Neil Patel Documentation

Rule of thumb: there are strings involved when inserting and checking. Comparing these strings takes up to O(n) time as n is the # of character in a string. But I will omit talking about this since it can get redundant.

What is the time complexity of each method in your implementation in the worst case in terms of Big O notation? [5 points]

Void PageRank(int n); The function prints the page rank of all pages after p power iterations in ascending alphabetical order. I iterate over the set with a for loop to print the website's name and the rank with 2 decimal places. The set contains "V" number of websites, where "V" is the number of websites in a set. Therefore, this is an O(V) time operation to print the information and even though this function is using space from the set, it is not creating any new space, so the space is O(1). The overall time and space complexity of this function is O(V) time and O(1) space.

Void insertEdge(string from, string to); This function pushes data into 3 containers. First, "map<string, vector<string>> list;" I push main graph information into an ordered map which is O(log v) in the worst case, where v is the number of elements currently in the map because std::map is a self-balancing binary search tree, has a logarithmic heigh. Also, I am pushing the value into a dynamic vector which has a worst-case time complexity of O(V) leading to resizing the vector where V is the number websites that are pointing to the key in the main map. Then in the if statement, the find operation has a O(log V) time operation in the worst case as it tries to find the "from" website in the map where there are V websites. The space complexity for this data structure will be O(n+m), where n is the total number of unique keys and m is the total number of elements across all vectors.

This function also pushed data into an outDegree map "map<string, int> outDegree" where the worst case for pushing data in here is O(log V) where V is the number of websites. This data structure will be useful for calculating the rank of every page. The worst-case space will be O(V), where V is the number of websites that will be pushed into the map. Essentially, every website will have to be pushed. I also used the find(to) function where the time complexity to find an element in the map is O(log V), where V is the number of elements currently in the map.

I also pushed into an ordered set "set<string> uniqueSites" to be able to print the data in an alphabetically order manner. I am inserting the "to" and "from" 2 times making the time complexity O(2 log V), which simplifies to O(log V), where V is the number of websites already in the set. This set is taking up worst-case O(V) space as I am pushing V number of websites, which is all of them into the set.

The overall time and space complexity for this function is O(log V) space, where V is the number of websites. Worst case time is O(e+2m) which simplifies to O(e+m), where e is the total number of edges and m is the number of unique websites.

Void calcRank(int p); In this function, I calculate the rank of every page, leading each page to have a page rank respectively given p power iterations. The parameters for "p" are (1 <= p <= 10000). Since p will always be greater than 1, we need to always initially calculate the rank of each page given p = 1. So, for each page, the rank is initialized to 1/number of unique websites. I did this using a for loop iterating through the to/from graph and storing the rank in a map data structure called pageRank (map<string, float> pageRank). The time complexity for calculating 1 power iteration here is O(V) where V is the number of unique websites, then inserting into the pageRank map is O(log V) where V is the number of unique websites and log because that is the height of the graph inserting into a red/black tree given V number of websites. Simplifies to O(V*log(V))

Then as the number of power iteration grows, I must calculate the new page rank for every power iteration. For this I use a while loop. For each while loop iteration, I calculate the new page rank for each page, and I do this calculation p times. Here is the step I take to calculate page rank when p > 1 below.

1. Iterate through the to/from graph.
    a. Time O(V), where V is the number of unique websites. O(1) space
2. For each key, calculate the rank by iterating through the vector of size "it->secont.size()" and store the value in a variable called total.
    a. Time O(outdegree(V)), outdegree(V) is the size of the vector. O(1) space
3. For each element in vector, add (previous rank/# of out-degrees) to total.
    a. Time is O(2 log V), simplifies to O(log V), where V is the number of unique websites. This is because we are retrieving values form the map data structure twice to divide them. Space O(1), no space used.
4. Afte the total has been calculated, store the new rank in a map "map<string, float> storage".
    a. Time is O(log V), where V is the # of unique websites and log V because I am inserting into an ordered map data structure. Space O(V), because I am inserting every unique website into the ordered map to store the new rank.
5. When the calculation for all keys is done, then update the pageRank map.
    a. Time is O(V*log V), where V is the # of unique websites. O(V) because the for-loop iterates through the storage map and then O(log V), as I update the pageRank ordered map data structure V times. Space O(1), non-used.

Therefore, the worst-case time is O(p * (n + n + m)) = O(p * (n + m)), where n is the number of vertices and m is the size of the vectors. The worst-cast space is O(V), where V is the number of vertices/websites, and the storage map is the only storage that I'm using.

What is the time complexity of your main method in your implementation in the worst case in terms of Big O notation? [5 points]

The main method does 3 things.

First it collects the data of from/to websites and inserts them into the graph calling the insertEdge function using a for loop. There are “n” number of from/to websites, making the for-loop time complexity O(n), where n is the number of times we are inserting. The insertEdge function has a worst-case time of O(e+m), where e is the total number of edges and m is the number of unique websites. Therefore, this part of the main function is O(n*(e+m)) time.

Secondly, the main function then calculates the page rank of all the websites. This calculation is done using the calcRank function. The time complexity for this is O(p * (n + m)), and space is O(V).

Lastly, the main function prints the page rank of all unique pages using the PageRank function. This function has a time complexity of O(V), where V is the # of unique pages this function needs to print.

Therefore, the worst-cast time is O(V * p * n * (n + m)). Since V and n are not the same thing, we cannot put them together, so this is the worst-cast time. The worst-cast space is O(4V) because I am using 4 different containers, simplified to O(V).

Describe the data structure you used to implement the graph and why. [2.5 points]

The data structure that I used to implement that graph was a map, that has a key of strings which were the website names and a value of vector that help strings. Instead of mapping from/to, I mapped to/from as the calculations of the PageRank became easier. For each website, we need the number of indegrees, which website is an indegree, and for those websites that are indegrees we need there outdegree. Having a graph data structure “map<string, vector<string>> list” store to/from data, make is so all the items stored in the vector are the websites that point to the key. This way we have most of the information and all we need now is the outdegree for each website. This data structure has an insertion time of O(log V + outdegree(V)) as I insert websites into the map O(log V) and in the vector which has O(outdegree(V)) in the worst-case. The space that this data structure takes up is O(n+m) where n increases linearly with the number of key/value pairs and m is the number of websites in each vector.

What did you learn from this assignment and what would you do differently if you had to start over? [2 points]

The coolest thing I learned form this project was how Google was founded on a question that revolutionized the way people search for information. I learned that to create something meaningful with tons of impact, you need to work on problems as complex as these. If I were to do this project over, I would try to use less space and make and make a function that gets the adjacent vertices to a given vertex in O(e) time where would the number of edges to that given vertex. This would make the graph map from/to vertices, which seems more pleasing as a graph structure to do more operation rather than just calculate the page rank.