Travelling Salesman Problem

**Brief Summary of The Problem:**

Given a list of cities and the distance between each city, find the shortest distance to travel to each city once and return to your starting location.

**Brute Force Method:**

In this method we will test for every possible path until we make a cycle. We will start from one point and move to another point and continue until we return to the starting point. Then we will take an alternate route and continue this until all possible routes are tested. This method takes the longest time.

**The Algorithm:**

The algorithm involves 2 different arrays. One array is the main array which is a 2d array used to store the distance between the nodes. The other one is a traversing array used to set the path in which the algorithm traverses the main array. Using brute force, we permutated the traversing array for all possible permutation. This ensures that we will cover all possible routes. For each possible route the algorithm will calculate the total distance and if the current distance it calculated is smaller than the smallest distance then the smallest distance will be replaced by the current distance and the route is printed.

**Execution time with different numbers:**

|  |  |  |
| --- | --- | --- |
| **Number of Nodes** | **Time Taken in Milliseconds** | **Time Taken in Microseconds** |
| 4 | 0 | 996 |
| 5 | 0 | 997 |
| 6 | 3 | 0 |
| 7 | 6 | 0 |
| 8 | 11 | 0 |
| 9 | 35 | 0 |
| 10 | 223 | 0 |
| 11 | 2520 | 0 |

**Nearest Neighbor Method:**

This method starts from one location and then takes the shortest path to the next location next. Then from there it goes from closest location until all the locations have been visited.

**The algorithm**:

The algorithm we used was one we made on our own. The code starts from our first location and shifts through the matrix to find the closest point to our location. Ones that point is found, the distance is saved, the location is we have started at is marked as visited and the new location is our new starting point. Then the code checks if we have been to all our locations. If we have been to all locations, our total distance is printed out and the code ends. If not then the code is run again through recursion with our new set destination. This time when the shortest distance is been found it checks if we have been to that location already as to not be stuck in loop.

**Execution time with different point numbers:**

For the purpose of this test. We created a random number generator to fill up our matrix depending on the number of points we are testing with.

|  |  |  |  |
| --- | --- | --- | --- |
| **# Points** | **Time in Microseconds** | **Time in Milliseconds** | **Time in Nanoseconds** |
| 5 | 0 | 0 | 0 |
| 10 | 0 | 0 | 0 |
| 20 | 15 | 15620 | 15620200 |
| 50 | 31 | 31215 | 31215800 |
| 100 | 62 | 62459 | 62459300 |
| 150 | 109 | 109348 | 109348900 |
| 300 | 249 | 249941 | 249941000 |
| 500 | 390 | 390533 | 390533100 |
| 600 | 453 | 453016 | 453016900 |

**Data Structure:**

The data structure we used is arrays and vectors instead of linked lists. We use arrays because of its fast-accessing time. We also considered the space it takes, but as all the nodes are connected to each other there will be no space advantage using linked list.

References:

To generate random numbers:

<https://stackoverflow.com/questions/10074879/rand-4000000000ul-giving-only-small-values>

To get the run time of function:

<https://www.geeksforgeeks.org/measure-execution-time-function-cpp/>

To check if an element is within a vector:

<https://stackoverflow.com/questions/3450860/check-if-a-stdvector-contains-a-certain-object>