Pradyun Sathish

Winter 2024

**Bruin Tour Project Report**

***load():***

The Big-O for the *load()* is **O(N \* Q)**. Assuming there are *N* number of street segments with each segment having a max number *Q* amount of POIs, and considering that the time complexity to insert street segments and connected points is about O(1) per insertion, the total time complexity for this process would be O(N \* Q).

***get\_connected\_points():***

The Big-O for *get\_connected\_points()* is **O(N)**. The time complexity for the *find* function in the *hashmap* is mostly around O(1). If we find the GeoPoint in the hashmap, the time complexity to copy the vector over depends on its size. Given that the vector has *N* amount of GeoPoints in it, the time complexity for copying the vector would be O(N). The total time complexity comes out to O(N) \* O(1) which is O(N).

***get\_street\_name():***

The Big-O for *get\_street\_name()* is **O(1)**. The time complexity for the *find* function to find the vector of *Segments* is O(1). If the *find* function returns true, then looking up the street name and returning it also has a time complexity of O(1).

***route():***

I implemented my *route()* function using the A\* algorithm. Firstly, I had a struct *GeoPos* that contained a GeoPoint and an int variable to hold the *f cost*. The data structures I used were a priority queue and 2 hashmaps. The priority queue, *openList* was used to contain *GeoPos* objects that were yet to be explored. Every time I was at a certain location, I would add the neighbors of that location to *openList*. One of the hashmaps, *closedList*, held *GeoPos* values that had already been explored, in other words, we have taken note of all its neighbors. The other hashmap, *locationOfPrevPoint*, held the GeoPoint of where we came from, or where we were before coming to the one we are currently evaluating. I also used a vector, *routeTrack*, to keep track of the GeoPoints on my route.