Assignment – 16

Maddula Rupa Sri Manohar
maddularupasrimanohar2001@gmail.com

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.

Program:

```
public class TowerOfHanoiProblem {

	public static void main(String[] args) {
		int n=3;
		towerOFHenoi(n,'A','B','C');
	}

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

}
```

Output:

```
Move disk 1 form A to C
Move disk 2 from A to B
Move disk 1 form C to B
Move disk 3 from A to C
Move disk 1 form B to A
Move disk 2 from B to C
Move disk 1 form 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.

Program:

```
import java.util.Arrays;

public class TravelingSalesmanProblem {



	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("Minimum cost of visited all cities:
					"+findMinCost(graph));
	}
	private static int findMinCost(int[][] graph) {
		int n=graph.length;
		int[][] dp=new int[1<<n][n];
		for(int[] d:dp) {
			Arrays.fill(d,Integer.MAX_VALUE/2);
		}
		dp[1][0]=0;
		for(int m=1;m<(1<<n);m+=2) {
			for(int i=0;i<n;i++) {
				if((m&(1<<i))!=0){
					for(int j=0;j<n;j++) {
						if((m&(1<<j))==0) {

	dp[m|(1<<j)][j]=Math.min(dp[m|(1<<j)][j],
				dp[m][i]+graph[i][j]);
					}
				}
			}
		}
		int mincost=Integer.MAX_VALUE;
		for(int i=1;i<n;i++) {
			mincost=Math.min(mincost, dp[(1<<n)-
				1][i]+graph[i][0]);
		}
		return mincost;
	}
}
```

Output:

```
Minimum cost of visited all cities: 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.

Program:

```
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;

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

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

public class JobSequencingProblem {

	public static void main(String[] args) {

		List<Job> jobs = new ArrayList<Job>();
		jobs.add(new Job(1, 2, 100));
		jobs.add(new Job(2, 1, 50));
		jobs.add(new Job(3, 2, 10));
		jobs.add(new Job(4, 1, 20));
		jobs.add(new Job(5, 3, 30));

		List<Job> sequence = jobSequencing(jobs);
		System.out.println("Maximum profit sequence of jobs: ");
		for (Job job : sequence) {
			System.out.println("Jod id: " + job.id + ",
                                Deadline: " + job.deadline +
                               ", Profit: " + job.profit);
		}
	}

	private static List<Job> jobSequencing(List<Job> jobs) {
		Collections.sort(jobs, (a, b) -> b.profit - a.profit);
		int maxdeadline = 0;
		for (Job job : jobs) {
			maxdeadline = Math.max(maxdeadline, job.deadline);
		}
		boolean[] slot = new boolean[maxdeadline];
		List<Job> sequence = new ArrayList<Job>();
```

```
            for (Job job : jobs) {
                  for (int i = Math.min(maxdeadline, job.deadline) - 1
                                          ; i >= 0; i--) {
                        if (!slot[i]) {
                              slot[i] = true;
                              sequence.add(job);
                              break;
                        }
                  }
            }
            return sequence;
      }

}
```

Output:

```
Maximum profit sequence of jobs:
Jod id: 1, Deadline: 2, Profit: 100
Jod id: 2, Deadline: 1, Profit: 50
Jod id: 5, Deadline: 3, Profit: 30
```