

Name : Tejaswini Anil kamble

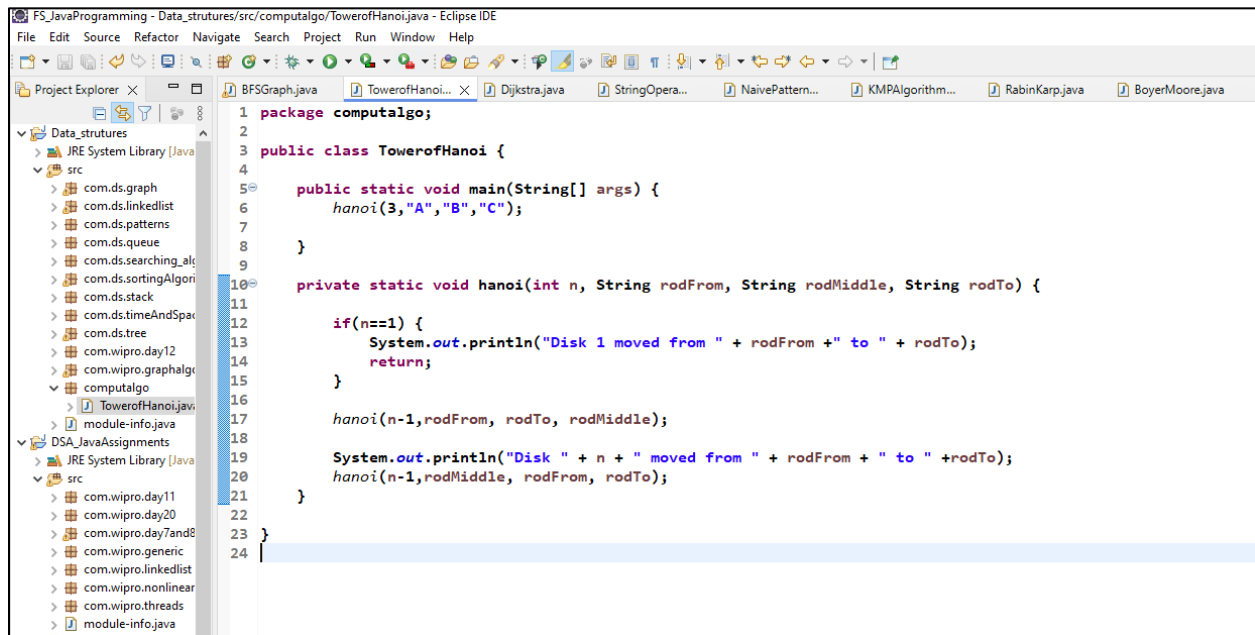
Email : teju000kamble@gmail.com

Assignments : Day 13 and 14

Task 1: Tower of Hanoi Solver

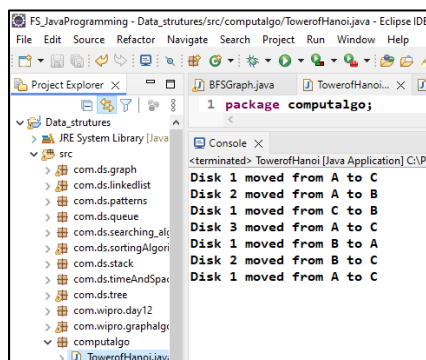
Create a program that solves the Tower of Hanoi puzzle for n disks. The solution should use recursion to move disks between three pegs (source, auxiliary, and destination) according to the game's rules. The program should print out each move required to solve the puzzle.

Ans : Source Code



```
1 package computalgo;
2
3 public class TowerofHanoi {
4
5     public static void main(String[] args) {
6         hanoi(3,"A","B","C");
7     }
8
9     private static void hanoi(int n, String rodFrom, String rodMiddle, String rodTo) {
10
11         if(n==1) {
12             System.out.println("Disk 1 moved from " + rodFrom + " to " + rodTo);
13             return;
14         }
15
16         hanoi(n-1,rodFrom, rodTo, rodMiddle);
17
18         System.out.println("Disk " + n + " moved from " + rodFrom + " to " +rodTo);
19         hanoi(n-1,rodMiddle, rodFrom, rodTo);
20     }
21 }
22
23
24
```

Output:

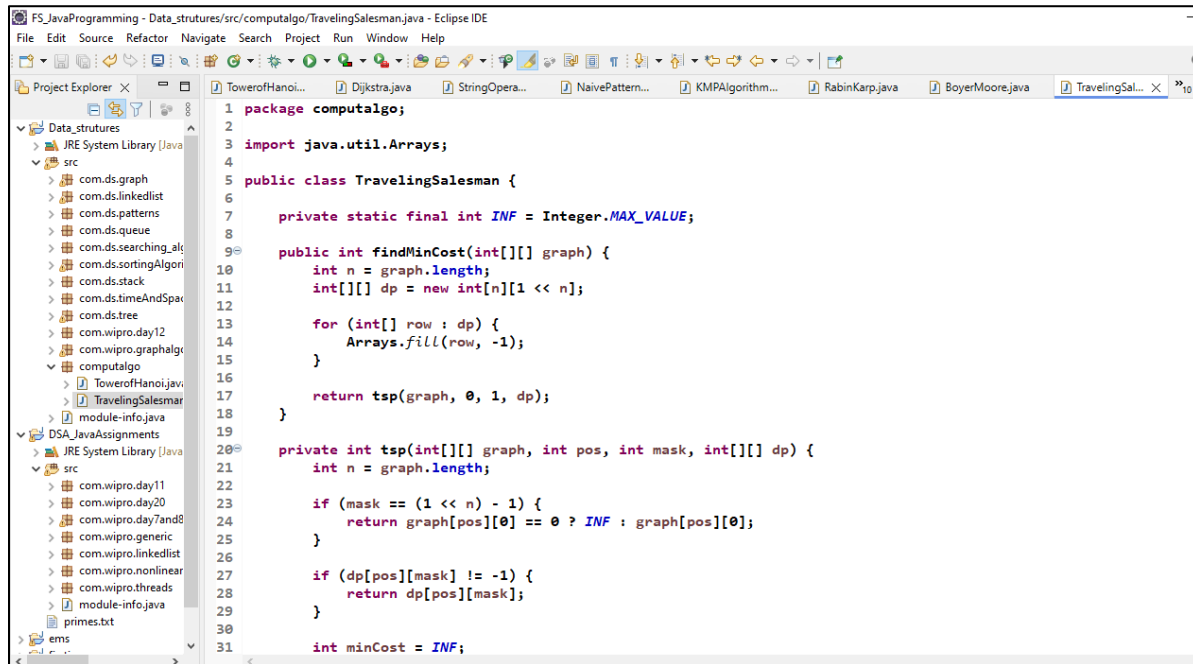


```
<terminated> TowerofHanoi [Java Application] C:\P...
Disk 1 moved from A to C
Disk 2 moved from A to B
Disk 3 moved from A to C
Disk 1 moved from B to A
Disk 2 moved from B to C
Disk 1 moved from A to C
```

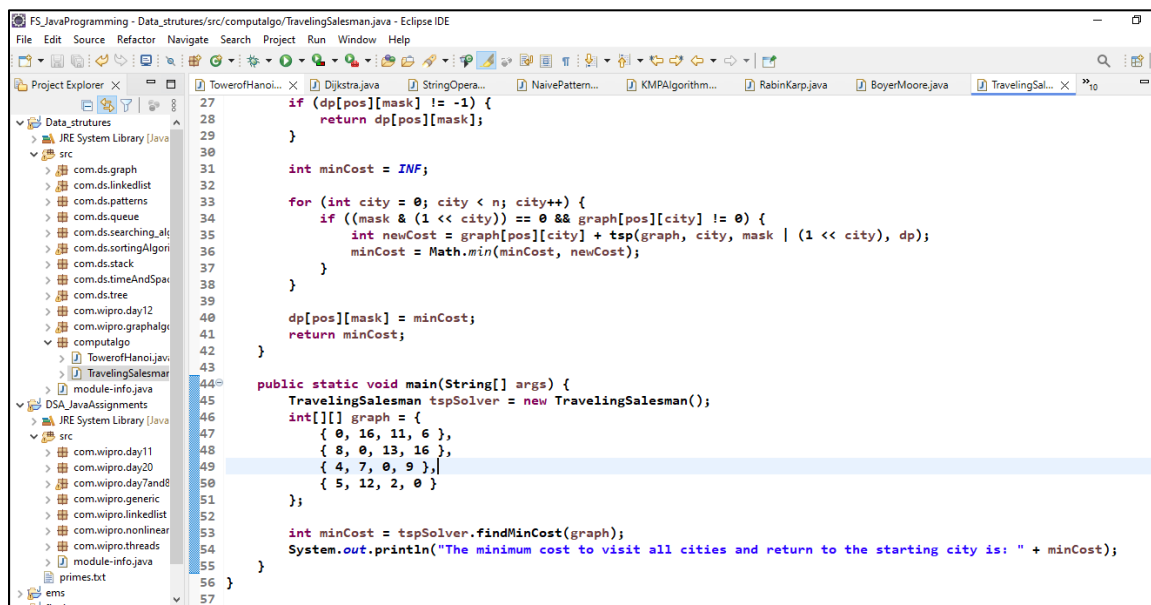
Task 2: Traveling Salesman Problem

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

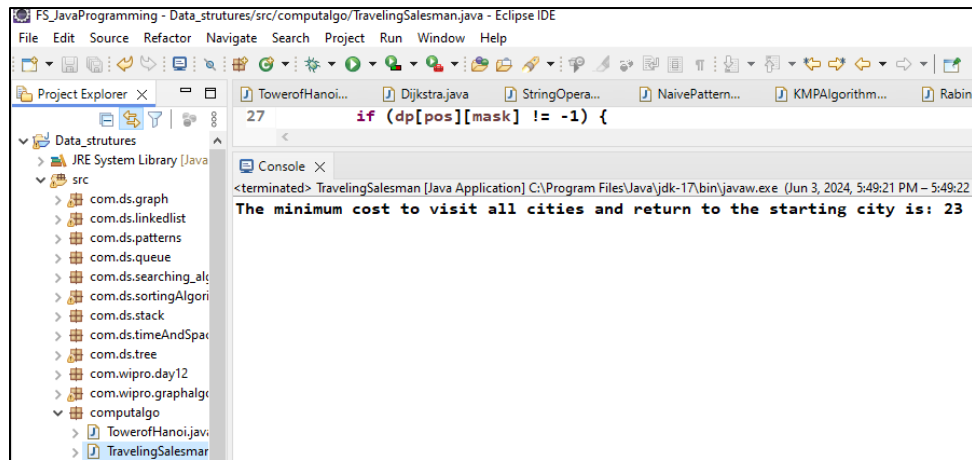


```
1 package computalgo;
2
3 import java.util.Arrays;
4
5 public class TravelingSalesman {
6
7     private static final int INF = Integer.MAX_VALUE;
8
9     public int findMinCost(int[][] graph) {
10         int n = graph.length;
11         int[][] dp = new int[n][1 << n];
12
13         for (int[] row : dp) {
14             Arrays.fill(row, -1);
15         }
16
17         return tsp(graph, 0, 1, dp);
18     }
19
20     private int tsp(int[][] graph, int pos, int mask, int[][] dp) {
21         int n = graph.length;
22
23         if (mask == (1 << n) - 1) {
24             return graph[pos][0] == 0 ? INF : graph[pos][0];
25         }
26
27         if (dp[pos][mask] != -1) {
28             return dp[pos][mask];
29         }
30
31         int minCost = INF;
```



```
27         if (dp[pos][mask] != -1) {
28             return dp[pos][mask];
29         }
30
31         int minCost = INF;
32
33         for (int city = 0; city < n; city++) {
34             if ((mask & (1 << city)) == 0 && graph[pos][city] != 0) {
35                 int newCost = graph[pos][city] + tsp(graph, city, mask | (1 << city), dp);
36                 minCost = Math.min(minCost, newCost);
37             }
38         }
39
40         dp[pos][mask] = minCost;
41         return minCost;
42     }
43
44     public static void main(String[] args) {
45         TravelingSalesman tspSolver = new TravelingSalesman();
46         int[][] graph = {
47             { 0, 16, 11, 6 },
48             { 8, 0, 13, 16 },
49             { 4, 7, 0, 9 },
50             { 5, 12, 2, 0 }
51         };
52
53         int minCost = tspSolver.findMinCost(graph);
54         System.out.println("The minimum cost to visit all cities and return to the starting city is: " + minCost);
55     }
56 }
57 }
```

Output:

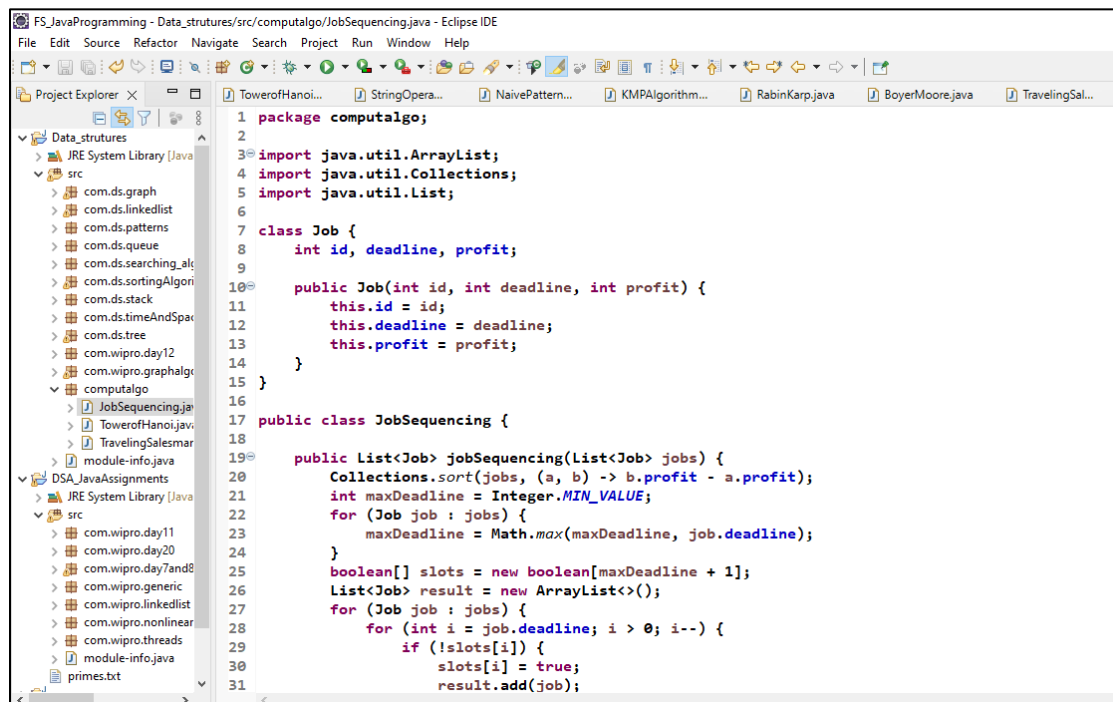


```
FS_JavaProgramming - Data_structures/src/computalgo/travelingSalesman.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Project Explorer
Data_structures
  JRE System Library [Java]
  src
    com.ds.graph
    com.ds.linkedlist
    com.ds.patterns
    com.ds.queue
    com.ds.searching_alg
    com.ds.sortingAlgori
    com.ds.stack
    com.ds.timeAndSpac
    com.ds.tree
    com.wipro.day12
    com.wipro.graphalge
    computalgo
      TowerofHanoi.jav
      TravelingSalesmar
TowerofHanoi...
Dijkstra.java
StringOpera...
NaivePattern...
KMPAlgorithm...
Rabin...
27    if (dp[pos][mask] != -1) {
Console
<terminated> TravelingSalesman [Java Application] C:\Program Files\Java\jdk-17\bin\javaw.exe (Jun 3, 2024, 5:49:21 PM - 5:49:22 PM)
The minimum cost to visit all cities and return to the starting city is: 23
```

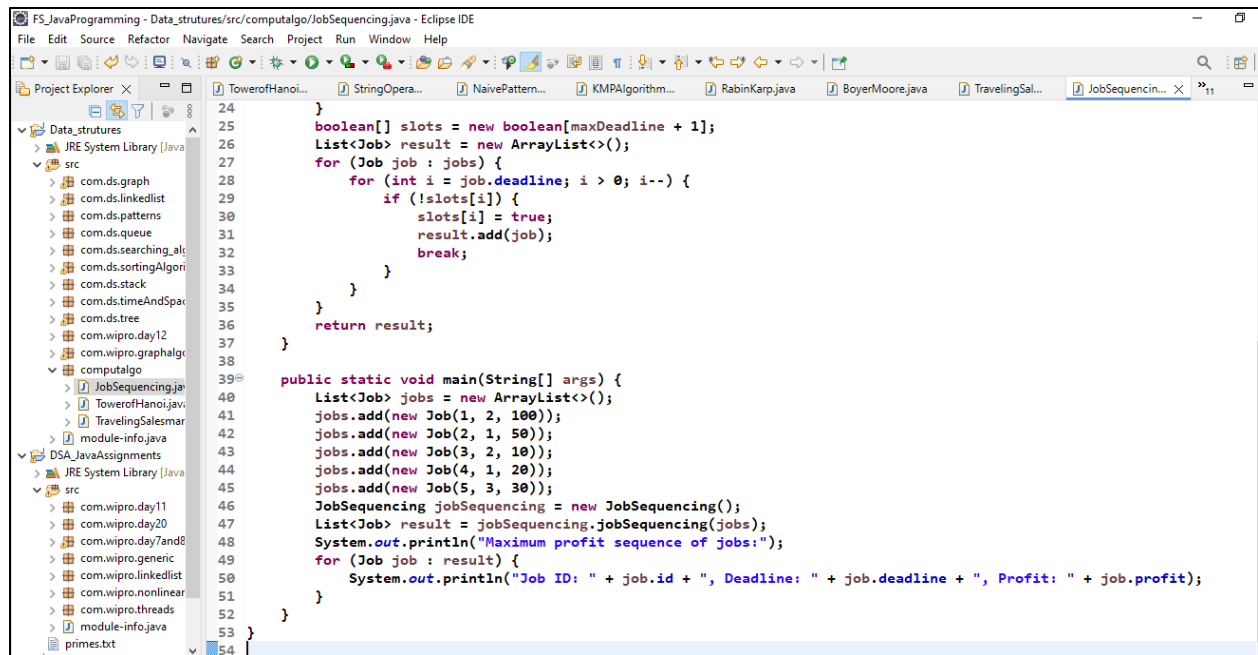
Task 3: Job Sequencing Problem

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

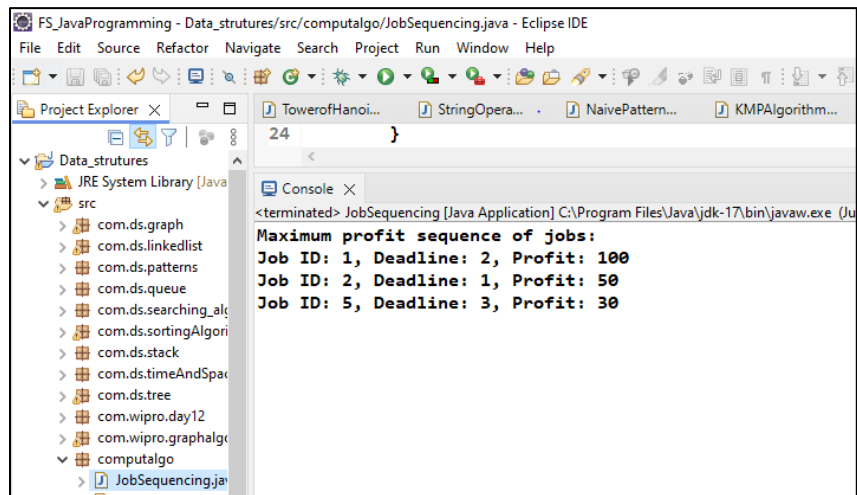


```
FS_JavaProgramming - Data_structures/src/computalgo/JobSequencing.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Project Explorer
Data_structures
  JRE System Library [Java]
  src
    com.ds.graph
    com.ds.linkedlist
    com.ds.patterns
    com.ds.queue
    com.ds.searching_alg
    com.ds.sortingAlgori
    com.ds.stack
    com.ds.timeAndSpac
    com.ds.tree
    com.wipro.day12
    com.wipro.graphalge
    computalgo
      JobSequencing.java
      TowerofHanoi.jav
      TravelingSalesmar
DSA_JavaAssignments
  JRE System Library [Java]
  src
    com.wipro.day11
    com.wipro.day20
    com.wipro.day7and8
    com.wipro.generic
    com.wipro.linkedlist
    com.wipro.nonlinear
    com.wipro.threads
    module-info.java
primes.txt
TowerofHanoi...
StringOpera...
NaivePattern...
KMPAlgorithm...
RabinKarp.java
BoyerMoore.java
TravelingSal...
1 package computalgo;
2
3 import java.util.ArrayList;
4 import java.util.Collections;
5 import java.util.List;
6
7 class Job {
8     int id, deadline, profit;
9
10    public Job(int id, int deadline, int profit) {
11        this.id = id;
12        this.deadline = deadline;
13        this.profit = profit;
14    }
15 }
16
17 public class JobSequencing {
18
19    public List<Job> jobSequencing(List<Job> jobs) {
20        Collections.sort(jobs, (a, b) -> b.profit - a.profit);
21        int maxDeadline = Integer.MIN_VALUE;
22        for (Job job : jobs) {
23            maxDeadline = Math.max(maxDeadline, job.deadline);
24        }
25        boolean[] slots = new boolean[maxDeadline + 1];
26        List<Job> result = new ArrayList<>();
27        for (Job job : jobs) {
28            for (int i = job.deadline; i > 0; i--) {
29                if (!slots[i]) {
30                    slots[i] = true;
31                    result.add(job);
32                }
33            }
34        }
35        return result;
36    }
37 }
```



```
24 }
25
26 boolean[] slots = new boolean[maxDeadline + 1];
27 List<Job> result = new ArrayList<>();
28 for (Job job : jobs) {
29     for (int i = job.deadline; i > 0; i--) {
30         if (!slots[i]) {
31             slots[i] = true;
32             result.add(job);
33             break;
34         }
35     }
36 }
37 return result;
38 }
39
40 public static void main(String[] args) {
41     List<Job> jobs = new ArrayList<>();
42     jobs.add(new Job(1, 2, 100));
43     jobs.add(new Job(2, 1, 50));
44     jobs.add(new Job(3, 2, 10));
45     jobs.add(new Job(4, 1, 20));
46     jobs.add(new Job(5, 3, 30));
47     JobSequencing jobSequencing = new JobSequencing();
48     List<Job> result = jobSequencing.jobSequencing(jobs);
49     System.out.println("Maximum profit sequence of jobs:");
50     for (Job job : result) {
51         System.out.println("Job ID: " + job.id + ", Deadline: " + job.deadline + ", Profit: " + job.profit);
52     }
53 }
54 }
```

Output:



```
<terminated> JobSequencing [Java Application] C:\Program Files\Java\jdk-17\bin\javaw.exe (Ju
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
```