Page Rank Project

Kathleen Tiley, 17 April 2022

A. Data Structure

To implement the graph, I used an unordered map connecting the index of the webpage an edge is directed toward (the "to" vertex) to a vector of Edge objects. The Edge class I created includes data members for the index of the webpage the edge comes from (the "from" vertex) and the weight of the edge. To generate the vertex indices, I used another unordered map connecting the webpage names, as strings, to their integer indices. Lastly, I used a third unordered map to track the outdegrees of each vertex. I chose to use unordered maps so I could insert and access the elements in constant time. Although this meant I would need to sort the webpages alphanumerically before printing, I judged that this would reduce the overall time complexity of my program since I needed to make many inserts and accesses and only needed to sort once. Using maps also allowed me to simplify my code by using the class's insert function when I needed to insert only if the element was not in the map and the [] operator when I wanted to insert or update the element. Using a map to convert webpage string names to integer vertices also simplified my code and made it easier to debug. I used the outdegrees map so I could update a vertex's outdegree count while adding an edge in the insertEdge function. Once all the edges were added with default weights of one, the updateWeights function assigned the correct weight, one over the outdegree count, to each edge. I used my own Edge class instead of a pair object so I could update the weight value while avoiding the more complex syntax needed to create and access elements in a vector of vectors.

B. Computational Complexity

In the following analysis, V represents the number of vertices in the graph, which is also the number of unique webpages, p represents the number of power iterations, and n represents the number of lines of input, or the number of edges in the graph. In the implementation, V is either numVertices or vertices.size(), p is either power_iterations or p, and n is no_of_lines. The single letters are used instead of the variables in the below analysis to improve clarity.

1. insertEdge

The insertEdge function operates in constant time O(1) for the best, worst and average cases. The unordered map insert function, [] operator, and vector push_back operation all have O(1) time complexity as they are used in the insertEdge function. The function also creates an Edge object, which is O(1) time in the best, worst and average cases because the Edge class constructor performs only constant time operations

2. updateWeights

To update the weight associated with each edge, updateWeights must traverse the list of edges for each vertex in the graph. In the best case, each vertex will be disconnected from all the others, meaning all the lists of edges have size zero. In the average and worst cases, the complexity of finding all of a vertex's adjacent vertices in an adjacency list graph implementation is O(V), since that vertex may be connected to every other vertex in the graph. Because updateWeights must conduct V adjacency operations at O(V) time complexity, the overall time complexity of the function is O(V^2) in the average and worst cases.

3. PageRank

The PageRank function includes three general steps. First, it creates a vector R0 of size V and initializes all entries, which takes O(V) time in all cases. Next, it conducts p – 1 matrix multiplications, where p is the number of power iterations. If p equals 1, this step is skipped. Lastly, it sorts the webpages alphanumerically and prints each webpage and its final rank.

The matrix multiplication step uses the same structure as updateWeights. For each vertex in the graph, it traverses that vertex's edges to calculate the rank associated with that vertex to be saved in the result matrix R1. In the best case, when the vertex is disconnected, this takes O(V) time. In the worst and average cases, this takes O(V^2) time. In the final step, PageRank initializes a string vector holding the names of all the webpages in the graph and then sorts it with the C++ sort function. The sort function operates in O(n*logn) time, where n is the input size, for all cases, so this takes O(V + V*logV) for all cases. Lastly, PageRank iterates through the vector of webpages to print them and their page rank, which adds an additional O(V) term. The complexity function for the worst and average cases is thus:

T(p, V) = V + (p – 1)*V^2 + V + V*logV + V

For the purposes of big-O time complexity, the linear terms can be removed and the function simplified to: T(p, V) = (p – 1)*V^2 + V*logV, or O((p – 1)*V^2) in the worst and average cases. In the best case, T(p, V) = (p – 1)*V + V*logV doesn't simplify further, since if p is very large the first term will grow faster than the second, but if p is small the opposite is true. PageRank's big-O time complexity in the best case is O((p – 1)*V + V*logV).

4. Main

The main function calls insertEdge n times, where in is the number of edges in the graph, while reading in the data. The insertEdge function runs in constant time in all cases, so this part is O(n) in all cases. It then calls the updateWeights and PageRank functions once, whose best, average, and worst case complexities are described above. All other operations in main are constant time operations.

Best Case: T(n, p, V) = n + V + (p-1)*V + V*logV

Worst, Avg Cases: T(n, p, V) = n + V^2 + (p – 1)*V^2

These functions simplify to O(n + (p-1)*V + V*logV) in the best case and O(n + (p – 1)*V^2) in the worst and average cases.

C. Reflection

If I could do the project again, I would spend more time considering ways to optimize the way the program calculates and stores the edge weights and try to find a way to update the weights during insertEdge. It seems inefficient to process all the edges and then update all the weights afterwards. I would also consider alternate ways of implementing the matrix multiplication process. Lastly, I think I would implement PageRank to return r_p-1 and create a different function to sort and print the URLs and ranks. I think this would improve the program's modularity and allow future users to change the program's output format without making changes to the PageRank function.