Nathan Schoedl

COP 3530

17 July, 2021

Project 2 Documentation

Data Structure Used

I decided to use a map with its key and value pair being a string and a vector respectively to represent the adjacency list for the graph. This implementation allows for the url associated with a vertex to be the key and provides a list (vector) of adjacent nodes. Keeping in mind the characteristics of directed graphs and the objective of the project, the list of adjacent vertices includes only the vertices that the key vertex (vertex which acts as the key for the map) is pointing to. Space is conserved by using a map to represent the adjacency list because the map only stores information regarding urls of linked webpages rather than storing information of all webpages, linked to or not, for each webpage. Such wasteful use of memory would occur if I used a two dimensional array as the data structure to store information regarding linkage. Additionally, using a vector as the "value" for the map is useful for its size function, as retrieving the size of a vector associated with a specific webpage (key) directly provides the outdegree of said webpage, which is useful in the computation of a page's rank.

Computational Complexity for Methods

InsertEdge(…) :

- Best - O(1). This would occur in the case that an edge being inserted is from and to vertices already inserted and at the top of the BST. Thus finding the from and to vertices and inserting an edge would take constant time.

- Average - O(log V) where V stands for the number of vertices or URLs in the map. Finding the from and two vertices and inserting and edge from find would all take log V time because map's are backed by Red Black Trees and log V reflects average case for these functions in a Red Black Tree.
- Worst - O(log V). In the worst case, finding and inserting an element at the very bottom of a Red Black Tree takes Log V time so this would be the worse case time complexity

PageRank(…) :

- Best - O(pV) where p stands for the number of power iterations and V stands for the number of vertices (URLs). This would occur when the map is sparse, or, in other words, the number of edges, n, is close to |V|. This complexity occurs in the CalcRank helper function and the instance of sparse verses dense graphs is relevant in the inner for loop when computing the rank since, for a sparse graph, the inner for loop will run proportional to n / |V| which is constant in this case.
- Average - O(pV^2). This complexity occurs under the assumption that every website has a total of V/2 links to other websites, thus n = V^2 / 2. Again the total number of edges becomes important for the amount of times the inner for loop runs and in the given case it runs V/2 times for all vertices which means the nested for loop a time complexity of V^2.
- Worst - O(pV^2) which occurs for a very dense graph, or in other words, when the amount of edges is proportional to |V|^2 for reasons similar to those given above.

<u>Computational Complexity for Main Method</u>

- Best - O(n(log(V)) + pV) where n stands for the number of insertions, V stands for the number of vertices and p stands for the amount of power iterations. The first term, n(log(V))

is referencing the insertion step. The loop runs n times and calls InsertEdge for every iteration. InsertEdge has a best case complexity of O(1) but this is for a special case insertion that is not possible to consistently occur for a simple graph undergoing n insertions. PageRank is called once so it can undergo its best case complexity, hence the second term.

- Average - O(n(log(V)) + pV^2) average case complexity for the main function follows the same logic behind average case complexity for InsertEdge and PageRank.
- Worst - O(n(log(V)) + pV^2), for similar reasoning as average case complexity for the main method.

Reflection

From this project I learned how upper level math can be used in a seemingly standard function and the importance of understanding these concepts. I'm a math major and I took Linear Algebra last semester so it was interesting to see an introduction to applying the concepts I had learned in a less abstract sense. My professor for Linear Algebra taught us a "secret weapon" for matrix multiplication which helps to explain my slightly unorthodox approach to computing the rank so this project helped solidify my understanding of such methods and of how PageRank works. If I were to start over I would try to think of ways to optimize space or reduce time complexity. I would also think about time complexity when writing methods instead of trying to deduce them after.