Joel George

Project 4: Report

**MyMap:**

Associate():

If MyMap holds N values, the time complexity of this function is O(log(N)), as the function uses a binary search to find the correct location to insert the node.

Find():

Again, if MyMap holds N values, the time complexity of this function is O(log(N)), as the function uses a binary search to find the desired key.

**AttractionMapper:**

Init():

If N is the number of StreetSegments that MyMap holds and A is the total number of attractions in MyMap, the time complexity of this function is O(N + A\*log(N)). The loop iterates through every StreetSegment, searching for attractions to associate into my MyMap (thus the N). In the end, this function ends up looping through every attraction available as well, and calling the associate function every loop.

getGeoCoord():

This function simply calls MyMap’s find function and returns a pointer based on the return value of the find function, so the time complexity of this function is just O(log(N)), N being the number of attractions in the MyMap.

**SegmentMapper: *init*(), *getSegments*()**

Init():

The time complexity of this function is O(N\*log(N) + A\*log(A)), where N is the total number of StreetSegments and A is the total number of attractions. For every StreetSegment, this function calls the associate and find functions, and for every attraction, this function also calls the associate and find functions. Since both of these have a time complexity of log(*H)*, *H* being the total number of objects that you’re iterating through, and since they are called every iteration through N and A, this results in the time complexity stated above.

getSegments():

Just like getGeoCoord(), this function simply calls MyMap’s find function and returns a vector based on the return value of the find function, so the time complexity of this function is just O(log(N)), N being the number of StreetSegments in the MyMap.

**Navigator:**

Navigate():

Navigate’s average time complexity is incredibly hard to calculate, mainly because an average time complexity depends on what an average “path” is defined as; even if somewhat average distance may not equate to an average path, because a curvy road of distance S may have many more Street Segments than a straight road of distance S. In essence, there doesn’t seem to be an obvious average number of segments per path, and that is the main constraint of this functions time complexities. The best I can provide is a worst case scenario, which is O(N\*S\*A), N being the total number of “nodes”, or GeoCoordinates in the map, S being the total number of StreetSegments in the map data, and A being the number of attractions in the map data file. Essentially, what this would look like is a path that goes through every section of road on the map, starting at an intersection through which ***every*** street in the map runs. Obviously, this is the worst case scenario. The best case scenario is much better, simply because these absurd geographical features aren’t found in the Westwood map data file.