Bruin Nav

Austin Guo

Project 4 Report

Time Complexities

MyMap:

*associate()*

If the MyMap binary search tree holds N nodes, and each key is only associated with one value, then *associate()* is **O(log N).** This is because to search through the sorted unbalanced binary search tree, the best case is O(1) and the worst case is O(N), but the average case time complexity is O(log N) to search through N nodes that only have two child nodes each. Further, my insert is constant time O(1), creating a new node, taking the node pointer and linking it to this new node, then returning out.

*find()*

If the MyMap binary search tree holds N nodes, and each key is only associated with one value, then *find()* is **O(log N).** This is because to search through the sorted unbalanced binary search tree, the best case is O(1) (the node to find is the root node) and the worst case is O(N) (all the nodes are on different levels with only one node per level), but on average the search will run through log N nodes. This is due to the fact that if there can only be 2 child nodes per node, then there are only 2 possible paths to take, therefore the number of nodes in each level of the tree is double of the previous level. Doubling the number of total nodes in the tree will then only increase the level of the binary search tree by one (on average), and this will only require one more node traversal to find the node. Thus, the search is of time complexity O(log N).

AttractionMapper:

*init*()

If the number of segments returned by MapLoader’s getNumSegments is given by N, and the total number of Attractions on the entire map given by mapdata.txt is A, and the length of each string is S on average, then *init()* will run on a time complexity of **O((N + A) log (A)).** It’s important to note that the S term is not considered in this final evaluation of time complexity because on average, it will be insignificant in magnitude compared to N and A and will thus be ignored in the final time complexity. Further, the time complexity of (N + A) arises because we must iterate through all N StreetSegments in the vector with size given by N, and at each position in this vector we must loop through some fraction of A, A being the *total* number of Attractions, not the average number per StreetSegment. Thus, to loop through all the Attractions on some arbitrary StreetSegment, it requires a time complexity of O(N + A/something), since it must loop through N elements to find the StreetSegment, then it must loop through A Attractions to access them. If we make it so that every StreetSegment will at some point play the role of “some arbitrary StreetSegment” in the explanation above (i.e. run through a loop of street segments), then summing all these time complexities we get O(some constant \* N + (A/something + A/something else + … A/something else)) = O(some constant \* N + A total attractions looped through) + O(N + A). However, we are not done at this point: my init method does not simply perform constant time operations every time it iterates through the A Attractions, but rather calls *associate()* from MyMap, which has its complexity described above as O(log (A)). Therefore, this O(log(A)) operation is performed O(N+A) times, and the time complexity is **O((N+A) log(A)).**

*getGeoCoord*()

Let the average length of input string parameter be S, and the total number of Attractions be A. Then, my *getGeoCoord()* function runs in **O(log (A))**. This is because on average, the length of each input string of attraction name will be very small compared to the number of Attractions in the MyMap private variable in AttractionMapper, so converting the string to lowercase is a negligible operation in the final time complexity. However, to find the string attraction name in the MyMap private variable, I traverse the sorted binary search tree that is MyMap. Because it’s sorted, binary search can be used and any element can be found in log(A) time. Therefore, the total time complexity is O(log(A)).

SegmentMapper:

*init*()

Letting the number of StreetSegments in the MapLoader be N, and A be the total number of attractions summed from those streets, the time complexity is **O((N+A)log(N+A))**. This is because my *init()* function iterates through all N of the StreetSegments and performs operations of O(logN) and O(log (N+A)) per iteration. The N+A comes from the fact that the number of GeoCoords that will eventually end up in SegmentMapper’s MyMap is N+1+A, since StreetSegments cannot be recounted and are connected by starts and ends (there will only be the shared start and ends equal to the number of streets minus one, then plus two more for the unshared start and end of the end streets). Note that the constant 1 term gets dropped because time complexity doesn’t include constant operations (O(N+1+A) = O(N) + O(1) + O(A) = O(N+A)). However, since each iteration has an operation of O(log (N+A)), the final time complexity will ignore O(log N) operations that have less effect on the final time complexity. Further, per iteration we will loop through some fraction of the total Attractions A, and perform log(N+A) operations per iteration in the loop. For the same reasoning as above, if per iteration of N we will loop through some fraction of A with each loop of A performing operations of log(N+A), the sum of all these iterations will get a term of order N+A log(N+A). Then, to conclude and bring all mentioned elements together, the previously mentioned O(log (N+A)) operations run on all iterations of N, and since A is the total number of attractions it will be equivalent to running A log(N+A) only once out of the N operations when they are summed, and the total final time complexity is O(N log(N+A) + A log(N+A)). When simplified (we can factor out the log) we get O((N+A) log(N+A)).

*getSegments*()

Letting N be the number of StreetSegments from MapLoader and A be the total number of attractions across these N StreetSegments, we get the time complexity of *getSegments()* to be **log(N+A).** This is because the total number of GeoCoords is N+A, as given by in previous explanations, since the number of StreetSegments + 1 is the total number of start and end GeoCoords, and each attraction A is a unique GeoCoord. Then, the function is composed of all O(1) operations except the MyMap find() function call on a MyMap of N+A GeoCoords. Thus, to perform binary search on N+A elements we get the time complexity to be log(N+A).

Navigator:

*navigate()*

Letting N be the number of StreetSegments in the MapLoader object, and A being the total sum of all the attractions in these N StreetSegments, the time complexity of navigate is **O((N+A)log(N+A))**. This is evident because I have broken up my navigate function into 3 parts, as it calls 3 helper functions to determine the path of GeoCoords, determine the StreetSegments and distances between these GeoCoords, and finally to turn these StreetSegments and distances into NavSegments, and the helper functions have time complexities of O( (N+A) log (N+A)), O((N+A) log (N+A)), and O(N) respectively. Adding these time complexities together, and only taking the most significant complexities, the total time complexity is just O((N+A) log (N+A)).

The first function *pathFind()* is O( (N+A) log (N+A)) complexity because it loops through a number of iterations to some fraction of N times, N being the number of streets. In each iteration, operations of log(N+A) are performed, such as associating and searching the binary search tree of MyMap, which are detailed and described above. The number of GeoCoords, which is what is keyed in the MyMap, is N+1+A with similar arguments as before, since the total number of streets encompasses overlapping GeoCoords, then adding 1 adds the last nonshared GeoCoord, and A accounts for all unique Attraction GeoCoords. Then, with similar logic as before, in each iteration a fraction of the attractions are looped through, performing association and search operations of log(N+A) in the MyMap of GeoCoords. Summed all together, these Attraction operations are summed to the initial N log (N+A) time complexity part of the function, which turns to N log (N+A) + A log(N+A) = (N+A) log (N+A). All other operations are of O(1) time or lesser complexity than previously described and will not be factored into the final time complexity.

Similarly, the second helper function loops through the GeoCoords of the path that pathfind() returns, which will be a fraction of N+1+A possible total GeoCoords. In each of these iterations, getSegments() is called, which has a time complexity of log(N+A) as there is some multiple of N+A GeoCoords in the SegmentMapper MyMap variable. Below this in my helper function I have nested loops that may have worst-case time complexity of O(N^2), but this is unrealistic as one street cannot possibly be connected to every other street in the map (which is where this worst case arises), and realistically can only be connected to adjacent streets and thus will be negligible when determining total time complexity (these iterations may run up to S^2 times, with S being less than 4 or 5 most of the time and in our mapdata file being only 2).

Further, in the last helper function I already have all NavSegments mapped out, so I only have to iterate through the multiple of N NavSegments and perform O(1) computations. This total helper function complexity is just O(N).

Thus, the total time complexity is as specified, as O((N+A) log (N+A)). This is the average case complexity, but usually it will run much faster by finding the best path with an adaptation of the A\* path-finding algorithm.