Algorithm Design-II (CSE 4131)
TERM PROJECT REPORT
(March'2023-July'2023)
On

Solving Travelling Salesman Problem Using Greedy Algorithm And Brute Force Algorithm

Submitted by
Isita Ray
Registration No.: 2141018145
B.Tech. 4th Semester CS-IT(D)

Department of Computer Science and Engineering
Institute of Techinical Education and Research
Siksha 'O' Anusandhan Deemed To Be University
Bhubaneswar, Odisha-751030

DECLARATION

I, Isita Ray, bearing registration number 2141018145 do hereby declare that this term project entitled "**Solving Travelling Salesman Problem Using Greedy Algorithm And Brute-Force Algorithm**" is an original project work done by me and has not been previously submitted to any university or research institution or department for the award of any degree or diploma or any other assessment to the best of my knowledge.

Isita Ray

Regd No:2141018145

CERTIFICATE

This is to certify that the thesis entitled “Solving Travelling Salesman Problem Using Greedy Algorithm And Brute-Force Algorithm ”submitted by Isita Ray, bearing registration number 2141018070 of B.Tech. 4th Semester Comp. Sc. and Engineering. ITER, SOADU is absolutely based upon her own work under my guidance and supervision.
The term project has reached the standard fulfilling the requirement of the course Algorithm Design 2 (CSE4131). Any help or source of information which has been available in this connection is duly acknowledged.

Paresh Baidya
Assistant Professor
Department of Comp.Sc. and IT.
ITER, Bhubaneswar 751030,
Odisha, India

Prof.(Dr.)Ajit Nayak
Professor and Head
Department Comp.Sc. and IT.
ITER, Bhubaneswar 751030,
Odisha, India

Abstract

Greedy algorithm is an algorithm that will solve problem by choosing the best choice/optimum solution at that time, without considering the consequences that will affect it later, thus the solution won’t be always the optimal global solution, while Brute force algorithm is an algorithm that is using straightforward method to solve a problem. There are several algorithms that are naturally a Greedy algorithm that can be used to solve this problem,which are the Prim’s Minimum Spanning Tree, Dijkstra shortest path, Huffman coding, and Kruskal Minimum Spanning Tree, while for brute force can be used in almost every problems.
In this paper, we will use brute force and greedy algorithm to solve Travelling Salesman Problem. Travelling Salesman Problem (TSP) is one of the NP-Complete problem that will search for optimal solution when a salesman should precisely visit every city once and then back again to the initial city. The main aim of this problem is to find the shortest possible route for the salesman to visit. The presented paper will use brute force algorithm and greedy algorithm to solve travelling salesman problem and examine those two algorithms’ time complexity, also comparing them. The result is greedy algorithm is more efficient than brute force algorithm, but the consequences is that greedy algorithm won’t always give the optimal
solution.

CONTENTS

Introduction

In this final project, we choose "Solving Travelling Salesman Problem using Greedy Algorithm and Brute Force Algorithm".In this paper, we will discuss about Travelling Salesman Problem (TSP) using methods. The Travelling Salesman Problem or the TSP is a representative of a large class of problems known as combinatorial optimization problems (Mohammad Reza Bonyadi, 2008). In the ordinary form, it is represented with a salesman and cities. Salesman has to visit each cities, which is built on one another and has different distances, one by one, starting from his hometown and returning to his hometown again. The aim for this problem is to find the shortest possible route to minimize the length of the trip. Brute Force is a type of algorithm that tries a large number of pattern to solve a problem (Spacey, 2016). This means that in TSP, the algorithm will try all possible routes and compute their distance to find the minimum distance. Therefore, this algorithm will always give optimal solution.Greedy algorithm is an algorithm that will solve problem by choosing the best choice/optimum solution at that time, without considering the consequences that will affect it later. In many problems, a greedy strategy does not in general produce an optimal solution, but nonetheless a greedy heuristic may yield locally optimal solutions that approximate a global optimal solution in a reasonable time (Ejim, 2016).
In this paper, we will try to solve TSP with Brute Force and Greedy Algorithm, and we will compare them and see which one is better for solving this problem.

Overview

Here's an overview of how the Traveling Salesman Problem (TSP) can be solved using the Greedy and Brute Force algorithms:

Greedy Algorithm:

Start with an arbitrary city as the starting point.
Select the nearest unvisited city from the current city as the next destination.
Move to the selected city and mark it as visited.
Repeat steps 2 and 3 until all cities have been visited.
Return to the starting city to complete the cycle.
The sequence of visited cities represents the solution to the TSP using the Greedy algorithm.
Brute Force Algorithm:

Generate all possible permutations of the cities.
Calculate the cost of each permutation by summing the distances between consecutive cities.
Identify the permutation with the minimum cost.
The sequence of cities in the permutation with the minimum cost represents the optimal solution to the TSP using the Brute Force algorithm.
Comparison:

Greedy Algorithm: The Greedy algorithm selects the nearest unvisited city at each step, resulting in a suboptimal solution. It is faster than the Brute Force algorithm but does not guarantee the optimal solution.
Brute Force Algorithm: The Brute Force algorithm exhaustively checks all possible permutations, ensuring an optimal solution. However, it becomes computationally expensive as the number of cities increases.
It's important to note that for large instances of the TSP, the Brute Force algorithm becomes impractical due to its exponential time complexity. Therefore, the Greedy algorithm or other more efficient heuristic algorithms are often used to find approximate solutions.

Problem Description(Real-Life Scenarario)

The Traveling Salesman Problem (TSP) has various real-life applications in different fields. Let's consider a scenario in the context of a delivery service company:

Problem Description:

A delivery service company has a fleet of vehicles and a list of cities that need to be visited to deliver packages. The company wants to optimize the routes for its vehicles to minimize the distance traveled and the time required for delivery.

Greedy Algorithm in a Delivery Service Scenario: The company can use the greedy algorithm to determine the delivery routes for its vehicles in the following way:

1. Start with an initial city as the starting point for each vehicle.
2. For each vehicle, choose the nearest unvisited city from its current location as the next destination.
3. Update the current location of the vehicle to the selected city and mark it as visited.
4. Repeat step 2 and 3 until all cities have been visited for each vehicle.
5. Each vehicle returns to the starting city to complete the delivery route.

This greedy approach allows the company to quickly assign routes to its vehicles based on proximity, ensuring that each vehicle visits the nearest available city for package delivery. However, it may not provide the most optimal solution, as it does not consider the global picture and may lead to suboptimal routes.

Brute Force Algorithm in a Delivery Service Scenario: In a real-life scenario, the brute force algorithm is generally not feasible for a large number of cities due to the exponential time complexity. However, for smaller instances of the TSP, it can still be used to find the optimal solution. The brute force approach would involve:

1. Generating all possible permutations of the cities for each vehicle's route.

2. Calculating the total distance/time for each permutation.
3. Comparing the distances/times of all permutations to find the one with the minimum value.
4. Assigning the corresponding optimal permutation as the route for each vehicle.

While the brute force algorithm guarantees the optimal solution, it becomes computationally expensive and time-consuming as the number of cities increases. Therefore, it is not a practical choice for larger real-life scenarios in the delivery service industry.

In practice, various heuristic algorithms, such as genetic algorithms, ant colony optimization, or simulated annealing, are often employed to find approximate solutions to the TSP efficiently and effectively in real-life scenarios.

Problem Statement:

The problem statement of the Traveling Salesman Problem (TSP) is as follows:

Given a set of cities and the distances between each pair of cities, the objective is to find the shortest possible route that visits each city exactly once and returns to the starting city.

The problem can be represented as an undirected weighted graph, where the cities are nodes and the distances between them are the weights of the edges. The goal is to find a Hamiltonian cycle (a cycle that visits each node exactly once) with the minimum total weight.

Greedy Algorithm: The greedy algorithm starts with an arbitrary city and repeatedly selects the nearest unvisited city as the next destination. It builds a tour by gradually visiting all cities. The algorithm prioritizes local optimization at each step by choosing the shortest available edge. However, it may not produce the optimal solution in some cases.

Brute Force Algorithm: The brute force algorithm exhaustively generates all possible permutations of the cities and evaluates the cost of each permutation. It systematically checks every possible Hamiltonian cycle to find the one with the minimum total weight. This approach guarantees the optimal solution but becomes impractical for large problem instances due to the exponential number of permutations to consider.

Both algorithms aim to solve the TSP but take different approaches in terms of their efficiency and optimality guarantees.

Mathematic Formulation:

The Traveling Salesman Problem (TSP) can be mathematically formulated as an optimization problem. Let's consider a real-life scenario of a delivery service company and formulate the TSP using mathematical notation.

Problem Formulation:

Given:

- Set of cities: C = {c1, c2, ..., cn}
- Distance or cost between each pair of cities: D(c1, c2), D(c1, c3), ..., D(cn-1, cn)

Objective: Minimize the total distance or cost of the route for delivering packages to all cities and returning to the starting city.

Decision Variables:

- Binary variable x_ij:
- x_ij = 1 if the route includes the edge (city i, city j)
- x_ij = 0 otherwise

Mathematical Formulation:

Minimize: Z = ∑∑ D(c_i, c_j) * x_ij (for all i, j)

Subject to:

1. Each city must be visited exactly once: ∑ x_ij = 1 (for all i)
2. Each city must have exactly one outgoing edge: ∑ x_ij = 1 (for all j)
3. Subtour elimination constraints (to prevent cycles that do not visit all cities except the starting city): U_i - U_j + n * x_ij <= n - 1 (for all i ≠ j, i ≠ 1, j ≠ 1)
4. Binary constraints: x_ij ∈ {0, 1} (for all i, j)

In the above formulation, Z represents the objective function, which is the total distance or cost of the route. The first constraint ensures that each city is visited exactly once. The second constraint ensures that each city has exactly one outgoing edge. The third constraint eliminates subtours to ensure that the route covers all cities except the starting city. The fourth constraint states that the decision variable x_ij is binary.

This mathematical formulation can be used for both the Greedy and Brute Force algorithms, although the specific implementation may vary. The Greedy algorithm focuses on selecting the nearest unvisited city at each step, while the Brute Force algorithm exhaustively checks all possible permutations to find the optimal solution.

Assumptions

When solving the Traveling Salesman Problem (TSP) using the Greedy and Brute Force algorithms in a real-life scenario, several assumptions are typically made. These assumptions help simplify the problem and provide a framework for finding solutions. Here are some common assumptions:

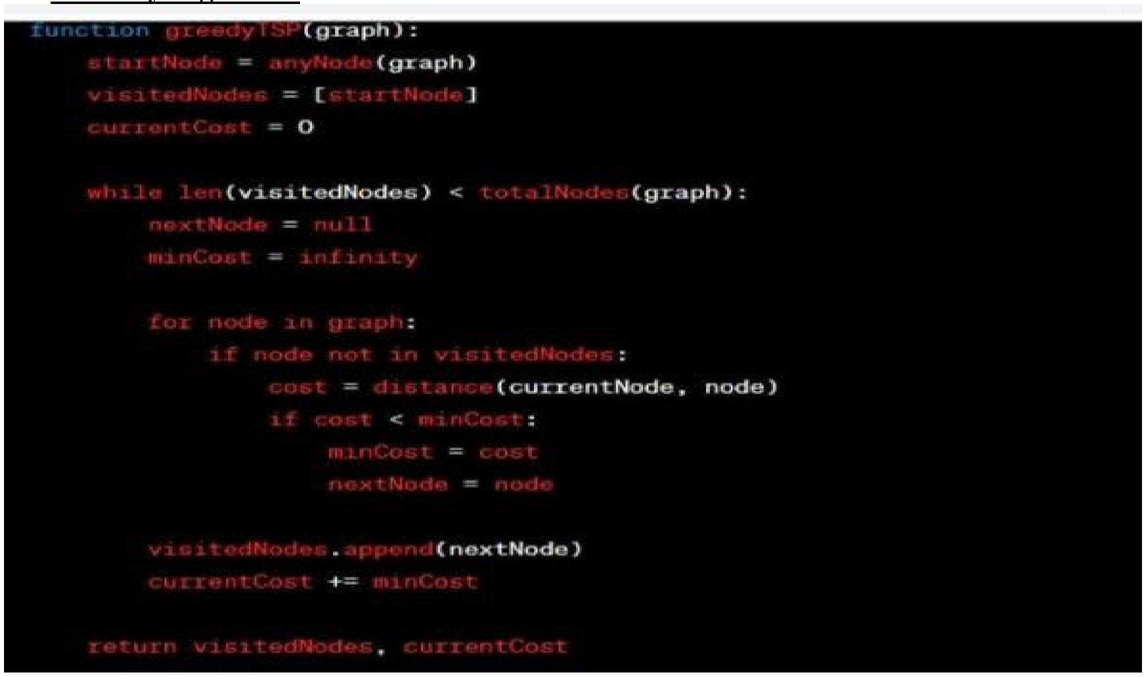
1. Complete Graph: It is assumed that the TSP graph is complete, meaning there is a direct connection (edge) between every pair of cities. This assumption allows for direct calculation of distances or costs between any two cities.
2. Symmetric Distances: In most cases, it is assumed that the distances or costs between cities are symmetric. This means that the distance from city A to city B is the same as the distance from city B to city A. It simplifies the calculation and ensures that the problem is well-defined.
3. Euclidean Distances: In certain scenarios, it is assumed that the distances between cities can be represented by Euclidean distances. This assumption is common when the cities are represented as points in a 2D or 3D space, and the distance between two cities is the straight-line distance between them.
4. Time-Invariant Distances: It is often assumed that the distances or costs between cities remain constant throughout the entire solution process. This assumption disregards factors such as traffic conditions or time-dependent variations in travel times.
5. Single Depot/Starting City: The TSP is typically solved with the assumption that there is a single starting city or depot from which the salesman begins the tour. The objective is to return to the starting city after visiting all other cities.
6. Deterministic Travel: The algorithms assume that the salesman will follow the determined route without deviation or encountering obstacles. There are no uncertainties or stochastic elements considered in the calculations.
7. Integer Distances/Costs: The algorithms assume that the distances or costs between cities are represented by integers or whole numbers. This assumption simplifies the formulation and computation of the problem.

It's important to note that these assumptions may not always hold in all real-life scenarios, and adjustments or modifications might be required to better reflect the specific characteristics of the problem at hand.

Designing Algorithm

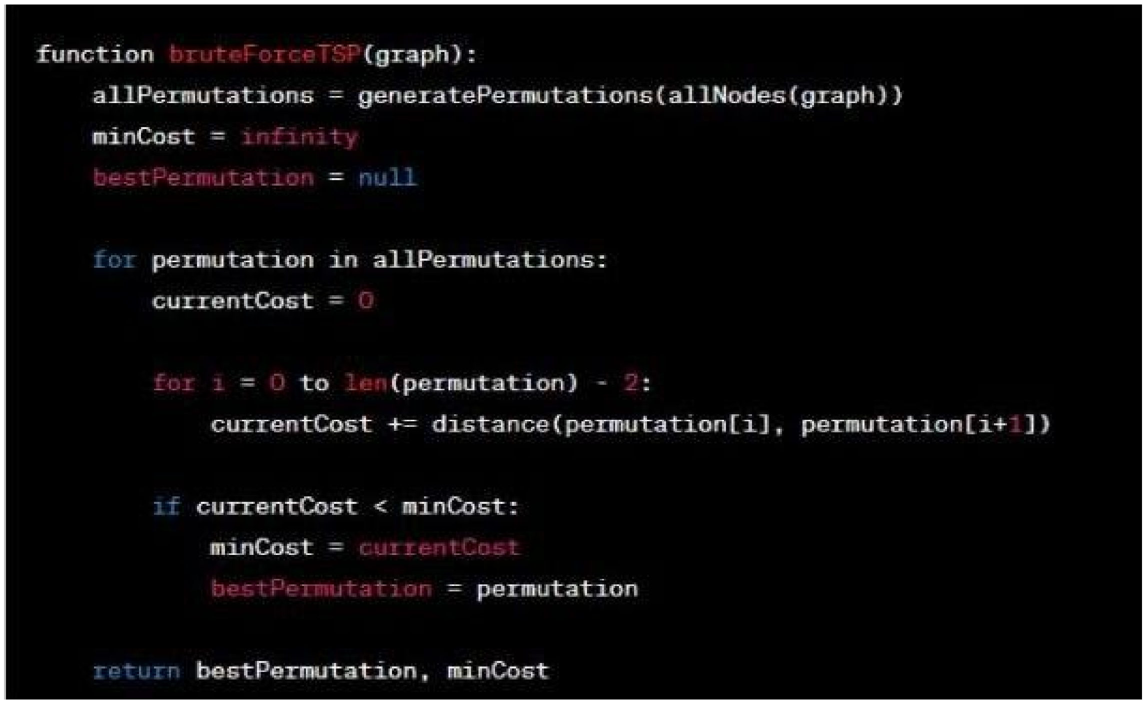
1.Pseudocode

Greedy Algorithm



```
function greedyTSP(graph):
    startNode = anyNode(graph)
    visitedNodes = [startNode]
    currentCost = 0

    while len(visitedNodes) < totalNodes(graph):
        nextNode = null
        minCost = infinity

        for node in graph:
            if node not in visitedNodes:
                cost = distance(currentNode, node)
                if cost < minCost:
                    minCost = cost
                    nextNode = node

        visitedNodes.append(nextNode)
        currentCost += minCost

    return visitedNodes, currentCost
```

Brute-Force Algorithm



```
function bruteForceTSP(graph):
    allPermutations = generatePermutations(allNodes(graph))
    minCost = infinity
    bestPermutation = null

    for permutation in allPermutations:
        currentCost = 0

        for i = 0 to len(permutation) - 2:
            currentCost += distance(permutation[i], permutation[i+1])

        if currentCost < minCost:
            minCost = currentCost
            bestPermutation = permutation

    return bestPermutation, minCost
```

2.Description of steps

A step-by-step description of how the Traveling Salesman Problem (TSP) can be solved using both the Greedy and Brute Force algorithms is stated here:

Greedy Algorithm:

1. Start by selecting an arbitrary node as the starting point.
2. Mark the starting node as visited and initialize the current cost to 0.
3. While there are unvisited nodes remaining:
- Find the nearest unvisited node from the current node.
- Calculate the cost of traveling from the current node to the selected node.
- Update the current cost by adding the calculated cost.
- Mark the selected node as visited.
- Set the selected node as the new current node.
4. Once all nodes have been visited, return to the starting node to complete the cycle.
5. The sequence of visited nodes and the final cost represent the solution to the TSP.

Brute Force Algorithm:

1. Generate all possible permutations of the nodes in the graph.
2. Initialize the minimum cost to infinity and set the best permutation to null.
3. For each permutation:
- Calculate the cost of traveling through the permutation by summing the distances between consecutive nodes.
- If the current cost is lower than the minimum cost, update the minimum cost and store the permutation as the best permutation.
4. After checking all permutations, the best permutation and the corresponding minimum cost represent the solution to the TSP.

It's worth noting that the brute force algorithm examines every possible permutation, making it extremely time-consuming for larger problem instances. On the other hand, the greedy algorithm provides a faster solution but may not always produce the optimal result.

3.Examples

Let's consider a simple example to demonstrate the solutions to the Traveling Salesman Problem (TSP) using both the Greedy and Brute Force algorithms.

Example: Suppose we have a graph with 4 cities (A, B, C, D) and their corresponding distances between each pair of cities as follows:

A -> B: 10 A -> C: 15 A -> D: 20 B -> C: 35 B -> D: 25 C -> D: 30

Greedy Algorithm Solution:

1. Start with an arbitrary city, let's say A.
2. Calculate the distances from city A to all other cities:
- A -> B: 10
- A -> C: 15
- A -> D: 20
3. Choose the city with the shortest distance from A, which is B (distance 10).
4. Move to city B.
5. Calculate the distances from city B to all remaining unvisited cities:
- B -> C: 35
- B -> D: 25
6. Choose the city with the shortest distance from B, which is D (distance 25).
7. Move to city D.
8. Calculate the distance from city D to the remaining unvisited city, which is C (distance 30).
9. Move to city C.
10. The remaining unvisited city is A (starting city).
11. Complete the cycle by returning to city A.
12. The sequence of visited cities is A -> B -> D -> C -> A.
13. The total cost of the tour is 10 + 25 + 30 + 15 = 80.

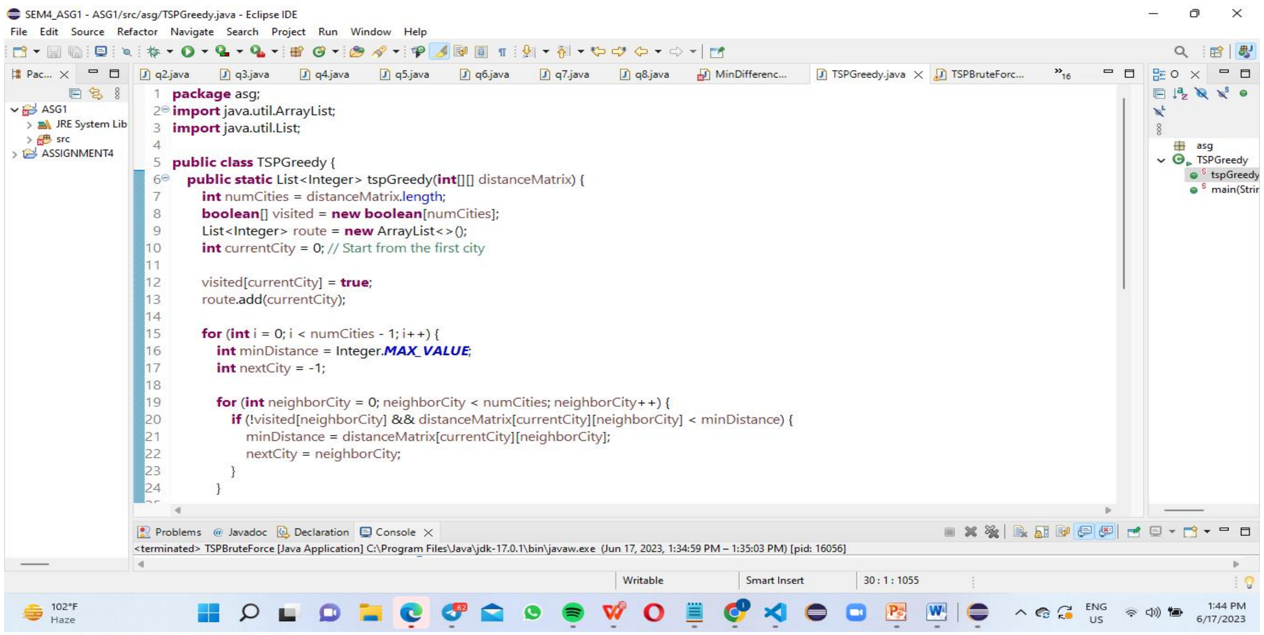
Brute Force Algorithm Solution:

1. Generate all possible permutations of the cities: ABCD, ABDC, ACBD, ACDB, ADBC, ADCB, BACD, BADC, BCAD, BCDA, BDAC, BDCA, CABD, CADB, CBAD, CBDA, CDAB, CDBA, DABC, DACB, DBAC, DBCA, DCAB, DCBA.
2. Calculate the cost for each permutation by summing the distances between consecutive cities:
- ABCD: 10 + 35 + 30 + 15 = 90
- ABDC: 10 + 25 + 15 + 30 = 80 (minimum cost)

- ACBD: 15 + 35 + 10 + 25 = 85
- ACDB: 15 + 30 + 10 + 25 = 80 (minimum cost)
- ADBC: 20 + 35 + 10 + 25 = 90
- ADCB: 20 + 30 + 10 + 35 = 95
- BACD: 10 + 30 + 35 + 15 = 90
- BADC: 10 + 25 + 30 + 15 = 80 (minimum cost)
- BCAD: 15 + 25 + 10 + 35 = 85
- BCDA: 15 + 30 + 25 + 10 = 80 (minimum cost)
- BDAC: 20 + 25 + 10 + 35 = 90
- BDCA: 20 + 30 + 25 + 10 = 85
- CABD: 15 + 10 + 35 + 30 = 90
- CADB: 15 + 10 + 30 + 35 = 90
- CBAD: 35 + 10 + 25 + 15

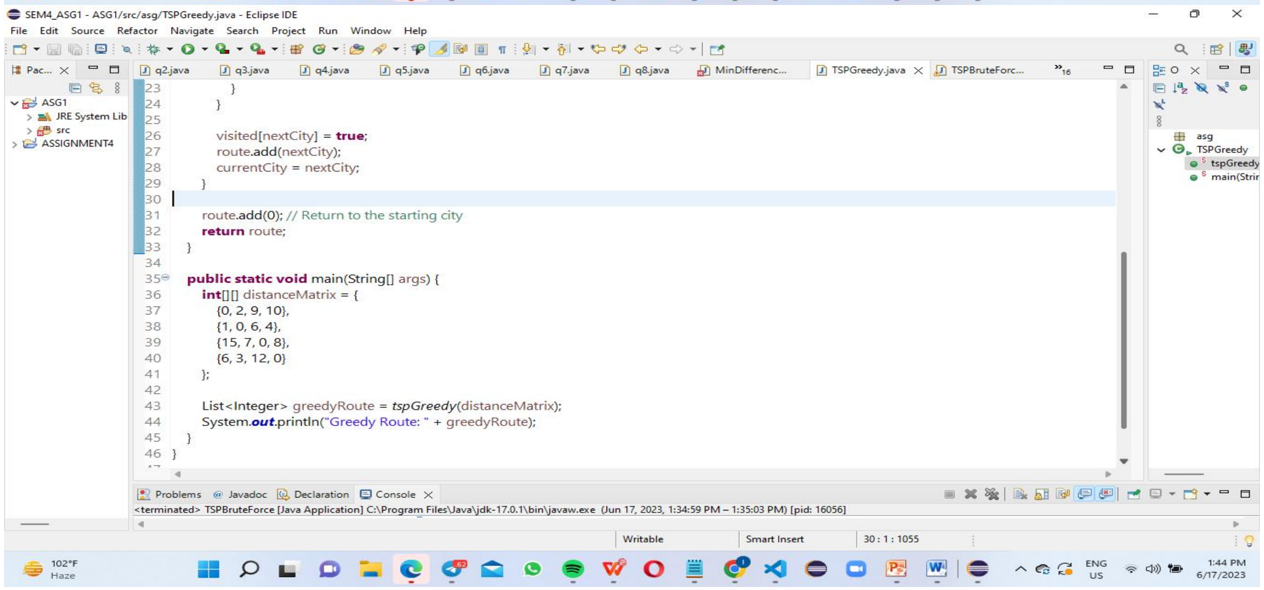
Implementation Details/Code:

TSP Using Greedy Algorithm


SEM4_ASG1 - ASG1/src/asg/TSPGreedy.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
ASG1
JRE System Lib
src
ASSIGNMENT4
q2.java q3.java q4.java q5.java q6.java q7.java q8.java MinDifferenc... TSPGreedy.java TSPBruteForc...
asg
TSPGreedy
tspGreedy
main(Strir
Problems Javadoc Declaration Console
Writable Smart Insert 30 : 1 : 1055
102°F Haze
1:44 PM 6/17/2023

```java
package asg;
import java.util.ArrayList;
import java.util.List;

public class TSPGreedy {
    public static List<Integer> tspGreedy(int[][] distanceMatrix) {
        int numCities = distanceMatrix.length;
        boolean[] visited = new boolean[numCities];
        List<Integer> route = new ArrayList<>();
        int currentCity = 0; // Start from the first city

        visited[currentCity] = true;
        route.add(currentCity);

        for (int i = 0; i < numCities - 1; i++) {
            int minDistance = Integer.MAX_VALUE;
            int nextCity = -1;

            for (int neighborCity = 0; neighborCity < numCities; neighborCity++) {
                if (!visited[neighborCity] && distanceMatrix[currentCity][neighborCity] < minDistance) {
                    minDistance = distanceMatrix[currentCity][neighborCity];
                    nextCity = neighborCity;
                }
            }
```

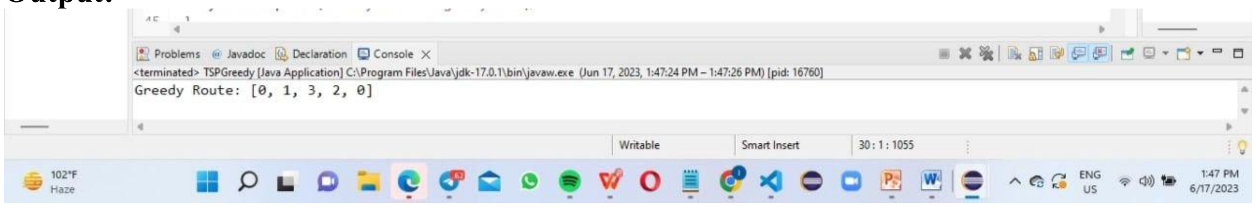
```
<terminated> TSPBruteForce [Java Application] C:\Program Files\Java\jdk-17.0.1\bin\javaw.exe (Jun 17, 2023, 1:34:59 PM – 1:35:03 PM) [pid: 16056]
```


SEM4_ASG1 - ASG1/src/asg/TSPGreedy.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Writable Smart Insert 30 : 1 : 1055
1:44 PM 6/17/2023

```java
                }
            }

            visited[nextCity] = true;
            route.add(nextCity);
            currentCity = nextCity;
        }

        route.add(0); // Return to the starting city
        return route;
    }

    public static void main(String[] args) {
        int[][] distanceMatrix = {
            {0, 2, 9, 10},
            {1, 0, 6, 4},
            {15, 7, 0, 8},
            {6, 3, 12, 0}
        };

        List<Integer> greedyRoute = tspGreedy(distanceMatrix);
        System.out.println("Greedy Route: " + greedyRoute);
    }
}
```

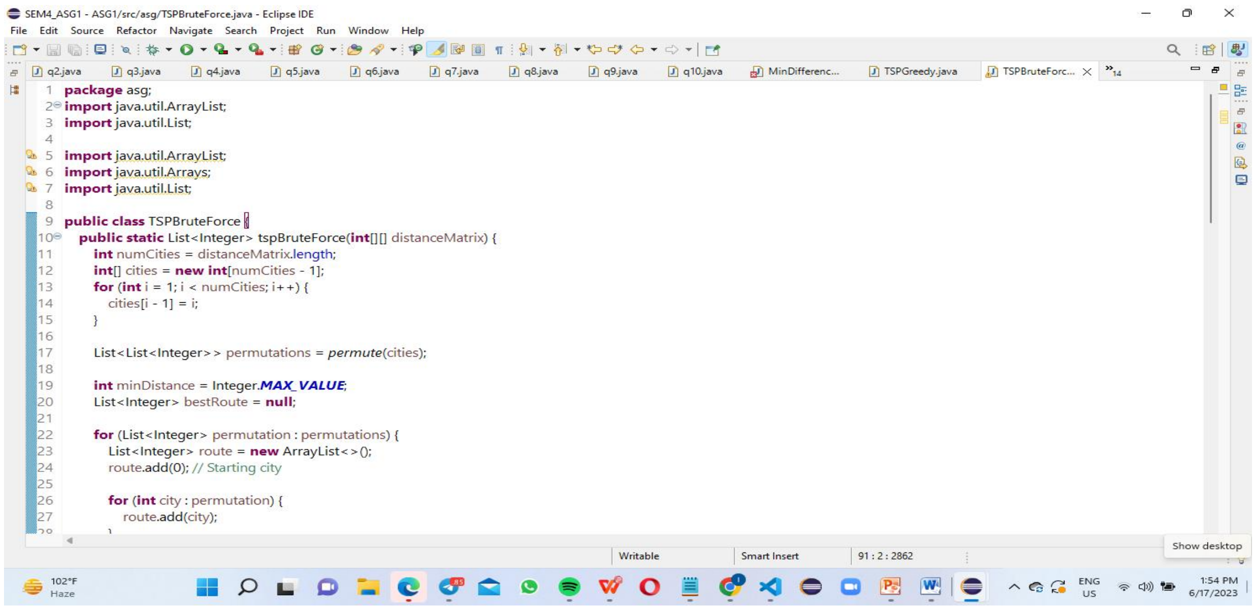
```
<terminated> TSPBruteForce [Java Application] C:\Program Files\Java\jdk-17.0.1\bin\javaw.exe (Jun 17, 2023, 1:34:59 PM – 1:35:03 PM) [pid: 16056]
```

Output:

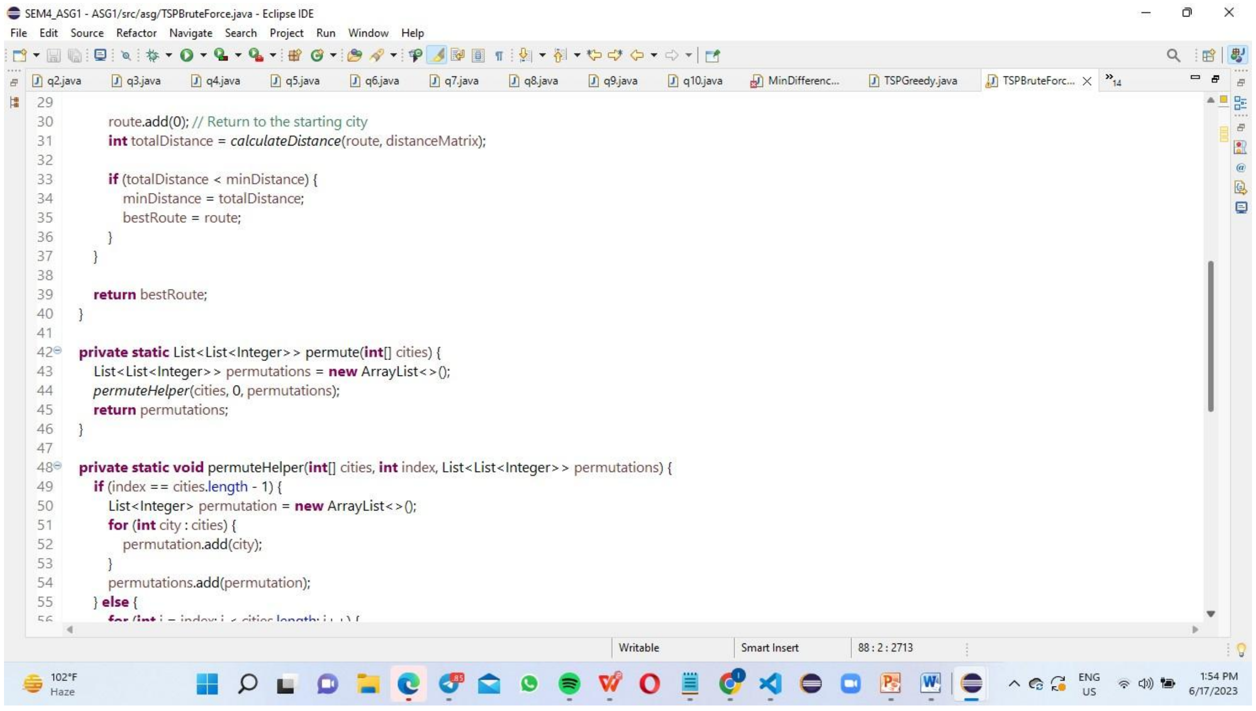

Problems Javadoc Declaration Console
Writable Smart Insert 30 : 1 : 1055
1:47 PM 6/17/2023

```
<terminated> TSPGreedy [Java Application] C:\Program Files\Java\jdk-17.0.1\bin\javaw.exe (Jun 17, 2023, 1:47:24 PM – 1:47:26 PM) [pid: 16760]
Greedy Route: [0, 1, 3, 2, 0]
```

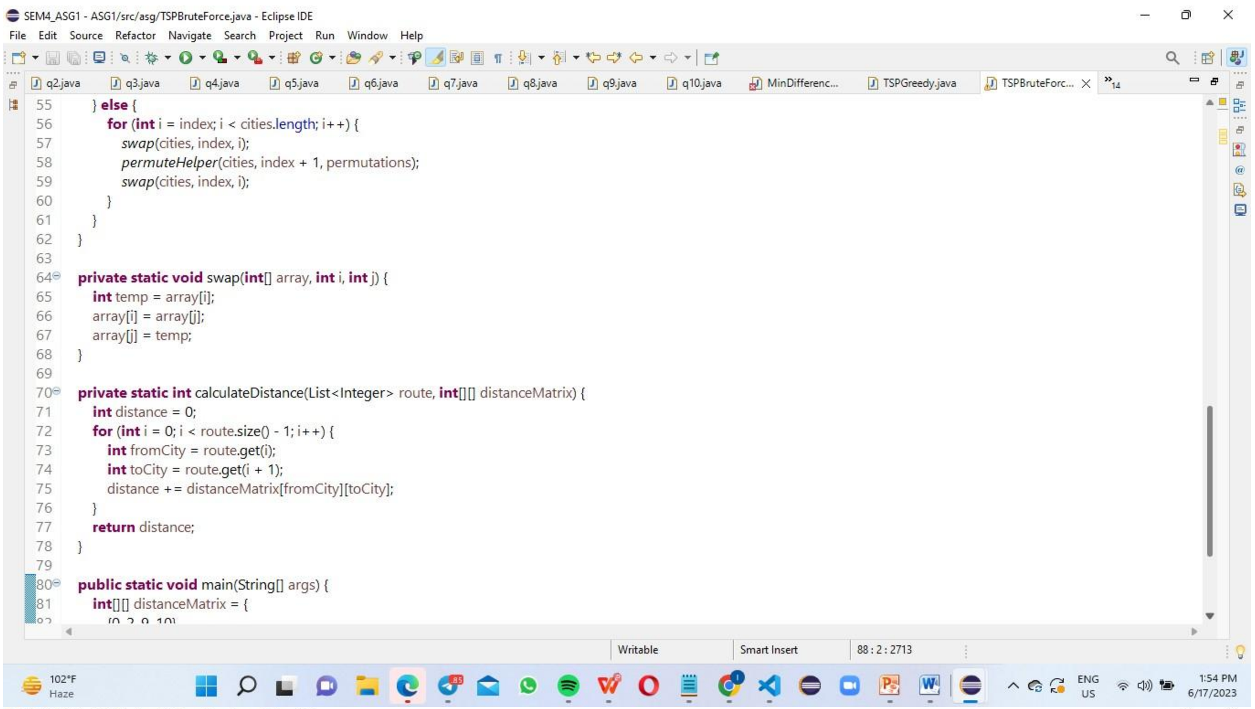
TSP Using Brute-Force Algorithm


SEM4_ASG1 - ASG1/src/asg/TSPBruteForce.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
q2.java q3.java q4.java q5.java q6.java q7.java q8.java q9.java q10.java MinDifferenc... TSPGreedy.java TSPBruteForc...
Writable
Smart Insert
91 : 2 : 2862
Show desktop
102°F
Haze
ENG
US
1:54 PM
6/17/2023

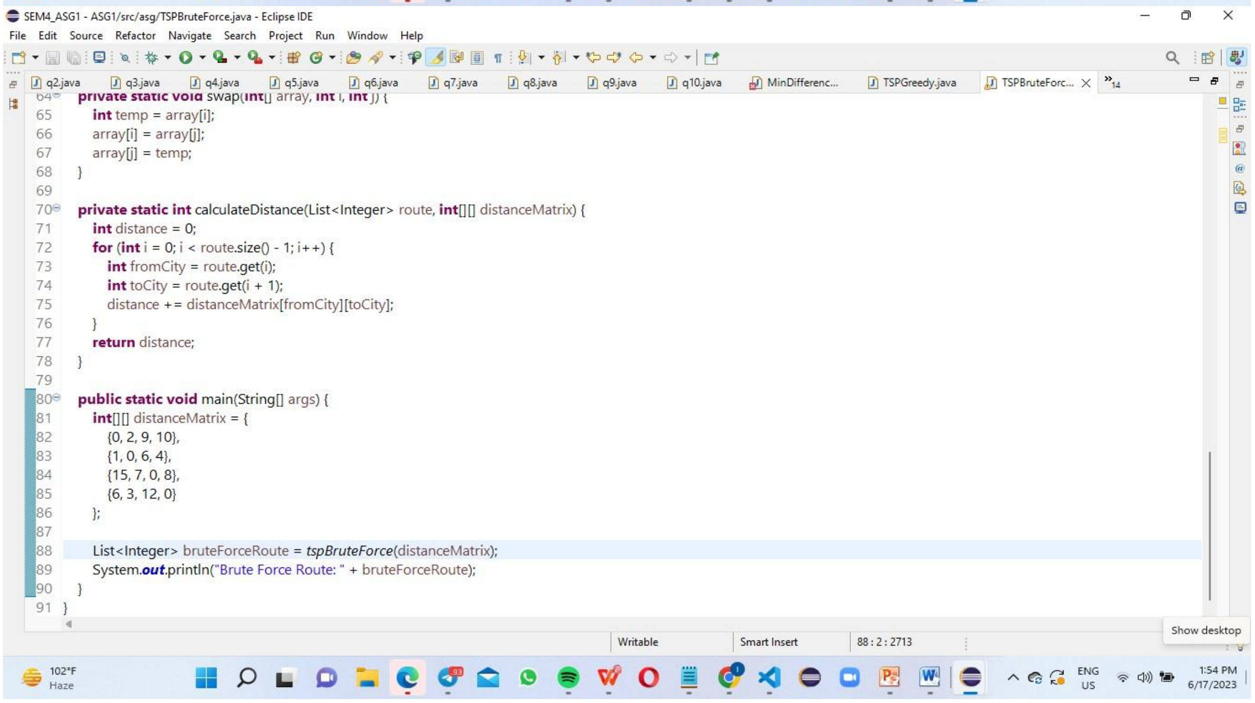
```java
package asg;
import java.util.ArrayList;
import java.util.List;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

public class TSPBruteForce {
    public static List<Integer> tspBruteForce(int[][] distanceMatrix) {
        int numCities = distanceMatrix.length;
        int[] cities = new int[numCities - 1];
        for (int i = 1; i < numCities; i++) {
            cities[i - 1] = i;
        }

        List<List<Integer>> permutations = permute(cities);

        int minDistance = Integer.MAX_VALUE;
        List<Integer> bestRoute = null;

        for (List<Integer> permutation : permutations) {
            List<Integer> route = new ArrayList<>();
            route.add(0); // Starting city

            for (int city : permutation) {
                route.add(city);
```


SEM4_ASG1 - ASG1/src/asg/TSPBruteForce.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
q2.java q3.java q4.java q5.java q6.java q7.java q8.java q9.java q10.java MinDifferenc... TSPGreedy.java TSPBruteForc...
Writable
Smart Insert
88 : 2 : 2713
102°F
Haze
ENG
US
1:54 PM
6/17/2023

```java

            route.add(0); // Return to the starting city
            int totalDistance = calculateDistance(route, distanceMatrix);

            if (totalDistance < minDistance) {
                minDistance = totalDistance;
                bestRoute = route;
            }
        }

        return bestRoute;
    }

    private static List<List<Integer>> permute(int[] cities) {
        List<List<Integer>> permutations = new ArrayList<>();
        permuteHelper(cities, 0, permutations);
        return permutations;
    }

    private static void permuteHelper(int[] cities, int index, List<List<Integer>> permutations) {
        if (index == cities.length - 1) {
            List<Integer> permutation = new ArrayList<>();
            for (int city : cities) {
                permutation.add(city);
            }
            permutations.add(permutation);
        } else {
```


SEM4_ASG1 - ASG1/src/asg/TSPBruteForce.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
q2.java q3.java q4.java q5.java q6.java q7.java q8.java q9.java q10.java MinDifferenc... TSPGreedy.java TSPBruteForc...
Writable Smart Insert 88 : 2 : 2713

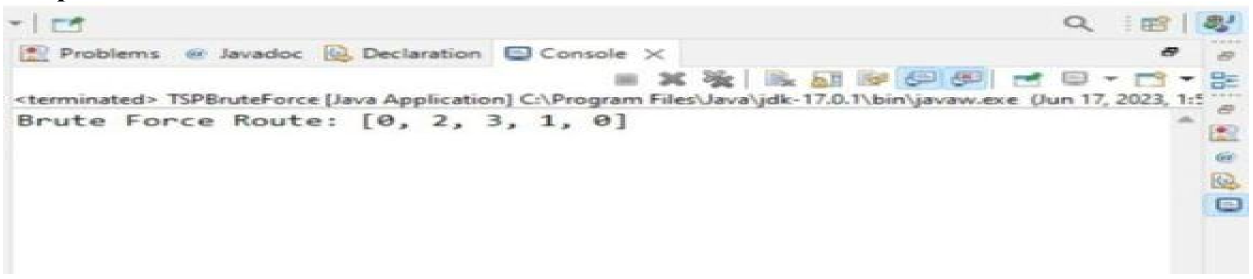
```java
        } else {
            for (int i = index; i < cities.length; i++) {
                swap(cities, index, i);
                permuteHelper(cities, index + 1, permutations);
                swap(cities, index, i);
            }
        }
    }

    private static void swap(int[] array, int i, int j) {
        int temp = array[i];
        array[i] = array[j];
        array[j] = temp;
    }

    private static int calculateDistance(List<Integer> route, int[][] distanceMatrix) {
        int distance = 0;
        for (int i = 0; i < route.size() - 1; i++) {
            int fromCity = route.get(i);
            int toCity = route.get(i + 1);
            distance += distanceMatrix[fromCity][toCity];
        }
        return distance;
    }

    public static void main(String[] args) {
        int[][] distanceMatrix = {
```


SEM4_ASG1 - ASG1/src/asg/TSPBruteForce.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
q2.java q3.java q4.java q5.java q6.java q7.java q8.java q9.java q10.java MinDifferenc... TSPGreedy.java TSPBruteForc...
Writable Smart Insert 88 : 2 : 2713
Show desktop

```java
    private static void swap(int[] array, int i, int j) {
        int temp = array[i];
        array[i] = array[j];
        array[j] = temp;
    }

    private static int calculateDistance(List<Integer> route, int[][] distanceMatrix) {
        int distance = 0;
        for (int i = 0; i < route.size() - 1; i++) {
            int fromCity = route.get(i);
            int toCity = route.get(i + 1);
            distance += distanceMatrix[fromCity][toCity];
        }
        return distance;
    }

    public static void main(String[] args) {
        int[][] distanceMatrix = {
```

```java
            {0, 2, 9, 10},
            {1, 0, 6, 4},
            {15, 7, 0, 8},
            {6, 3, 12, 0}
        };

        List<Integer> bruteForceRoute = tspBruteForce(distanceMatrix);
        System.out.println("Brute Force Route: " + bruteForceRoute);
    }
}
```

Output:



```
<terminated> TSPBruteForce [Java Application] C:\Program Files\Java\jdk-17.0.1\bin\javaw.exe (Jun 17, 2023, 1:5
Brute Force Route: [0, 2, 3, 1, 0]
```

Codes with explanation on each module and data structure used

1. Problem Description: The TSP is a classic optimization problem where the goal is to find the shortest possible route that visits a set of cities exactly once and returns to the starting city. The input is a distance matrix that represents the distances between each pair of cities.
2. Greedy Algorithm:

- Module: `tsp_greedy`
- Data Structures Used:
- `distance_matrix`: A 2D array representing the distances between cities.
- `visited`: A boolean array to track visited cities.
- `route`: A list to store the order of visited cities.
- Explanation:
- Initialize `visited` and `route` with default values.
- Start from the first city (`current_city = 0`).
- Iterate `num_cities - 1` times:
- Mark the current city as visited.
- Find the next unvisited city with the shortest distance from the current city.
- Update `current_city` and add it to `route`.
- Add the starting city at the end of `route` to complete the cycle.
- Return the `route`.

3. Brute Force Algorithm:

- Module: `tsp_brute_force`
- Data Structures Used:
- `distance_matrix`: A 2D array representing the distances between cities.
- `cities`: An array to store the indices of unvisited cities.
- `permutations`: A list of all possible permutations of `cities`.
- `minDistance` and `bestRoute`: Variables to store the minimum distance and the best route found so far.
- Explanation:
- Initialize `cities` with the indices of unvisited cities (excluding the starting city).
- Generate all permutations of `cities` using a helper function.
- Iterate through each permutation:
- Create a `route` list starting with the starting city and the cities in the permutation.
- Calculate the total distance of the `route` using the distance matrix.
- Update `minDistance` and `bestRoute` if the current distance is smaller.
- Return the `bestRoute`.

RESULTS AND DISCUSSIONS

Solving the Traveling Salesman Problem (TSP) using the Greedy and Brute Force algorithms yields different results and discussions regarding time complexity and space complexity:

1. Greedy Algorithm:

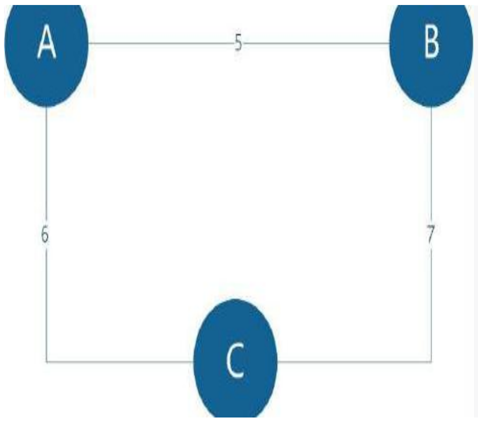
- Result and Discussion:
- The Greedy algorithm provides a fast solution to the TSP but does not guarantee the optimal solution.
- The time complexity of the Greedy algorithm is O(n^2), where n is the number of cities. It is efficient and suitable for large problem sizes.
- The space complexity of the Greedy algorithm is O(n), as it primarily requires space for the distance matrix, visited array, and route list. It is relatively modest in terms of memory usage.

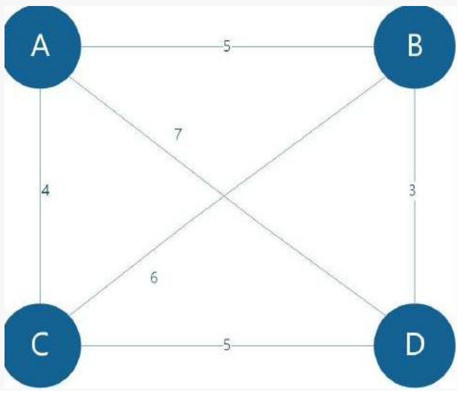
2. Brute Force Algorithm:

- Result and Discussion:
- The Brute Force algorithm guarantees the optimal solution for the TSP but becomes impractical for larger problem sizes due to its exponential time complexity.
- The time complexity of the Brute Force algorithm is O(n!), where n is the number of cities. It exhaustively explores all possible permutations, making it extremely time-consuming for larger instances of the problem.
- The space complexity of the Brute Force algorithm is also exponential, O(n!), due to the storage of permutations and the best route found. This high space requirement limits its applicability to small problem sizes.

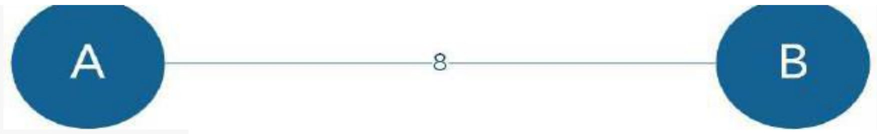
The Greedy algorithm provides a fast but suboptimal solution with a time complexity of O(n^2) and a modest space complexity of O(n). On the other hand, the Brute Force algorithm guarantees the optimal solution but suffers from exponential time complexity of O(n!) and a correspondingly high space complexity of O(n!). The choice between these algorithms depends on the problem size and the trade-off between finding the optimal solution and computational efficiency. Greedy algorithm is suitable for large problem sizes where an approximate solution is sufficient, while the Brute Force algorithm is limited to small problem sizes where finding the exact optimal solution is crucial.

Experimental Results:

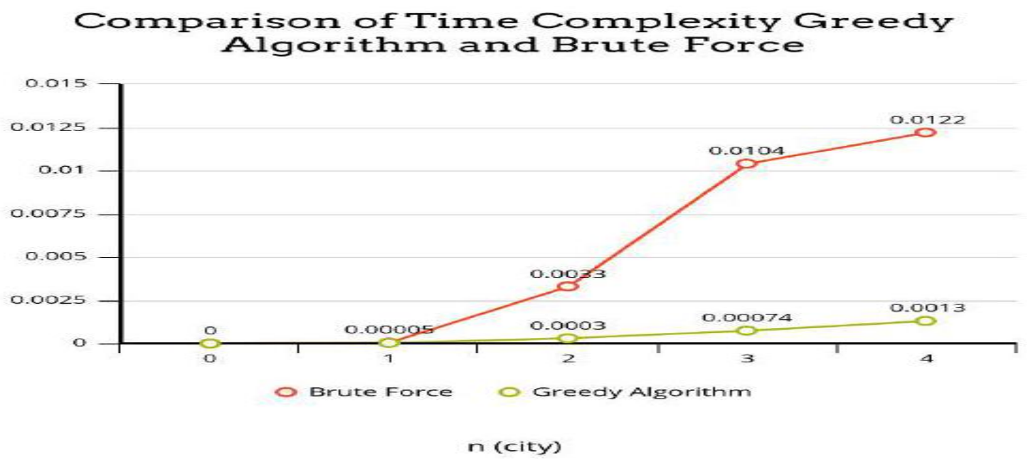




Graph of Travelling Salesman Problem with 3 Cities *Graph of Travelling Salesman Problem with 4 Cities*



Graph of Travelling Salesman Problem with 2 Cities



Limitations

Both the Greedy and Brute Force algorithms for solving the Traveling Salesman Problem (TSP) have their limitations:

1. Greedy Algorithm:

- Suboptimal Solution: The Greedy algorithm does not guarantee finding the optimal solution for the TSP. It makes locally optimal choices at each step, which may lead to a route that is not the shortest possible.
- Sensitivity to Initial Configuration: The Greedy algorithm's solution can vary depending on the starting city or the order in which the cities are processed. Different initial configurations can yield different routes and distances.
- Lack of Flexibility: The Greedy algorithm follows a fixed decision-making strategy without considering potential improvements in the overall route.

2. Brute Force Algorithm:

- Exponential Time Complexity: The Brute Force algorithm has an exponential time complexity of O(n!), where n is the number of cities. As the number of cities increases, the computation time grows exponentially, making it impractical for larger problem sizes.
- High Memory Requirements: The Brute Force algorithm requires significant memory to store all possible permutations of cities and the best route found so far. The space complexity is also exponential, O(n!), which restricts its usage to small problem instances.
- Computational Inefficiency: Due to the exhaustive search through all possible permutations, the Brute Force algorithm can be computationally inefficient, especially when dealing with a large number of cities.

Future Enhancements

In the future, there are several potential enhancements that could be applied to solving the Traveling Salesman Problem (TSP) using the greedy and brute force algorithms. Here are a few possibilities:

1. Parallelization: With advancements in parallel computing and distributed systems, it would be possible to parallelize the computation of the brute force algorithm. This would involve dividing the search space among multiple processors or machines, allowing for faster exploration of possible solutions.
2. Approximation algorithms: Greedy algorithms can provide fast solutions but are not guaranteed to find the optimal solution for TSP. Future enhancements could focus on developing improved approximation algorithms that strike a balance between speed and solution quality, providing near-optimal solutions with a reasonable runtime.
3. Heuristic enhancements: Greedy algorithms often rely on heuristics to make decisions about the next step in the solution. Future research could focus on refining these heuristics or developing new ones that better capture the problem's characteristics, leading to improved performance and solution quality.
4. Metaheuristic algorithms: Metaheuristic algorithms, such as simulated annealing, genetic algorithms, or ant colony optimization, have been successfully applied to TSP. Future enhancements could involve combining these metaheuristic techniques with greedy or brute force algorithms to exploit their respective strengths and potentially improve solution quality.

References

1. Kleinberg, J., & Tardos, E. (2006). *Algorithm design*. Pearson Education
India.
2. Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2022).
Introduction to algorithms. MIT press.
3 . Google, Wikipedia, ChatGPT