COMP 1405Z Project Analysis

Xuan Zheng

crawler.py

To solve the problem of how to store the information of the crawl. I decided to make a class that would contain all the data I need from a document. This class, urlData, would then be converted to json and written onto a file. This avoids the need to create many different files for each separate document and it makes retrieving that data much easier.

Function Name	Runtime Complexity	Space Complexity	Explanation
calculateTf	O(n*m)	O(m)	n is the number of URLs and m is the number of words in a URL. For space complexity, m represents the number of unique words stored
calculateIdf	O(n)	O(m)	Loops through all unique words in every document represented by n m is the number of unique words
calculateTfidf	O(n*m)	O(n*m)	n is the number of URLs and m is the number of words in the URL
calculatePageRank	O(n^3)	O(n^3)	N is the number of URLs in the crawl
seed	O(n^3)	O(n^3)	Time complexity is the

			largest one of its subfunctions, calculatePageRank

searchdata.py

Function Name	Runtime Complexity	Space Complexity
get_outgoing_links	O(1)	O(n)
get_incoming_links	O(1)	O(n)
get_page_rank	O(1)	O(n)
get_idf	O(1)	O(n)
get_tf	O(1)	O(n)
get_tf_idf	O(1)	O(n)

search.py

For the search process, I chose to loop through the top ten list, going through each element one by one, to find the smallest score in the top ten. There are a couple reasons I chose this method. First, I thought about the possibility of using python's version of priority queue to do this, however I was not allowed to import the heapq module for this project. My second idea was to use binary search, but after implementing it, the code was long and not very clear to read. That's when I realized, if I just simply looped through each element, it wouldn't affect the time complexity much at all.

The time complexity for binary search and looping through each element in my program is O(log10) and O(10) respectively. However, when you look at

the fact that the number of documents we need to compare will most likely be >10, the time complexity for returning the top ten list is going to be either O(nlog10) or O(10n) where n represents the number of documents that we are looping through. So the time complexity for both binary search and looping through each element would be O(n) in the end. Thus I simply looped through each element because it made for code that was easier to read and understand.

Another thing to add is that python's insert() is also in O(n) time where n is the number of elements in the list. In our case it would also really be O(10) which is negligible.

Function Name	Runtime Complexity	Space Complexity	Explanation
tfidfQuery	O(n)	O(n)	n is the number of unique words in the query
cosineNum	O(n)	O(n)	n is the number of unique words in the query
cosineDenom	O(n)	O(n)	n is the number of unique words in the query
search	O(n)	O(n)	n is the number of documents