

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.

```
package com.wipro.computalgo;
```

```
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);

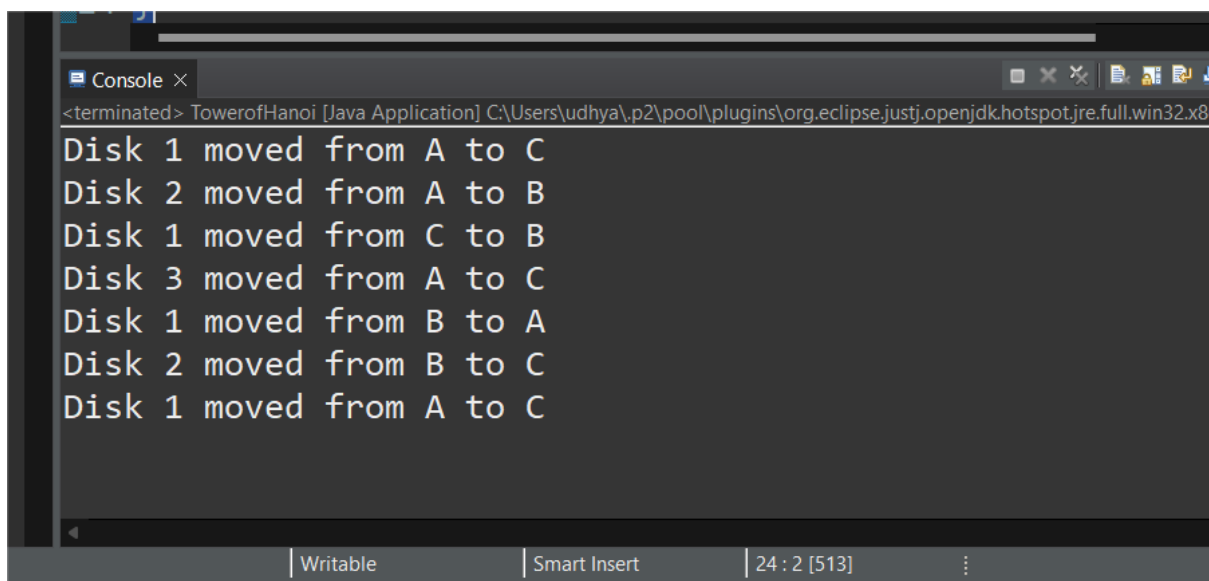
        hanoi(n-1,rodMiddle, rodFrom, rodTo);

    }

}

```

OUTPUT:



```

<terminated> TowerofHanoi [Java Application] C:\Users\udhya\p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64.jdk\bin\java.exe
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.

```
package com.wipro.computalgo;
```

```
import java.util.Arrays;
```

```
public class TravelingSalesman {
```

```
private static final int INF = Integer.MAX_VALUE;
```

```
public static int tsp(int[][] graph) {
```

```
    int n = graph.length;
```

```
    int VISITED_ALL = (1 << n) - 1;
```

```
    int[][] dp = new int[n][(1 << n)];
```

```
    for (int[] row : dp) {
```

```
        Arrays.fill(row, INF);
```

```
    }
```

```
    for (int i = 0; i < n; i++) {
```

```
        dp[i][1 << i] = graph[0][i];
```

```
    }
```

```
    for (int mask = 1; mask < (1 << n); mask++) {
```

```
        for (int u = 0; u < n; u++) {
```

```
            if ((mask & (1 << u)) != 0) { // Check if u is in the subset represented by  
mask
```

```
                for (int v = 0; v < n; v++) {
```

```
                    if ((mask & (1 << v)) == 0) { // Check if v is not in the subset
```

```
                        dp[v][mask | (1 << v)] = Math.min(dp[v][mask | (1 << v)],  
dp[u][mask] + graph[u][v]);
```

```

        }
    }
}
}
}

```

```

int minCost = INF;
for (int i = 0; i < n; i++) {
    minCost = Math.min(minCost, dp[i][VISITED_ALL - 1] + graph[i][0]);
}

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}
    };

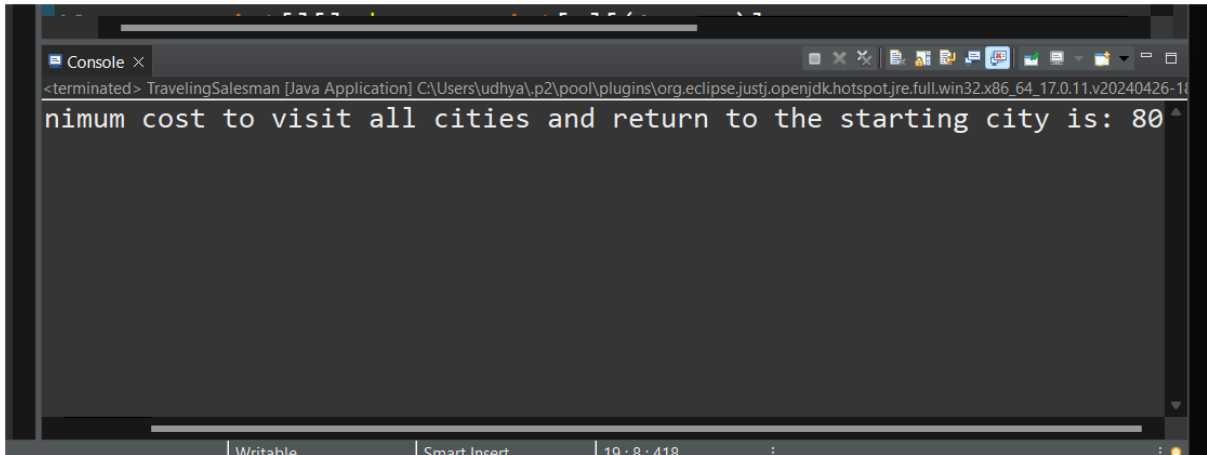
    int result = tsp(graph);

    System.out.println("The minimum cost to visit all cities and return to the
starting city is: " + result);
}

```

```
}
```

OUTPUT:



```
<terminated> TravelingSalesman [Java Application] C:\Users\udhya\p2\pool\plugins\org.eclipse.justi.openjdk.hotspot.jre.full.win32.x86_64_17.0.11.v20240426-1f
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.

```
package com.wipro.computalgo;
```

```
import java.util.ArrayList;
```

```
import java.util.Arrays;
```

```
import java.util.Collections;
```

```
import java.util.Comparator;
```

```
import java.util.List;
```

```
class Job {
```

```
    int Id;
```

```
    int Deadline;
```

```
int Profit;
```

```
public Job(int id, int deadline, int profit) {  
    Id = id;  
    Deadline = deadline;  
    Profit = profit;  
}
```

```
@Override
```

```
public String toString() {  
    return "Job{" +  
        "Id=" + Id +  
        ", Deadline=" + Deadline +  
        ", Profit=" + Profit +  
        '}';  
}  
}
```

```
public class JobSequencing {
```

```
    public static List<Job> jobSequencing(List<Job> jobs) {
```

```
        Collections.sort(jobs, (a, b) -> b.Profit - a.Profit);
```

```
        int n = jobs.size();
```

```
int maxDeadline = jobs.stream().mapToInt(job ->
job.Deadline).max().orElse(0);
```

```
Job[] result = new Job[maxDeadline];
```

```
boolean[] slot = new boolean[maxDeadline];
```

```
Arrays.fill(slot, false);
```

```
for (Job job : jobs) {
```

```
    for (int j = Math.min(maxDeadline - 1, job.Deadline - 1); j >= 0; j--) {
```

```
        if (!slot[j]) {
```

```
            slot[j] = true;
```

```
            result[j] = job;
```

```
            break;
```

```
        }
```

```
    }
```

```
}
```

```
List<Job> jobSequence = new ArrayList<>();
```

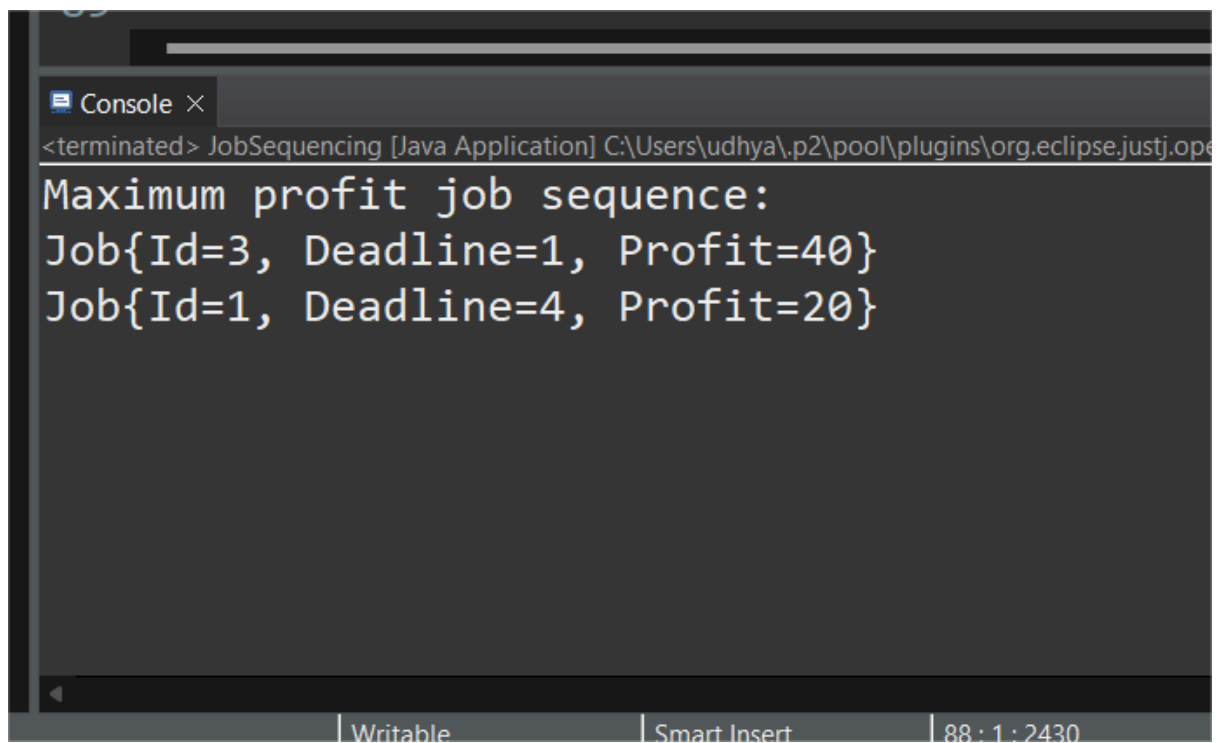
```
for (Job job : result) {
```

```
        if (job != null) {  
            jobSequence.add(job);  
        }  
    }  
}
```

```
    return jobSequence;  
}
```

```
public static void main(String[] args) {  
    List<Job> jobs = Arrays.asList(  
        new Job(1, 4, 20),  
        new Job(2, 1, 10),  
        new Job(3, 1, 40),  
        new Job(4, 1, 30)  
    );  
  
    List<Job> jobSequence = jobSequencing(jobs);  
  
    System.out.println("Maximum profit job sequence:");  
    for (Job job : jobSequence) {  
        System.out.println(job);  
    }  
}
```


OUTPUT:



The screenshot shows the Eclipse IDE's console window. The title bar of the console tab reads "Console X". The text in the console is as follows:

```
<terminated> JobSequencing [Java Application] C:\Users\udhya\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64.jdk-17.0.2-1020220810  
Maximum profit job sequence:  
Job{Id=3, Deadline=1, Profit=40}  
Job{Id=1, Deadline=4, Profit=20}
```

At the bottom of the console window, there is a status bar with the following elements: a left arrow icon, the text "Writable", the text "Smart Insert", and the text "88 : 1 : 2430".