

## **Project Report – Building a Search Engine**

The purpose of this project is to build a search engine, like Google, to take a string input from the user and outputs the relevant websites for his keywords. The websites should be sorted according to the score of each website. The score has several variables that change its value with the user's interactions. The number of clicks, number of views, and page rank of the webpages are the three variables that enter into calculating the final score.

In the next pages, I'm going to present the project's details using a collection of questions and answers to clarify all of the information in implementing the search engine.

### **1- What to need before using the search engine?**

- The project must be provided with three main files which acts as a data base for the search engine. All of the files are in the csv style.
  - The first file is "Keyword file.csv". This file stores the webpages' names and the adjacent keywords for each webpage.
  - The second file is "Number of impressions file.csv". This file stores the webpages' names and the adjacent number of impressions for each webpage. It's required to increment the impression by one, when the webpage is viewed in the search results to the user.
  - The third file is "Web graph file.csv". This file store the links between the webpages' pairs. Every webpage can include a hyperlink to another webpage. This hyperlink is used to calculate the page rank of the webpage. The more links inside a specific webpage, the greater rank it could gain.
- The project needs an extra action before starting, it needs a fourth file called "clicked.csv" to allow the project to update the numbers of clicks. The file should have the name of webpages with the adjacent number of clicks (will be initially equal zero). So, we will create this file by copying "Number of impressions file.csv", changing its name to "clicked.csv", and make all the values equal zero.

## 2- What are the main data structures used for implementing the project?

Implementation of the search engine is using several data structures to deal with input processing and give the final output. **Classes, vectors, maps, and pointers are the main data structures used in this project.**

- First, we have a **class of webpages** that can make a webpage object with several variables that distinguish it.
  - Every object will have a name, impression, clicked, CTR, PR, and score variables. In addition, it has a vector of keywords, a vector of the webpages which are pointing to it (ToWeb), and a vector of the webpages from which it is pointed (FromWeb).
  - The class have a collection of setters, getters, and other functions which help the coder to access and control any information for any webpage.
- Second, we initialize a **vector of objects** to be the main database through the whole code. Then, we start to read the input files and update the webpages information using the vector of objects.
  - The numbers of impression, clicks, and the keywords are taken from the input file in a **map of strings and whatever the second element is**. In all cases, the string will represent the name of the webpage.
  - The “Wep graph file.csv” information will be taken and used to fill the two vectors in the webpage classes (ToWeb & FromWeb). These vectors are **vectors of pointers**, pointing to the objects in the main vector we have. By this data structure, we can, starting from any node, access the nodes pointing at this node and all of their information. Similarly, we can access the nodes from which this node is pointed.

### 3- How is the search engine implemented in C++?

Starting from readingFiles function, we execute the steps shown in the previous section to create the vector of webpages' objects with all their information. By this point, we are updating the name, impression, clicks, keywords, and the edges.

- Next, we calculate the page rank for each webpage. Calling calculate\_PR function with passing webpages vector as a parameter by reference will complete this task.
  - Calculating the page rank is done by initializing it to (1/the number of webpages). Then, we loop over every node and calculate the sum of (PageRank of the pointed webpage / the webpages' number which this current page is pointing to).
  - This process should be repeated with a number of iterations equals the webpages' number – 1.
  - Consequently, we should sort the pages according to their ranks, and adjust these ranks by consecutive numbers from 1 to the webpages' number.
- At this point, we are done with nearly 70% of the work. The next step is giving the webpages vector (which contains all of our work until now) to the four interfaces function to continue dealing with the user.
  - The four interfaces function is responsible for interacting with the user's input and resulting the corresponding output.
    - 1- startFace function:** Used to welcome the user and give him the options: new search, or exit the program.
    - 2- searchFace function:** Used to take the input from the user, execute the search matching function to choose the relevant webpages to print, sort these webpages according to their scores, and go to the next interface.
    - 3- resultFace function:** Used to show the search results to the user, update the impression for each resulted page, and give the user three options: choose a webpage to open, return to a new search, or exit the program.

**4- viewFace function:** Used to open the chosen webpage, update the number of clicks for this webpage, and give the user three options: back to search results, go to new search, or exit the program.

- The four functions are connected to each other by several instantiations and passing-by-reference parameters, so that the score can be update during the user interaction. In addition, there are functions used to update the impression and number of clicks in the input files, so that when the program is closed then the user wants to use again, the last updates can be saved.

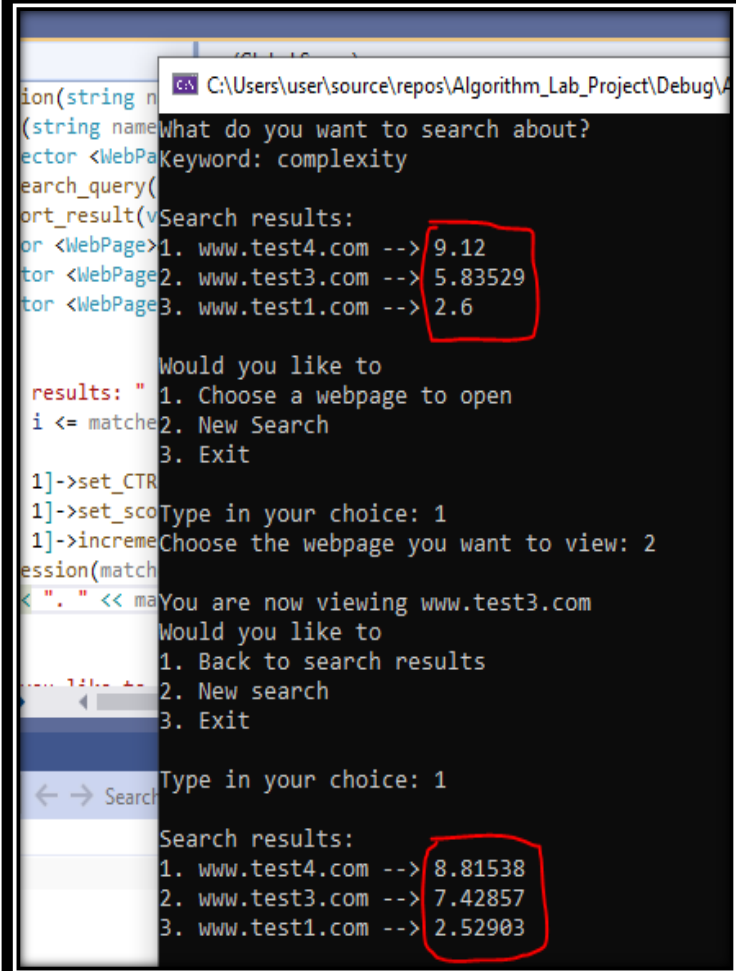
## **4- How does the search-matching function works?**

- Simply, it takes the input string from the user and starts by detecting if there are quotes in the beginning and ending of the string. If that happens, it erases these quotations and push the whole string in an empty vector of strings.
  - If no quotations detected, the function will start to find “AND” in the string. If that happens, it erases this AND, and take the next word in a variable called ANDS. Then, it separated the remaining sentence into single words and push them to the empty string vector.
  - If the previous if statement is false, the function will start to find “OR” in the string, then erase it. Consequently, it separated the remaining sentence into single words and push them to the empty string vector.
- After preparing the string vector and the ANDS variable, the program will loop over each string in the vector and search for it in each webpage’s keywords. If there is a match, the function checks if there is a word in ANDS variable or not. If it happens, the matched webpage must have this string to be selected for the results.

## 5- How can we check if the search engine works properly by mathematical proof?

We can print the results webpages with their adjacent scores next to them to be able to trace the updates executed in the Back End (I commented the code for this task in the cpp file). Starting from a new search, we can write any keyword to make the search engine outputs a number of relevant websites with their score.

- At this point, the impression value for the resulting webpages should be increased by 1. This change should decrease the total score by a little percentage. To make sure this is happening, we will choose to start a new search and write the same keyword. We will notice that the scores are decreased.
- What if the user chooses to open a specific webpage? This step will increase the number of clicks by 1. This change should increase the total score by a remarkable parentage. If we do these steps, we will notice an increase in the previously-selected webpage, while a little decrease in the remaining scores resulted in the first point. This is actually how any search engine sorts its results by.



```
C:\Users\user\source\repos\Algorithm_Lab_Project\Debug\A
ion(string n
(string nameWhat do you want to search about?
ector <WebPaKeyword: complexity
earch_query(
ort_result(vSearch results:
or <WebPage>1. www.test4.com --> 9.12
tor <WebPage>2. www.test3.com --> 5.83529
tor <WebPage>3. www.test1.com --> 2.6

Would you like to
results: " 1. Choose a webpage to open
i <= matche2. New Search
3. Exit

1]->set_CTR
1]->set_scoType in your choice: 1
1]->incremeChoose the webpage you want to view: 2
ession(match
< ". " << maYou are now viewing www.test3.com
Would you like to
1. Back to search results
2. New search
3. Exit

Type in your choice: 1

Search results:
1. www.test4.com --> 8.81538
2. www.test3.com --> 7.42857
3. www.test1.com --> 2.52903
```

## 6- What about time complexity for indexing and ranking algorithm?! (including pseudo codes)

About the ranking algorithm, the pseudo code can be presented as follows:

// For calculating the Page Rank

```
From i = 0 to webpages.size do
    arr[i].PR ← (1.0 / webpages.size);
    arr[i].PRtrial ← (1.0 / arr.size());

From j = 1 to webpages.size - 1 do {
    From i = 0 to webpages.size do
        double PRe ← 0.0
        from k = 0 to webpages.FromWeb.size do
            PRe ← PRe + arr[i].FromWeb[k]->PR / arr[i].FromWeb[k]->ToWeb.size
        arr[i].PRtrial ← PRe

    From i = 0 to webpages.size do
        arr[i].PR ← arr[i].PRtrial
}
```

// For sorting the webpages

```
multimap <double, int> numbers
multimap <double, int>::iterator it
    From i = 0 to webpages.size do
        numbers.insert(pair <double, int> (arr[i].get_PR(), i))

int i ← 1
from it = numbers.begin() to numbers.end()-1 do
{
    arr[it->second].set_PR(i);
    i++;
}
```

- If we consider the number of webpages equals  $n$ , so the function is looping first time to initialize the page ranks in time complexity of  $O(n)$ .
- Then, the function is looping three chain loops to processing the page rank. The first loop is the number of iterations used to output the rank with an accurate value, and it takes  $n - 2$  iterations. The second loop is used to processing on each node in the webpages vector, taking  $n$  iterations. The third loop is to visit the pointed nodes and increment the page rank, taking a maximum of  $n - 1$  iteration (This case will happen if the current node is being pointed from all the remaining nodes).
- These three loops will make the worst-case time complexity equals  $O(n^3)$

- About the sorting part, there is two loops, each of them takes  $n$  iterations, happening concurrently.

**Hence, the final time complexity is  $O(n) + O(n^2) + O(n^3) = O(n^3)$**

About initializing the input webpages and updating their values, it contains several functions (as mentioned above in Question 2). All of them is happening concurrently. The biggest time complexity of these functions is that one which are responsible to add the edges, which execute one for loop with  $n$  iterations and call AddEdge function. This function is calling FindByName function which takes  $O(n)$  complexity. **Hence, the time complexity for reading the files and initializing the pages is bounded by  $O(n^2)$**