Ryan Hulke
21 July 2023

Page Rank

Data Structure: Adjacency list (HashMap of URLs to a vector of adjacent nodes)

I chose this implementation because the HashMap's O(1) average time complexity for random access keeps the time complexity of both the AddEdge and PageRank functions down an order of magnitude compared to other potential implementations. Also, using an adjacency list as opposed to an adjacency matrix significantly decreases the computations required when doing power iterations in cases where there would be a lot of 0's.

AddEdge time complexity: O(n), n = number of nodes in graph

Since we are performing search and access in the HashMap, the worst case is O(n) when all the elements are hashed into the same basket, though the average case is O(1).

PageRank time complexity: O(I * n^3), I = number of power iterations, n = number of nodes

The while-loop that runs for every power iteration contains nested for-loops to check every neighbor of every node. Thus, if all the nodes are connected to each other, that results in n-squared iterations. But inside the nested for-loop, we use the [] operator on the HashMap, which technically has a worst-case time complexity of O(n). So, then we multiply all of those together to get I * n * n * n.

Main time complexity: O(I * n^3), I = number of power iterations, n = number of nodes

The loop adding edges can only ever reach O(n^2) time complexity (when every new edge has a new node and they all get hashed into the same basket), so calling the PageRank function will have the higher magnitude.

Conclusions:

I learned that using an adjacency list as opposed to an adjacency matrix can be much more efficient in some scenarios. I also learned about practical implementation of a HashMap for certain uses. If we started over, I would not have done anything differently for this project.