Daniel Dovale

11/13/2023

Simplified PageRank

My Implementation:

The data structure I implemented for the graph was an adjacency list which was tailored to be used for the PageRank algorithm. The three different unordered_maps I used ranks, indegree, and outdegree serve their own purposes, although the core graph implementation is done through the indegree map. The indegree map, maps a node (represents a website) to a vector of nodes that have edges pointing to it. This is essential for the algorithm since the rank of a node is influenced by the nodes that link to it and its outdegree. As for the outdegree map, this keeps track of the number of edges going out to each node which is also used for the calculation of the rank. Lastly, the ranks map stores the rank of each node which is helpful to easily display the ranks later on since the map data structure sorts automatically in ascending order.

The reason I used an adjacency list representation using an unordered map for the graph in this case is that website links are typically sparse which makes adjacency lists more space-efficient compared to an adjacency matrix. Unordered maps also offer average constant time complexity for insertion and access which is beneficial if this graph were to be larger. Additionally, creating two additional maps for the outdegree and the ranks simplified the process of calculating the rank and then later displaying the ranks.

Time Complexities:

void addEdge(string from, string to):

- Worst-Case Time Complexity: O(n)
- Finding an element in an unordered_map is O(n) in the worst case due to potential hash collisions although with a good hash function the time complexity would be close to O(1)
- n in this case would be the number of elements in the map

void PageRank(int p):

- Worst-Case Time Complexity: O($p * v^2 * n$)
- The outer loop runs p times, where p is the number of power iterations.
- Inside the loop, iterating over indegree is O(v), where v is the number of vertices.
- For each vertex, iterating over its adjacent vertices (the inner loop) is O(v) in the worst case if the graph is dense

- Accessing outdegree.find(link) is O(n) in the worst case for each link.

void displayRanks():

- Worst-Case Time Complexity: O($v * log(v)$)
- Iterating through the ranks map is O(v * log(v)) time where v is the number of vertices which is due to the sorted traversal of a map

int main():

- Worst-Case Time Complexity: O(n^2) + O($p * v^2 * n$) + O($v * log(v)$)
- The for loop runs n time where n is the number of edges and inside the loop the add edge method is called which the worst case of this method is O(n) which causes the overall loop to be O(n^2).
- The pagerank calculation "graph.PageRank(p)" is then O(p * v^2 * n) and the displayRanks "graph.displayRanks()" is O(vlogv) which was discussed before.
- Overall you would add up these time complexities to get the the worst case time complexity for the main function.

Reflections:

After completing this project I feel like I have a better understanding of utilizing and adjacency list implementation for graphs. I feel like this was the hardest part for me to grasp at first but after I was able to properly design the class using the 3 different maps to make things easier for me, it was pretty simple. Additionally, I realized the importance of considering the efficiency of algorithms for the addEdge and PageRank calculations since these become crucial with larger datasets. Lastly, if I had to start over the project I would spend more time on deciding which data structures are the most adequate for the pagerank calculations since I wasted a lot of time coding due to this (I realized later on that a certain data structure would not work for what I wanted).