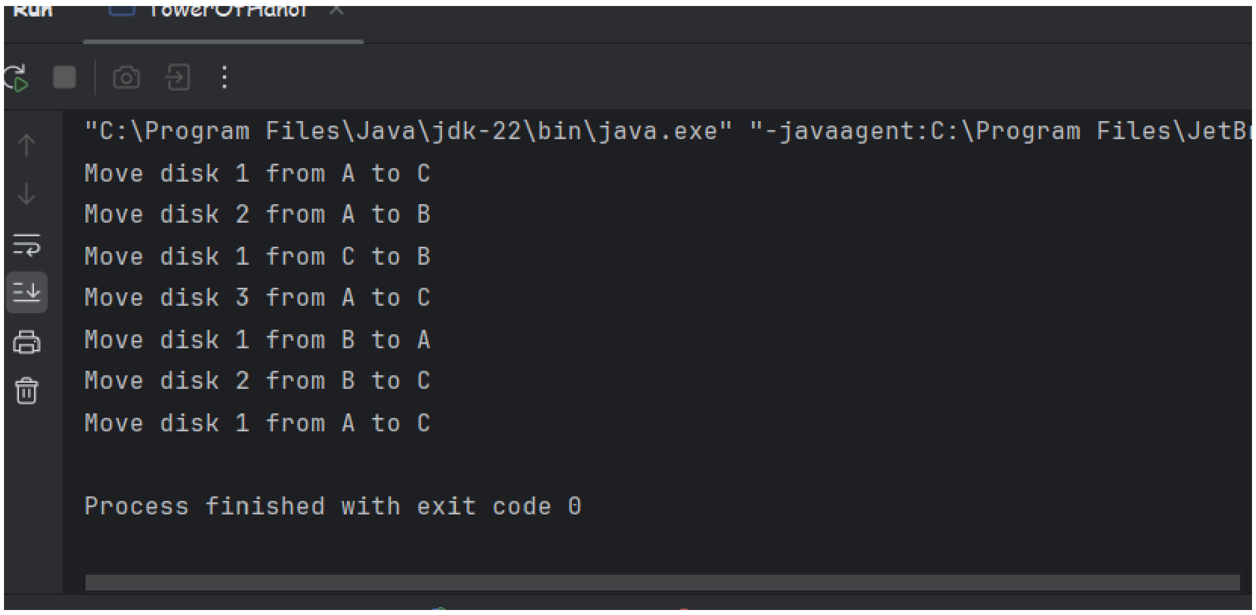
Task 1: Tower of Hanoi Solver

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:

```
package Day13_14;

public class TowerOfHanoi {

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

        solveTowerOfHanoi(n - 1, source, destination, auxiliary);

        System.out.println("Move disk " + n + " from " + source + " to " + destination);

        solveTowerOfHanoi(n - 1, auxiliary, source, destination);
    }

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

Output:



```
"C:\Program Files\Java\jdk-22\bin\java.exe" "-javaagent:C:\Program Files\JetB
Move disk 1 from A to C
Move disk 2 from A to B
Move disk 1 from C to B
Move disk 3 from A to C
Move disk 1 from B to A
Move disk 2 from B to C
Move disk 1 from A to C

Process finished with exit code 0
```

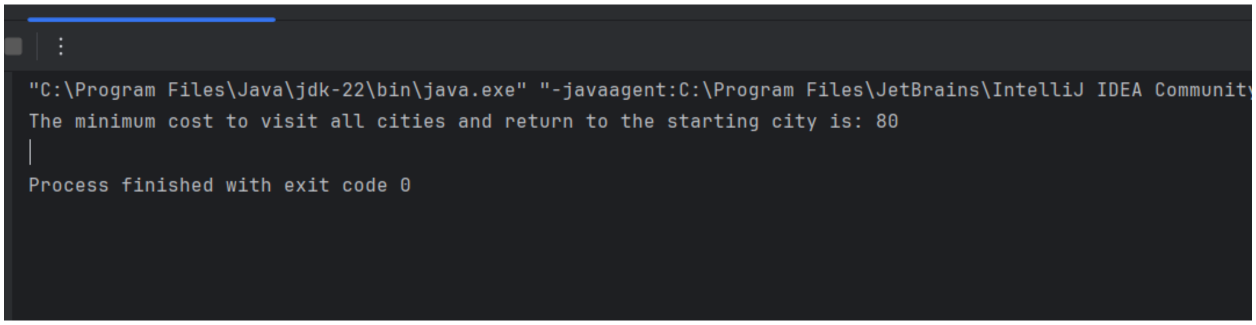
Task 2: Traveling Salesman Problem

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.

```
package Day13_14;

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 city, int mask, int[][] graph, int[][] dp, int VISITED_ALL) {
    if (mask == VISITED_ALL) {
            return graph[city][0];
        }

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

        int minCost = Integer.MAX_VALUE;

        for (int nextCity = 0; nextCity < graph.length; nextCity++) {
            if ((mask & (1 << nextCity)) == 0) { // if nextCity is not visited
                int newCost = graph[city][nextCity] + tsp(nextCity, mask | (1 << nextCity), graph, dp,
VISITED_ALL);
                minCost = Math.min(minCost, newCost);
            }
        }

        dp[city][mask] = minCost;
        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 = FindMinCost(graph);
        System.out.println("The minimum cost to visit all cities and return to the starting city is: " + result);
    }
}
```

Output:



```
"C:\Program Files\Java\jdk-22\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community
The minimum cost to visit all cities and return to the starting city is: 80

Process finished with exit code 0
```

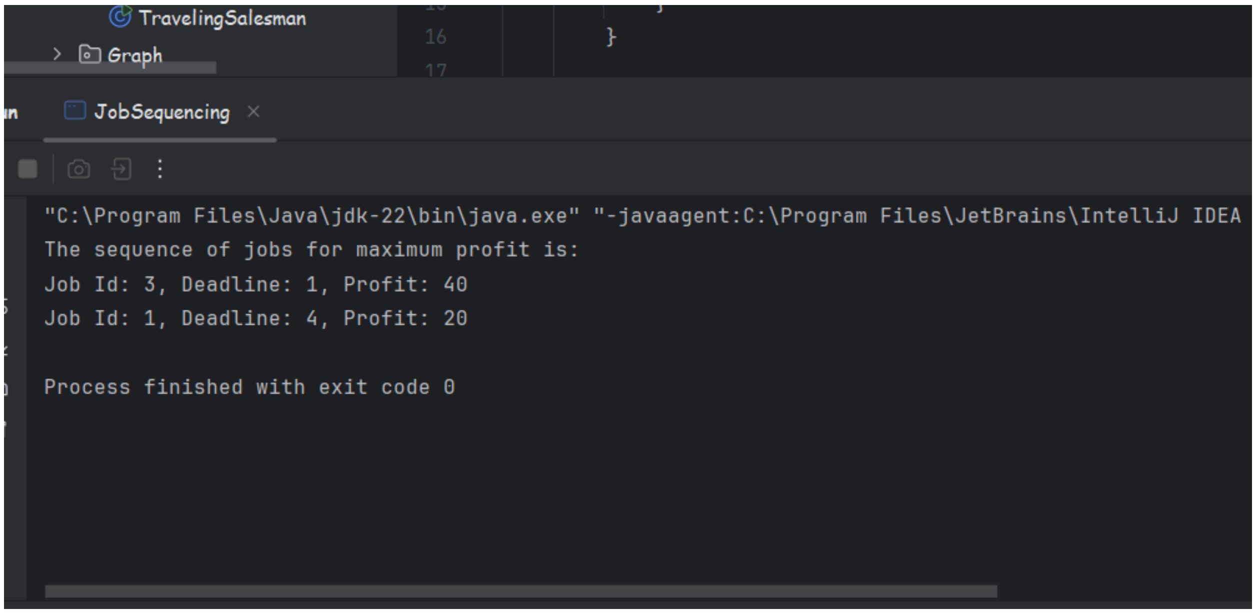
Task 3: Job Sequencing Problem

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 Day13_14;

import java.util.*;

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[] slots = new boolean[maxDeadline];

    for (Job job : jobs) {

      for (int j = Math.min(maxDeadline, job.deadline) - 1; j >= 0; j--) {
        if (!slots[j]) {
          slots[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("The sequence of jobs for maximum profit is:");
    for (Job job : jobSequence) {
        System.out.println("Job Id: " + job.id + ", Deadline: " + job.deadline + ", Profit: " + job.profit);
    }
}
}
```

Output:



```
"C:\Program Files\Java\jdk-22\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA
The sequence of jobs for maximum profit is:
Job Id: 3, Deadline: 1, Profit: 40
Job Id: 1, Deadline: 4, Profit: 20

Process finished with exit code 0
```