DAY-13,14 JAVA ASSIGNMENT

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.

TowerOfHanoi.java

```java
package com.wipro.computationalgo;

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);
    }

}
```

Console

```
<terminated> TowerOfHanoi [Java Application] C:\Program Files\Java\jdk-17\bin\javaw.exe  (4 Jun 2024, 4:33:48 pm – 4:33:48 pm) [pid
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.

TowerOfHanoi.java | TravelingSalesman.java

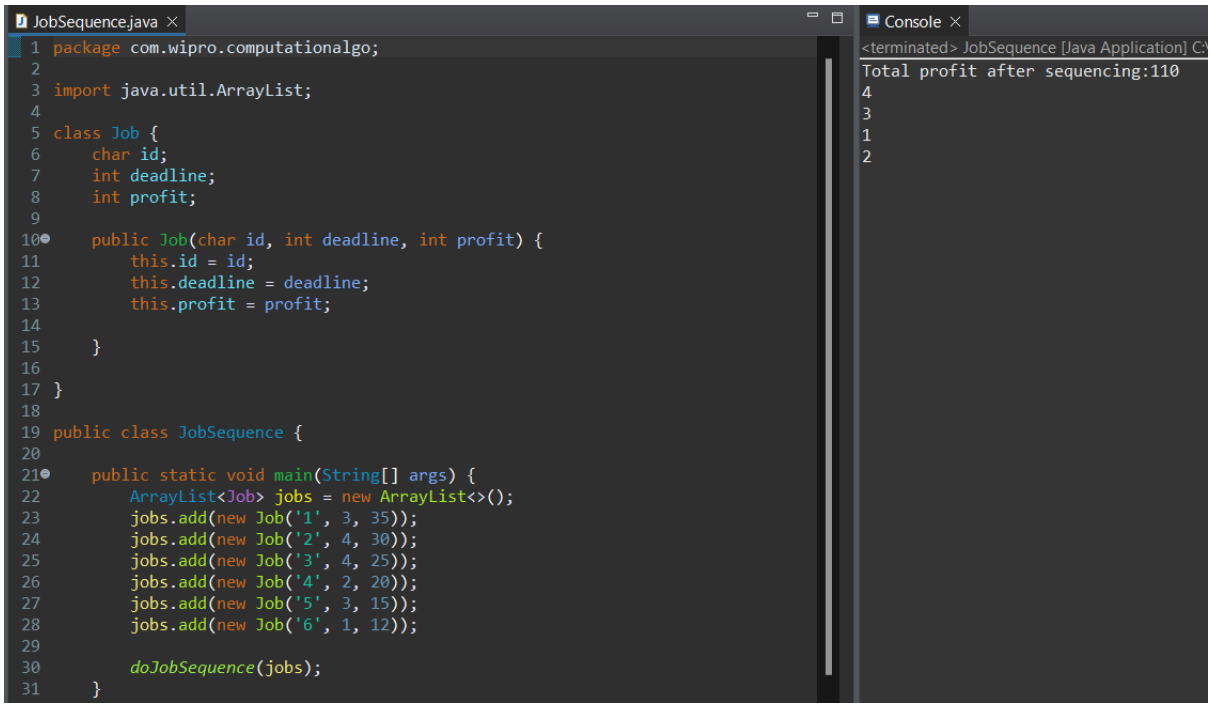
```java
package day13;

public class TravelingSalesman {

    public static void main(String[] args) {
        int[][] graph = {
            {0, 10, 15, 20},
            {10, 0, 35, 25},
            {15, 35, 0, 30},
            {20, 25, 30, 0}
        };
        System.out.println("The minimum cost to visit all cities and return to the starting city is: " + findMinCost(graph));
    }

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

        // Initialize dp array with -1
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < (1 << n); j++) {
                dp[i][j] = -1;
            }
        }

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

    private static int tsp(int pos, int mask, int[][] dp, int[][] graph, int VISITED_ALL) {
        if (mask == VISITED_ALL) {
            return graph[pos][0]; // return to starting point
        }

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

        int minCost = Integer.MAX_VALUE;

        // Visit all the unvisited cities and choose the one with the minimum cost
        for (int city = 0; city < graph.length; city++) {
            if ((mask & (1 << city)) == 0) { // if city is not visited
                int newCost = graph[pos][city] + tsp(city, mask | (1 << city), dp, graph, VISITED_ALL);
                minCost = Math.min(minCost, newCost);
            }
        }

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

Console

```
<terminated> TravelingSalesman [Java Application] C:\Program Files\Java\jdk-17\bin\javaw.exe  (4 Jun 2024,
The minimum cost to visit all cities and return to the starting city is: 80
```

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.




```java
package com.wipro.computationalgo;

import java.util.ArrayList;

class Job {
    char id;
    int deadline;
    int profit;

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

    }

}

public class JobSequence {

    public static void main(String[] args) {
        ArrayList<Job> jobs = new ArrayList<>();
        jobs.add(new Job('1', 3, 35));
        jobs.add(new Job('2', 4, 30));
        jobs.add(new Job('3', 4, 25));
        jobs.add(new Job('4', 2, 20));
        jobs.add(new Job('5', 3, 15));
        jobs.add(new Job('6', 1, 12));

        doJobSequence(jobs);
    }
```

```java

    private static void doJobSequence(ArrayList<Job> jobs) {
        jobs.sort((a, b) -> b.profit - a.profit);

        int maxDeadLine = Integer.MIN_VALUE;
        for (Job job : jobs) {
            maxDeadLine = Math.max(maxDeadLine, job.deadline);
        }

        boolean[] filledSlots = new boolean[maxDeadLine];

        char[] results = new char[maxDeadLine];
        int totalProfit = 0;
        for (Job job : jobs) {
            for (int i = job.deadline - 1; i >= 0; i--) {
                if (!filledSlots[i]) {
                    filledSlots[i] = true;
                    results[i] = job.id;
                    totalProfit += job.profit;
                    break;
                }
            }
        }
        System.out.println("Total profit after sequencing:" + totalProfit);
        for (char id : results) {
            System.out.println(id + "  ");
        }

    }
}
```