EE599 TrojanMap

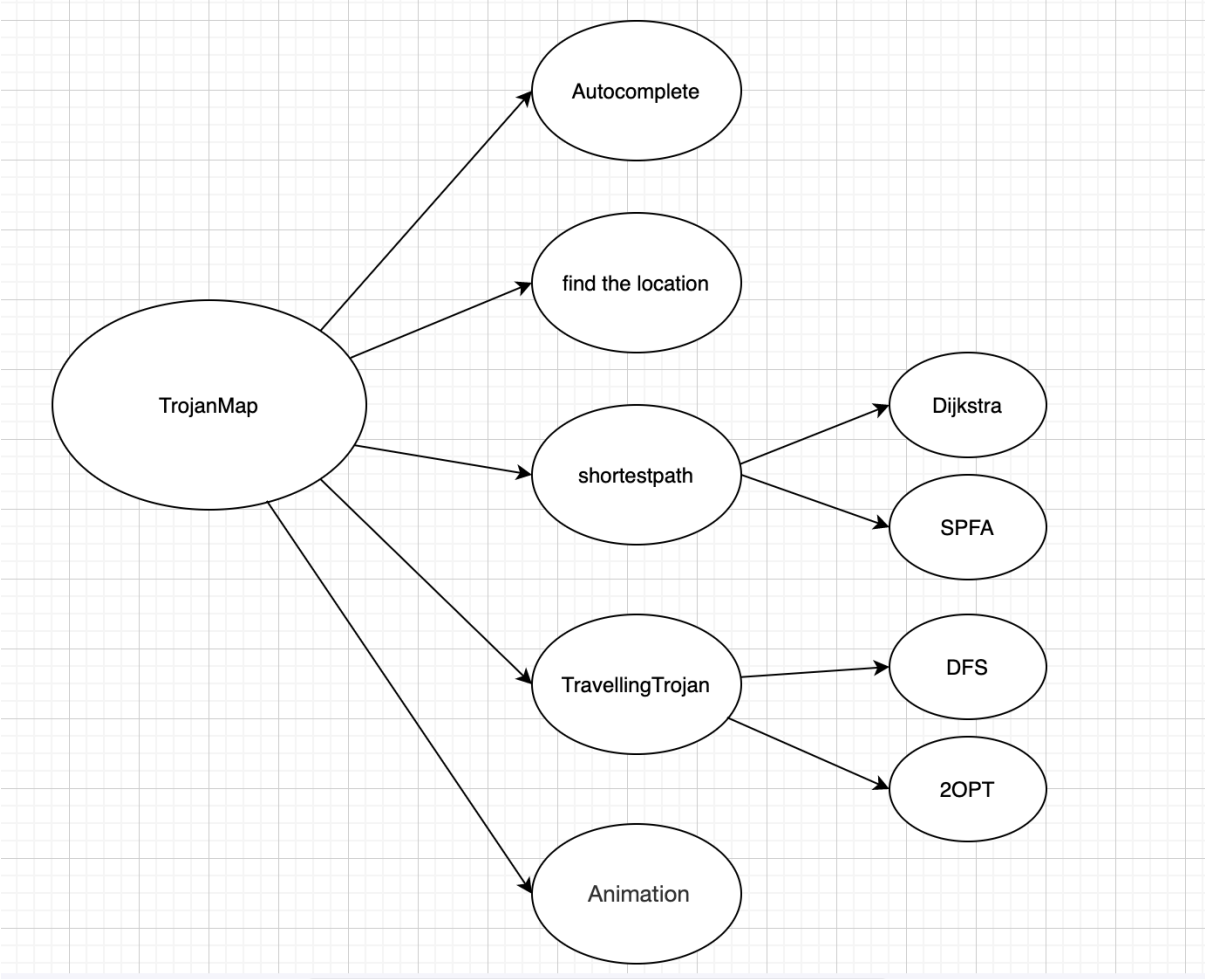
Team: Please AK

Team member :Yuhang Lu

USCID: 4683352754

Decription

- This project mainly focuses on graph algorithm. Based on an area around USC, I implemented four main functions (`autocomplete`, `findthelocation`, `findshortestpath` and `TravellingTrojan`)



  - autocomplete:
    - If you input some letters, I will output all the location name whose prefix is same as your input. Another thing need to notice is that It is case insensitive,
    - I used map to store all those nodes. And since it is case insensitive, I need to travese all the nodes to check if they match. So the time Complexity is $O(n)$
  - findthelocation

        - If you input a location name, I will output the location's latitude and longtitude. Meanwhile, I will circle the place in the map and show it to you. It is case sensitive.
        - I used map to help with me. Since it is case sensitive, I could just make use of the Map data structure. And the time complexity is $O(logn)$.
            - Since the map is small, there are around 4000 nodes. I prefer to use map instead of hash map. Hash map has a large constant running time.
    - ShortestPath
        - If you input two location name start and end, I will output shortest path for you and meanwhile, I will annote the path in the map and show it to you.. It is case sensitive.
        - Dijkstra
            - It is an algorithm based on greedy. Everytime, you find the closest node which is in the view of the updated nodes to update other nodes
            - the time complexity is $O(V^2)$ and of course you could use a heap to optimize the process of finding the closest node and the time complxity would be $O(VlogV)$. While since it is a small map, $O(V^2)$ is also acceptable.
        - SPFA(Shortest Path Fast Algorithm)
            - It is a new algorithm not covered in class but it is a very good one. It is kind of an improvement of Bellman-Ford algorithm. SPFA uses a queue to optimize the process.
            - The average time complexity is $O(E)$ which is a big improvement from Bellman-Ford algorithm. But, in the worst case, it is same as Bellman-Ford algorithm $O(VE)$
            - here is the pseudo-code from wikipedia

```
procedure Shortest-Path-Faster-Algorithm(G, s)
 1    for each vertex v ≠ s in V(G)
 2        d(v) := ∞
 3    d(s) := 0
 4    push s into Q
 5    while Q is not empty do
 6        u := poll Q
 7        for each edge (u, v) in E(G) do
 8            if d(u) + w(u, v) < d(v) then
 9                d(v) := d(u) + w(u, v)
10                if v is not in Q then
11                    push v into Q
```

    - TravellingTrojan
        - It is a very classic problem called travelling salesman and It is a NP problem. The function would randomly pick several nodes and I will output the path which covers all the locations once and goes back to the start point.
            - The graph here is a complete graph which means every node is connected to other nodes

        - DFS
            - I use pruning to optimize my algorithm. If we just try brute-force algorithm, It will check all the possible path and the time complexity is $O(n!)$ which is terrible. But in some cases, if we just iterate part of nodes and the total distance is larger then the current optimal solution, this kind of case will not be able to optimize the results. And we could just return.
            - But in the worst case, the time complexity is still $O(n!)$
        - 2opt
            - It is a local search algorithm, we just keep swaping the nodes until no improvement is made.
            - The time complexity is $O(n^2)$ but we should notice that it is a location search algorithm which means sometimes we could just find the local optimal result instead of the global optimal one
    - Animation
        - OpenCV is a very powerful tool. And TAs provide us some APIs, we could make use of these APIs.

Results

runtime

Function	Seconds
Autocomplete	4.7e-05
Gettheposition	2.9e-05
Dijkstra(Ralphs->Target)	0.136291
SPFA(Ralphs->Target)	0.046358

We could find those four functions are super fast.

Comparison between Dijkstra and SPFA

- Dijkstra's running time is almost three times SPFA's. This is because Dijkstra's running time is $O(V^2)$ and SPFA's is $O(E)$ but SPFA needs a queue to help.
- And of course, their results are the same.

Travelling Trojan	DFS	2opt
N = 3	5.2e-05s	8.5e-05s
N = 5	0.000375s	0.000528s
N = 8	0.012844s	0.002913s
N = 10	0.400187s	0.012082s
N = 11	2.11063s	0.010827s
N = 12	9.28311s	0.012624s
N = 14	261.776s	0.025028s

Comparison between DFS and 2opt

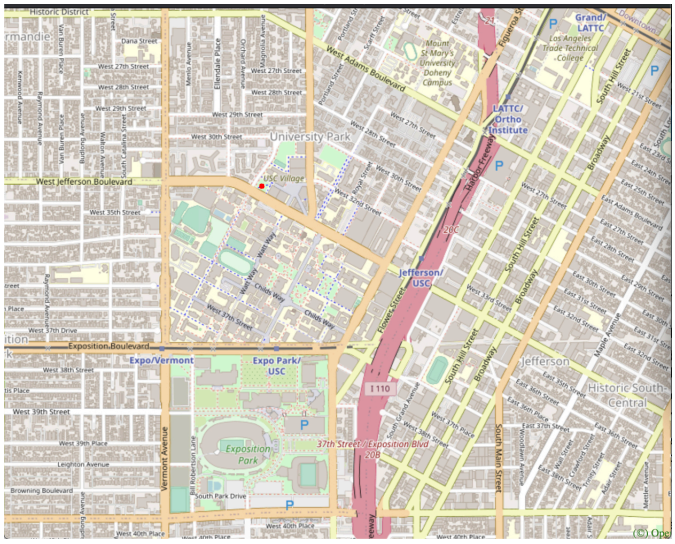
- We could find that when n is 5 dfs runs faster than 2opt, this is possible because of different constant running time.
- when n is increasing, running time of dfs is keep increasing and increasing speed is very fast. We could find that when N = 14, the running time is 261s which is not acceptable. While, for 2opt, the increasing speed is slow. This is because time complexity for DFS is $O(n!)$ and that for 2opt is $O(2n)$
- DFS could always find the optimal result but 2opt cannot.

DEMO

- Autocomplete
  - Input "la" or "LA" or "lA" or "La"
  - output

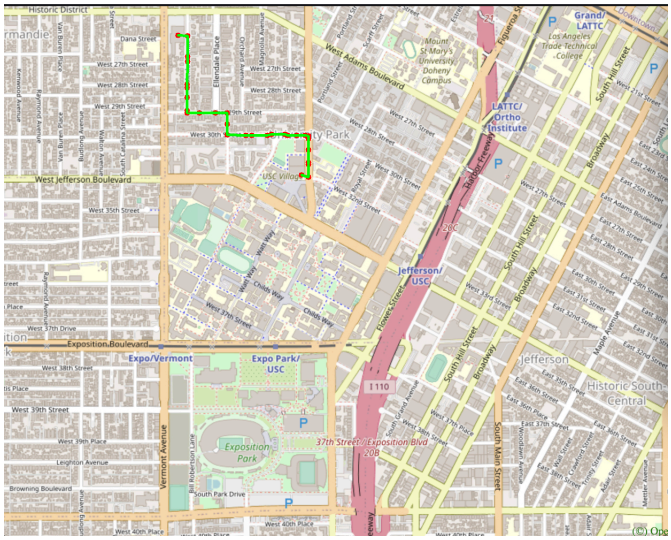
```
LATTCOrtho Institute
Lawn
```

- find the position
  - Input "Lawn"
  - output

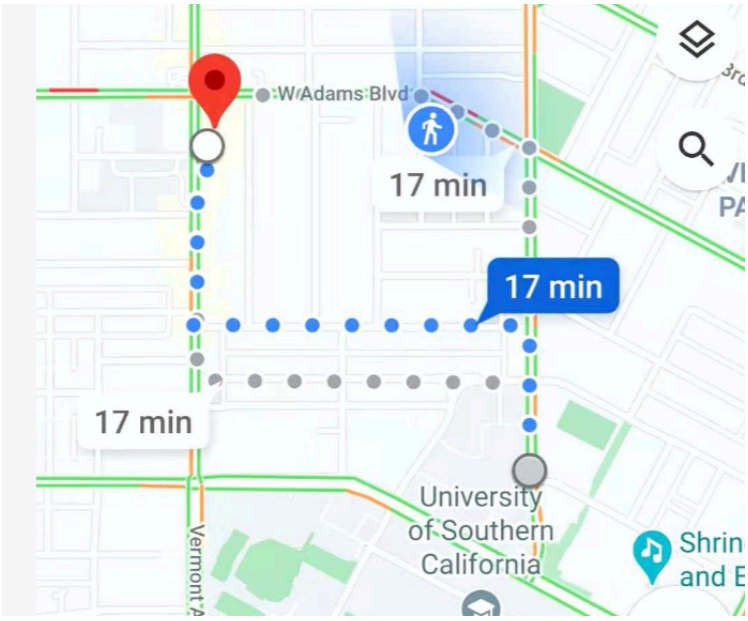



```
Latitude: 34.0253 Longitude: -118.286
```

  - Input "lawn"
  - Output: (-1, -1) it is case sensitive
- shortest path
  - Input: "Target" to "Ralphs"
  - both algorithms' output



google map's output



it is slightly different from out algorithm's output. And google map would take real-time traffic status into consideration.

- Travelling Trojan
  - The demo results are in `Result` part above.
- Lots of other test cases are shown in `trojanmap_test_student.cc`
- when we try to test our code, we should check the logic first. It is not good to try different cases to update our code because not all the time we could cover all the corner cases.

Highlight

- Since all the information are stored in Node structure, if we want to get some information we need to use map to find the node and then get the information. If we do all these things in algorithm, it will slow down our algorithm. And actually, those functions doesn't care those things so we could map those nodes to integers in the begining and when the algorithm is done, just convert them back.
  - ```
    std::map<std::string, Node> name2node;
    std::map<std::string, int> id2num;
    std::map<int, std::string> num2id;
    std::vector<std::vector<double>> G;
    ```
  - With the help of some Map structures, I could easily convert nodes to corresponding integer index. And I could store the distance in the 2-D vector which is what the function really needs.
- I used a new algorithm SPFA(shortest path fast algorithm) which is faster
- I used pruning to optimize DFS.
- I change the menu so that you could easily choose which algorithm to use

```
*1.Dijkstra
*2.spfa
```
```

```
**********
*1.DFS
*2.2opt
```