**CSCE 2203: Search Engine Project Report**

**Nada El-Shenawy**

**ID: 900182782**

**Section A: Pseudo Codes**

**Indexing:**

The search input is parsed into a vector of vectors ***qu***:

* If the input is not in quotation marks, each word is placed in a separate vector in ***qu***.
* If there are multiple words in quotes, they are parsed into one vector together in ***qu.***

The function ***indexing()***:

* If the vector inside ***qu*** has only one word (no quotes)
  + Find word in unordered map ***keywords***
  + Loop over all urls that contain this word and insert them in unordered set ***result*** (if not already in the set)
* Else if the vector contains multiple words (in quotes)
  + For each word:
    - Find word in unordered map ***keywords***
    - Loop over all urls that contain this word and insert them in unordered map ***temp***
    - The value of the unordered map counts how many times this url appeared while iterating
  + Loop over ***temp***
    - For each url: if the count is equal to the number of words in the vector (in quotes), insert url in unordered set ***result.*** (if not already in the set)(if count == number of words, then this url contains all the words in quotes)

**Ranking:**

Function ***pageRank***:

* Calculate value for each page: ***val*** = 1 / number of pages
* For each url:
  + Check the set of urls that contain links to this url
  + update ***pagerank*** for this url by summing *val/number of outbound links* for url in the set

Function ***sortDisplay***:

* loops over all url (pagerank+1) values to find minimum
* calculate normalized pagerank for each url: ***normPR*** = (***pagerank*** + 1)/***min***  
  (each pagerank is incremented by 1 to avoid division by 0)
* loops over all urls to calculate page score for each by the given formula:  
  Score(Page) = 0.4*PR_{norm}+ ((1-\frac{0.1*imp}{1+0.1*imp})*PR_{norm} + \frac{0.1*imp}{1+0.1*imp}*CTR)*0.6
* The unordered set containing the search results is then copied into an array of structs containing urls and page scores
* The array of structs is sorted by function ***sort***
* The sorted array of results is printed

**Section B: Time and Space Complexity Analysis**

Let U = number of urls and K = number of keywords.

**Indexing:**

**Time complexity:**

* Loop to parse input: worst case O(K)
* Loop over vector containing parsed input: worse case O(K)

Inner loop:

* If no quotes:
  + Loop over urls in each keyword searched: worst case O(U)
* Else:
  + Loop over urls in each keyword searched to increment counter: worst case O(U)
  + Loop over urls to obtain the ones containing all the words in quotes: worst case O(U)

Time complexity = O(K) + O(K)\*(O(U)+O(U)) = O(KU)+O(K) = O(KU)

**Space complexity:**

* Vector for search input: worst case O(K)
* Unordered set for results: worst case O(U)
* Unordered map to check overlap of urls (in case of quotations): worst case O(U)

Space complexity = O(K + U)

**Ranking:**

**Time complexity:**

* Loop over urls to update pagerank: worst case O(U)
* Inner loop over urls linked to this url: worst case O(U)
* Loop over urls to find minimum pagerank: O(U)
* Loop over urls to calculate normalized pagerank and page score O(U)
* Loop over results to copy to array: worst case O(U)
* Sort results array O(U logU)
* Loop over results to print them: worst case O(U)

Time complexity = O(U)\*O(U) + O(U logU) + 4\*O(U) = O(U2)

**Space complexity:**

* Array to store results and sort them: worst case O(U)

Space complexity = O(U)

**Section C: Data Structures**

The main data structures used by my algorithm are unordered maps, unordered sets and vectors. For searching, I used the unordered map of keywords, containing unordered sets of urls that match this keyword, to make the search mode as fast as possible. The sets and maps used are all unordered because for this algorithm, an alphabetically ordered set of urls or keywords is unnecessary, so choosing unordered data structures would significantly decrease the complexity of insertion and deletion. I used unordered maps and sets instead of vectors because searching using the find() function would have constant complexity as opposed to linear complexity in vectors.

**Section D: Design Choices**

I chose to store the graph in an unordered map, in which each url has an unordered set of urls which contain links to this url (key). This facilitates the calculation of PageRank, as I can easily see which urls contain links to this url. In addition each url contains a counter of the outbound links (in the struct), which is also needed for the calculation f pagerank.

Although storing the keywords in both the struct of url data and a separate unordered map would increase space complexity, it greatly decreases time complexity when deleting a url. This is because in the deletion of a url, we have to remove it from each list of matching keywords. If the keywords weren’t stored in the struct, we would have to loop over all the keywords to find the url we are deleting and then remove it. Having the keywords stored in the struct of url data, we will know the exact keywords that need updating, without looping over all keywords.