ASSIGNMENT: 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:

```
public class TowerOfHanoi {


   public static void solveHanoi(int n, char source, char auxiliary, char destination) {
      if (n == 1) {
         System.out.println("Move disk 1 from " + source + " to " + destination);
         return;
      }
      solveHanoi(n - 1, source, destination, auxiliary);
      System.out.println("Move disk " + n + " from " + source + " to " + destination);
      solveHanoi(n - 1, auxiliary, source, destination);
   }


   public static void main(String[] args) {
      int n = 5;
      solveHanoi(n, 'A', 'B', 'C');
   }
}


//code_by_RUBY
```

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:

```
import java.util.Arrays;


public class TravelingSalesman {


   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[] row : dp) {
         Arrays.fill(row, -1);
      }


      return tsp(0, 1, graph, dp, VISITED_ALL);
   }


   private static int tsp(int pos, int mask, int[][] graph, int[][] dp, int VISITED_ALL) {
      if (mask == VISITED_ALL) {
         return graph[pos][0];
      }
```

```
    if (dp[pos][mask] != -1) {
        return dp[pos][mask];
    }

    int ans = Integer.MAX_VALUE;

    for (int city = 0; city < graph.length; city++) {
        if ((mask & (1 << city)) == 0) {
            int newAns = graph[pos][city] + tsp(city, mask | (1 << city), graph, dp, VISITED_ALL);
            ans = Math.min(ans, newAns);
        }
    }

    return dp[pos][mask] = ans;
}

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 is: " + FindMinCost(graph));
    }
}
//code_by_RUBY
```

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:

```
import java.util.*;

class Job {
    int id;
    int deadline;
    int profit;

    public Job(int id, int deadline, int profit) {
        this.id = id;
        this.deadline = deadline;
        this.profit = profit;
    }
}

public class JobSequencing {
```

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

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



    int maxDeadline = 0;
    for (Job job : jobs) {
        if (job.deadline > maxDeadline) {
            maxDeadline = job.deadline;
        }
    }



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

```
                slot[j] = true;
                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("Job Sequence for maximum profit:");

        for (Job job : jobSequence) {

            System.out.println("Job ID: " + job.id + ", Profit: " + job.profit);

        }

    }

}

//code_by_RUBY
```