Dean Jones

005-299-127

**Project 4 Report**

**Big O**

StreetMap

load():

Runs in O(n) time, where n is the number of lines in the input mapping data file. This is because the code must traverse over every line, to get the street segment coordinates and its street name. Inserting each street segment into the ExpandableHashMap takes O(1) in the average case, so with roughly 2n insertions to make (one for the street segment in the map and one for its reverse version) Big O evaluates to O(n).

getSegmentsThatStartWith():

Runs in O(1) typically. This calls the ExpandableHashMap’s find() function, which on average takes O(1) time to return a pointer to the vector of StreetSegments. If it isn’t nullptr, then true is returned and the input vector<StreetSegment> takes on the value the pointer refers to; otherwise, false is returned and the vector is unchanged.

PointToPointRouter

generatePointToPointRoute():

As I implemented the A\* algorithm, I will describe the data structures I used instead of attempting to reveal the Big O. I used an open list and closed list with objects of the struct PointNode. The struct held a certain GeoCoord, a pointer to its parent PointNode, and the different costs associated with it (f cost, g cost, and h cost). The open list took on neighbors of pointers that were explored, representing those points that should be examined in more detail, while the closed list represented all locations that had already been traversed. A vector of StreetSegments is necessary for getting the routes to neighboring points for a GeoCoord.

DeliveryOptimizer

optimizeDeliveryOrder():

I used simulated annealing, so the Big O will not be discussed. New vectors of DeliveryRequests were generated from the initial input parameter, reversing the elements between randomly chosen indexes, to check whether their overall crow distance was less than the initial crow distance.