Name : Tejaswini Anil kamble

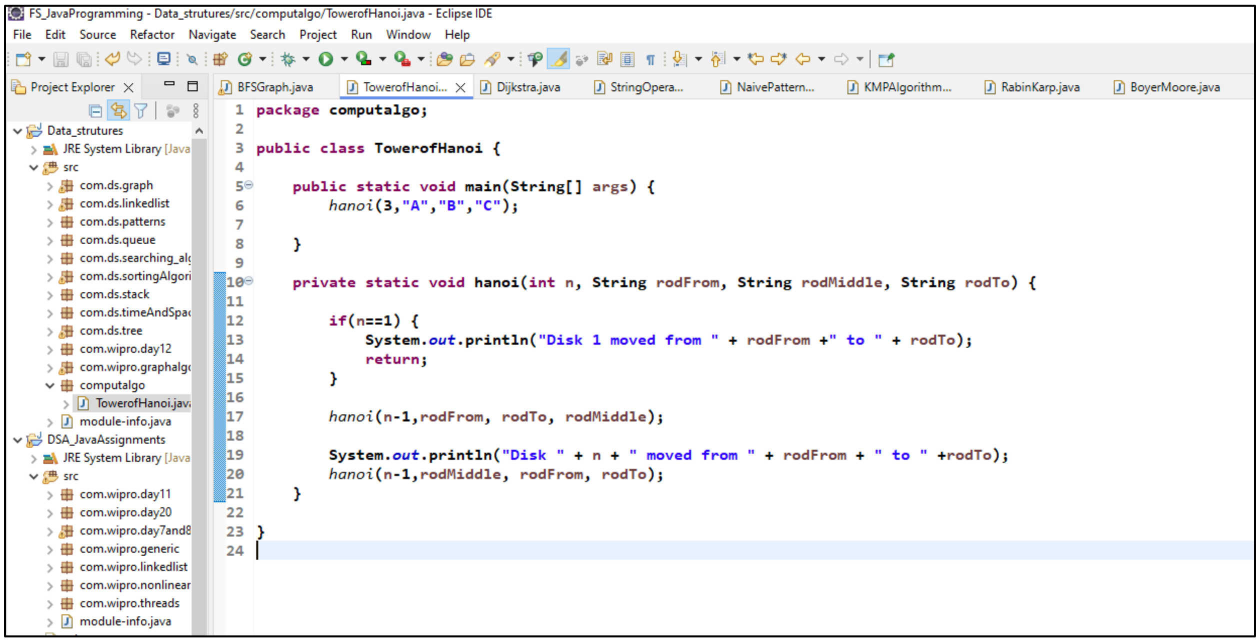
Email : teju000kamble@gmail.com

Assignments : Day 13 and 14

Task 1: Tower of Hanoi Solver

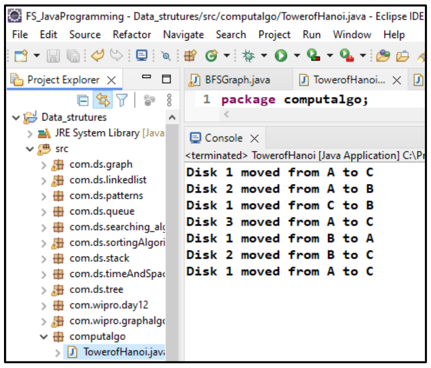
Create a program that solves the Tower of Hanoi puzzle for n disks. The solution should use recursion to move disks between three pegs (source, auxiliary, and destination) according to the game's rules. The program should print out each move required to solve the puzzle.

Ans : Source Code



```java
package computalgo;

public class TowerofHanoi {

    public static void main(String[] args) {
        hanoi(3,"A","B","C");

    }

    private static void hanoi(int n, String rodFrom, String rodMiddle, String rodTo) {

        if(n==1) {
            System.out.println("Disk 1 moved from " + rodFrom +" to " + rodTo);
            return;
        }

        hanoi(n-1,rodFrom, rodTo, rodMiddle);

        System.out.println("Disk " + n + " moved from " + rodFrom + " to " +rodTo);
        hanoi(n-1,rodMiddle, rodFrom, rodTo);
    }

}
```

Output:

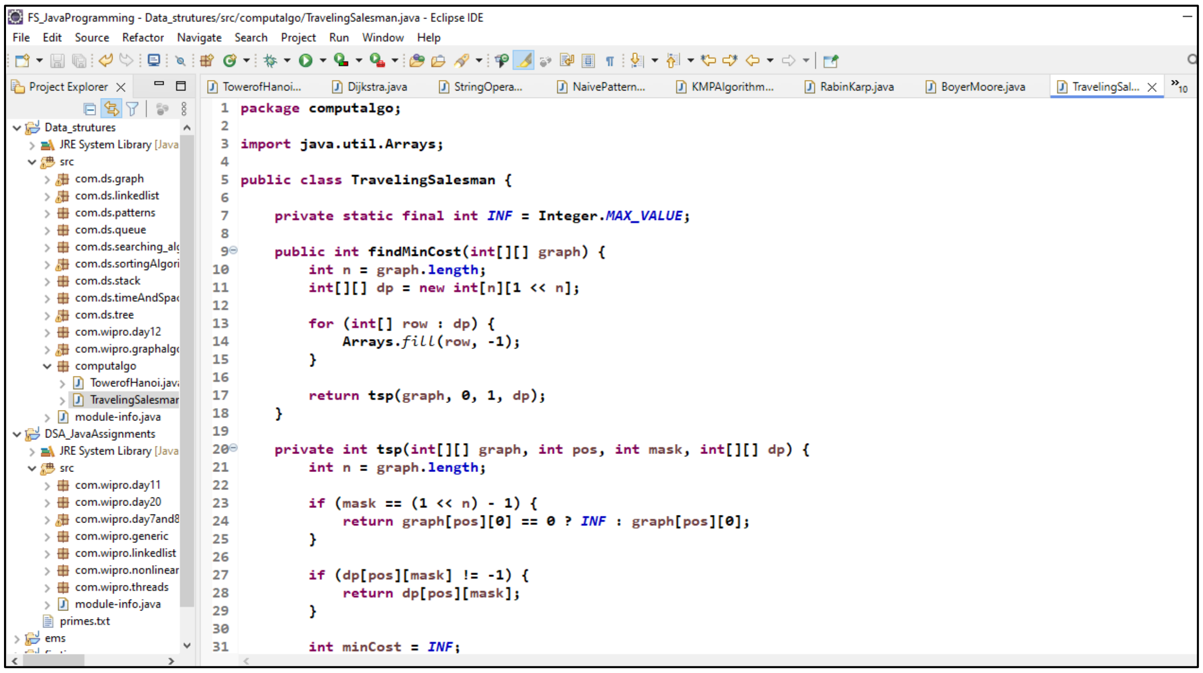


```
Disk 1 moved from A to C
Disk 2 moved from A to B
Disk 1 moved from C to B
Disk 3 moved from A to C
Disk 1 moved from B to A
Disk 2 moved from B to C
Disk 1 moved from A to C
```

Task 2: Traveling Salesman Problem

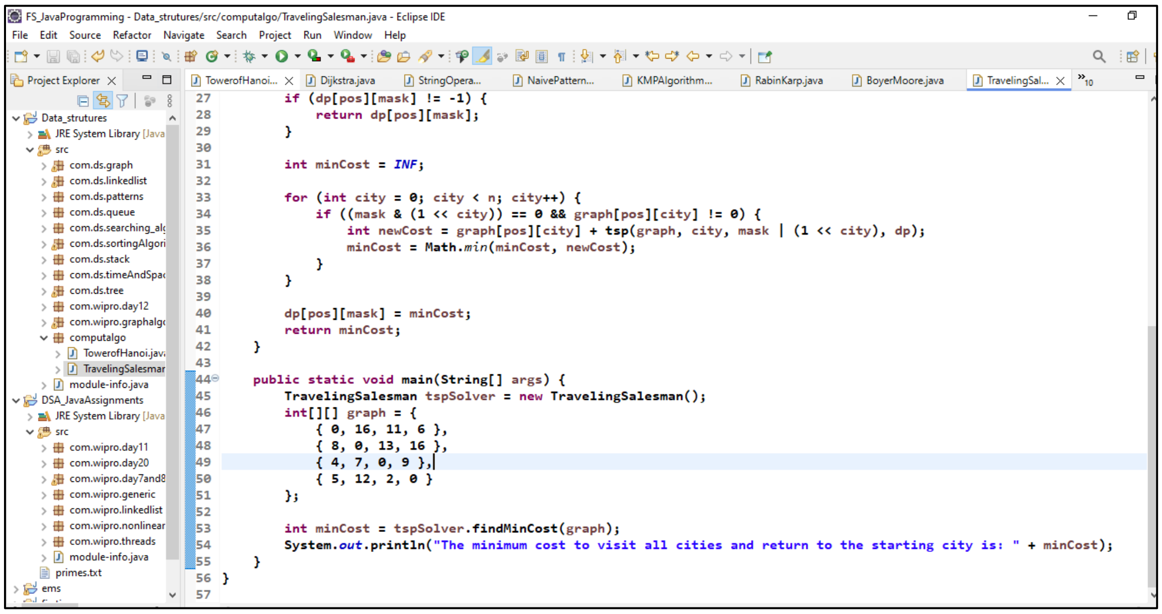
Create a function int FindMinCost(int[,] graph) that takes a 2D array representing the graph where graph[i][j] is the cost to travel from city i to city j. The function should return the minimum cost to visit all cities and return to the starting city. Use dynamic programming for this solution.

Ans: Source Code



FS_JavaProgramming - Data_strutures/src/computalgo/TravelingSalesman.java - Eclipse IDE

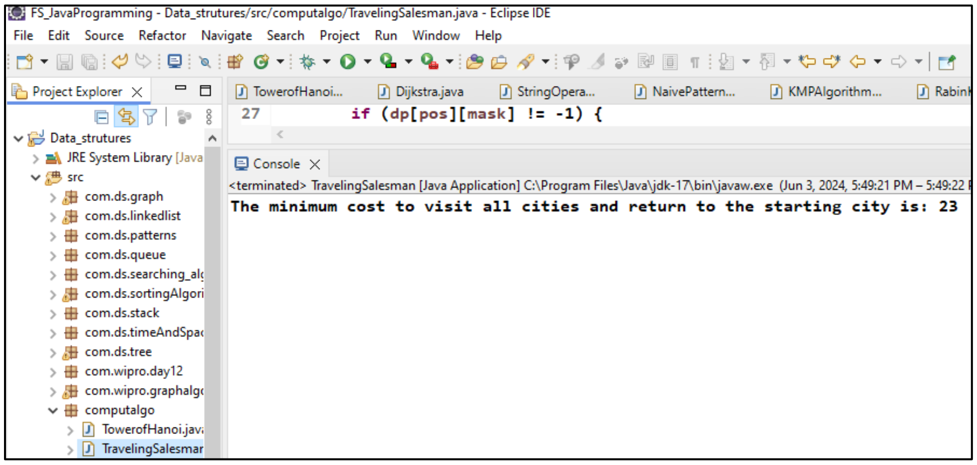
```java
package computalgo;

import java.util.Arrays;

public class TravelingSalesman {

    private static final int INF = Integer.MAX_VALUE;

    public int findMinCost(int[][] graph) {
        int n = graph.length;
        int[][] dp = new int[n][1 << n];

        for (int[] row : dp) {
            Arrays.fill(row, -1);
        }

        return tsp(graph, 0, 1, dp);
    }

    private int tsp(int[][] graph, int pos, int mask, int[][] dp) {
        int n = graph.length;

        if (mask == (1 << n) - 1) {
            return graph[pos][0] == 0 ? INF : graph[pos][0];
        }

        if (dp[pos][mask] != -1) {
            return dp[pos][mask];
        }

        int minCost = INF;
```



FS_JavaProgramming - Data_strutures/src/computalgo/TravelingSalesman.java - Eclipse IDE

```java
        if (dp[pos][mask] != -1) {
            return dp[pos][mask];
        }

        int minCost = INF;

        for (int city = 0; city < n; city++) {
            if ((mask & (1 << city)) == 0 && graph[pos][city] != 0) {
                int newCost = graph[pos][city] + tsp(graph, city, mask | (1 << city), dp);
                minCost = Math.min(minCost, newCost);
            }
        }

        dp[pos][mask] = minCost;
        return minCost;
    }

    public static void main(String[] args) {
        TravelingSalesman tspSolver = new TravelingSalesman();
        int[][] graph = {
            { 0, 16, 11, 6 },
            { 8, 0, 13, 16 },
            { 4, 7, 0, 9 },
            { 5, 12, 2, 0 }
        };

        int minCost = tspSolver.findMinCost(graph);
        System.out.println("The minimum cost to visit all cities and return to the starting city is: " + minCost);
    }
}
```

Output:



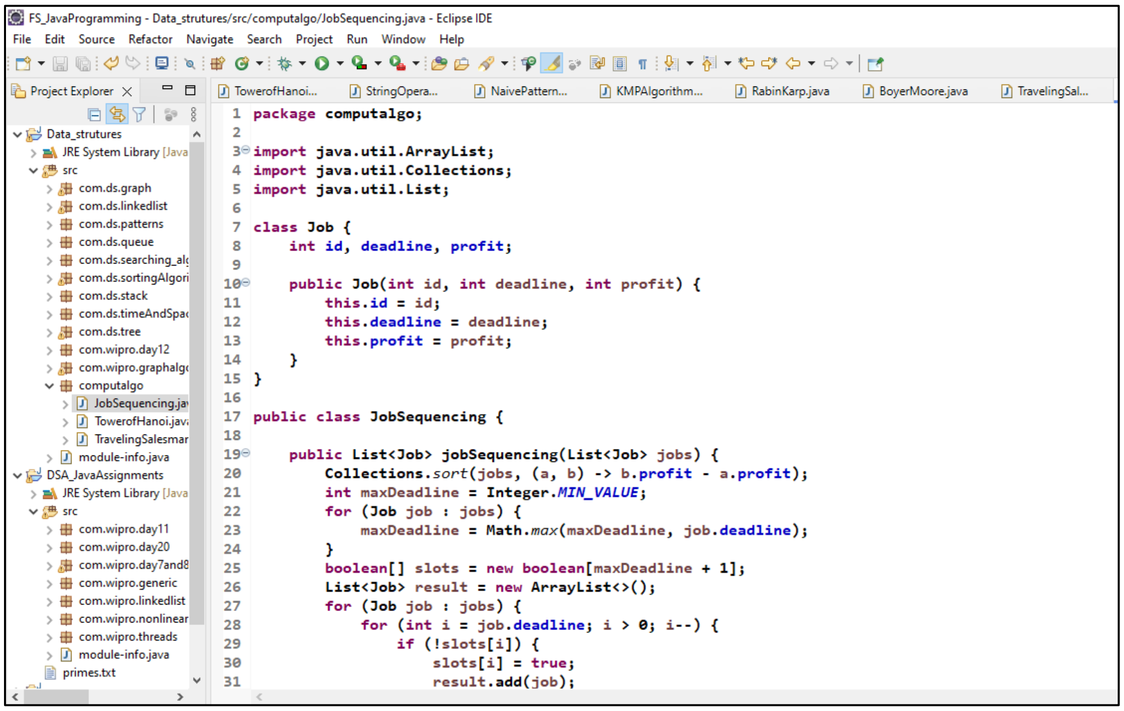
```
27          if (dp[pos][mask] != -1) {
```

```
<terminated> TravelingSalesman [Java Application] C:\Program Files\Java\jdk-17\bin\javaw.exe (Jun 3, 2024, 5:49:21 PM – 5:49:22
The minimum cost to visit all cities and return to the starting city is: 23
```

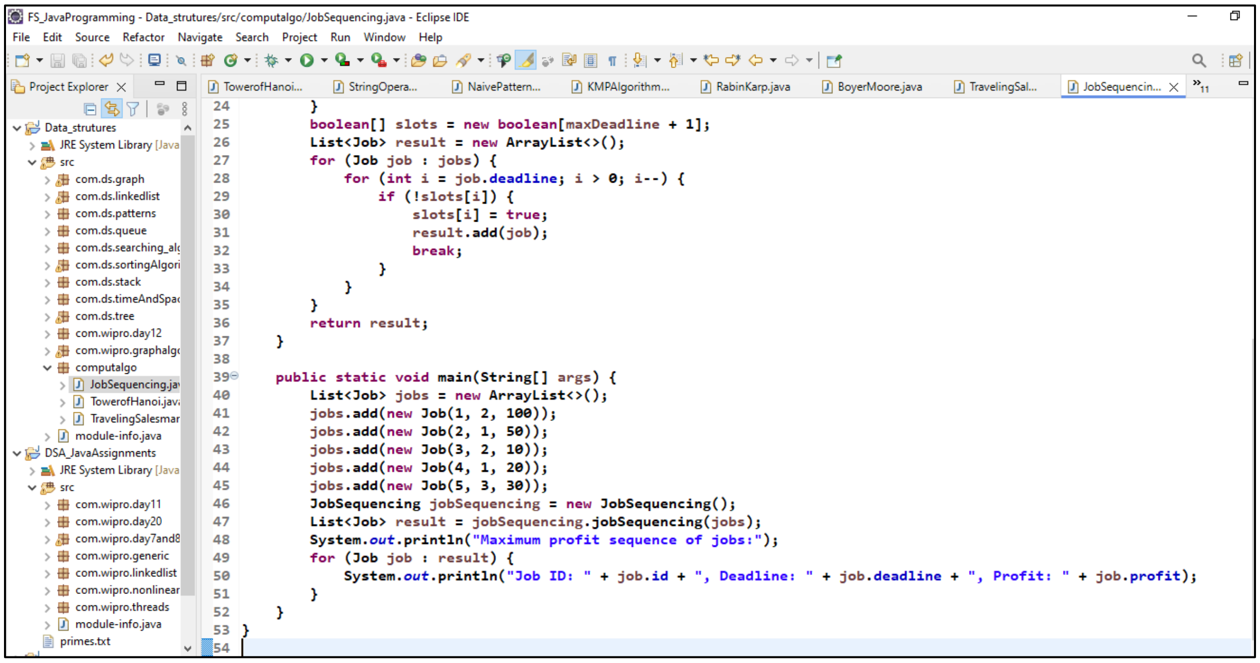
Task 3: Job Sequencing Problem

Define a class Job with properties int Id, int Deadline, and int Profit. Then implement a function List<Job> JobSequencing(List<Job> jobs) that takes a list of jobs and returns the maximum profit sequence of jobs that can be done before the deadlines. Use the greedy method to solve this problem.

Ans : Source Code

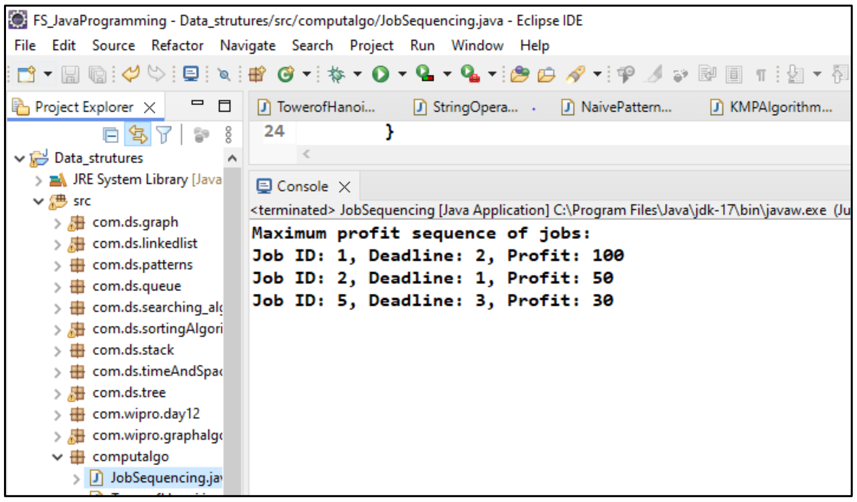


```java
package computalgo;

import java.util.ArrayList;
import java.util.Collections;
import java.util.List;

class Job {
    int id, deadline, profit;

    public Job(int id, int deadline, int profit) {
        this.id = id;
        this.deadline = deadline;
        this.profit = profit;
    }
}

public class JobSequencing {

    public List<Job> jobSequencing(List<Job> jobs) {
        Collections.sort(jobs, (a, b) -> b.profit - a.profit);
        int maxDeadline = Integer.MIN_VALUE;
        for (Job job : jobs) {
            maxDeadline = Math.max(maxDeadline, job.deadline);
        }
        boolean[] slots = new boolean[maxDeadline + 1];
        List<Job> result = new ArrayList<>();
        for (Job job : jobs) {
            for (int i = job.deadline; i > 0; i--) {
                if (!slots[i]) {
                    slots[i] = true;
                    result.add(job);
```



```
        }
        boolean[] slots = new boolean[maxDeadline + 1];
        List<Job> result = new ArrayList<>();
        for (Job job : jobs) {
            for (int i = job.deadline; i > 0; i--) {
                if (!slots[i]) {
                    slots[i] = true;
                    result.add(job);
                    break;
                }
            }
        }
        return result;
    }

    public static void main(String[] args) {
        List<Job> jobs = new ArrayList<>();
        jobs.add(new Job(1, 2, 100));
        jobs.add(new Job(2, 1, 50));
        jobs.add(new Job(3, 2, 10));
        jobs.add(new Job(4, 1, 20));
        jobs.add(new Job(5, 3, 30));
        JobSequencing jobSequencing = new JobSequencing();
        List<Job> result = jobSequencing.jobSequencing(jobs);
        System.out.println("Maximum profit sequence of jobs:");
        for (Job job : result) {
            System.out.println("Job ID: " + job.id + ", Deadline: " + job.deadline + ", Profit: " + job.profit);
        }
    }
}
```

Output:



```
<terminated> JobSequencing [Java Application] C:\Program Files\Java\jdk-17\bin\javaw.exe
Maximum profit sequence of jobs:
Job ID: 1, Deadline: 2, Profit: 100
Job ID: 2, Deadline: 1, Profit: 50
Job ID: 5, Deadline: 3, Profit: 30
```