**University of Southern California (USC)**

**EE599 – Computing Principles for Electrical Engineering**

**Final Project Report/README**

**Group 2: Runtime Terror**

**Name: Kunal Sheth (3656712856), Rohit Singh (6689387841)**

* **Abstract**

This project focuses on Graph algorithms & is based on area near University of Southern California (USC), Los Angeles.

* **What we changed**

There are 4 functions provided: AutoComplete, Get the position, Shortest Path, Travelling Trojan.

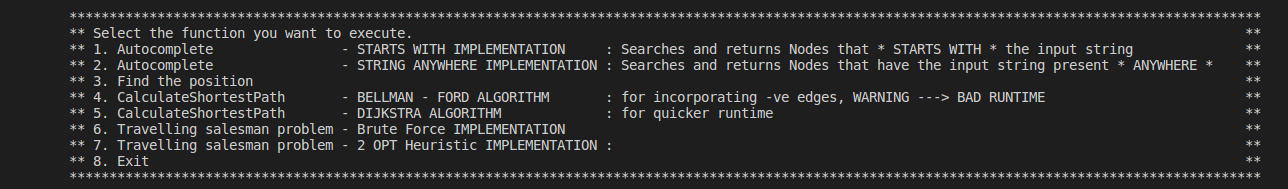
1. Main Menu: We modified the “CreateGraphfromCSVfile” - menu to incorporate various Helper functions.
2. AutoComplete: Added case insensitivity and corner cases for Testing boundary conditions.
3. Get the position: Added corner cases for Testing boundary conditions.
4. Shortest Path Dijkstra: Added corner cases for Testing boundary conditions.
5. Shortest Path Bellman Ford: Incorporated Bellman Ford algorithm.
6. Travelling Trojan: For this as there were 2 methods needed to be implemented: Brute-Force & 2\_opt, we modified the function call in the “PrintMenu” function of “trojanmap.cc” file. We also added “Travellingtrojan\_2opt” in “trojanmap.cc” & “trojanmap.h” files.

* **Individual Contributions**

1. Main Menu Alterations : Rohit
2. AutoComplete and GTest : Rohit
3. AutoComplete\_2 and GTest : Rohit
4. Get the position and GTest : Rohit
5. Shortest Path Dijkstra and GTest : Rohit
6. Shortest Path Bellman Ford and GTest : Rohit
7. Travelling Trojan Brute-Force and GTest : Kunal
8. Travelling Trojan 2\_opt and GTest : Kunal

* **How we implemented**

1. **MENU:**



1. **AutoComplete:**

We incorporated Case Insensitivity and considered all the corner test cases. We created a vector to store all the data into it. We utilized the vector to operate the AutoComplete function. To optimize the Auto Complete Function, we added a new function – that test whether the input string is anywhere in the Nodes field.

**Time Complexity**: O(n) as we are working with vectors.

A picture containing text

Description automatically generated

Fig: 1.1

A picture containing text

Description automatically generated

Fig: 1.2

A picture containing text

Description automatically generated

Fig: 1.3

A picture containing text

Description automatically generated

Fig: 1.4

1. **Get the position**: We return the latitude and longitude for the given input string.

**Time Complexity**: O(logn) as we are using the map data.

Text

Description automatically generated

Fig: 2.1

A picture containing text

Description automatically generated

Fig: 2.2

A circuit board

Description automatically generated

Fig: 2.3

A picture containing shape

Description automatically generated

Fig: 2.4

1. **Shortest Path**:
2. **Dijkstra Algorithm:** In this algorithm we are interesting from the starting position and greedily deciding the edges, according to the min. distance from the source to the child of the current node. The min heap accepts the nodes and it’s corresponding distance from the source. As we insert nodes into the heap, it automatically sorts it, and prepares the min. distance node at the top of the heap. This saves the runtime for calculating the min. distance from source. Additionally, we compare the weights every time we work will all the children of the current node.

**Time Complexity**: O(m + n log n)

1. **Bellman Ford Algorithm:** In this algorithm we are traversing all nodes present. Similar to Dijkstra’s Algorithm, we compare the min. distance from the source to the child of the current node. We lack the advantage of the priority que, in bellman ford because we are not using that. We traverse between all nodes and its children, comparing the weights every time we work will all the children of the current node.

**Time Complexity**: O(m\*n)

1. **Travelling Trojan**
2. **Brute Force Method**: Over here on this method, whenever the current path length is larger than the current optimal result we will just return. The input to the function was a “<vector <string>>” and the output was a “pair of <double, vector<vector<string>>>”. Here we firstly built a graph. Then as the graph is a cyclic one, random location was chosen out of all the provided one’s as a starting and ending location. We then generated all (n-1)! of the locations & calculated the current\_\_path weight of every possible permutation keeping the track of the minimum weight permutation & finally returned the minimum cost out of all. This method is the best possible method for the Travelling salesmen as every possible scenario will be covered. The time complexity for the brute force method is mentioned below. The output of the brute force method for 9 location is shown below. However, as the number of input location increases, this method is too slow, and we switch to different method. One such method is 2\_opt discussed next.

**Time complexity**: O(n!)

Map

Description automatically generated

Fig: 4.1.1

1. **2\_opt Heuristic Method**: Over here, the input as well as output remain the same as that of the brute force method. This method is a heuristic one where we keep swapping the nodes till the time there is no improvement. Sometimes it can find local optimal results instead of a global one. One such picture of the concept is shown below:

A picture containing shape

Description automatically generated

Fig: 4.2.1

The important thing to notice over here is that if we start or end at a particular node, then we must pop this from the search as an eligible candidate for swapping, as reversing the order will cause an invalid path. Here the time taken of large sets of input location is very less compared to that of the brute force method. The time complexity and the output from our implementation are mentioned & shown below for 2\_opt. This concept can also be extended to other heuristic method called “3\_opt”, but that is currently out of scope at this point of time.

Below is also the comparison & visualization between the Brute force & 2\_opt method.

Chart, line chart

Description automatically generated

Fig: 4.2.2(left one: brute force, right one: 2\_opt)

**Time Complexity**: O(n^2)

A picture containing map

Description automatically generated

Fig: 4.2.3

* **What did we learn?**

1. The most important thing we learned is “think and implement” & not “implement & think”.
2. In depth understanding of the usage of majorly used data structures and algorithms.
3. Debugging skills, we faced a lot of challenges & thus debugging skills for us improved a lot over our implementation as after that we were able to visualize about some corner cases that we missed.
4. Presentation skills. How we present ourselves is the most important thing which we learnt from this project.
5. Got to learn about the practical scenario & real-world implementation for shortest path algorithms beyond our project like for IP routing to find the open shortest path first, robotic path etc.

* **References**

1. <https://www.geeksforgeeks.org/traveling-salesman-problem-tsp-implementation/>
2. <https://en.wikipedia.org/wiki/2-opt>
3. <http://cs.indstate.edu/~zeeshan/aman.pdf>