# Report

# Algorithm Lab Project


# Abdelrahman Shaaban

# 900183004

# Introduction:

I was asked to build a small search engine that takes a search query, processing its writing way: whether it includes quotations or not, “AND” and “OR” words, and returns the most related and useful results to the user. Such a ranking of which the most useful and the most related depend on two algorithms I was asked to implement, which are the PageRank algorithm and then computing the overall score of each page that depends on the computed PR, and the Click-Through-Rate (CTR) that is mainly computed using the number of impressions of each page, and its number of clicks.

As the program begins, I need to initialize the files that the user’s search will be done through. Four files are initialized and work as the big data base that all the processes into my program mainly depend on them. The first file is the web graph file that include all the web pages I have and describes how they all are connected to each other: Exe. Hyperlinks. Initializing this file is the backbone to build my web graph [Vertices→The Web Pages –Edges→ The Connections between them]. Building this web graph is very important in computing the Page Rank of each web page. The algorithm actually is dependent of it.

The second file I need to initialize is the key words file. This file is important in connecting each web page to a few key words that user mainly will search for. Using the pair and vector data structures to read all these data into is done to be able to use them in the searching process. The last two files I need to initialize are the impression and the click files that show the number of impressions and clicks of each page. Reading these files into two maps data structure is very useful in computing the CTR and the overall ranking score of each page.

# Pseudo-Code:

PageRank:

Input:

1- **Vector (Web_pages) t**
2- **Vector of Edges (src→dest) as src and dest are web pages.**

Output:

1- A vector of pairs <string, double> with each page is connected to its PR value

Assumptions: Calculating the PR require 100 iterations according to my research.

Logic:

```
Graph(edges,v){
map <string, int> M;   //to index each page
map <string, double> out;   //to get how many vertices this webpage points to;
vector of vectors v1;     //adj_list
vector of vectors v2;     //temp adj_list
loop(i=0→v-1)M[web_pages[i]]=i;     //indexing  each web_page
loop (i=0→edges_vector.size-1) {
          increment(out[edge[i].source]);
          push edge[i].source into v1[edge[i].destination] //to get the value of each vertex connected to
Rank:          //compute the PR
arr[v][100];
loop (i=0→v-1)A[i][0]=1/v   //first column 1/number of web pages
loop(j=1→99)
  loop(i=0→v-1)
        v2=v1
        while(v2[web_page[i]].size!=0)
             get the sum of all the values of the connected this vertex and divide by out[vertex]
             A[i][j]+=A[M[vertix]][j-1]/out[vertix];

loop (i=0→v-1)map[web_page[i]] to A[i][99]
```

**Score Calculator:**

Input: vector of all related web pages W to the search queue, map of impressions, a map of computed PR, and a map of clicks all connected to the different web pages

Output: a priority queue of the web pages results that will be shown

```
loop (i=0→web_pages.size()-1)
CTR= click[W[i]]/Imp[W[i]]
v1= 0.4*PR[W[i]]
continue compute this equation
pair<double, string> score, web_page
push the pair into the priority queue -→sort it directly
```

$PR_{norm}$ represents the normalized PageRank value across all webpages

$$score(page) = 0.4 \times PR_{norm} + \left(\left(1 - \frac{0.1 \times impressions}{1 + 0.1 \times impressions}\right) \times PR_{norm} + \frac{0.1 \times impressions}{1 + 0.1 \times impressions} \times CTR\right) \times 0.6$$

## Time Complexity:

### Ranking Algorithm PR:

Loop(j=1→n-1)  n is the number of iterations
  Loop (i=0→v-1)  v is the number of webpage
So this function has a complexity of **O(n*v)**

**Space Complexity:**

2 vectors adjacent list of worst case of V,
a vector of web pages numbers V,
and a map of V size
So worst case complexity is **O(v)**

### Indexing:

Loop(i=0→web_page_num)  n the number of web pages
  Mapping the webpage in log(n)
So this function has a complexity of **O(n log n)**

**Space complexity is O(1)**

## The main data structures that are used:

**I tend to use many of C++ STL data structures that helped me to effectively implement my algorithms, and easily build the engine.**

**Vectors: to contain the webpages I have and process them**

**Pairs: to connect two variables effectively and then push them in a vector, using pairs instead of maps is very useful when it comes to sorting.**

**Maps: very effective and easy to connect any two variables to each other**

**Priority queue: very useful to automatically sort the webpages I have based on their scores in a logarithmic time.**

**Set: It was mainly used for not repeating any webpage to be chosen twice while processing the OR search.**