Search engine

This is a basic search engine that takes a search query from the user and show results according to the websites' rankings. The initial ranks for websites are given according to Page Rank algorithm. The program updates the impressions and views for each website continuously. The score for each website is continuously updated.

Complexities:

Page rank:

Time complexity:

$\Omega(n)$ if each page has only one page pointing at it

$O(n^2)$ if the graph is a dense graph and each node is connected to all the others

Space complexity:

The function itself has a space complexity of O(n)

It defines an array to hold temporary ranking values which is of size n (the number of websites in our database)

CTR calc score:

Time complexity:

$O(n)$ where n is the number of the websites in our database

Space complexity:

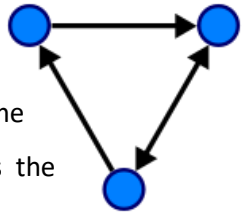
The function itself has a space complexity of O(1)

Showing the pages sorted:

Time complexity:

$O(n^2)$ where n is the number of websites that satisfy the search query

Main data structures used:



Graph using adjacency list: it represented the relations between the websites and adjacency list was especially used as in the page ranking if the graph is sparse the time complexity of the algorithm becomes $\Omega(n)$ and in other functions it also improves the performance.

Hash Map: the hash map acted like a DNS as it mapped the names of the websites to the indices representing their position to access any website in O(1) according to its position which was given to us from the map in O(1). This improved the performance in many instances.

Unordered_set: the unorederd sets represented the keywords in each website this was helpful to search if keywords matched the search query in O(1) which improved the performance.

Vectors: vectors were used to have the websites, views, impressions and other helping values. While the program is working no deletions or additions happen in the main vectors but we extensively access these vectors so they were used as accessing complexity is O(1).

Design tradeoffs:

Sacrificing some space to have better complexity:

some vectors holding different values like atThis or fromThis were added to the code to have better complexities in different functions. For example, when calculating the page rank we needed to know the number of nodes pointing at this node and pointed at from this node so recalculating these values each time we needed them would have been very time consuming so they were all calculated once in the constructor with O(n) complexity to avoid the very time consuming process of calculating them again each time we needed them.

Unordered sets and case sensitivity:

There were two options. The first is to change the data structure of unordered_set to a vector and when matching the search query with the keywords iterate over all keywords in each website to know if it exists or not. This would have made case-insensitive comparing (by uppercasing both sides of the comparison each time) possible. Yet, it means that for each word in the query we need to do O(n) comparisons for each word in the keywords. However, unordered sets gave us the option to check for the existence of the keyword in O(1) which is much better. The only problem is that we can't upper both sides to do the case-insensitive comparison as the checking does not work that way.

A work around solution was done to solve this problem. Each time a new keyword was added to the keywords set another word in the form "?case?uppercase(word)?insensitive? " was added. The two words ?case? and ?insensitive? were added as a keywords to reduce the chance of having a wrong search to its minimum probability with having a good complexity and the given word was uppercased and put in the middle. This made us able to make case insensitive searches while having complexity across all the websites of O(n).