Day 13 and 14

1. **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.**

```java
package m5_core_java_programming.mycollection;

public class Hanoi {
    public static void start(int disk) {

        diskMove(disk, "A", "B", "C");
    }

    private static void diskMove(int disk, String A, String B, String C) {
        if (disk == 1) {
            System.out.println("Disk 1 is moved from " + A + " to " + C);
            return;
        }
        diskMove(disk - 1, A, C, B);
        System.out.println("Disk " + disk + " is moved from " + A + " to " + C);
        diskMove(disk - 1, B, A, C);
    }

}
```

```java
package m5_core_java_programming.day_19_part_1;

import m5_core_java_programming.mycollection.Hanoi;

/*
    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.
*/
public class Assignment_1 {
    public static void main(String[] args) {
        System.out.println("Hanoi with 4 disk : ");
        Hanoi.start(4);
    }
}
```

Output

```
C:\Users\coolr\.jdks\openjdk-22.0.1\bin\java.exe "-javaagent:C:\Program Files\JetBrains\Inte
Hanoi with 4 disk :
Disk 1 is moved from A to B
Disk 2 is moved from A to C
Disk 1 is moved from B to C
Disk 3 is moved from A to B
Disk 1 is moved from C to A
Disk 2 is moved from C to B
Disk 1 is moved from A to B
Disk 4 is moved from A to C
Disk 1 is moved from B to C
Disk 2 is moved from B to A
Disk 1 is moved from C to A
Disk 3 is moved from B to C
Disk 1 is moved from A to B
Disk 2 is moved from A to C
Disk 1 is moved from B to C

Process finished with exit code 0
```

---

2. **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.**

```java
package m5_core_java_programming.day_19_part_1;

/*
   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.
*/

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;

public class Assignment_2 {

   public static long FindMinCost(int [][] graph){
       int n = graph.length;
       int start = 0;
```

```
        long[][] dp = new long[1 << n][n];
        HashMap<Integer,HashMap<Integer,Integer>> tracer = new HashMap<>();

        for (int i = 0; i < (1 << n); i++) {
            for (int j = 0; j < n; j++) {
                dp[i][j] = Long.MAX_VALUE;
                tracer.put(1<<i,new HashMap<>());
                tracer.get(1<<i).put(j,n+1);

            }
        }


        dp[1 << start][start] = 0;


        for (int mask = 0; mask < (1 << n); mask++) {
            for (int u = 0; u < n; u++) {
                if ((mask & (1 << u)) != 0) {
                    for (int v = 0; v < n; v++) {
                        if ((mask & (1 << v)) == 0) {
                            int nextMask = mask | (1 << v);
                            if (dp[mask][u] != Long.MAX_VALUE && dp[mask][u] +
graph[u][v] < dp[nextMask][v]) {
                                dp[nextMask][v] = dp[mask][u] + graph[u][v];
                                if(!tracer.containsKey(nextMask)){
                                    tracer.put(nextMask,new HashMap<>());
                                }
                                tracer.get(nextMask).put(v,u);
                            }
                        }
                    }
                }
            }
        }


        long result = Long.MAX_VALUE;
        int end = n+1;
        for (int i = 0; i < n; i++) {
            if (dp[(1 << n) - 1][i] != Long.MAX_VALUE && dp[(1 << n) - 1][i] +
graph[i][start] < result) {
                result = dp[(1 << n) - 1][i] + graph[i][start];
                end = i;
            }
        }

        List<Integer> path = new ArrayList<>();
        int mask = (1 << n) - 1;
```

```
        int city = end;
        while (true) {
            path.add(city);
            int curr = city;
            city = tracer.getOrDefault(mask,new
HashMap<>()).getOrDefault(curr,n+1);
            mask ^= (1 << curr);
            if(city == n+1){
                break;
            }
        }

        System.out.println(path);
        return result;
    }

    public static void main(String[] args) {
        int [][] graph =  {
                {0, 29, 20, 21},
                {29, 0, 15, 17},
                {20, 15, 0, 28},
                {21, 17, 28, 0}
        };
        System.out.println(FindMinCost(graph));
    }
}
```

Output

```
C:\Users\coolr\.jdks\openjdk-22.0.1\bin\java.exe "-javaagent:C:\Program Files\JetBrains\In
[2, 1, 3, 0]
73

Process finished with exit code 0
```

---

3. **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 m5_core_java_programming.day_19_part_1;

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

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

class Job {
   int id;
   int deadLine;
   int profit;

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

   @Override
   public String toString() {
       return "Job{" +
               "id=" + id +
               ", deadLine=" + deadLine +
               ", profit=" + profit +
               '}';
   }
}

public class Assignment_3 {

   private static LinkedList<Job> JobSequencing(List<Job> jobs) {
       boolean[] slots = new boolean[5];

       PriorityQueue<Job> pq = new PriorityQueue<>((Job a, Job b) -> b.profit -
a.profit);
       LinkedList<Job> timeline = new LinkedList<>();
       int totalProfit = 0;

       for (int i = jobs.size() - 1; i >= 0; i--) {
           int slot = (i == 0 ? jobs.getFirst().deadLine : jobs.get(i).deadLine
- jobs.get(i - 1).deadLine);
           pq.offer(jobs.get(i));

           while (slot > 0 && !pq.isEmpty()) {
               Job temp = pq.poll();
               slot--;
               timeline.add(temp);
               totalProfit += temp.profit;
           }
       }

       Collections.sort(timeline, ((Job a, Job b) -> a.deadLine - b.deadLine));
       System.out.println("Total Profit : " + totalProfit);
```

```
        return timeline;
    }

    public static void main(String[] args) {

        LinkedList<Job> jobs = new LinkedList<>();

        jobs.add(new Job(1, 3, 35));
        jobs.add(new Job(2, 4, 30));
        jobs.add(new Job(3, 4, 25));
        jobs.add(new Job(4, 2, 20));
        jobs.add(new Job(5, 3, 15));
        jobs.add(new Job(6, 1, 12));

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

        LinkedList<Job> timeline = JobSequencing(jobs);

        System.out.println("Scheduled jobs are : ");
        for (Job job : timeline) {
            System.out.println(job);
        }


    }
}
```

Output

```
C:\Users\coolr\.jdks\openjdk-22.0.1\bin\java.exe "-javaagent:C:\Progra
Total Profit : 110
Scheduled jobs are :
Job{id=4, deadLine=2, profit=20}
Job{id=1, deadLine=3, profit=35}
Job{id=2, deadLine=4, profit=30}
Job{id=3, deadLine=4, profit=25}

Process finished with exit code 0
```

Tools Used :
IntelliJ IDE
java version "1.8.0_411"
Java(TM) SE Runtime Environment (build 1.8.0_411-b09)
Java HotSpot(TM) Client VM (build 25.411-b09, mixed mode, sharing)