EE599 FINAL PROJECT TROJANMAP

GROUP NAME: RUNTIME TERROR

NAME: ROHIT SINGH(RKSINGH@USC.EDU),
KUNAL SHETH(KUNALCHE@USC.EDU)

USC University of Southern California

CONTENTS

1 Introduction

2 Auto Complete - Rohit/ **3**

3 Get Position - Kunal / **4**

4 Shortest Path Dijkstra –Rohit / **5**

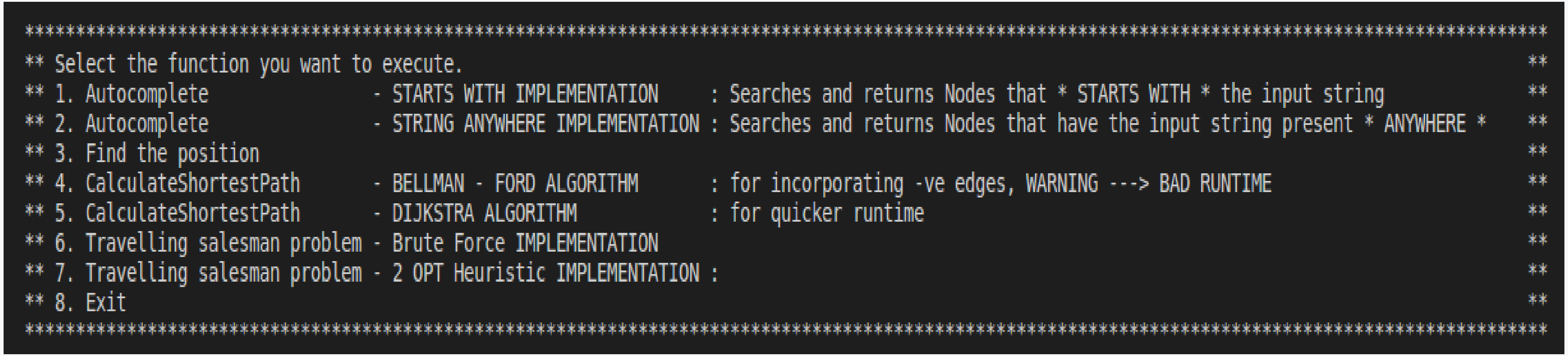
5 Shortest Path Bellman Ford –Rohit / **8**

6 Travelling Trojan Algorithm –Kunal, Rohit/ **6**

MAIN MENU

Modifying the Menu to incorporate all the helper functions

Modified the code for user adaptability.



```
******************************************************************************************************************************************************
** Select the function you want to execute.                                                                                                         **
** 1. Autocomplete                - STARTS WITH IMPLEMENTATION     : Searches and returns Nodes that * STARTS WITH * the input string               **
** 2. Autocomplete                - STRING ANYWHERE IMPLEMENTATION : Searches and returns Nodes that have the input string present * ANYWHERE *     **
** 3. Find the position                                                                                                                             **
** 4. CalculateShortestPath       - BELLMAN - FORD ALGORITHM       : for incorporating -ve edges, WARNING ---> BAD RUNTIME                          **
** 5. CalculateShortestPath       - DIJKSTRA ALGORITHM             : for quicker runtime                                                            **
** 6. Travelling salesman problem - Brute Force IMPLEMENTATION                                                                                      **
** 7. Travelling salesman problem - 2 OPT Heuristic IMPLEMENTATION :                                                                                **
** 8. Exit                                                                                                                                          **
******************************************************************************************************************************************************
```

AUTO COMPLETE

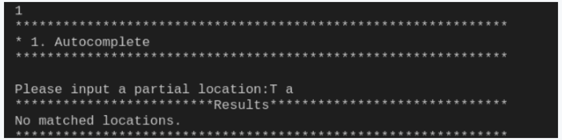
Generating all the possible Nodes, according to the partial search data

Case Sensitivity

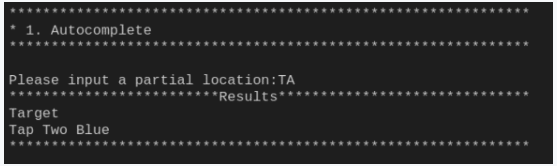
Starts With

Time complexity: O(n)

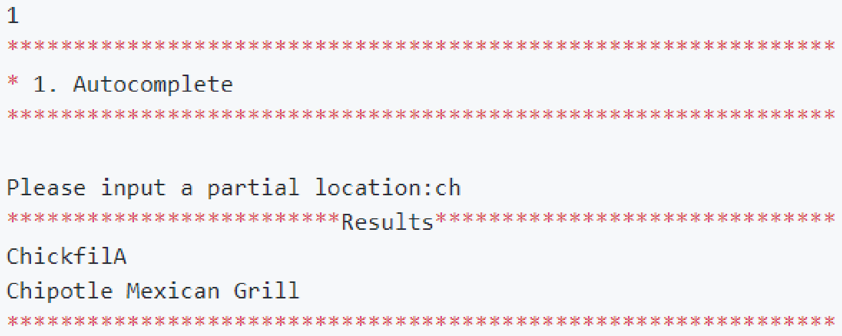
Corner Cases



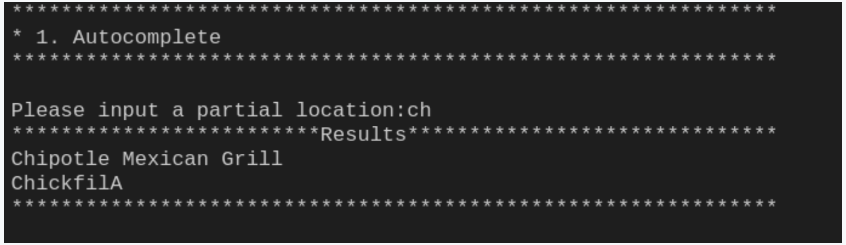
```
1
****************************************************************
* 1. Autocomplete
****************************************************************

Please input a partial location:T a
*************************Results******************************
No matched locations.
****************************************************************
```



```
*****************************************************************
* 1. Autocomplete
*****************************************************************

Please input a partial location:TA
*************************Results******************************
Target
Tap Two Blue
*****************************************************************
```



```
1
**************************************************************
* 1. Autocomplete
**************************************************************

Please input a partial location:ch
*************************Results******************************
ChickfilA
Chipotle Mexican Grill
**************************************************************
```



```
**************************************************************
* 1. Autocomplete
**************************************************************

Please input a partial location:ch
*************************Results******************************
Chipotle Mexican Grill
ChickfilA
**************************************************************
```

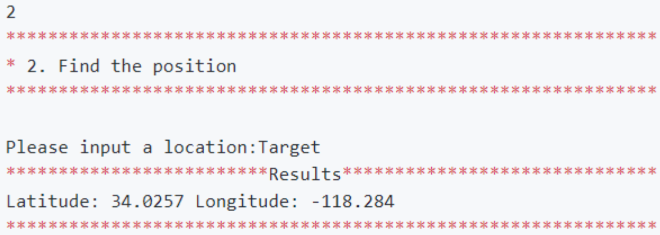
GET POSITION

Returning Position (Latitude and Longitude) for a given Nodes

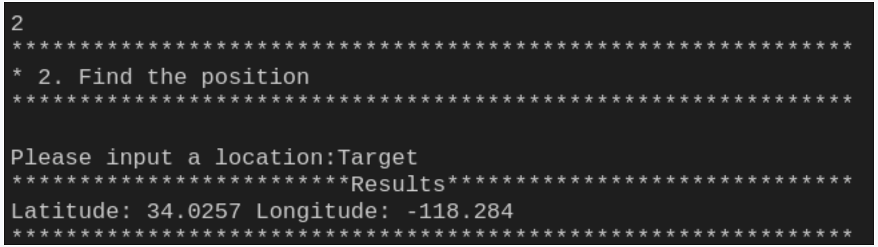
Matches exact Output mentioned

Runtime : O(logn)

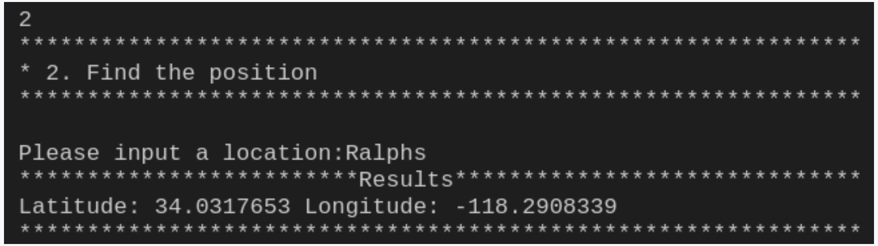
Corner Cases



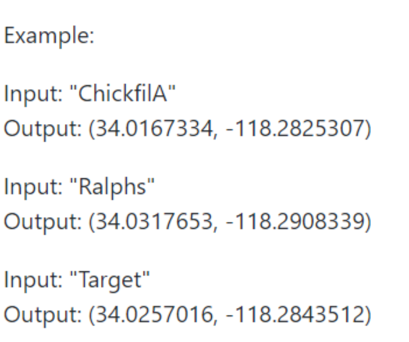
```
2
**************************************************************
* 2. Find the position
**************************************************************

Please input a location:Target
*************************Results******************************
Latitude: 34.0257 Longitude: -118.284
**************************************************************
```



```
2
**************************************************************
* 2. Find the position
**************************************************************

Please input a location:Target
*************************Results******************************
Latitude: 34.0257 Longitude: -118.284
**************************************************************
```



```
2
**************************************************************
* 2. Find the position
**************************************************************

Please input a location:Ralphs
*************************Results******************************
Latitude: 34.0317653 Longitude: -118.2908339
**************************************************************
```

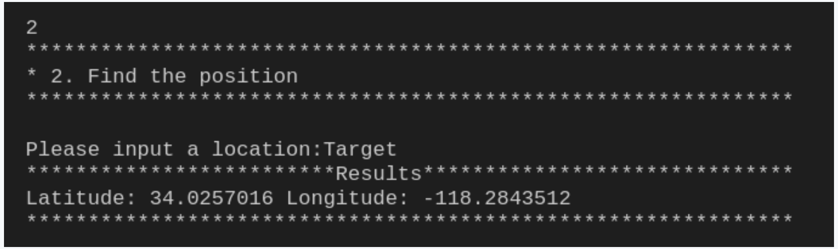


Example:

Input: "ChickfilA"
Output: (34.0167334, -118.2825307)

Input: "Ralphs"
Output: (34.0317653, -118.2908339)

Input: "Target"
Output: (34.0257016, -118.2843512)



```
2
**************************************************************
* 2. Find the position
**************************************************************

Please input a location:Target
*************************Results******************************
Latitude: 34.0257016 Longitude: -118.2843512
**************************************************************
```

SHORTEST PATH ALGORITHM(DIJKSTRA'S ALGORITHM)

- Min heap implementation helped to get the shortest path in a greedy manner, because in each step we pick the vertex with minimum distance from current vertex.

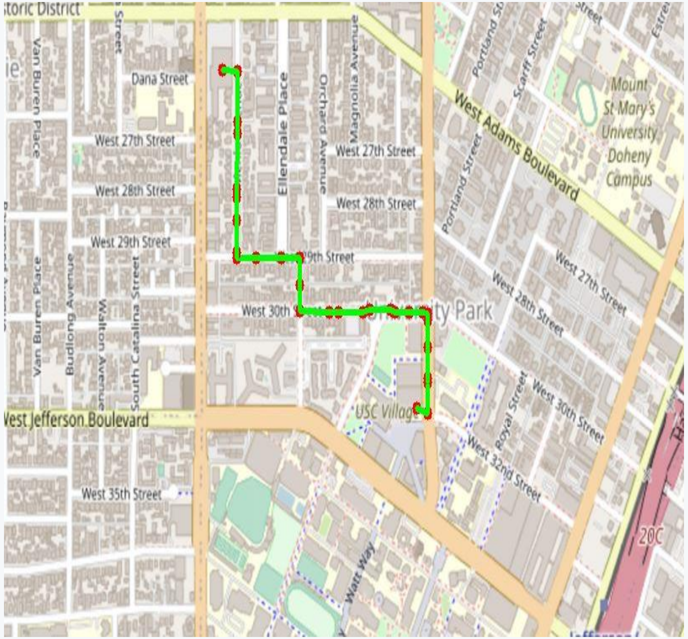
- Time Complexity: O(v^2)

- Comparison with google maps(next slide)

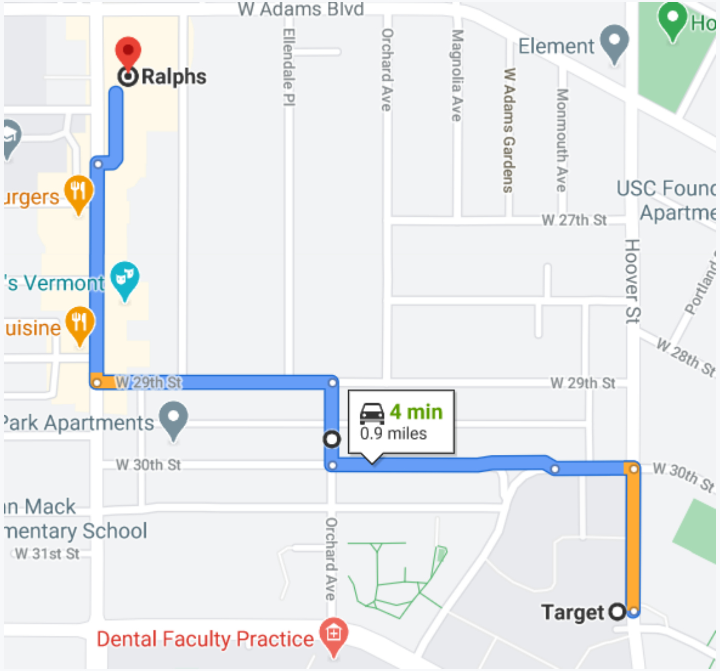
SHORTEST PATH ALGORITHM(COMPARISON WITH GOOGLE MAPS)

Implementing Djikstra Algorithm

- Input -> Target



Output -> Ralphs



SHORTEST PATH ALGORITHM(BELLMAN FORD’S ALGORITHM)

- Recursive algorithm iterating all the edges in the graph.
- Time Complexity: O(m*n)
- Graph generated is like that of Dijkstra’s.

TRAVELLING TROJAN (BRUTE FORCE - DFS)

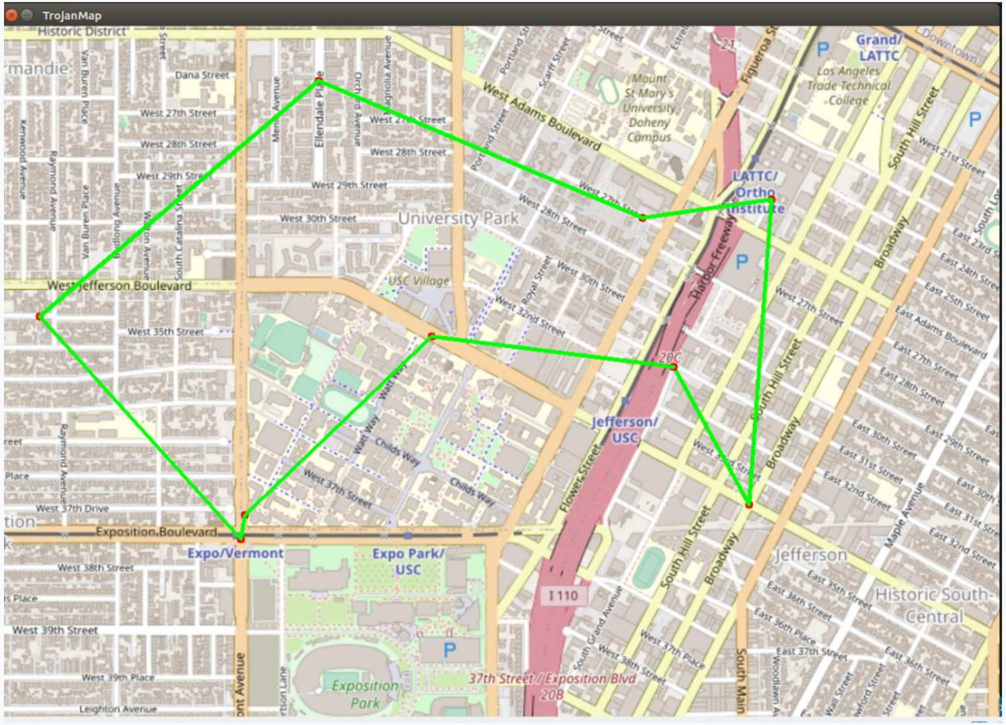
Returning Position (Latitude and Longitude) for a given Nodes

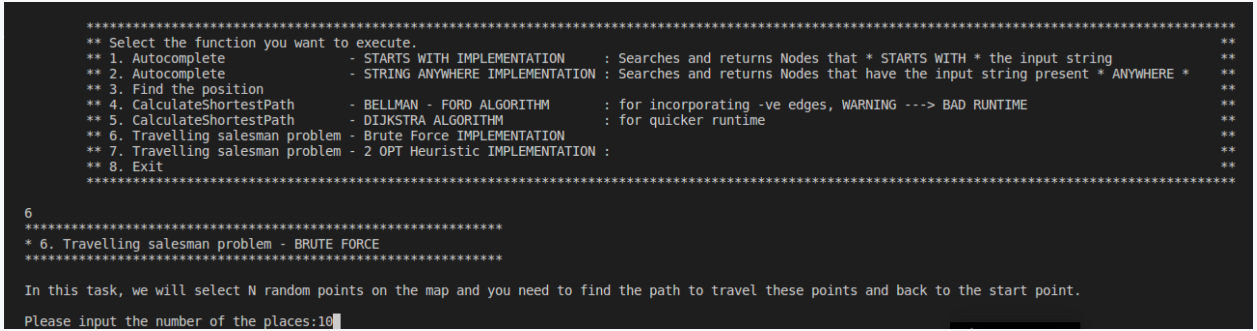
- In this method we try each and every possible permutations.
- Further, whenever the current path length is larger than the current optimal result, we need to return
- Graph data structure is used, and that graph is a cyclic one meaning that the starting point and the ending point is the same. Note, any point can be selected as a starting point.
- Finally after calculating the weight, we return the most minimum weight of all.
- Time complexity: O(n!)
- Not good for large data sets as the time to execute the code is very large

TRAVELLING TROJAN (OUTPUT - BRUTE FORCE - DFS)

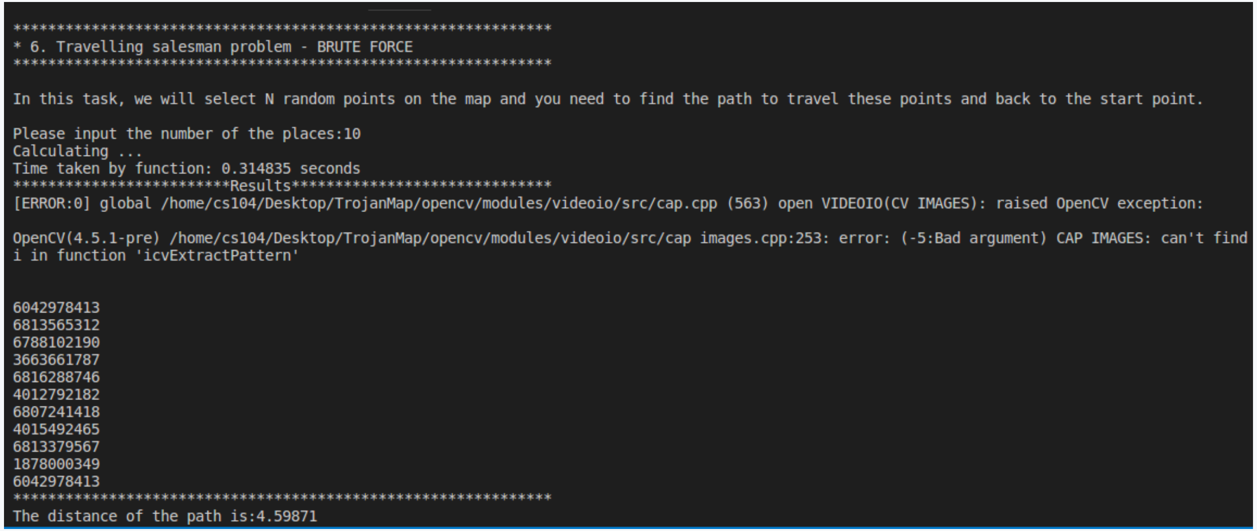
Returning Position (Latitude and Longitude) for a given Nodes

- Below is one such output from our implementation with the number of locations as ‘9’.

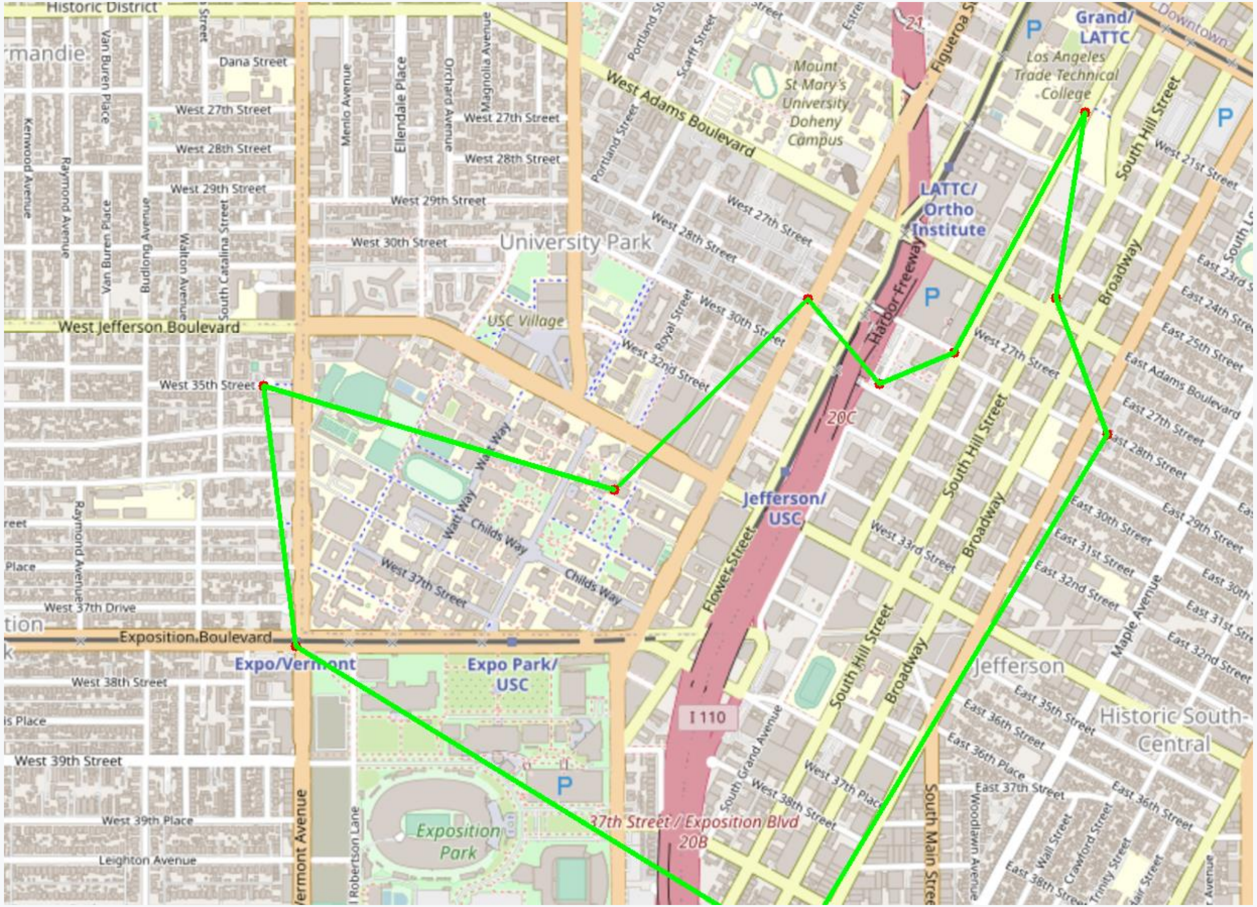




```
        **************************************************************************************************************************************************
        ** Select the function you want to execute.                                                                                                     **
        ** 1. Autocomplete                 - STARTS WITH IMPLEMENTATION     : Searches and returns Nodes that * STARTS WITH * the input string          **
        ** 2. Autocomplete                 - STRING ANYWHERE IMPLEMENTATION : Searches and returns Nodes that have the input string present * ANYWHERE *    **
        ** 3. Find the position                                                                                                                         **
        ** 4. CalculateShortestPath        - BELLMAN - FORD ALGORITHM       : for incorporating -ve edges, WARNING ---> BAD RUNTIME                     **
        ** 5. CalculateShortestPath        - DIJKSTRA ALGORITHM             : for quicker runtime                                                       **
        ** 6. Travelling salesman problem - Brute Force IMPLEMENTATION                                                                                  **
        ** 7. Travelling salesman problem - 2 OPT Heuristic IMPLEMENTATION :                                                                            **
        ** 8. Exit                                                                                                                                      **
        **************************************************************************************************************************************************

 6
 **************************************************************
 * 6. Travelling salesman problem - BRUTE FORCE
 **************************************************************

 In this task, we will select N random points on the map and you need to find the path to travel these points and back to the start point.

 Please input the number of the places:10
```



```
**************************************************************
* 6. Travelling salesman problem - BRUTE FORCE
**************************************************************

In this task, we will select N random points on the map and you need to find the path to travel these points and back to the start point.

Please input the number of the places:10
Calculating ...
Time taken by function: 0.314835 seconds
*************************Results******************************
[ERROR:0] global /home/cs104/Desktop/TrojanMap/opencv/modules/videoio/src/cap.cpp (563) open VIDEOIO(CV IMAGES): raised OpenCV exception:

OpenCV(4.5.1-pre) /home/cs104/Desktop/TrojanMap/opencv/modules/videoio/src/cap images.cpp:253: error: (-5:Bad argument) CAP IMAGES: can't find
i in function 'icvExtractPattern'


6042978413
6813565312
6788102190
3663661787
6816288746
4012792182
6807241418
4015492465
6813379567
1878000349
6042978413
**************************************************************
The distance of the path is:4.59871
```

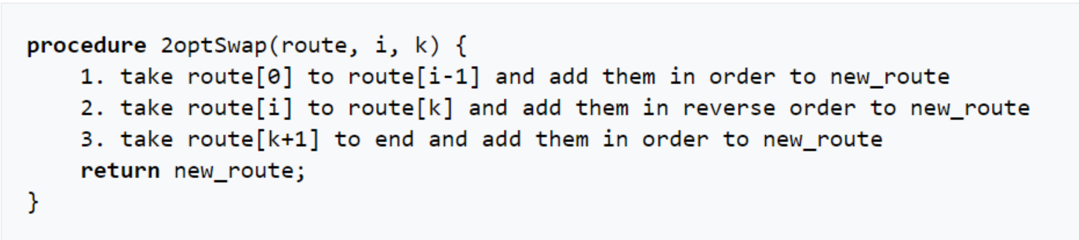


Generated Travelling Salesman Graph

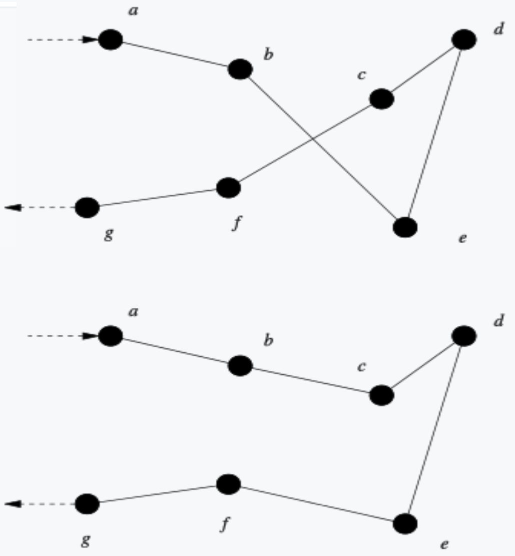
TRAVELLING TROJAN(2-OPT HEURISTIC)

Returning Position (Latitude and Longitude) for a given Nodes

- This method is a heuristic one as we keep swapping the nodes till the time there is no improvement.
- The time complexity is: O(n^2)
- Time taken of large sets of input locations is very less compared to that of brute force.



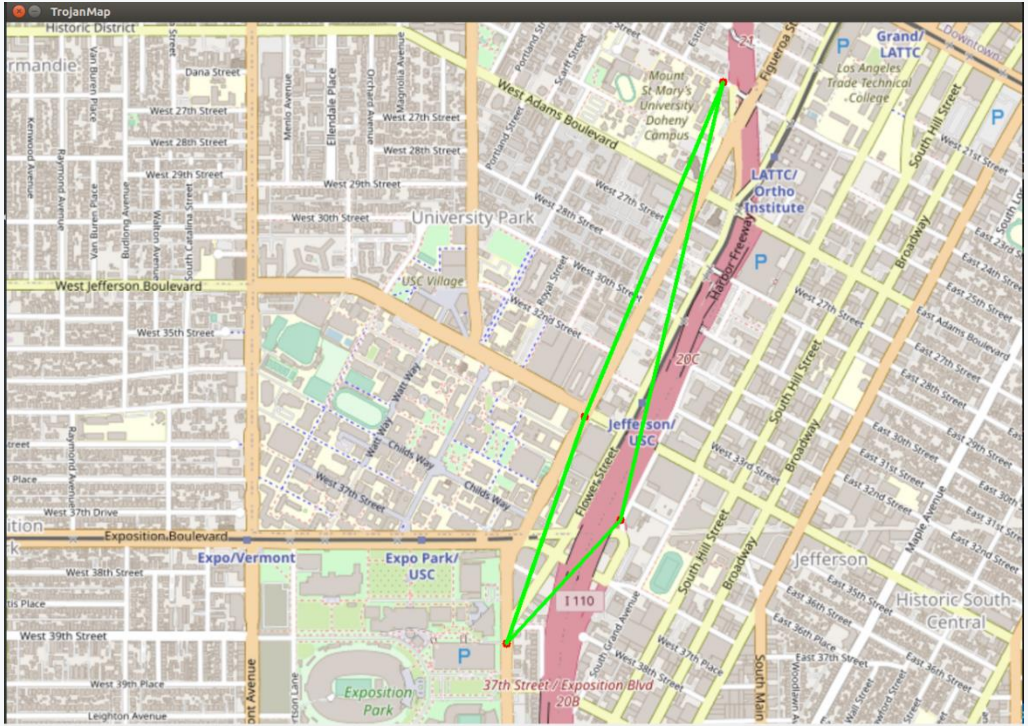
```
procedure 2optSwap(route, i, k) {
    1. take route[0] to route[i-1] and add them in order to new_route
    2. take route[i] to route[k] and add them in reverse order to new_route
    3. take route[k+1] to end and add them in order to new_route
    return new_route;
}
```

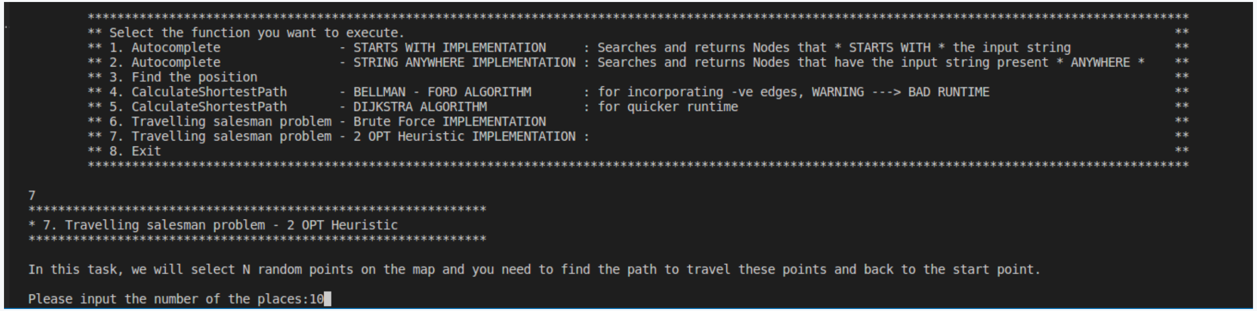


Ref: https://en.wikipedia.org/wiki/2-opt

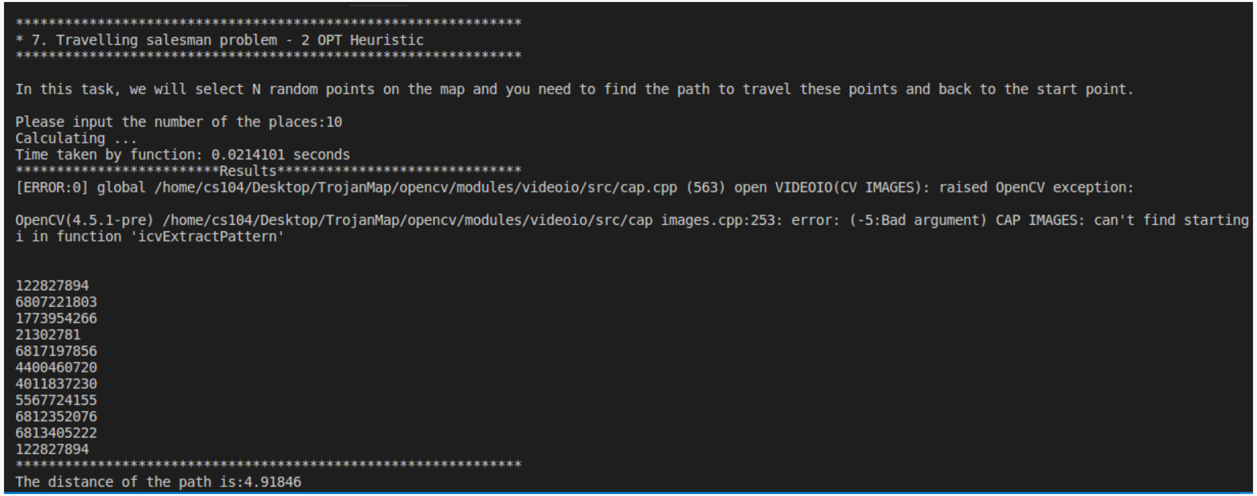
TRAVELLING TROJAN(OUTPUT – 2_OPT HEURISTIC)

- Below is the output of our one such implementation with number of locations as ‘4’.

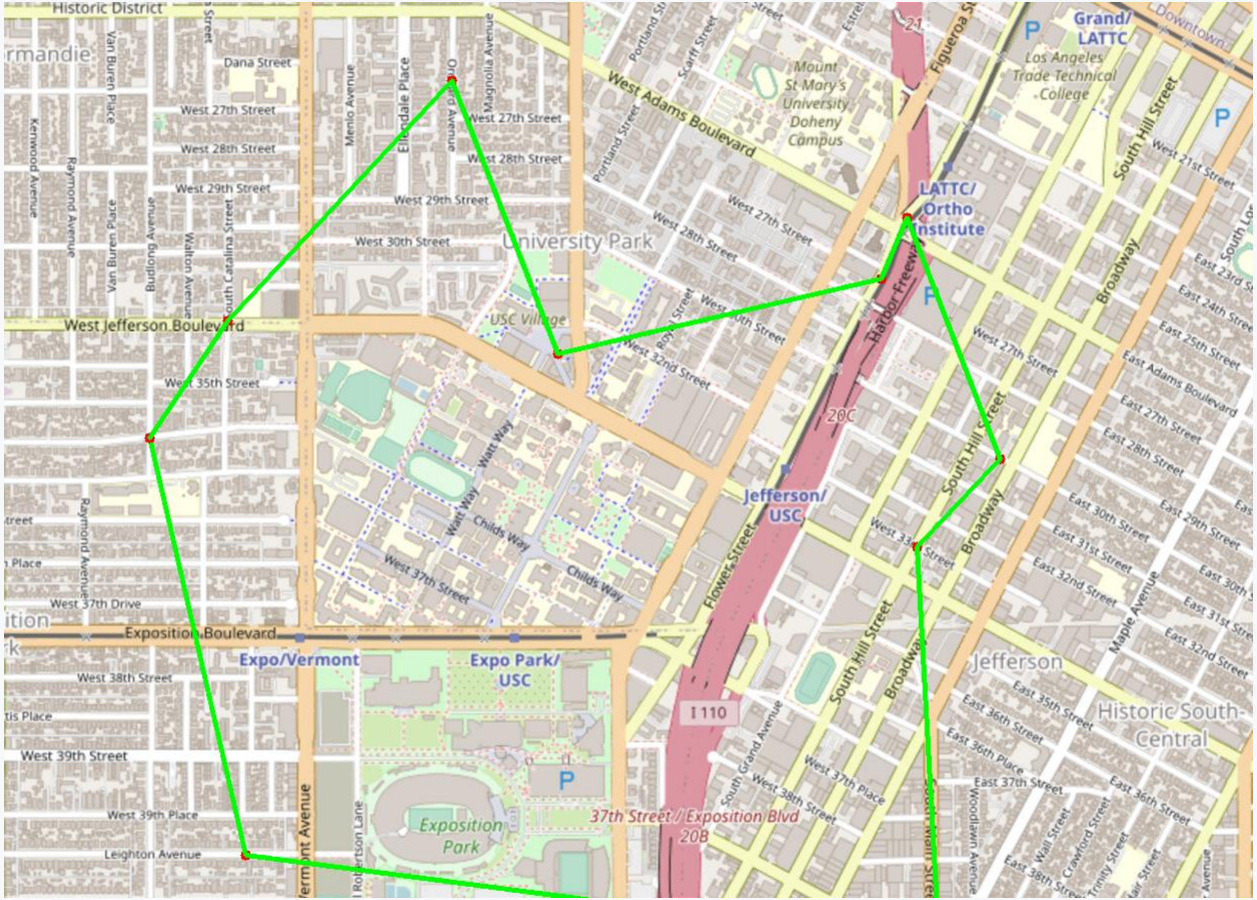




```
        ****************************************************************************************************
        ** Select the function you want to execute.                                                       **
        ** 1. Autocomplete                - STARTS WITH IMPLEMENTATION     : Searches and returns Nodes that * STARTS WITH * the input string          **
        ** 2. Autocomplete                - STRING ANYWHERE IMPLEMENTATION : Searches and returns Nodes that have the input string present * ANYWHERE *   **
        ** 3. Find the position                                                                           **
        ** 4. CalculateShortestPath       - BELLMAN - FORD ALGORITHM       : for incorporating -ve edges, WARNING ---> BAD RUNTIME                  **
        ** 5. CalculateShortestPath       - DIJKSTRA ALGORITHM             : for quicker runtime          **
        ** 6. Travelling salesman problem - Brute Force IMPLEMENTATION                                    **
        ** 7. Travelling salesman problem - 2 OPT Heuristic IMPLEMENTATION :                              **
        ** 8. Exit                                                                                        **
        ****************************************************************************************************

7
**************************************************************
* 7. Travelling salesman problem - 2 OPT Heuristic
**************************************************************

In this task, we will select N random points on the map and you need to find the path to travel these points and back to the start point.

Please input the number of the places:10
```



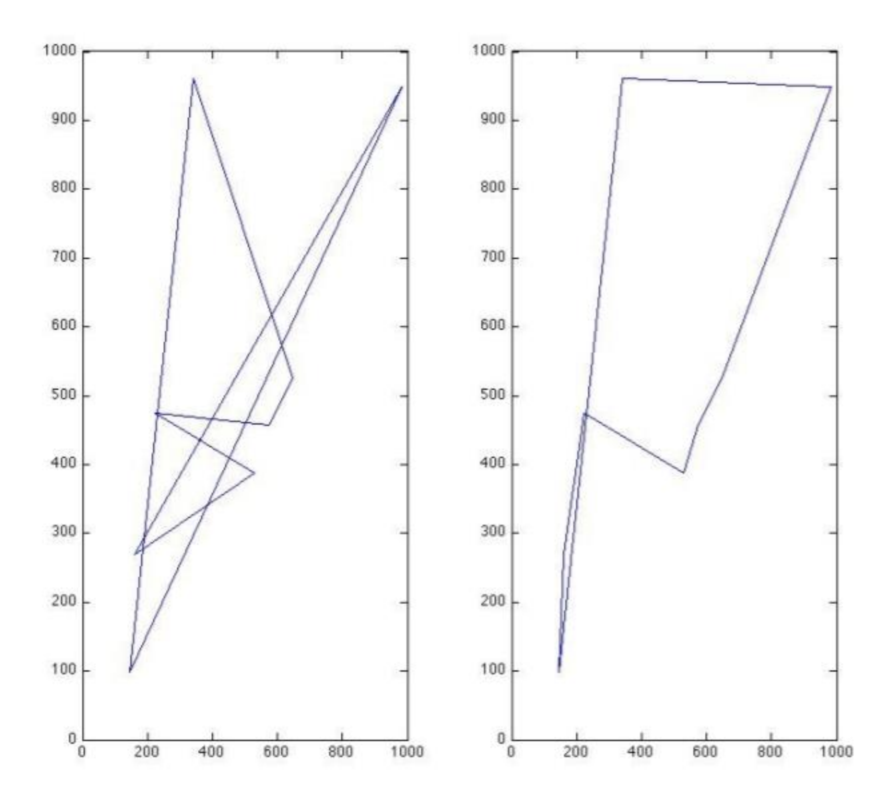
```
**************************************************************
* 7. Travelling salesman problem - 2 OPT Heuristic
**************************************************************

In this task, we will select N random points on the map and you need to find the path to travel these points and back to the start point.

Please input the number of the places:10
Calculating ...
Time taken by function: 0.0214101 seconds
*************************Results******************************
[ERROR:0] global /home/cs104/Desktop/TrojanMap/opencv/modules/videoio/src/cap.cpp (563) open VIDEOIO(CV IMAGES): raised OpenCV exception:

OpenCV(4.5.1-pre) /home/cs104/Desktop/TrojanMap/opencv/modules/videoio/src/cap images.cpp:253: error: (-5:Bad argument) CAP IMAGES: can't find starting
i in function 'icvExtractPattern'


122827894
6807221803
1773954266
21302781
6817197856
4400460720
4011837230
5567724155
6812352076
6813405222
122827894
**************************************************************
The distance of the path is:4.91846
```



Generated Travelling Salesman Graph

TRAVELLING TROJAN(COMPARISON: BRUTE FORCE VS 2_OPT)

Returning Position (Latitude and Longitude) for a given Nodes



Ref: http://cs.indstate.edu/~zeeshan/aman.pdf

Comparison:

Shortest Path – Bellman Ford	127 sec
Shortest Path – Dijkstra Algorithm	0.2 sec

Travelling Trojan – Brute Force	0.314 sec
Travelling Trojan – 2 OPT	0.02 sec

Thank You

SAFE TOGETHER

TROJAN FAMILY

WE FIGHT
AS ONE

USC University of Southern California