Day 13 and 14

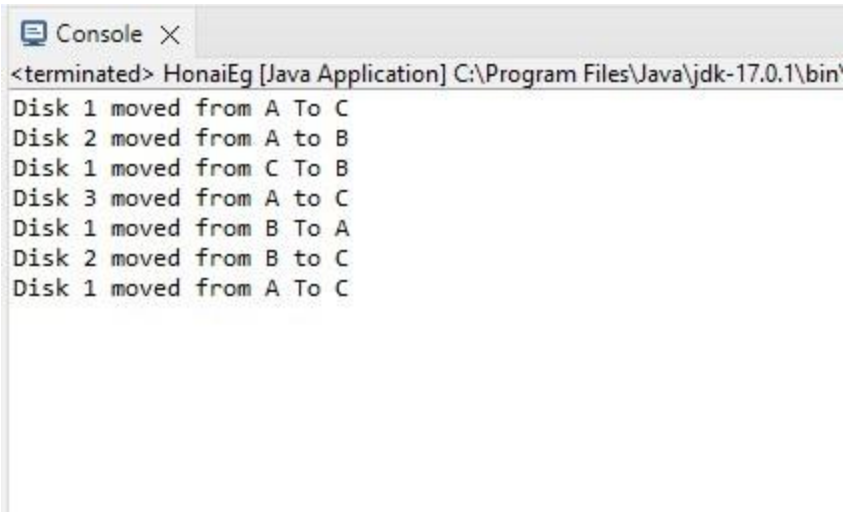
1] Task 1: Tower of Hanoi Solver
Create a program that solves the Tower of Hanoi puzzle for n disks. The solution should use recursion to move disks between three pegs (source, auxiliary, and destination) according to the game's rules. The program should print out each move required to solve the puzzle.

Solution:-
Code -

HonaiEg.java

```java
package com.wipro.computealgo;

public class HonaiEg {


    public static void main(String[] args) {
        // TODO Auto-generated method stub
        honai(3,"A","B","C");

    }
    private static void honai(int n, String rodFrom, String rodMiddle, String rodTo) {
        // TODO Auto-generated method stub
        if(n==1) {
            System.out.println("Disk 1 moved from "+rodFrom +" To "+rodTo);
            return ;
        }
        honai(n-1,rodFrom,rodTo,rodMiddle);

        System.out.println("Disk "+n+" moved from "+rodFrom+" to "+rodTo);

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

    }

}

```

Output -



2] 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:-
Code -

TravellingSalesmanEg.java

```java
package com.wipro.computealgo;

import java.util.ArrayList;
public class TravellingSalesmanEg {

    private static int[][] distances = {
            {0, 16, 11, 6},
            {8, 0, 13, 16},
            {4, 7, 0, 9},
            {5, 12, 2, 0}
        };

        public static List<Integer> findShortestPath(int[][] graph) {
            int n = graph.length;
            int[] cities = new int[n];
            for (int i = 0; i < n; i++) {
                cities[i] = i;
            }

            List<Integer> shortestPath = new ArrayList<>();
            int shortestDistance = Integer.MAX_VALUE;

            do {
                int currentDistance = calculatePathDistance(cities, graph);
                if (currentDistance < shortestDistance) {
                    shortestDistance = currentDistance;
```

TravellingSalesmanEg.java

```java
                if (currentDistance < shortestDistance) {
                    shortestDistance = currentDistance;
                    shortestPath.clear();
                    for (int city : cities) {
                        shortestPath.add(city);
                    }
                }
            } while (nextPermutation(cities));

            return shortestPath;
        }

        private static int calculatePathDistance(int[] path, int[][] graph) {
            int distance = 0;
            for (int i = 0; i < path.length - 1; i++) {
                distance += graph[path[i]][path[i + 1]];
            }
            distance += graph[path[path.length - 1]][path[0]]; // Return to the starting city
            return distance;
        }

        private static boolean nextPermutation(int[] array) {
            int i = array.length - 2;
            while (i >= 0 && array[i] >= array[i + 1]) {
                i--;
            }
```

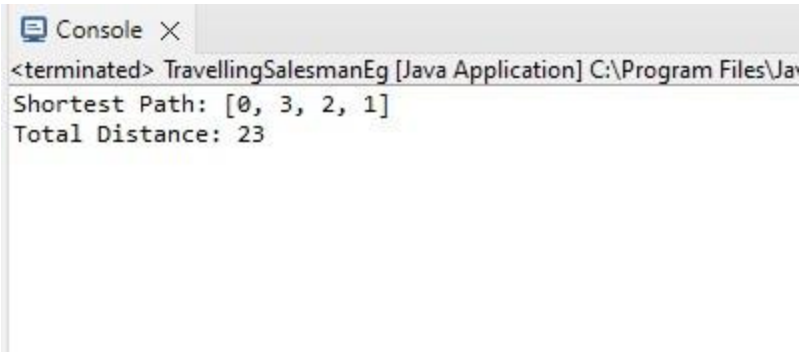
TravellingSalesmanEg.java

```
                i--;
            }
            if (i < 0) {
                return false;
            }
            int j = array.length - 1;
            while (array[j] <= array[i]) {
                j--;
            }
            swap(array, i, j);
            reverse(array, i + 1);
            return true;
        }

        private static void swap(int[] array, int i, int j) {
            int temp = array[i];
            array[i] = array[j];
            array[j] = temp;
        }

        private static void reverse(int[] array, int start) {
            int i = start;
            int j = array.length - 1;
            while (i < j) {
                swap(array, i, j);
                i++;
```

*TravellingSalesmanEg.java

```
        private static void swap(int[] array, int i, int j) {
            int temp = array[i];
            array[i] = array[j];
            array[j] = temp;
        }

        private static void reverse(int[] array, int start) {
            int i = start;
            int j = array.length - 1;
            while (i < j) {
                swap(array, i, j);
                i++;
                j--;
            }
        }

        public static void main(String[] args) {
            List<Integer> shortestPath = findShortestPath(distances);
            System.out.println("Shortest Path: " + shortestPath);
System.out.println("Total Distance: " + calculatePathDistance(shortestPath.stream().mapToInt(i -> i).toArray(), distances));
        }
    }
```

Output -



3] 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:-
Code -

JobSequencingProblem.java

```java
package com.wipro.computealgo;

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 JobSequencingProblem {

    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));
```

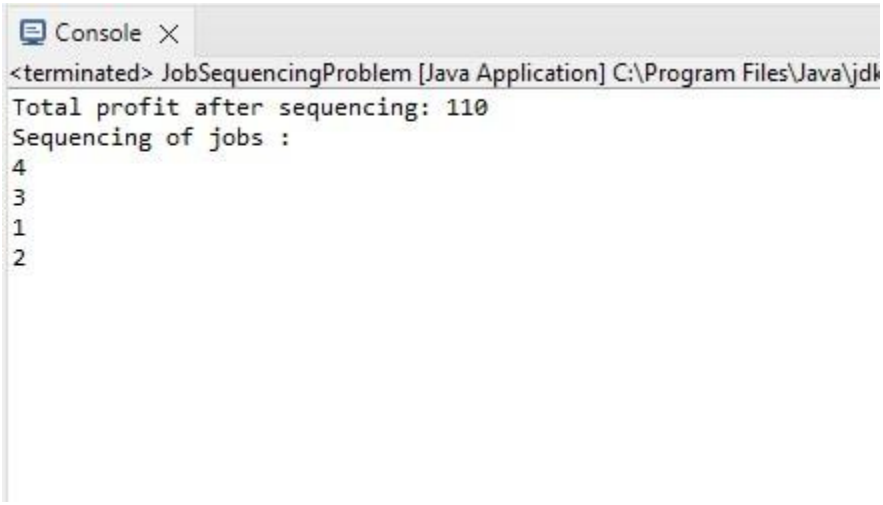
JobSequencingProblem.java

```
        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);
    }

    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;
```

JobSequencingProblem.java

```
        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);
        System.out.println("Sequencing of jobs :");
        for(char id: results) {
            System.out.println(id + "  " );
        }


    }
}
```

Output -


Console
<terminated> JobSequencingProblem [Java Application] C:\Program Files\Java\jdk
Total profit after sequencing: 110
Sequencing of jobs :
4
3
1
2