# Analysis of Search Engine

Omar Elwaliely

November 2021

**Abstract**

During the month of November, I created a search engine that is built on a static set of websites and keywords. In this paper I analyze my algorithm and find the potential trade-offs and improvements that could be made. It is found that the majority of the issues arise at initialization time or lie in the space that the program takes.

# Contents

# Chapter 1

# Introduction

In the modern day world we rely on search engines to efficiently extract information for us from a multitude of sources; however it is easy to overlook just how much efforts go into finding the information you need. In order to figure out how these search engines work. I created a smaller version of a basic search engine.

## 1.1 Specifications of Search Engine

The engine that I will use will not be complex. Firstly, the data is assumed to be static. This is for simplicity sake. In order to calculate Page Rank, it is also assumed that each website is clicked at least once and is pointing to at least one website. There are thirty websites and a cumulative 17 search terms that a user can make.

# Chapter 2

# Page Rank Algorithm

According to [3], namely section 4.1, The ranking algorithm works as follows: It checks a parent node for nodes connected to it. It then goes to those nodes and checks what their current rank is, then it divides that rank by the number of nodes this child is connected to. It sums up all these values and then stores it as the parents current rank. It does this for all nodes and then repeats this process until everything is checked.
This can be represented as:

$$PR_{t+1} = \sum_{P_j} \frac{PR_t(P_j)}{C(P_j)}$$

The issue with this is as the paper states is, in very big graphs it becomes a very large value of $O(n^3)$ so instead we make it $O(100n^2)$ which is actually $O(n^2)$. To do this we instead check each node and divide its page rank by the amount of connections. Next we multiply this by 0.85 also referred to as "damping factor"; we keep track of these number someplace lets call it "multiplier" and go to a parent node and check whats connected to it. For anything connected to it we sum these multipliers. Once the iteration is done we check if the currentPR is less than the previous iteration by a value of 0.01 or more for all the websites. If this happens the Page rank has stabilized. Otherwise we keep doing this. If we break 100 iterations we stop. The equation for this would be:

$$W_i = (1 - d) + d \sum_{j=1} I_{i,j} \frac{W_i}{n_i}$$

After each page has a $PR$ they will be normalized using the equation:

$$PRnorm = \frac{PR - MinimumPR}{MaximumPR - MinimumPR}$$

Finally this PR will be turned into a Page Rank using the equation:

$$\begin{aligned} PageRank =& 0.4 \times PRnorm + [(1 - \frac{0.1 \times impressions}{1 + 0.1 \times impressions}) \\ & \times PRnorm + (1 - \frac{0.1 \times impressions}{1 + 0.1 \times impressions}) \times CTR] \end{aligned} \tag{2.1}$$

# Chapter 3

# Data Structures Used

The space and time complexities were found according to GeeksForGeeks.[1]

## 3.1 Multimap

The multimap data structure is a Standard Template Library representation of a Red Black tree, but keys can have multiple values.[2] This is the main data structure used when a keyword is searched.The time complexity for searching in a multimap is $O(\log n)$ and space complexity for search is $O(1)$. Insertion time and space complexity is $O(1)$.

## 3.2 Hashmap

The hashmap data structure is considered an associative array. It matches keys to values.[2] In the program it is used to find if websites are present in multiple keywords. The time complexity of insertion and search is $O(1)$. The space complexity for insertion is $O(1)$, but for search it is $O(n)$.

## 3.3 Map

The map is the Standard Template Library of a normal Red Black tree.[2] In the program it is used during initialization as a helper to bind strings to a website struct. The time complexity for searching in a map is $O(\log n)$ and

space complexity for search is $O(1)$. Insertion time and space complexity is $O(1)$.

## 3.4 Vector

The vector data structure is a type of dynamic array that can be changed throughout the program.[2] It is used in order to store the websites temporarily while the user decides if he wants to click a website. Additionally, it is used to store the graph during the initialization of the Page Rank algorithm. The time and space complexity for insertion is $O(n)$ since it copies previous array to new array. The time and space complexity for search is $O(1)$.

# Chapter 4

# Psuedocode

## 4.1 General Explanation

At initialization, the program reads from three files. The search engine reads from a file and stores the information into a multimap data structure, the key is the keyword, and the values are the websites. Regarding the ranking algorithm, If the Page Ranking was done at least once, then the file to use is the file with already prepared ranking and clicks. If the ranking wasn't done, then inputting files is done in the search engine part of the code, then the program will use a file with websites presented as connected edges to create an adjacency list. After that, it takes in the impressions from a separate file, and it uses the ranking algorithm presented in the first chapter rank the pages. Finally, it sorts the pages according to page rank, and copies them to a new map. In terms of searching, once the user gives an input, it searches through the already sorted multimap, depending on the inputs. If the input contains "AND "then the program goes to the first keyword (up until it finds "AND") and looks for it in the multimap. Once the keyword is found, it takes all the values and puts them into a hashmap. Next, it searches for the keyword after the "AND" and compares it in the multimap. If there is a collision, this website contains both keywords since it was both of the keywords' nodes, and since they are already in order, all these websites' impressions increase by one, and they are all saved in a vector and printed. This vector is used when a user decides to "click" a website. We need a data structure to know which website is clicked. In the case of "OR" the terms are also searched in a multimap; however, before printing, they are stored in

vectors, and since they are already sorted, a basic merge is made, and then they are finally printed. If a quotation is entered the word is searched in the sorted multimap and all the values are printed. In case a user decides to click a website, it is just compared to the vectors, and it refers to where each website is stored and increases clicks by one. Additionally, after each print the sorting is done one more time. At the exit of the program all the website data is outputted to a file that will be used at the next run.

## 4.2 Programming

Throughout this code you will see that "website" is sometimes referenced, you can assume that these websites are stored in a map at the start for the sake of usability; however, the fetching of this information can be done in a multitude of ways so it is represented as "website.object" in the psuedocode.

Page Rank:

graph g //in program this is a vector of vectors

while(you can read from file)
    copy a line and store it in a stringstream
    use first word to get website.name and use this name as a reference.
    second word is website.impression.

while(you can read from file)
    take in first word
    take in second word
    g[first].pushback[second]
    increment second.connections by one (used in the ranking algorithm)

for int k=0 → 100//iteration to stabilize, keep in mind this is is a worst case
    for int i=0 → n-1
        multiplier = pr of current iteration * 0.85 / websiteconnections
        for int j=0 → n-1
            if (j is in graph[i])
                sum = sum + multiplier[j]

for int i=0 → n-1
if(website.pr[i] +0.01 ¡ website.pr[j] for all i)
break loop since the values have stabalized
set all websites previous page ranks to websites.pageranks

getmax() of websites
getmin() of websites
normalize page ranks using max and min and use page rank equation and store them

If the pagerank was already initialized, ignore all the previous and do as follows:

while(in pageranks output file)
first value → websitename
second value → websitename.impressions
third value → websitename.clicks
fourth value → websitename.pagerank

Search Engine:

while(you can read from file)
store line in stringstream
store first word as website name
while(still in stringstream)
store second
put in multimap where first is value and second is key

//now the sorting will be done

vector keyvalue
for(any key value pair in multimap)
if(in a new key)
sort(keyvalue)
store in multimap sortedmap[key].keyvalue[index]
else
store value in keyvalue vector

```
//now the searching will be done

if(search terms contains "AND")
while(you can take in a word from whole search term)
        take in a word
        sortedmap(word) //this goes to the node with the values
        while(in node)
            store in hashmap
            if(collision)
        increase impressions of word //word is website
                print word
                store word in vector
if(search term contains "OR")
    create vectorA and vectorB
while(you can take in a word from whole search term)
        take in a word
        sortedmap(word) //this goes to the node with the values
        while(in node)
        increase impressions of word //word is website
            store word in vectorA if A is empty and B if it is not
        merge A and B
if(search terms contains quotations)
        print all values in sortedmap(word)
else
    Do case "OR"

At exit:
Take in all website.objects and store in page ranks output file
```

# Chapter 5

# Time and Space Complexity

## 5.1 Initialization Time Complexity

To calculate the time complexity of the initialization we will look for which part of the initialization uses the most theoretical time. Assuming the Page Ranking initialization has never been done the highest complexity is be reliant on the search engine class. Assuming the page rank was initialized before, the time complexity will be $O(n^2 \log n)$. We conclude this also using the Search Engine class. Reading from files takes $O(n)$ time and another $O(n)$ time to store that in a string stream. Iterating through a string stream takes $O(n)$ time and storing the data to a multimap takes $O(1)$ time. Up until this point the time complexity is $O(n)$; however, the complexity increases when we begin to sort these websites. Iterating through a multimap takes $O(n)$ time and sorting takes $O(n \log n)$ time, so to sort an entire multimap it would take $O(n^2 \log n)$ time. Therefore we arrive at a best case complexity and worst case complexity for time at initialization:

$O(n^2 \log n)$ and $\Omega(n^2 \log n)$

## 5.2 Initialization Space Complexity

To calculate the space complexity of the initialization we will look for which data structure in the initialization uses the most theoretical space. Assuming the Page Ranking initialization has never been done the space complexity will be $O(n^2)$. We conclude this from the graph data structure, which is com-

prised of a vector of vectors. Since we can think of vectors as dynamic arrays. Then we have a structure which requires N $\times$ N space. If the Page Ranking initialization was done, then the space complexity becomes $O(n)$. We know this because the most demanding data structures are now the vectors which take $O(n)$ space. Therefore we arrive at a best case complexity and worst case complexity for space at initialization:

$O(n^2)$ and $\Omega(n)$

## 5.3 Search Time Complexity

To calculate the time complexity of the search we will look for which part of the search algorithm uses the most theoretical time. The search time complexity is $O(n)$. We find this by going to the worst case which is that someone uses the "OR" keyword. In this scenario the program will first iterate through the whole keyword in $O(n)$ time it will then transfer it to a stringstream in $O(n)$ time. After iterating through if a word is found it will put it in the multimap, the search time for a multimap is $O(\log n)$ time. After the key is found it iterates through all the values once in $O(n)$ and copies to a vector which is also $O(n)$, finally it merges two vectors in $O(n)$ so the cumulative search time is $O(n)$. The best case is if the keyword doesn't exist in the map. In this scenario the word still must be read from the user in $O(n)$ time, and the search is $O(\log n)$, so together there is a $O(n)$ time. Therefore we arrive at a best case complexity and worst case complexity for time of search:

$O(n)$ and $\Omega(n)$

## 5.4 Search Space Complexity

To calculate the space complexity of the search we will look for which data structure of the search algorithm uses the most theoretical space. The space complexity is $O(n)$ we know this because in the worst case scenario a hashmap is being used which takes $O(n)$ space. The best case space complexity is found in the case that a user uses quotations. Then the only data structure used is a multimap which takes O(1) space. Therefore we arrive at a best case complexity and worst case complexity for space of search:

$O(n)$ and $\Omega(1)$

This can be represented as:

Table 5.1: Time

| **Search** | **Initialization** |
|---|---|
| $O(n)$ | $O(n^2 \log n)$ |
| $\Omega(n)$ | $\Omega(n^2 \log n)$ |

Table 5.2: Space

| **Search** | **Initialization** |
|---|---|
| $O(n)$ | $O(n^2)$ |
| $\Omega(1)$ | $\Omega(n)$ |

# Chapter 6

# Trade-Offs

Like most algorithms there seems to be an inverse relation between space and time complexity. The main data structure used was the determining factor of the program in terms of time complexity, but in terms of the space complexity, the determining factor was the helper data structures.

## 6.1 Initialization

Assuming the file was never initialized the general space complexity is $O(n^2)$ due to the use of a vector of vectors; however there were not many alternatives. An example alternative would be a two dimensional array; however, since we can specify the size of the vectors at initialization time, the complexities would be the same for both; so the justification for using a vector of vectors is that it is easy to use and it has about the same complexities as a two dimensional array. The time complexity could also be improved using a compressor and; however, since the base PR stays the same after the initial run we can ignore this factor of the program; however for arguments sake, in order to improve this, we can use an encoder and decoder; however since this is a general experiment, and due to a time constraint I decided to keep it as is. Assuming the file was initialized before the space complexity relies on the helper functions involved in the search engine. The largest data structure is the vector which takes $O(n)$ space. The only way of improving this would be to use the original map; however iterating through this map would due to this instance would take $O(n^2)$ time. So, due to time, the helper functions (hashmaps and vectors) were preferred during initialization while sacraficing

a small amount of space.

## 6.2 Search

The space complexity of search is dependant on the helper data structure, the hashmap. In order to improve the space complexity the hashmap would have to be replaced with a data structure which takes less space. One such example would be to use the same main tree; however replacing the hashmap with that tree would severely increase the time complexity. The use case of the hashmap is to efficiently see if a website is common in two different searches. Using a tree would mean that you would have to search though the map n times. Since the search time is $O(\log n)$ for a tree, the final time complexity, assuming you use a tree, would be $O(n \log n)$. Since the time complexity of a hashmap search is $O(1)$ So in this case, the helper data structure is the hashmap due to its lower time complexity, and slight decrease in space complexity in comparison to time complexity when related to the tree.

# Chapter 7

# Potential Improvements and Conclusion

There were also a few shortcomings in the project. The main shortcoming was the long initialization time. Another shortcoming was the generally high space complexity. In the future if someone were to replicate in this project on a larger scale they maybe benefit from using a better technique of creating a web graph, and using one very reliable data structure. To conclude, a lot was learned from this experiment. On a very small scale it was still very complex in terms of content. This is because a lot went into maintaining a search engine with a complexity that is both low in space and low in time when searching.

# Bibliography

[1] Analysis of time and space complexity of c stl containers. https://www.geeksforgeeks.org/analysis-of-time-and-space-complexity-of-stl-containers/, Jun 2021.

[2] The c standard template library (stl), Nov 2021. URL: https://www.geeksforgeeks.org/the-c-standard-template-library-stl/.

[3] Amy N. Langville and C. D. Meyer. *Google's PageRank and beyond: the science of search engine rankings*. Princeton University Press, Princeton [N.J.], 2012;2009;2011;2006;.