Jai Patel

Describe the data structure you used to implement the graph and why?

The graph is represented using an adjacency list stored as unordered_map<string, vector<string>> graph. This unordered_map puts the key as a string which represents a page and its associated value is a vector of strings which represents a destination page that the key page links to. I used this because it saves space and provides constant time access to the connections.

I also used these two other unordered maps:

The unordered_map<string, int> out_deg is used to store the URLs with its own outdegree value. This unordered_map puts the key as a string and the value as an integer. This makes it quick and easy to know how many pages each page links to. I used this because it is easy to look up and saves space.

The unordered_map<string, double> rank keeps the current PageRank score for each page. This unordered_map puts the key as a string and the value as a double. Each page can be updated and accessed quickly using its URL as the key. I used this because finding the PageRank number is easier this way

What is the computational complexity of each method in your implementation in the worst case in terms of Big O notation?

The void addEdge(const string& fromEdge, const string& toEdge) method is used to add a link between two pages which is basically making a connection between those two pages. It first checks if the starting and ending pages of the link are in the graph already or not. If they do not, then it creates its entries for them. Since adding a link and checking if a page exists in an unordered_map is in constant time, then the complexity is O(1).

The void GetRank() method shows off by having a PageRank value for every page in the graph by assigning an equal rank to each page. It goes through each page in the graph and sets its starting rank, which takes O(V) time, where V is the total number of pages.

The string PageRank(int n) method is called first with n=1,where n is the number put in by the user and is the “no_of_lines” variable. It gets the rank values and then sorts them. Sorting the ranks takes O(V(log(V))), where V is the total number of pages. But, the PageRank is called with multiple iterations so it needs to repeatedly change each page’s rank based on the ranks of other pages that link to it. For each iteration, it resets

for each page, which takes O(V). Then, it distributes each page's rank across its outgoing links by iterating over each page and its connections, which takes O(V+E), where E is the total number of links between pages. If PageRank is run for p iterations, the complexity becomes O(p×(V+E)). After the iterations, it sorts the pages again, which adds O(VlogV) to the total complexity. So, the time complexity of PageRank is O(p×(V+E)+VlogV).

The destructor ~AdjacencyList is cleaning up the AdjacencyList object when it's done being used. It doesn't do any complex work because C++ automatically handles memory for the unordered_map and vector. This means the time complexity is O(1).

What is the computational complexity of your main method in your implementation in the worst case in terms of Big O notation?

The function reads the number of edges and the number of iterations, power_iterations, and then iterates over the input lines to read each edge. It calls the addEdge method to add these edges to the graph. The time complexity is O(no_of_lines), where no_of_lines is the number of lines that are read, and the addEdge method has the time complexity of O(1) since you are inserting into an unordered map. After, the function calls PageRank, which has a time complexity of O(power_iterations×(V+E)+VlogV), where V is the number of pages, E is the number of edges, and power_iterations is the number of times the rank calculation happens. The O(power_iterations×(V+E)) portion of the time complexity recalculates page ranks for all edges based on their links. The (V(log(V)) part is for sorting the ranks for the output.
So, the time complexity of this function is O(no_of_lines)+O(power_iterations×(V+E)+VlogV).

What did you learn from this assignment, and what would you do differently if you had to start over?

I learned from this assignment about how to implement and design a graph through an adjacency list using maps and vectors. I learned how to go through maps. I learned how to show what an adjacency list is and the time complexities. I also used the use of a graph and how to use it in my own way to get what task needs to get done.

If I had to start over, I would start the project earlier to give myself more time. I should have found a better way to go through the graph and have a smaller time complexity. I also should have made some more functions instead of putting most of my code into my PageRank function. I also should have done some more testing for my code to really grasp my function and see if everything checks out.