Problem Statement:

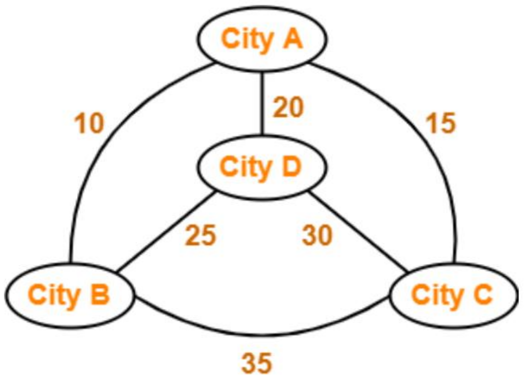
Different exact and approximation algorithms for Travelling-Sales-Person Problem.

Introduction:

Travelling Sales Person Problem

The travelling salesman problem is a graph computational problem where the salesman needs to visit all cities (represented using nodes in a graph) in a list just once and the distances (represented using edges in the graph) between all these cities are known. The solution that is needed to be found for this problem is the shortest possible route in which the salesman visits all the cities and returns to the origin city.

If you look at the graph below, considering that the salesman starts from the city 'A', they need to travel through all the remaining cities B, C, D and get back to 'A' while making sure that the cost taken is minimum.



If salesman starting city is A, then a TSP tour in the graph is-

A → B → D → C → A

Cost of the tour

= 10 + 25 + 30 + 15

Software Requirement Specification:

- Operating System: Windows
- Language: Java
- IDE: VS Code

Hardware Requirement Specification:

- Processor: Intel (i5, i7, i9)
- HDD: Min 32 GB; Recommended 64 GB or more
- RAM: Min 1 GB; Recommended 4 GB or above

About Project:

Algorithmic Exploration: The project explores two different types of algorithms to solve the TSP:

- Exact Algorithms:
  1. Optimality: Exact algorithms aim to find the optimal solution, meaning they guarantee the shortest possible tour.
  2. Computation: These algorithms explore all possible solutions or use advanced mathematical techniques like dynamic programming, branch and bound, or integer linear programming.
  3. Efficiency: Exact algorithms can be very time-consuming and computationally intensive, especially for a large number of cities. They are not practical for TSP instances with a large number of cities.
  4. Guaranteed Solution: The primary advantage of exact algorithms is that they provide a provably optimal solution if given enough time to complete.

  Examples of exact algorithms include the Brute Force.

- Approximation Algorithms:
    1. Optimality: Approximation algorithms provide a suboptimal solution, meaning they find a tour that is shorter or equal to a certain factor times the optimal tour length. The factor is often referred to as the approximation ratio.
    2. Computation: These algorithms use heuristics and greedy strategies to quickly find a solution without exploring all possibilities.
    3. Efficiency: Approximation algorithms are usually much faster and are suitable for large TSP instances.
    4. Performance: The quality of the solution depends on the specific approximation algorithm used. Some approximation algorithms can provide very good solutions in practice, even if they don't guarantee optimality.

Examples of approximation algorithms include the Nearest Neighbor algorithm.

Algorithms:

1. Exact TSP (**Brute-Force**) Algorithm:

Objective: The exact TSP algorithm aims to find the optimal (shortest) route that visits all cities exactly once and returns to the starting city. It does this by exhaustively searching through all possible permutations of city visit orders.

Algorithm Overview: It starts by generating all possible permutations of city visit orders. This is done using a recursive function (backtrack). For each permutation, it calculates the total distance (length of the route) by summing the distances between consecutive cities. It keeps track of the shortest route found and its corresponding distance. The algorithm returns the shortest route and its total distance.

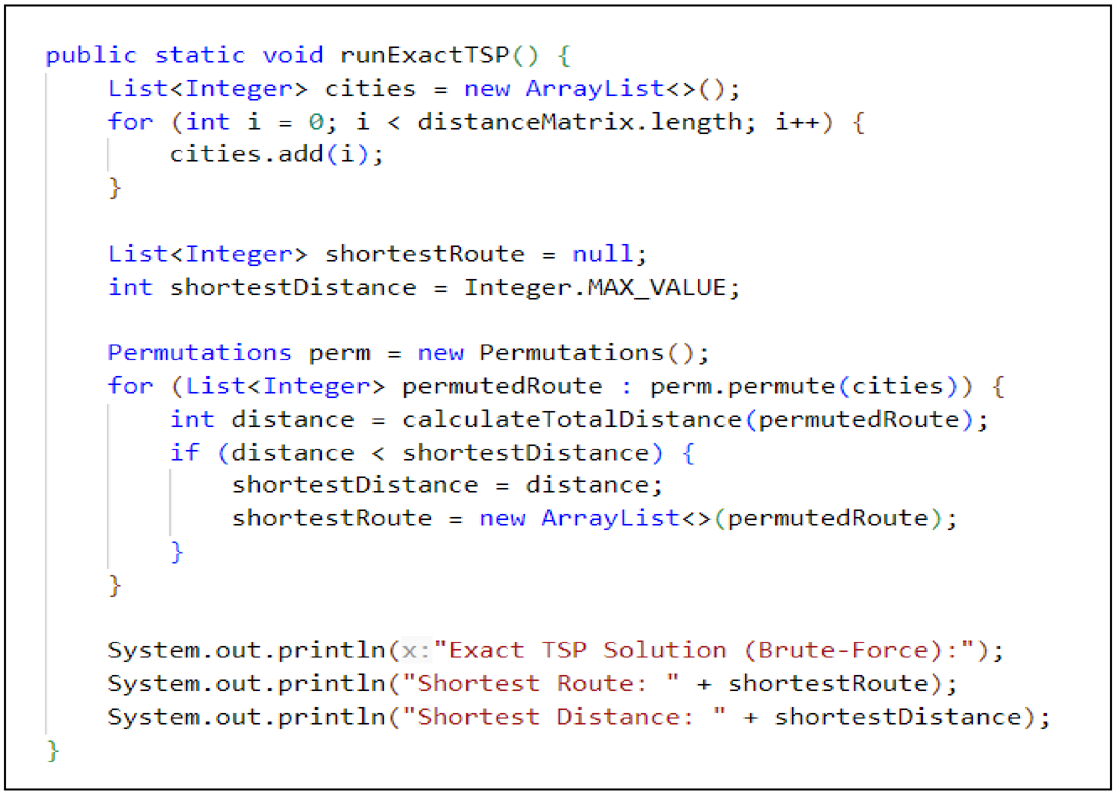
Pros:

Guarantees an optimal solution since it explores all possibilities.

Suitable for small to moderate-sized instances of the TSP.

Cons:

Computationally intensive and slow for large TSP instances due to factorial growth in the number of permutations.



```java
public static void runExactTSP() {
    List<Integer> cities = new ArrayList<>();
    for (int i = 0; i < distanceMatrix.length; i++) {
        cities.add(i);
    }

    List<Integer> shortestRoute = null;
    int shortestDistance = Integer.MAX_VALUE;

    Permutations perm = new Permutations();
    for (List<Integer> permutedRoute : perm.permute(cities)) {
        int distance = calculateTotalDistance(permutedRoute);
        if (distance < shortestDistance) {
            shortestDistance = distance;
            shortestRoute = new ArrayList<>(permutedRoute);
        }
    }

    System.out.println(x:"Exact TSP Solution (Brute-Force):");
    System.out.println("Shortest Route: " + shortestRoute);
    System.out.println("Shortest Distance: " + shortestDistance);
}
```

2. Approximation TSP (**Nearest Neighbor**) Algorithm:

Objective: The approximation TSP (Nearest Neighbor) algorithm provides a suboptimal (approximate) solution by quickly constructing a tour that is close to the optimal route. It is computationally more efficient than the exact algorithm.

Algorithm Overview: It starts from an initial city and iteratively selects the nearest unvisited city as the next destination. It continues this process until all cities are visited, and the tour is complete by returning to the starting city.

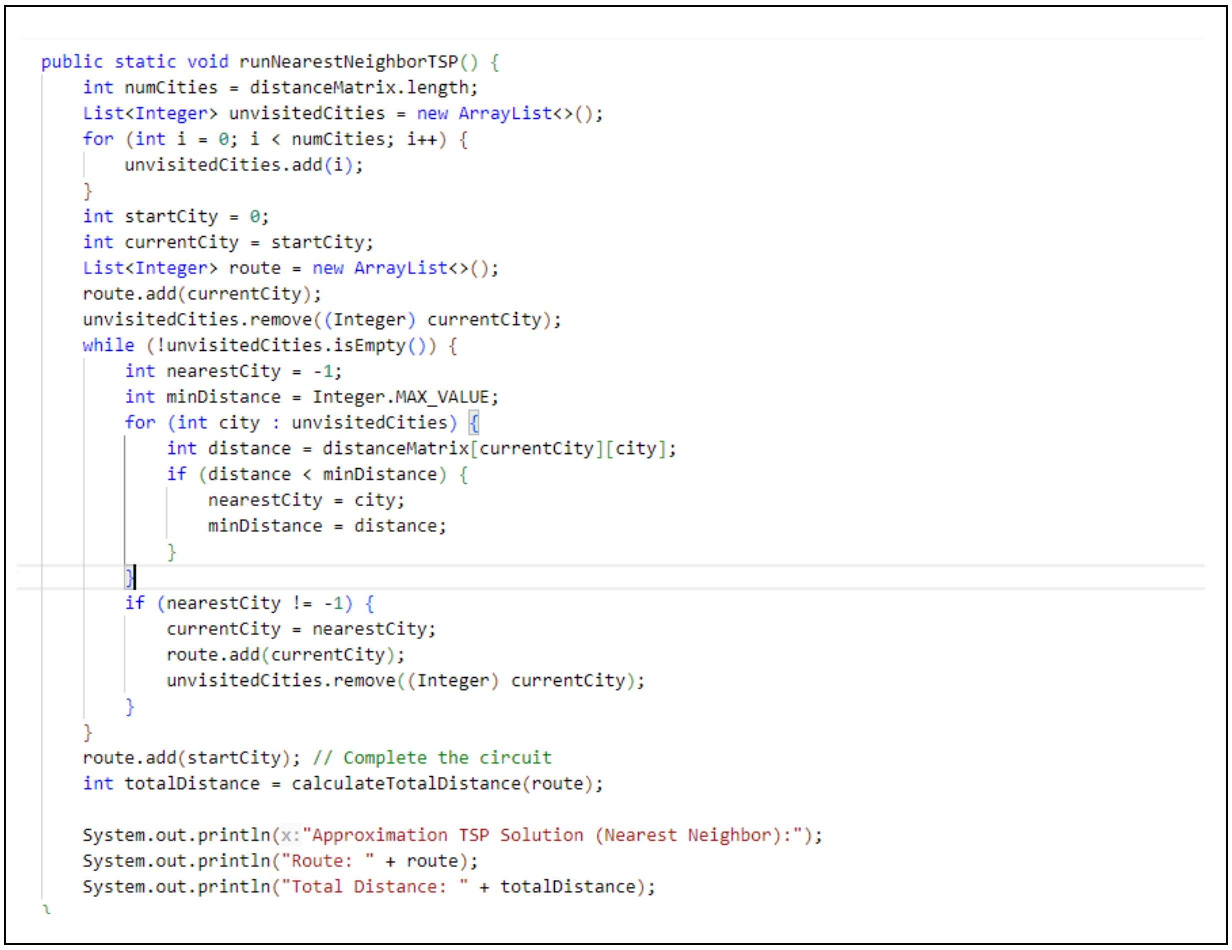
Pros:

Much faster than exact algorithms, making it suitable for larger TSP instances.

Often provides reasonably good solutions in practice.

Cons:

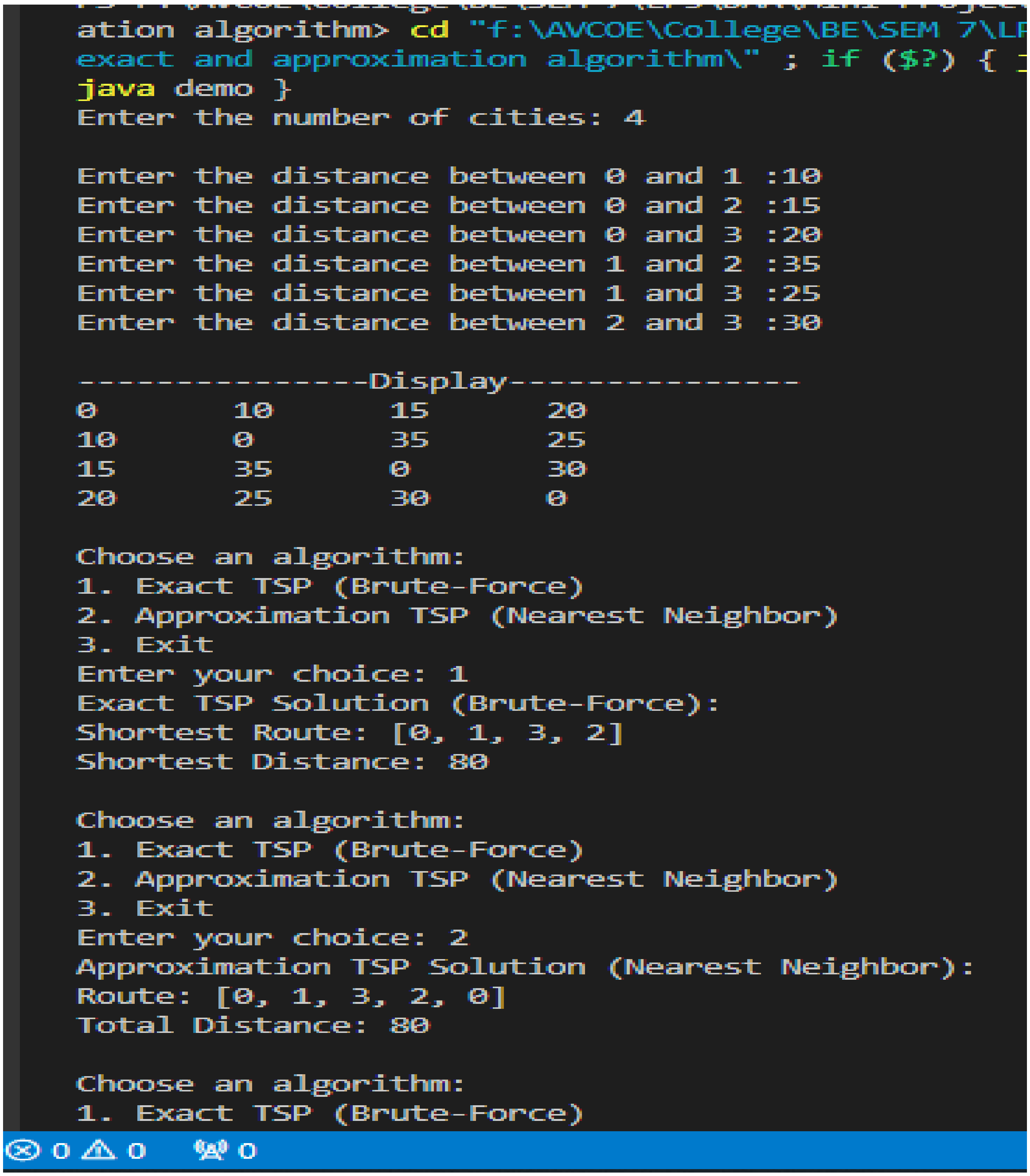
Not guaranteed to find the optimal solution, and the quality of the solution may vary depending on the starting city and the distribution of cities.



```java
public static void runNearestNeighborTSP() {
    int numCities = distanceMatrix.length;
    List<Integer> unvisitedCities = new ArrayList<>();
    for (int i = 0; i < numCities; i++) {
        unvisitedCities.add(i);
    }
    int startCity = 0;
    int currentCity = startCity;
    List<Integer> route = new ArrayList<>();
    route.add(currentCity);
    unvisitedCities.remove((Integer) currentCity);
    while (!unvisitedCities.isEmpty()) {
        int nearestCity = -1;
        int minDistance = Integer.MAX_VALUE;
        for (int city : unvisitedCities) {
            int distance = distanceMatrix[currentCity][city];
            if (distance < minDistance) {
                nearestCity = city;
                minDistance = distance;
            }
        }
        if (nearestCity != -1) {
            currentCity = nearestCity;
            route.add(currentCity);
            unvisitedCities.remove((Integer) currentCity);
        }
    }
    route.add(startCity); // Complete the circuit
    int totalDistance = calculateTotalDistance(route);

    System.out.println(x:"Approximation TSP Solution (Nearest Neighbor):");
    System.out.println("Route: " + route);
    System.out.println("Total Distance: " + totalDistance);
}
```

Output:



```
ation algorithm> cd "f:\AVCOE\College\BE\SEM 7\LP
exact and approximation algorithm\" ; if ($?) { 
java demo }
Enter the number of cities: 4

Enter the distance between 0 and 1 :10
Enter the distance between 0 and 2 :15
Enter the distance between 0 and 3 :20
Enter the distance between 1 and 2 :35
Enter the distance between 1 and 3 :25
Enter the distance between 2 and 3 :30

---------------Display---------------
0       10      15      20
10      0       35      25
15      35      0       30
20      25      30      0

Choose an algorithm:
1. Exact TSP (Brute-Force)
2. Approximation TSP (Nearest Neighbor)
3. Exit
Enter your choice: 1
Exact TSP Solution (Brute-Force):
Shortest Route: [0, 1, 3, 2]
Shortest Distance: 80

Choose an algorithm:
1. Exact TSP (Brute-Force)
2. Approximation TSP (Nearest Neighbor)
3. Exit
Enter your choice: 2
Approximation TSP Solution (Nearest Neighbor):
Route: [0, 1, 3, 2, 0]
Total Distance: 80

Choose an algorithm:
1. Exact TSP (Brute-Force)
```

Time Complexity:

- **Exact TSP (Brute-Force) Algorithm:**

The exact TSP algorithm explores all possible permutations of city visit orders, which results in a factorial time complexity. Specifically, the time complexity is O(n!) for n cities.

Explanation: For each of the n cities, there are (n-1)! possible permutations of visiting the remaining cities. This leads to an exponentially growing number of permutations as n increases. As a result, the algorithm becomes impractical for large values of n.

- **Approximation TSP (Nearest Neighbor) Algorithm:**

The approximation TSP algorithm has a time complexity of O(n^2), where n is the number of cities.

Explanation: The algorithm involves iterating through all cities to find the nearest neighbor for each city. This requires comparing distances between the current city and all unvisited cities. As there are n cities and n-1 unvisited cities for each, the total number of distance comparisons is proportional to n * (n-1), resulting in a quadratic time complexity.

Conclusion:

Hence, we successfully implemented Different exact and approximation algorithms for Travelling-Sales-Person Problem.