

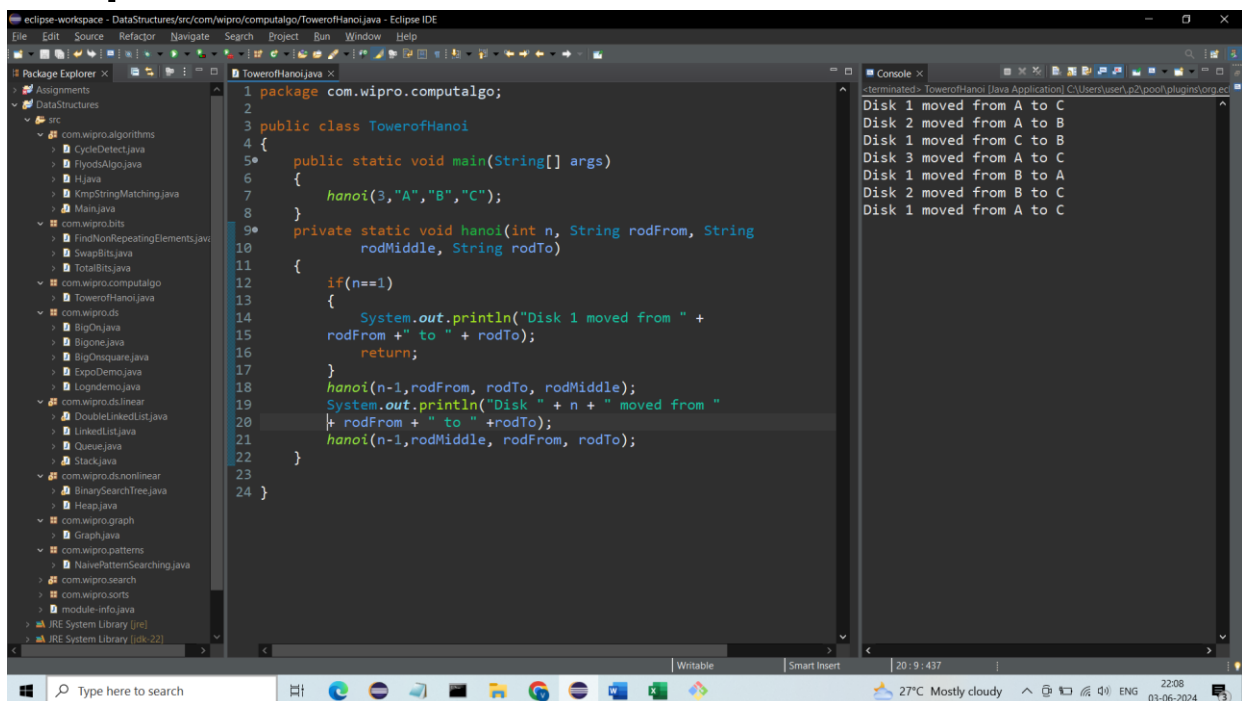
Task 1: Tower of Hanoi Solver

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.

```
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);
}
```

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

Output: -



The screenshot shows the Eclipse IDE with the TowerofHanoi.java file open. The code is the same as shown in the previous block. The Package Explorer on the left shows the project structure. The Console on the right displays the output of the program, which is a sequence of disk moves for a 3-disk Tower of Hanoi problem. The output is as follows:

```
terminated: TowerofHanoi [Java Application] C:\Users\user\p2\pool\plugins\org.ec
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.

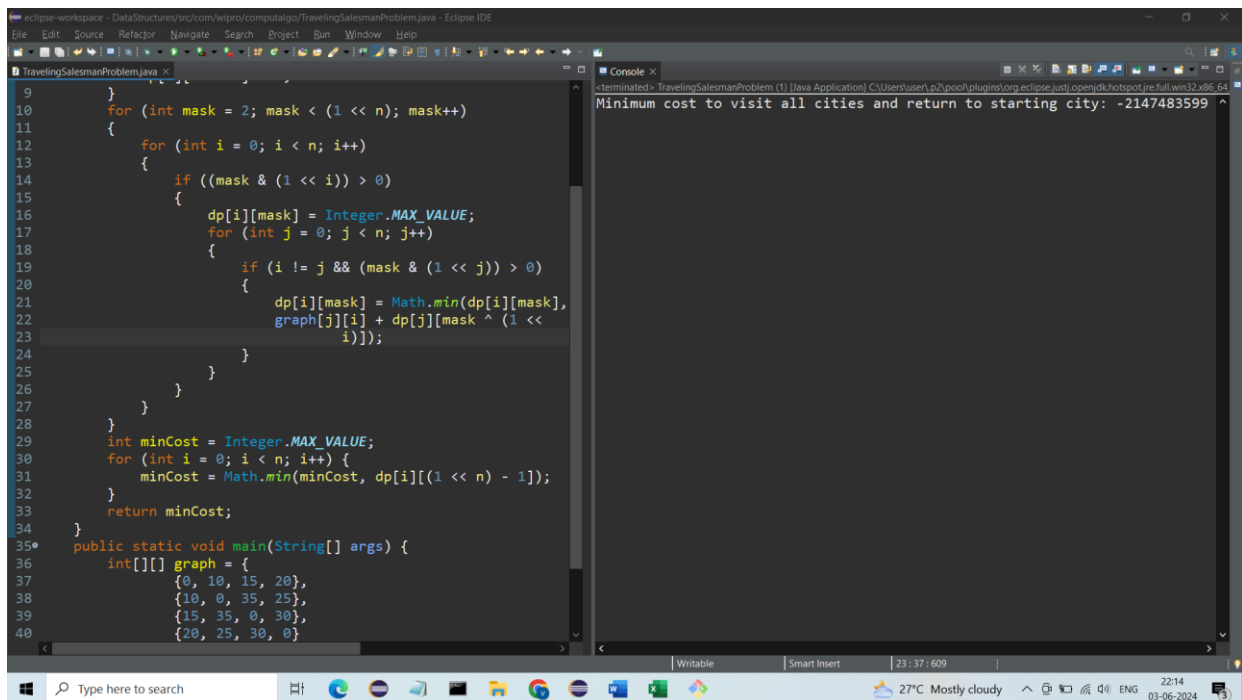
Function: -

```

public static int findMinCost(int[][] graph) {
    int n = graph.length;
    int[][] dp = new int[n][1 << n];
    for (int i = 0; i < n; i++) {
        dp[i][1 << i] = 0;
    }
    for (int mask = 2; mask < (1 << n); mask++)
    {
        for (int i = 0; i < n; i++)
        {
            if ((mask & (1 << i)) > 0)
            {
                dp[i][mask] = Integer.MAX_VALUE;
                for (int j = 0; j < n; j++)
                {
                    if (i != j && (mask & (1 << j)) > 0)
                    {
                        dp[i][mask] = Math.min(dp[i][mask],
                            graph[j][i] + dp[j][mask ^ (1 << i)]);
                    }
                }
            }
        }
    }
    int minCost = Integer.MAX_VALUE;
    for (int i = 0; i < n; i++) {
        minCost = Math.min(minCost, dp[i][(1 << n) - 1]);
    }
    return minCost;
}

```

Output: -



```
9
10     for (int mask = 2; mask < (1 << n); mask++)
11     {
12         for (int i = 0; i < n; i++)
13         {
14             if ((mask & (1 << i)) > 0)
15             {
16                 dp[i][mask] = Integer.MAX_VALUE;
17                 for (int j = 0; j < n; j++)
18                 {
19                     if (i != j && (mask & (1 << j)) > 0)
20                     {
21                         dp[i][mask] = Math.min(dp[i][mask],
22                                                 graph[j][i] + dp[j][mask ^ (1 <<
23                                                 i)]);
24                     }
25                 }
26             }
27         }
28     }
29     int minCost = Integer.MAX_VALUE;
30     for (int i = 0; i < n; i++) {
31         minCost = Math.min(minCost, dp[i][(1 << n) - 1]);
32     }
33     return minCost;
34 }
35* public static void main(String[] args) {
36     int[][] graph = {
37         {0, 10, 15, 20},
38         {10, 0, 35, 25},
39         {15, 35, 0, 30},
40         {20, 25, 30, 0}
35*
36
37
38
39
40
35*
36
37
38
39
40
```

Console Output: Minimum cost to visit all cities and return to starting city: -2147483599

Task 3: Job Sequencing Problem

Define a class Job with properties int Id, int Deadline, and int Profit. Then implement a function List JobSequencing(List 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.

Class: -

```

5 class Job {
6     public int id;
7     public int deadline;
8     public int profit;
9     public Job(int id, int deadline, int profit) {
10         this.id = id;
11         this.deadline = deadline;
12         this.profit = profit;
13     }
14 }

```

Function: -

```

public class JobSequencing {
    public static List<Job> scheduleJobs(List<Job> jobs) {
        jobs.sort((job1, job2) -> Integer.compare(job2.profit, job1.profit));
        int maxDeadline = 0;
        for (Job job : jobs) {
            maxDeadline = Math.max(maxDeadline, job.deadline);
        }
        boolean[] slots = new boolean[maxDeadline + 1];
        List<Job> scheduledJobs = new ArrayList<>();
        for (Job job : jobs) {
            for (int deadline = job.deadline; deadline > 0; deadline--) {
                if (!slots[deadline]) {
                    slots[deadline] = true;
                    scheduledJobs.add(job);
                    break;
                }
            }
        }
        return scheduledJobs;
    }
}

```

Output: -

