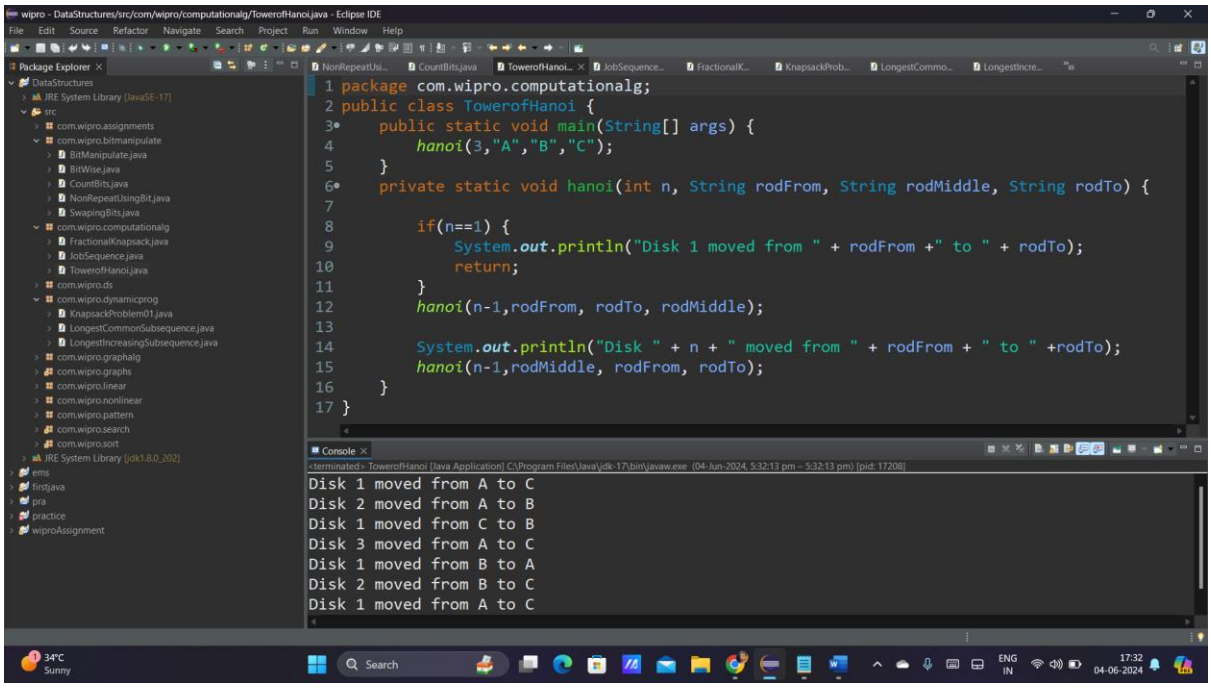
Assignment-Day13,14

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.



Task 2: Traveling Salesman Problem

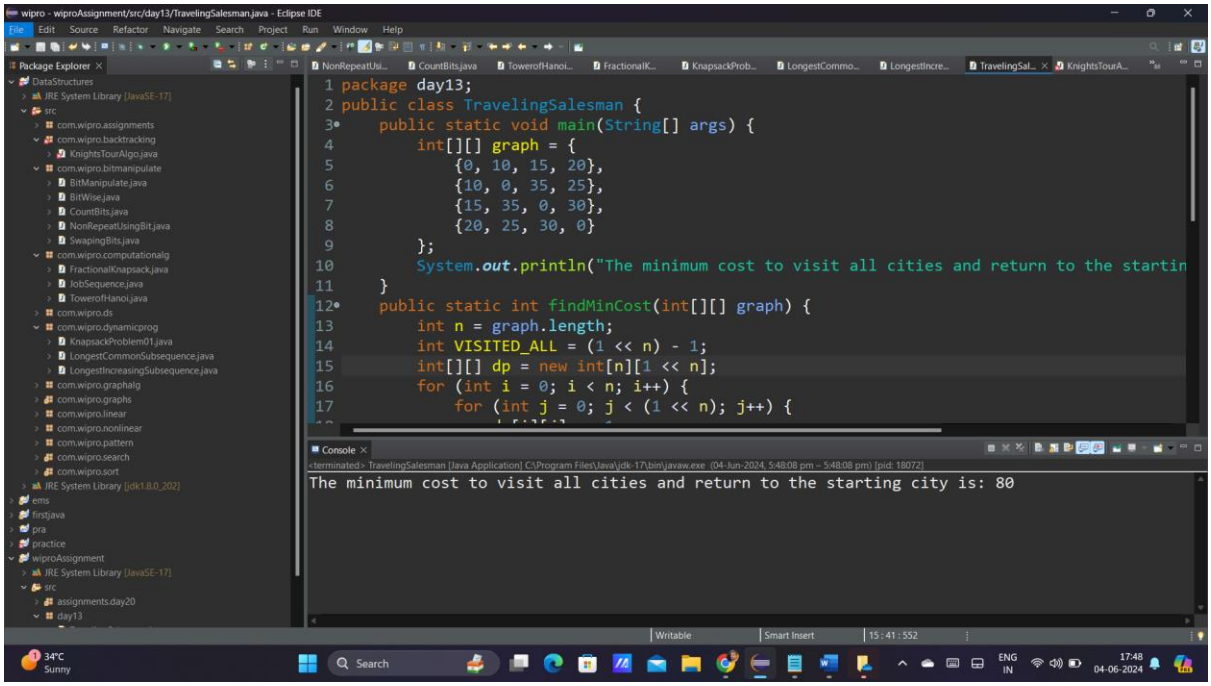
Create a function int FindMinCost(int[,] graph) that takes a 2D array representing the graph where graph[i][j] is the cost to travel from city i to city j. The function should return the minimum cost to visit all cities and return to the starting city. Use dynamic programming for this solution.

wipro - wiproAssignment/src/day13/TravelingSalesman.java - Eclipse IDE

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

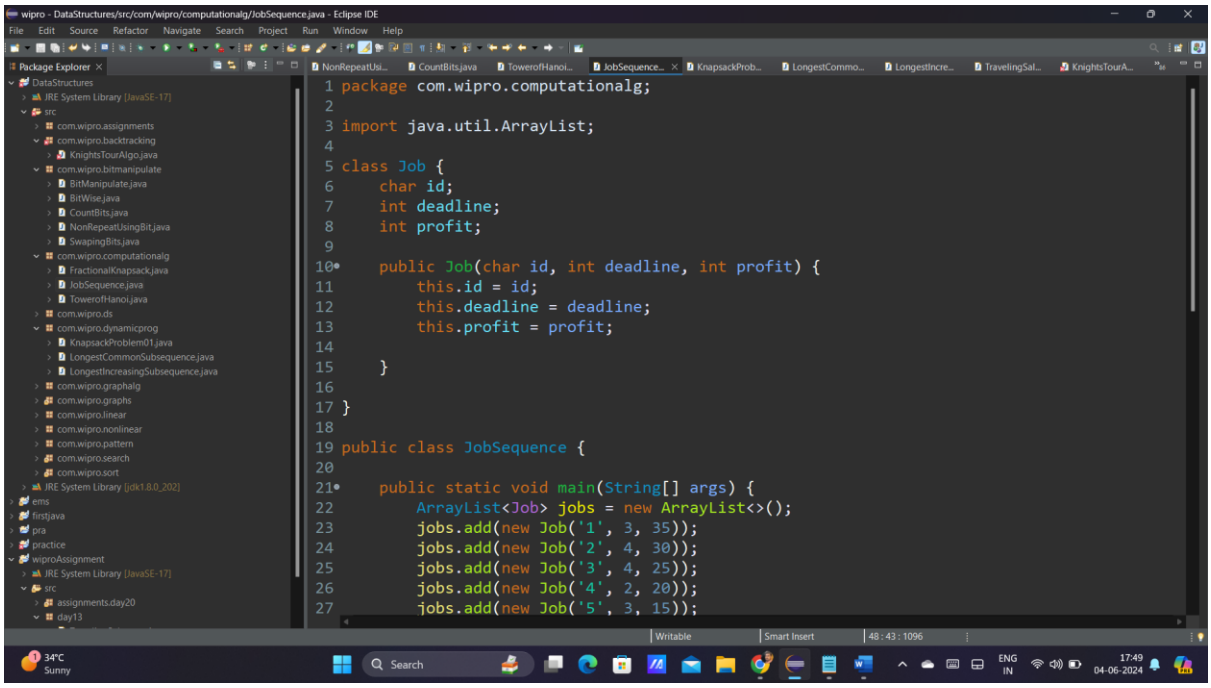
wipro - wiproAssignment/src/day13/TravelingSalesman.java - Eclipse IDE

```java
        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];
        }
        if (dp[pos][mask] != -1) {
            return dp[pos][mask];
        }
        int minCost = Integer.MAX_VALUE;
        for (int city = 0; city < graph.length; city++) {
            if ((mask & (1 << city)) == 0) {
                int newCost = graph[pos][city] + tsp(city, mask | (1 << city), dp, graph,
                minCost = Math.min(minCost, newCost);
            }
        }

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



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.



wipro - DataStructures/src/com/wipro/computationalg/JobSequence.java - Eclipse IDE

```java
        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;
                    totalProfit += job.profit;
                    break;
                }
```

wipro - DataStructures/src/com/wipro/computationalg/JobSequence.java - Eclipse IDE

```java
            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 + "  " );
        }

    }
}
```

Console

```
<terminated> JobSequence [Java Application] C:\Program Files\Java\jdk-17\bin\javaw.exe (04-Jun-2024, 5:50:21 pm – 5:50:24 pm) [pid: 13780]
Total profit after sequencing:110
4
3
1
2
```