**CS32 Project 4 Report**

**Akshay Smit**

**804641231**

Note: For the navigation, the A\* algorithm along with a heuristic has been implemented. The heuristic is the great-circle distance in miles between two points on Earth.

**template<typename KeyType, typename ValueType>**

**void MyMap<KeyType, ValueType>::associate(const KeyType &key, const ValueType &value)**

Note that this function delegates all the work to the helper function associateAux. The auxiliary function does O(1) amount of work before making one recursive call. If the BST had N items in it already, then assuming the data was randomly distributed so that the tree would not be too far out of balance, the depth of the call tree would on average be log2 (N). So, the overall time complexity is just O(log N) on average.

**template<typename KeyType, typename ValueType>**

**const ValueType\* MyMap<KeyType, ValueType>::find(const KeyType &key) const**

Note that this function delegates all the work to the helper function findAux. The auxiliary function does O(1) amount of work before making one recursive call. Once again, each recursive call ignores one entire subtree, so that the average depth of the call tree would be log2 (N) for randomly distributed data. So the overall time complexity is O(log N) on average.

**void AttractionMapperImpl::init(const MapLoader& ml)**

The innermost loop converts a string from uppercase to lowercase. But no attraction is likely to have a name that is longer than, say, 50 characters, So the innermost loop does at most O(1) amount of work in total. The call to MyMap’s associate function is O(log B) where B is the number of attractions in the MyMap object. Now for any StreetSegment, the second loop executes once for each attraction on that StreetSegment. The outermost loop runs for each StreetSegment and does O(1) amount of work by calling MapLoader’s getSegment. So if N is the number of StreetSegment objects in the mapdata.txt file and A is the total number of attractions, the overall time complexity is:

O(N) + O(log(A-1)) + O(log(A-2)) + O(log(A-3)) + … = O(N) + O(log A! ) = O(N) + O(A log A) = O(N + A log A).

**bool AttractionMapperImpl::getGeoCoord(string attraction, GeoCoord& gc) const**

This function first converts the attraction name to lowercase, but as before this is at most O(1) as no attraction is likely to have a name with over 50 characters. Then this function does O(1) amount of work and also makes a call to MyMap’s find function. So if A is the total number of attractions in the mapdata.txt file, this function has complexity O(log A).

**void SegmentMapperImpl::init(const MapLoader& ml)**

The innermost loop does O(1) work before calling MyMap’s find function. If A is the total number of attractions and N the total number of street segments in mapdata.txt, then the call to MyMap’s find function is certainly O(log (N+A)), because each GeoCoord must correspond to the start/end of a StreetSegment or the location of an Attraction. Then either MyMap’s associate function is called, or O(1) work is done. But the associate function is O(log (N+A)) so the inner loop has complexity O(log (N+A)).

Before the inner loop is called, the outer loop first calls MapLoader’s getSegment function, which is O(1). Then MyMap’s find function is called, which is O(log (N+A)). After this, either O(1) amount of work is done or MyMap’s associate function is called, which is O(log (N+A)). Then the inner loop is called, but note that the inner loop runs A times in total.

So the overall time complexity is O(N log(N+A)) + O(A log (N+A)) = O( (N+A) log (N+A) ).

**vector<StreetSegment> SegmentMapperImpl::getSegments(const GeoCoord& gc) const**

This function does O(1) work and calls MyMap’s find function. As there at most 2N+A distinct GeoCoord objects, where N is the total number of street segments and A total number of attractions in mapdata.txt file, this function has complexity O( log (N+A)).

**NavResult NavigatorImpl::navigate(string start, string end, vector<NavSegment> &directions) const**

Let A be the total number of attractions and N the total number of street segments in mapdata.txt.

The function calls AttractionMapper’s getGeoCoord two times, and this is of complexity O(log (A)) Then the helper function pathFinder is called, which implements the A\* algorithm.

In the pathFinder function, O(1) work is done before the while loop gets executed. This is because initially the priority\_queue and the MyMap object were empty, so inserting a single item is just O(1). Inside the while loop, looking up the top item in a priority\_queue is just O(1), while removal of the top item is O(log T) where T is the number of items in the priority\_queue.

The subsequent if statement checks whether the final destination has been reached, and the code inside this statement gets **executed at most once**. When this code is executed, the path is traced back from the destination back to the start, using a while loop. Each iteration of the loop calls MyMap’s find function, which is O(log R), where R is the total number of vertices encountered by the algorithm. Since the A\* algorithm with a heuristic is being used, R is actually much smaller than the total number of GeoCoord objects in the graph (which is at most 2N+A). So the complexity is O(R log (R)) as the while loop gets executed O(R) times.

O(R log (R)) = O((N+A) log (N+A)) but note that R is actually much smaller than N+A because the A\* algorithm evaluates far fewer vertices than the total number of vertices. So the constant of proportionality is very small.

Next, a call to SegmentMapper’s getSegment function is made, which is O(log (N)). After this, a loop iterates through all the segments associated with the current GeoCoord object. Now the most work is done if the loop has to call MyMap’s find function and associate function, which are both O(log (N+A)), but with a very small constant of proportionality due to the same reason mentioned above. In addition the loop calls the isDestOnStreetSegment function, which loops through all attractions on the current StreetSegment. It is highly unlikely that a single street will have, say, more than 100 important attractions. Even Google Maps does not show 100 attractions on a single street. So the amount of work done by isDestOnStreetSegment is bounded above by a constant, so it is O(1). Lastly, a new item is added to the priority\_queue, and this is O(log T) where T is number of items in priority\_queue. T is certainly bounded by 2N+A, so this is O(log (N+A)) although in reality T is much smaller than 2N+A because of the A\* algorithm.

Now note that this loop will, at most, loop through every StreetSegment in the graph, which would happen in the case that we examine all vertices in the graph. So the overall loop has time complexity O ((A+N) log (A+N)), but actually the constant of proportionality is very small because the A\* algorithm will not evaluate all vertices. In fact, it will evaluate far fewer vertices than the total number of vertices.

Finally, calls are made to MyMap’s find and associate functions, and items are added to the priority\_queue. There are no loops here, and so the complexity here is O ((N+A)\*log(N+A)) overall, again with a very small constant of proportionality.

Thus in total we see that although the complexity is O((N+A)\*log(N+A)), the constant of proportionality is very small. This is because the A\* algorithm does not usually check all vertices. It only checks very few vertices relative to the total number of vertices, at it is guided by a heuristic function and always chooses the minimum weight path.