

**East West University**

**Department of Computer Science and Engineering**

**PROJECT REPORT ON**

**“Travelling salesman problem”**

**Course: Algorithms**

**Course Code: CSE246**

**Semester: Fall 2023**

**Section: 03**

**Under the Guidance Of:**

Amit Mandal

**Submitted By:**

Md Arnob Tanvir (ID: 2020-1-60-220)

**Date of Submission:** 26-12-2023

**Introduction**

The Traveling Salesman Problem (TSP) is like solving a puzzle for finding the shortest route to visit several cities and return home. We can Think of it as planning the most efficient way for a person to travel through different places.

Whether it's a salesperson or anyone with multiple destinations, the challenge is to figure out the quickest path, especially as the number of cities increases. In this project report, we'll delve into the Traveling Salesman Problem, understanding its significance, and exploring how computer algorithms can help solve this intriguing problem efficiently.

**Related works**

Solving the Traveling Salesman Problem (TSP) involves different ways. Some methods, like Dynamic Programming, give precise answers but are slow for big problems. Branch and Bound carefully looks for the best solution. Greedy ways, such as Nearest Neighbor Algorithm, pick the closest city step by step. Minimum Spanning Tree makes an approximation by creating a special tree. There are also variations, like Asymmetric TSP and Time-Dependent TSP. Concorde TSP Solver exactly solves small problems. Parallel methods handle bigger ones. These different ways show how TSP can be solved in many clever ways.

**Methodology**

Methodology of the Greedy TSP Algorithm Implementation:

1. Initialization:

- Initialize variables, including `visited` and `visitedCities`.

- Set the starting city, cost, and current city.

2. Greedy Selection:

- Iterate through unvisited cities to find the nearest one.

- Update `minDistance` and `nextCity` if a shorter distance is found.

3. Tour Construction:

- Add the selected `nextCity` to the tour.

- Mark `nextCity` as visited.

- Update the total `cost` with the minimum distance.

- Set `currentCity` to `nextCity`.

4. Completion of Tour:

- Ensure all cities are visited.

- Return to the starting city.

- Update the `cost` and add the starting city to the tour.

5. Output:

- Display the minimum tour distance and the sequence of visited cities.

6. Error Handling:

- Validate user inputs (positive number of cities, non-negative distances).

7. Result Display:

- Display an error message if a valid tour cannot be found.

- Otherwise, present the minimum distance and the sequence of visited cities.

This summarizes the main steps and logic of the Greedy TSP Algorithm implemented in the provided C++ code.

**Proposed work**

This program is like a smart travel planner. We tell it how many cities we want to visit and the distances between them. It figures out the quickest way to visit all the cities and come back to the origin city. The program uses a clever method to choose the next city at each step, making it efficient. After running, it tells us the shortest distance for the salesman’s tour and the order in which he should visit the cities. It's like having a helpful guide to plan his road trip, making sure he can cover all the places with the least cost.

**Includes**

The code starts by adding some libraries.

* iostream is the standard input/output library in C++, which provides functions for reading and writing data.
* Vector is the library that includes resizable array. Also we can use it like stacks
* `<climits>` is a header file in C++ that provides constants representing the limits constants like `INT\_MAX` and `INT\_MIN`, which represent the maximum and minimum values for the `int` data type.

**Variables**

- `n`: Number of cities, input by the user.

- `distancesMatrix`: A 2D vector representing the distance matrix between cities.

- `visited`: A boolean vector indicating whether a city has been visited during the tour.

- `cost`: The total cost or distance of the tour.

- `currentCity`: The current city being processed during the tour.

- `visitedCities`: A vector to store the sequence of cities visited during the tour.

- `nextCity`: The city to be visited next during the tour.

- `minDistance`: The minimum distance to the next city.

Functions:

- `tspGreedy`: Implements the greedy approach for the traveling salesman problem, finding the minimum distance tour starting from a given city.

**Algorithm**

1. Read the number of cities n.
2. Read the distance matrix distancesMatrix for the cities.
3. Read the starting city startCity.
4. Initialize an empty vector visitedCities to store the sequence of visited cities.
5. Initialize a boolean vector visited to track visited cities, setting startCity as visited.
6. Initialize variables cost and currentCity to 0 and startCity, respectively.
7. Loop until all cities are visited:
8. Find the next unvisited city with the minimum distance from the current city.
9. Update nextCity with the index of the next city.
10. Add nextCity to visitedCities.
11. Set visited[nextCity] to true.
12. Update cost with the distance to nextCity.
13. Set currentCity to nextCity.
14. Add the distance from the last visited city to the starting city to cost.
15. Display the minimum distance (cost) and the sequence of visited cities (visitedCities).

The program utilizes a greedy algorithm to determine the optimal tour, starting from a user-specified city and visiting each city exactly once.

**Psuedocode for the tspGreedy function**

Function tspGreedy(startCity, n, distancesMatrix):

visitedCities = [startCity]

visited = (false,n)

visited[startCity] = true

cost = 0

currentCity = startCity

while length(visitedCities) < n:

nextCity = -1

minDistance = ∞

for i = 0 to n-1:

if ->not visited[i] && distancesMatrix[currentCity][i] != -1&& distancesMatrix[currentCity][i] < minDistance:

minDistance = distancesMatrix[currentCity][i]

nextCity = i

if -> nextCity == -1:

print("Error: Unable to find the next city.")

return

visitedCities.append(nextCity)

visited[nextCity] = true

cost += minDistance

currentCity = nextCity

cost += distancesMatrix[currentCity][startCity]

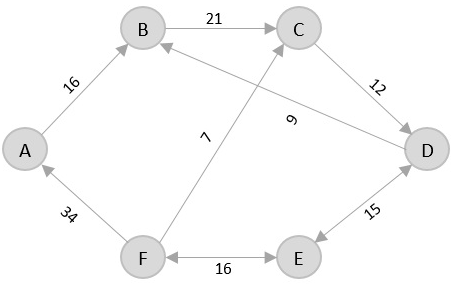
visitedCities.append(startCity)

return (cost, visitedCities)

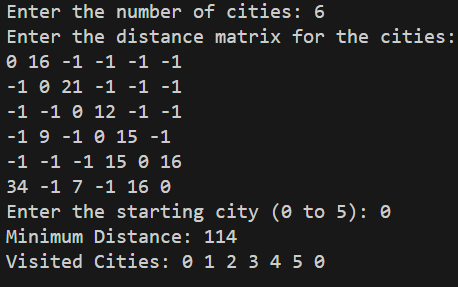
**Summary**

This C++ program solves the Traveling Salesman Problem (TSP) using a greedy algorithm. The user inputs the number of cities, and the distances between them. The algorithm starts from a chosen city, iteratively selects the nearest unvisited city, and forms a tour. The program then outputs the minimum distance traveled and the order of visited cities, providing an efficient route for the salesman to cover all cities. If there's an error or no valid tour is found, it alerts the user.

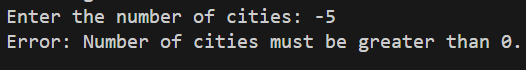
**Experimental result**

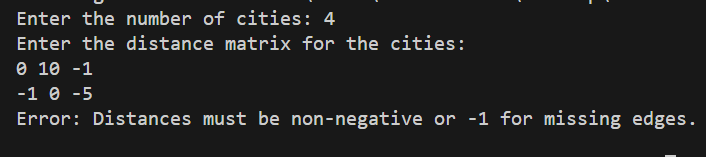


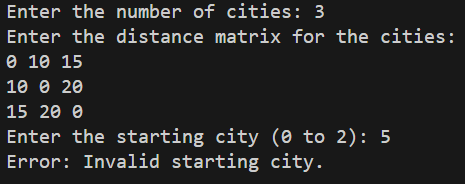
We get the following result. For this graph

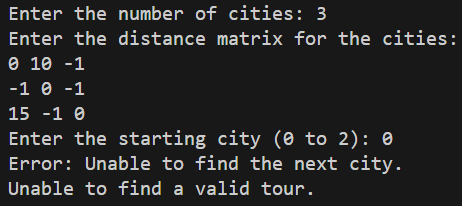


If we give invalid inputs then the program will warn us.









**Conclusion**

This program efficiently finds the shortest path to visit all given cities exactly once, known as the traveling salesman problem. It employs a user-friendly approach, taking inputs for city details and displaying the minimum distance tour and visited cities sequence. The algorithm gracefully handles errors, ensuring practical use in diverse scenarios.

**References**

1. [Travelling Salesman Problem (tutorialspoint.com)](https://www.tutorialspoint.com/design_and_analysis_of_algorithms/design_and_analysis_of_algorithms_travelling_salesman_problem.htm)
2. [trackobit.com](https://trackobit.com/blog/what-is-a-traveling-salesman-problem-explained)

**Appendix(code)**

#include <iostream>

#include <vector>

#include <climits>

using namespace std;

pair<int, vector<int>> tspGreedy(int startCity, int n, vector<vector<int>> &distancesMatrix)

{

vector<bool> visited(n, false);

visited[startCity] = true;

int cost = 0;

int currentCity = startCity;

vector<int> visitedCities = {startCity};

while (visitedCities.size() < n)

{

int nextCity = -1;

int minDistance = INT\_MAX;

for (int i = 0; i < n; ++i)

{

if (!visited[i] && distancesMatrix[currentCity][i] != -1 && distancesMatrix[currentCity][i] < minDistance)

{

minDistance = distancesMatrix[currentCity][i];

nextCity = i;

}

}

if (nextCity == -1)

{

cout << "Error: Unable to find the next city.\n";

return make\_pair(-1, vector<int>());

}

visitedCities.push\_back(nextCity);

visited[nextCity] = true;

cost += minDistance;

currentCity = nextCity;

}

cost += distancesMatrix[currentCity][startCity];

visitedCities.push\_back(startCity);

return make\_pair(cost, visitedCities);

}

int main()

{

int n;

cout << "Enter the number of cities: ";

cin >> n;

if (n <= 0)

{

cout << "Error: Number of cities must be greater than 0.\n";

return 1;

}

vector<vector<int>> distancesMatrix(n, vector<int>(n, 0));

cout << "Enter the distance matrix for the cities:\n";

for (int i = 0; i < n; ++i)

{

for (int j = 0; j < n; ++j)

{

cin >> distancesMatrix[i][j];

if (distancesMatrix[i][j] < -1)

{

cout << "Error: Distances must be non-negative or -1 for missing edges.\n";

return 1;

}

}

}

int startCity;

cout << "Enter the starting city (0 to " << n - 1 << "): ";

cin >> startCity;

if (startCity < 0 || startCity >= n)

{

cout << "Error: Invalid starting city.\n";

return 1;

}

pair<int, vector<int>> result = tspGreedy(startCity, n, distancesMatrix);

int minDistance = result.first;

vector<int> visitedCities = result.second;

if (minDistance == -1)

{

cout << "Unable to find a valid tour.\n";

}

else

{

cout << "Minimum Distance: " << minDistance << endl;

cout << "Visited Cities: ";

for (int i = 0; i < visitedCities.size(); ++i)

{

cout << visitedCities[i] << " ";

}

cout << endl;

}

return 0;

}