Name- Amit Kumar Parhi

Email- amitkparhi07@gmail.com

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.

Solution:

```
package com.wipro;

public class TowerOfHanoi {

    // Recursive function to solve Tower of Hanoi puzzle
    public static void solveHanoi(int n, char source, char auxiliary, char
destination) {
        if (n == 1) {
            System.out.println("Move disk 1 from " + source + " to " + destination);
            return;
        }

        // Move n-1 disks from source to auxiliary, so they are out of the way
        solveHanoi(n - 1, source, destination, auxiliary);

        // Move the nth disk from source to destination
        System.out.println("Move disk " + n + " from " + source + " to " +
destination);

        // Move the n-1 disks that we left on auxiliary to destination
        solveHanoi(n - 1, auxiliary, source, destination);
    }

    public static void main(String[] args) {
        int n = 4; // Number of disks
        solveHanoi(n, 'A', 'B', 'C'); // A, B and C are names of rods
    }
}
```

Output:

Move disk 1 from A to B

Move disk 2 from A to C

Move disk 1 from B to C

Move disk 3 from A to B

Move disk 1 from C to A

Move disk 2 from C to B

Move disk 1 from A to B

Move disk 4 from A to C

Move disk 1 from B to C

Move disk 2 from B to A

Move disk 1 from C to A

Move disk 3 from B to C

Move disk 1 from A to B

Move disk 2 from A to C

Move disk 1 from B 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.

Solution:

```
package com.wipro;

import java.util.Arrays;

public class TravelingSalesmanProblem {

    // Function to find the minimum cost to visit all cities and return to the
starting city
    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[] row : dp) {
            Arrays.fill(row, -1);
        }
        return tsp(0, 1, graph, dp, VISITED_ALL);
    }

    private static int tsp(int currentCity, int mask, int[][] graph, int[][] dp, int
VISITED_ALL) {
        if (mask == VISITED_ALL) {
```

```
            return graph[currentCity][0]; // Return to starting city
        }

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

        int minCost = Integer.MAX_VALUE;

        // Try to go to every other city
        for (int city = 0; city < graph.length; city++) {
            // If the city is not visited
            if ((mask & (1 << city)) == 0) {
                int newCost = graph[currentCity][city] + tsp(city, mask | (1 <<
city), graph, dp, VISITED_ALL);
                minCost = Math.min(minCost, newCost);
            }
        }

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

    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));
    }
}
```

Output:

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.

Solution:

```java
package com.wipro;

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

class Job {
    int Id;
    int Deadline;
    int Profit;

    // Constructor
    public Job(int id, int deadline, int profit) {
        this.Id = id;
        this.Deadline = deadline;
        this.Profit = profit;
    }
}

public class JobSequencing {

    public static List<Job> jobSequencing(List<Job> jobs) {
        // Sort jobs in descending order of profit
        Collections.sort(jobs, (a, b) -> b.Profit - a.Profit);

        // Find the maximum deadline to determine the range of the timeline
        int maxDeadline = 0;
        for (Job job : jobs) {
            if (job.Deadline > maxDeadline) {
                maxDeadline = job.Deadline;
            }
        }

        // Create a result array to keep track of the job scheduled at each time slot
        Job[] result = new Job[maxDeadline];

        // Iterate through the sorted jobs and place them in the latest possible slot
        for (Job job : jobs) {
            for (int j = job.Deadline - 1; j >= 0; j--) {
                if (result[j] == null) {
                    result[j] = job;
                    break;
                }
            }
        }

        // Collect the jobs in the result array into a list to return
        List<Job> jobSequence = new ArrayList<>();
        for (Job job : result) {
            if (job != null) {
                jobSequence.add(job);
            }
        }

        return jobSequence;
```

```
    }

    public static void main(String[] args) {
        List<Job> jobs = new ArrayList<>();
        jobs.add(new Job(1, 4, 20));
        jobs.add(new Job(2, 1, 10));
        jobs.add(new Job(3, 1, 40));
        jobs.add(new Job(4, 1, 30));

        List<Job> jobSequence = jobSequencing(jobs);

        System.out.println("The job sequence for maximum profit is:");
        for (Job job : jobSequence) {
            System.out.println("Job Id: " + job.Id + ", Deadline: " + job.Deadline +
", Profit: " + job.Profit);
        }
    }
}
```

Output:

The job sequence for maximum profit is:

Job Id: 3, Deadline: 1, Profit: 40

Job Id: 1, Deadline: 4, Profit: 20