareTo(root.city.getName()) < 0)
            root.left = deleteRec(root.left, name);
        else if (name.compareTo(root.city.getName())
> 0)
            root.right = deleteRec(root.right, name);
        else {
            if (root.left == null)
                return root.right;
            else if (root.right == null)
                return root.left;
```

```
            root.city = minValue(root.right);
            root.right = deleteRec(root.right,
root.city.getName());
        }
        return root;
    }

    private City minValue(CityNodeBST root) {
        City minv = root.city;
        while (root.left != null) {
            minv = root.left.city;
            root = root.left;
        }
        return minv;
    }

    public City retrieve(String name) {
        CityNodeBST res = retrieveRec(root, name);
        return res != null ? res.city : null;
    }

    private CityNodeBST retrieveRec(CityNodeBST root,
String name) {
        if (root == null ||
root.city.getName().equals(name))
            return root;
        if (root.city.getName().compareTo(name) > 0)
            return retrieveRec(root.left, name);
        return retrieveRec(root.right, name);
    }

    // Utility method to print the BST in-order
(sorted)
    public void printInOrder() {
        printInOrderRec(root);
    }

    private void printInOrderRec(CityNodeBST node) {
        if (node != null) {
            printInOrderRec(node.left);
            System.out.println(node.city);
            printInOrderRec(node.right);
```

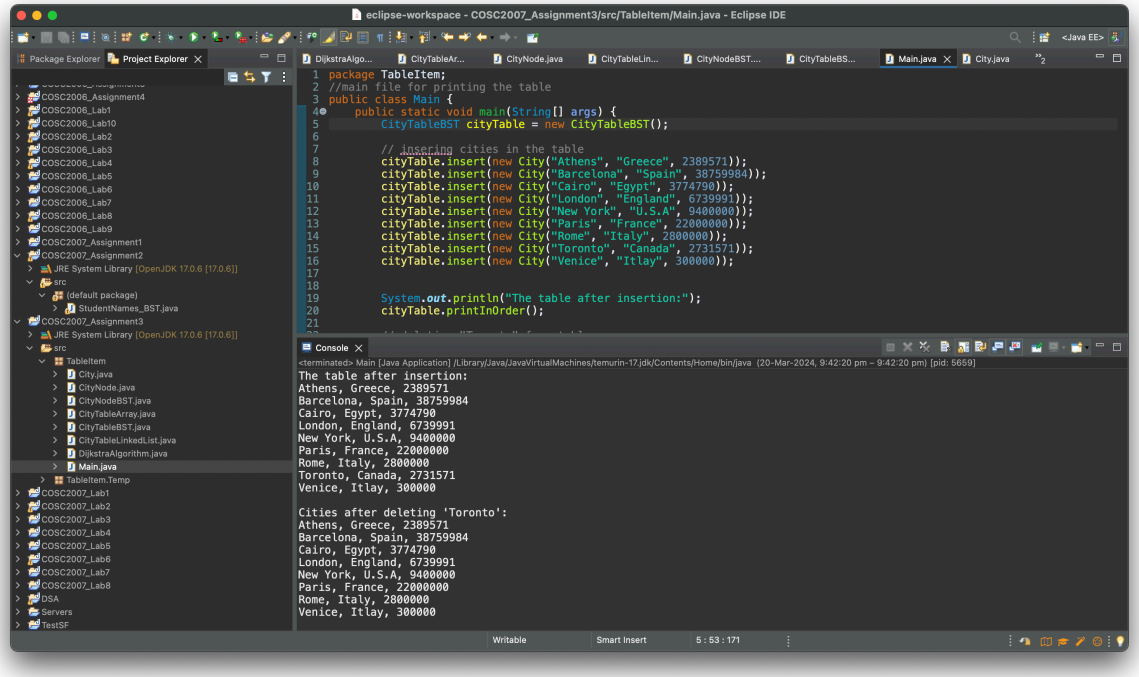
```
        }
    }
}
```

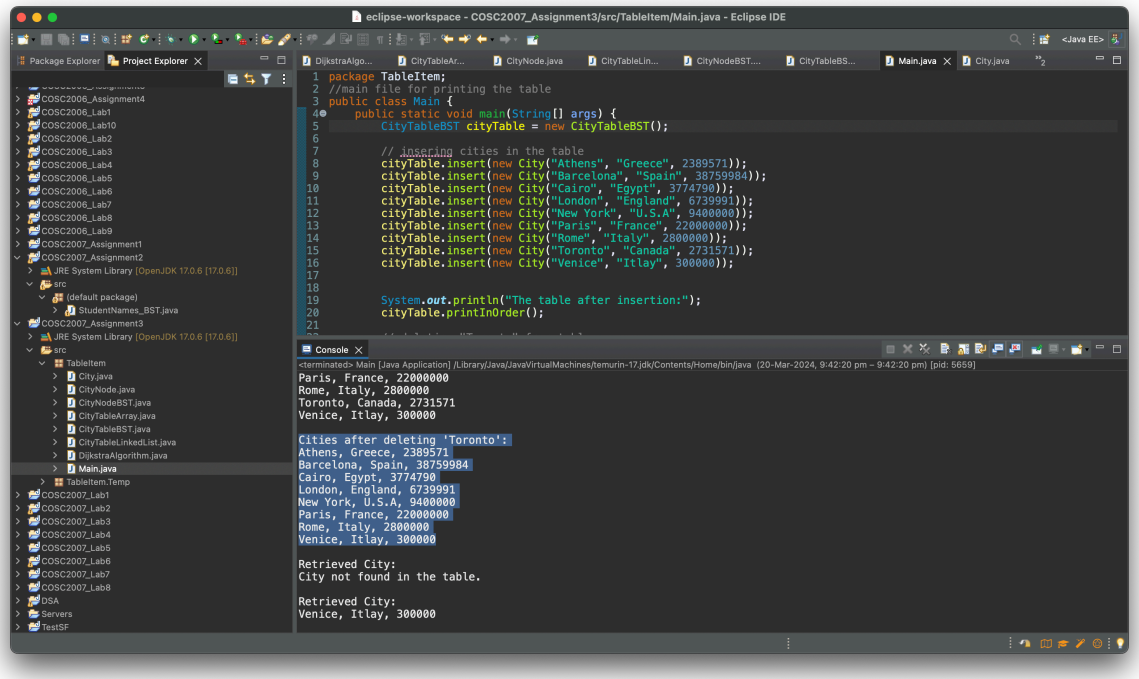
Main.java

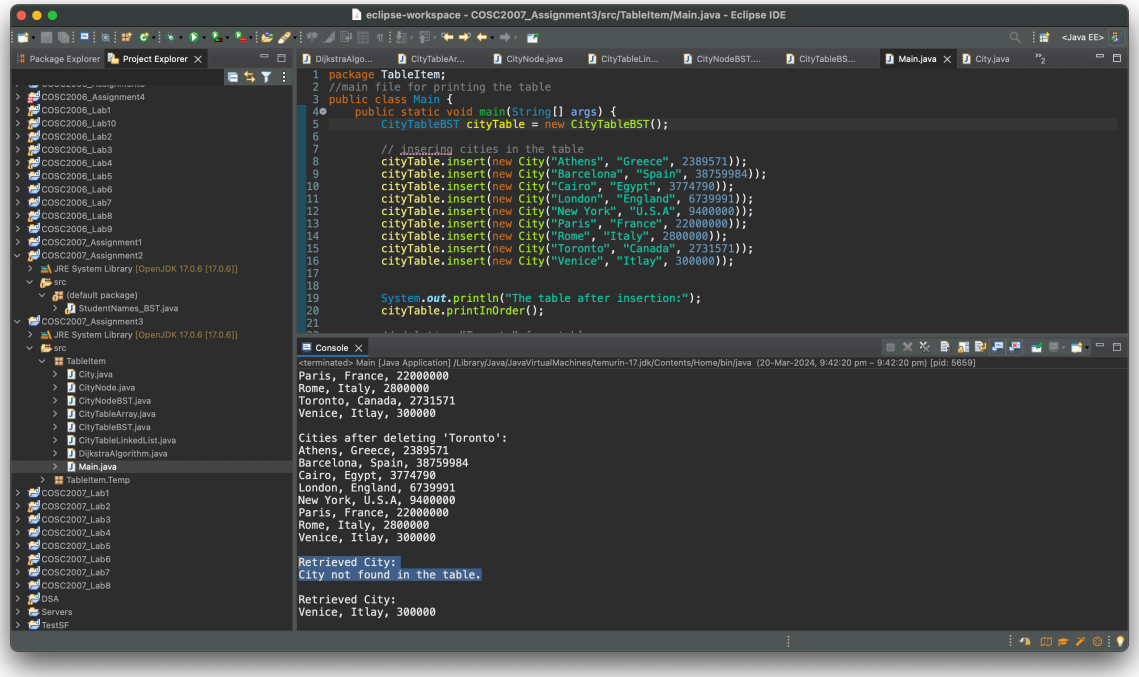
```
package TableItem;
//main file for printing the table
public class Main {
    public static void main(String[] args) {
        CityTableBST cityTable = new CityTableBST();

        // insering cities in the table
        cityTable.insert(new City("Athens", "Greece",
2389571));
        cityTable.insert(new City("Barcelona",
"Spain", 38759984));
        cityTable.insert(new City("Cairo", "Egypt",
3774790));
        cityTable.insert(new City("London",
"England", 6739991));
        cityTable.insert(new City("New York",
"U.S.A", 9400000));
        cityTable.insert(new City("Paris", "France",
22000000));
        cityTable.insert(new City("Rome", "Italy",
2800000));
        cityTable.insert(new City("Toronto",
"Canada", 2731571));
        cityTable.insert(new City("Venice", "Itlay",
300000));


        System.out.println("The table after
insertion:");
        cityTable.printInOrder();

        // deleting "Toronto" from table
        cityTable.delete("Toronto");
        System.out.println("\nCities after deleting
'Toronto':");
        cityTable.printInOrder();
```

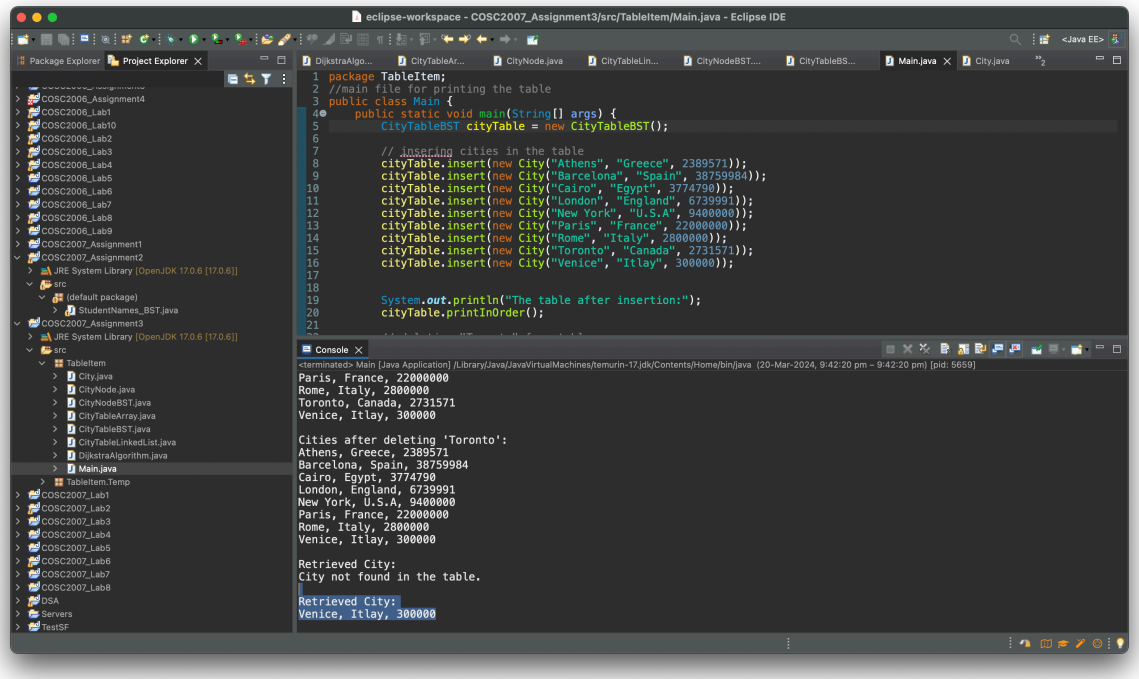
```
        // retriving "toronto" from the table
        City retrievedCity =
cityTable.retrieve("Toronto");
        System.out.println("\nRetrieved City:");
        System.out.println(retrievedCity != null ?
retrievedCity : "City not found in the table.");

        City retrievedCity2 =
cityTable.retrieve("Venice");
        System.out.println("\nRetrieved City:");
        System.out.println(retrievedCity2 != null ?
retrievedCity2 : "City not found in the table.");
    }
}
```

Output:




eclipse-workspace - COSC2007_Assignment3/src/TableItem/Main.java - Eclipse IDE
Package Explorer
Project Explorer
DijkstraAlgo...
CityTableAr...
CityNode.java
CityTableLin...
CityNodeBST....
CityTableBS...
Main.java
City.java
<Java EE>
COSC2006_Assignment4
COSC2006_Lab1
COSC2006_Lab10
COSC2006_Lab2
COSC2006_Lab3
COSC2006_Lab4
COSC2006_Lab5
COSC2006_Lab6
COSC2006_Lab7
COSC2006_Lab8
COSC2006_Lab9
COSC2007_Assignment1
COSC2007_Assignment2
JRE System Library [OpenJDK 17.0.6 [17.0.6]]
src
(default package)
StudentNames_BST.java
COSC2007_Assignment3
JRE System Library [OpenJDK 17.0.6 [17.0.6]]
src
TableItem
City.java
CityNode.java
CityNodeBST.java
CityTableArray.java
CityTableBST.java
CityTableLinkedList.java
DijkstraAlgorithm.java
Main.java
TableItem.Temp
COSC2007_Lab1
COSC2007_Lab2
COSC2007_Lab3
COSC2007_Lab4
COSC2007_Lab5
COSC2007_Lab6
COSC2007_Lab7
COSC2007_Lab8
DSA
Servers
TestSF
1 package TableItem;
2 //main file for printing the table
3 public class Main {
4 public static void main(String[] args) {
5 CityTableBST cityTable = new CityTableBST();
6
7 // inserting cities in the table
8 cityTable.insert(new City("Athens", "Greece", 2389571));
9 cityTable.insert(new City("Barcelona", "Spain", 38759984));
10 cityTable.insert(new City("Cairo", "Egypt", 3774790));
11 cityTable.insert(new City("London", "England", 6739991));
12 cityTable.insert(new City("New York", "U.S.A", 9400000));
13 cityTable.insert(new City("Paris", "France", 22000000));
14 cityTable.insert(new City("Rome", "Italy", 2800000));
15 cityTable.insert(new City("Toronto", "Canada", 2731571));
16 cityTable.insert(new City("Venice", "Itlay", 300000));
17
18
19 System.out.println("The table after insertion:");
20 cityTable.printInOrder();
21
Console
<terminated> Main [Java Application] /Library/Java/JavaVirtualMachines/temurin-17.jdk/Contents/Home/bin/java (20-Mar-2024, 9:42:20 pm – 9:42:20 pm) [pid: 5659]
Paris, France, 22000000
Rome, Italy, 2800000
Toronto, Canada, 2731571
Venice, Itlay, 300000
Cities after deleting 'Toronto':
Athens, Greece, 2389571
Barcelona, Spain, 38759984
Cairo, Egypt, 3774790
London, England, 6739991
New York, U.S.A, 9400000
Paris, France, 22000000
Rome, Italy, 2800000
Venice, Itlay, 300000
Retrieved City:
City not found in the table.
Retrieved City:
Venice, Itlay, 300000


eclipse-workspace - COSC2007_Assignment3/src/TableItem/Main.java - Eclipse IDE
Package Explorer
Project Explorer
DijkstraAlgo...
CityTableAr...
CityNode.java
CityTableLin...
CityNodeBST....
CityTableBS...
Main.java
City.java
<Java EE>
COSC2006_Assignment4
COSC2006_Lab1
COSC2006_Lab10
COSC2006_Lab2
COSC2006_Lab3
COSC2006_Lab4
COSC2006_Lab5
COSC2006_Lab6
COSC2006_Lab7
COSC2006_Lab8
COSC2006_Lab9
COSC2007_Assignment1
COSC2007_Assignment2
JRE System Library [OpenJDK 17.0.6 [17.0.6]]
src
(default package)
StudentNames_BST.java
COSC2007_Assignment3
JRE System Library [OpenJDK 17.0.6 [17.0.6]]
src
TableItem
City.java
CityNode.java
CityNodeBST.java
CityTableArray.java
CityTableBST.java
CityTableLinkedList.java
DijkstraAlgorithm.java
Main.java
TableItem.Temp
COSC2007_Lab1
COSC2007_Lab2
COSC2007_Lab3
COSC2007_Lab4
COSC2007_Lab5
COSC2007_Lab6
COSC2007_Lab7
COSC2007_Lab8
DSA
Servers
TestSF
1 package TableItem;
2 //main file for printing the table
3 public class Main {
4 public static void main(String[] args) {
5 CityTableBST cityTable = new CityTableBST();
6
7 // inserting cities in the table
8 cityTable.insert(new City("Athens", "Greece", 2389571));
9 cityTable.insert(new City("Barcelona", "Spain", 38759984));
10 cityTable.insert(new City("Cairo", "Egypt", 3774790));
11 cityTable.insert(new City("London", "England", 6739991));
12 cityTable.insert(new City("New York", "U.S.A", 9400000));
13 cityTable.insert(new City("Paris", "France", 22000000));
14 cityTable.insert(new City("Rome", "Italy", 2800000));
15 cityTable.insert(new City("Toronto", "Canada", 2731571));
16 cityTable.insert(new City("Venice", "Itlay", 300000));
17
18
19 System.out.println("The table after insertion:");
20 cityTable.printInOrder();
21
Console
<terminated> Main [Java Application] /Library/Java/JavaVirtualMachines/temurin-17.jdk/Contents/Home/bin/java (20-Mar-2024, 9:42:20 pm – 9:42:20 pm) [pid: 5659]
Paris, France, 22000000
Rome, Italy, 2800000
Toronto, Canada, 2731571
Venice, Itlay, 300000
Cities after deleting 'Toronto':
Athens, Greece, 2389571
Barcelona, Spain, 38759984
Cairo, Egypt, 3774790
London, England, 6739991
New York, U.S.A, 9400000
Paris, France, 22000000
Rome, Italy, 2800000
Venice, Itlay, 300000
Retrieved City:
City not found in the table.
Retrieved City:
Venice, Itlay, 300000



Conclusion:
For the Table ADT, the investigation of Array-based, Reference-based (Linked List), and Binary Search Tree (BST) implementations offered insightful information about choosing data structures according to application requirements. The array-based method has scalability issues but is fast and simple to use. While the BST combines effective search with balanced performance for dynamic operations, making it a flexible option for a variety of use cases, the Linked List performs better in dynamic operations but has slower searches. The significance of selecting the appropriate data structure based on particular requirements, such as operation frequency, data size, and performance criteria, was highlighted by this exercise.

Exercise No: 2
Question: Write the detailed algorithm and convert into Java code for the solution of Dijkstra's Algorithm. Use the adjacency matrix concept for the graph representation.

Algorithm/Pseudocode:

```
Algorithm Dijkstra'sAlgorithm(graph, startVertex):
   nVertices = number of vertices in graph
   Initialize shortestDistances[nVertices]
   Initialize added[nVertices]

   For each vertex v in graph:
      shortestDistances[v] = INFINITY
      added[v] = false

   shortestDistances[startVertex] = 0
   parents[startVertex] = NO_PARENT

   For each vertex v in graph except startVertex:
      nearestVertex = Find unvisited vertex with smallest known distance from startVertex
      added[nearestVertex] = true

      For each unvisited neighbor u of nearestVertex:
         edgeDistance = graph[nearestVertex][u]
         newDistance = shortestDistances[nearestVertex] + edgeDistance

         If newDistance < shortestDistances[u]:
            shortestDistances[u] = newDistance
            parents[u] = nearestVertex

   printSolution(startVertex, shortestDistances, parents)

Function printSolution(startVertex, distances, parents):
   For each vertex v, starting from startVertex:
      Print "startVertex -> v: distance, path"

Function printPath(currentVertex, parents):
   If currentVertex has NO_PARENT:
      return
   Else:
      Recursively print path from start vertex to parent of currentVertex
      Print currentVertex
```

Code:

```java
package TableItem;

/**
 * Name: Jeel Tikiwala
 * Student ID: 239659420
```

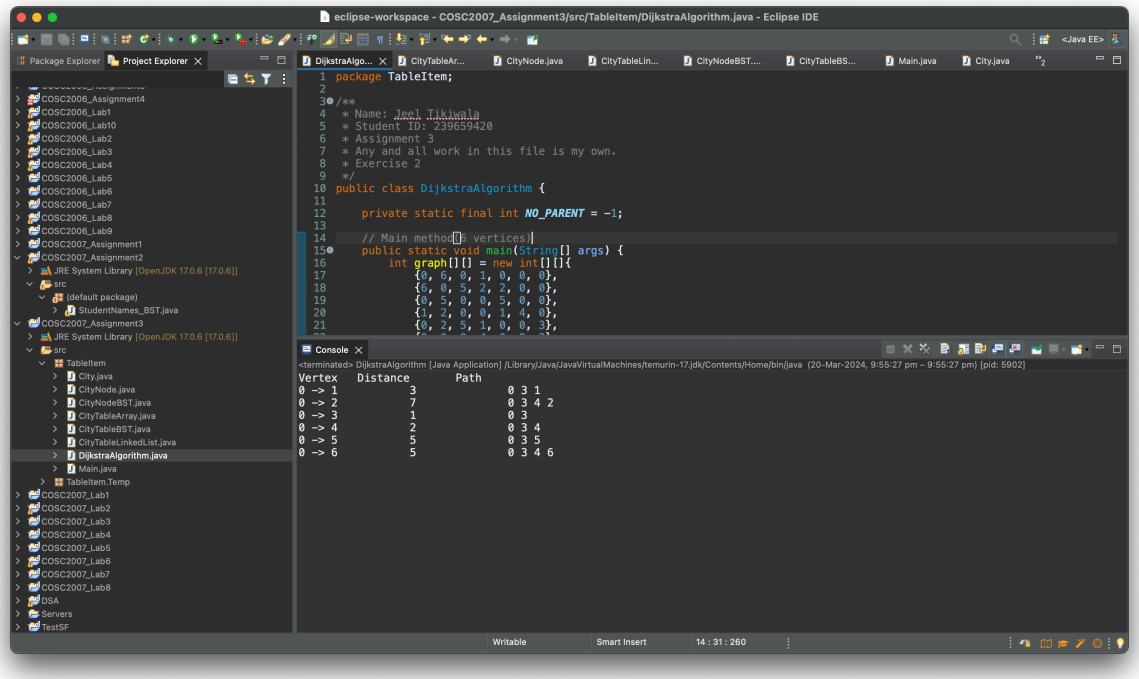
```java
 * Assignment 3
 * Any and all work in this file is my own.
 * Exercise 2
 */
public class DijkstraAlgorithm {

    private static final int NO_PARENT = -1;

    // Main method(6 vertices)
    public static void main(String[] args) {
        int graph[][] = new int[][]{
                {0, 6, 0, 1, 0, 0, 0},
                {6, 0, 5, 2, 2, 0, 0},
                {0, 5, 0, 0, 5, 0, 0},
                {1, 2, 0, 0, 1, 4, 0},
                {0, 2, 5, 1, 0, 0, 3},
                {0, 0, 0, 4, 0, 0, 2},
                {0, 0, 0, 0, 3, 2, 0}
        };

        dijkstraAlgorithm(graph, 0); // 0 is the
source vertex
    }

    // Function that implements Dijkstra's algorithm
    // graph[][]: adjacency matrix representation of
the graph
    // startVertex: source vertex to find shortest
path from
    public static void dijkstraAlgorithm(int[][]
graph, int startVertex) {
        int nVertices = graph[0].length;

        // Shortest distances from startVertex to i
        int[] shortestDistances = new int[nVertices];

        // Added[i] will be true if vertex i is
included in shortest path
        boolean[] added = new boolean[nVertices];

        // Initialize all distances as INFINITE and
added[] as false
```

```
        for (int vertexIndex = 0; vertexIndex <
nVertices; vertexIndex++) {
            shortestDistances[vertexIndex] =
Integer.MAX_VALUE;
            added[vertexIndex] = false;
        }

        // Distance of source vertex from itself is
always 0
        shortestDistances[startVertex] = 0;

        // store shortest path tree
        int[] parents = new int[nVertices];

        // The starting vertex does not have a parent
        parents[startVertex] = NO_PARENT;

        // to get shortest path for all vertices
        for (int i = 1; i < nVertices; i++) {

            // min distance vertex from the set of
vertices, not yet processed
            int nearestVertex = -1;
            int shortestDistance = Integer.MAX_VALUE;
            for (int vertexIndex = 0; vertexIndex <
nVertices; vertexIndex++) {
                if (!added[vertexIndex] &&
shortestDistances[vertexIndex] < shortestDistance) {
                    nearestVertex = vertexIndex;
                    shortestDistance =
shortestDistances[vertexIndex];
                }
            }


            added[nearestVertex] = true;

            // Update dist value of the adjacent
vertices of the picked vertex.
            for (int vertexIndex = 0; vertexIndex <
nVertices; vertexIndex++) {
```

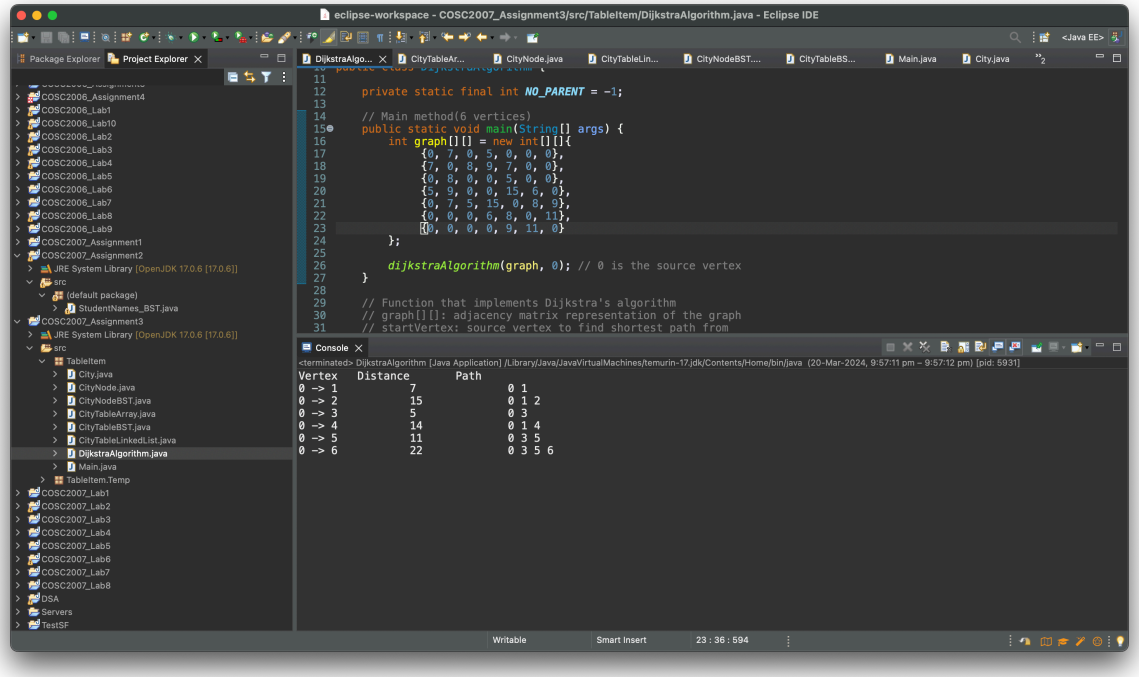
```
                int edgeDistance =
graph[nearestVertex][vertexIndex];

                if (edgeDistance > 0 &&
((shortestDistance + edgeDistance) <
shortestDistances[vertexIndex])) {
                    parents[vertexIndex] =
nearestVertex;
                    shortestDistances[vertexIndex] =
shortestDistance + edgeDistance;
                }
            }
        }

        printSolution(startVertex, shortestDistances,
parents);
    }


    private static void printSolution(int
startVertex, int[] distances, int[] parents) {
        int nVertices = distances.length;
        System.out.print("Vertex\t Distance\tPath");

        for (int vertexIndex = 0; vertexIndex <
nVertices; vertexIndex++) {
            if (vertexIndex != startVertex) {
                System.out.print("\n" + startVertex +
" -> ");
                System.out.print(vertexIndex + " \t\t
");

System.out.print(distances[vertexIndex] + "\t\t");
                printPath(vertexIndex, parents);
            }
        }
    }

    // fn to print shortest path from source to
currentVertex using parents array
    private static void printPath(int currentVertex,
int[] parents) {
```

```
            if (currentVertex == NO_PARENT) {
                return;
            }
            printPath(parents[currentVertex], parents);
            System.out.print(currentVertex + " ");
        }
}
```

Output:
Output -1



Output - 2



CONCLUSION:

Using an adjacency matrix to represent the graph, the Java implementation of Dijkstra's algorithm successfully illustrates how to find the shortest path from a single source vertex to every other vertex in a weighted graph. This method effectively finds the shortest paths in dense graphs where edge weights are non-negative, demonstrating the usefulness and applicability of Dijkstra's algorithm in a variety of optimisation problems, including network routing and geographic mapping. This algorithm highlights the significance of algorithmic strategy in computational problem-solving by managing dynamic graph structures and guaranteeing optimal pathfinding.