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.
Ans)
Code:-

```
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:-

```
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.

Ans)

Code:-

```
package WiprpTask;
import java.util.*;
class Edge {
  String vertex;
  int weight;
  public Edge(String vertex, int weight) {
     this.vertex = vertex;
     this.weight = weight;
  }
}
public class TravelingSalesman {
  HashMap<String, ArrayList<Edge>> adjList = new HashMap<>();
  public void addVertex(String vertex) {
     if (!adjList.containsKey(vertex)) {
        adjList.put(vertex, new ArrayList<Edge>());
     }
  }
  public void addEdge(String vertex1, String vertex2, int weight) {
     if (adjList.containsKey(vertex1) && adjList.containsKey(vertex2)) {
        adjList.get(vertex1).add(new Edge(vertex2, weight));
        adjList.get(vertex2).add(new Edge(vertex1, weight));
     }
  }
  public int[][] getGraph() {
     int n = adjList.size();
     int[][] graph = new int[n][n];
     String[] vertices = adjList.keySet().toArray(new String[0]);
     for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
           if (i == j) {
              graph[i][j] = 0;
           } else {
              graph[i][j] = Integer.MAX_VALUE;
           }
        }
     }
     for (int i = 0; i < n; i++) {
        for (Edge edge : adjList.get(vertices[i])) {
           int j = Arrays.asList(vertices).indexOf(edge.vertex);
           graph[i][j] = edge.weight;
        }
     }
     return graph;
  }
  public int[] FindMinCost(int[][] graph) {
```

```
    int n = graph.length;
    int dp[][] = new int[1 << n][n];
    int prev[][] = new int[1 << n][n]; // To store the previous vertex
    for (int[] row : dp) {
        Arrays.fill(row, Integer.MAX_VALUE);
    }
    dp[1][0] = 0;
    for (int mask = 0; mask < (1 << n); mask++) {
        for (int i = 0; i < n; i++) {
            if (((mask >> i) & 1) == 1) {
                for (int j = 0; j < n; j++) {
                    if (i != j && ((mask >> j) & 1) == 0) {
                        int newMask = mask | (1 << j);
                        if (dp[mask][i] + graph[i][j] < dp[newMask][j]) {
                            dp[newMask][j] = dp[mask][i] + graph[i][j];
                            prev[newMask][j] = i; // Update previous vertex
                        }
                    }
                }
            }
        }
    }
    int minCost = Integer.MAX_VALUE;
    int endVertex = -1;
    for (int i = 1; i < n; i++) {
        int cost = dp[(1 << n) - 1][i] + graph[i][0];
        if (cost < minCost) {
            minCost = cost;
            endVertex = i;
        }
    }
    // Reconstruct path
    int[] path = new int[n + 1];
    int mask = (1 << n) - 1;
    int current = endVertex;
    int index = n;
    while (mask != 1) {
        path[index--] = current;
        int next = prev[mask][current];
        mask ^= (1 << current);
        current = next;
    }
    path[0] = 0;
    path[n] = endVertex;
    return path;
}
public static void main(String[] args) {
    TravelingSalesman myGraph = new TravelingSalesman();
```

```
        myGraph.addVertex("Calcutta");
        myGraph.addVertex("Bihar");
        myGraph.addVertex("Delhi");
        myGraph.addVertex("Goa");
        myGraph.addEdge("Calcutta", "Bihar", 10);
        myGraph.addEdge("Calcutta", "Delhi", 15);
        myGraph.addEdge("Calcutta", "Goa", 20);
        myGraph.addEdge("Bihar", "Delhi", 35);
        myGraph.addEdge("Bihar", "Goa", 30);
        myGraph.addEdge("Delhi", "Goa", 25);
        int[][] graph = myGraph.getGraph();
        int[] path = myGraph.FindMinCost(graph);
        System.out.println("Minimum cost: " + path[0]);
        System.out.print("Path: ");
        for (int i = 0; i < path.length; i++) {
            System.out.print(path[i]);
            if (i < path.length - 1) {
                System.out.print(" -> ");
            }
        }
    }
}
```

OUTPUT:-

```
Minimum cost: 0
Path: 0 -> 3 -> 2 -> 0 -> 1
```

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.

Ans)

Code:-

```
package com.wipro.patterns;
import java.util.ArrayList;
public class JobSequence {
    class Job {
        String name;
        int deadline;
        int profit;
        public Job(String name, int deadline, int profit) {
            this.name = name;
            this.deadline = deadline;
            this.profit = profit;
        }
    }
```

```
  public static void main(String[] args) {
     JobSequence jobSequence = new JobSequence();
     ArrayList<Job> jobs = new ArrayList<>();
     jobs.add(jobSequence.new Job("Devs", 3, 35));
     jobs.add(jobSequence.new Job("HR", 4, 30));
     jobs.add(jobSequence.new Job("Testing", 4, 25));
     jobs.add(jobSequence.new Job("Data AI", 2, 20));
     jobs.add(jobSequence.new Job("Quality Control", 3, 15));
     jobs.add(jobSequence.new Job("Call center", 1, 12));
     int totalProfit = doJobSequence(jobs);
     System.out.println("Total profit after sequencing = " + totalProfit);
  }
  private static int 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];
     int[] profits = new int[maxDeadLine];
     String[] results = new String[maxDeadLine];
     for (Job job : jobs) {
        for (int i = job.deadline - 1; i >= 0; i--) {
           if (!filledSlots[i]) {
              filledSlots[i] = true;
              results[i] = job.name;
              profits[i] = job.profit;
              break;
           }
        }
     }
     int totalProfit = 0;
     for (int profit : profits) {
        totalProfit += profit;
     }
     for (String result : results) {
        if (result != null) {
           System.out.print(result + " ");
        }
     }
     System.out.println();
     return totalProfit;
  }
}
```

OUTPUT:-

```
Data AI Testing Devs HR
```

Total profit after sequencing = 110