THE AMERICAN UNIVERSITY IN CAIRO

School of Sciences and Engineering

Analysis Lab Project

By:

Shahd Elmahallawy ID: 900194441

CSCE 2203- Analysis and Design of Algorithms Lab

Spring 2022

1. Indexing Algorithms:

To get the websites names, I didn't make a specific data structure to save them. I got them whenever needed from the keys of the unordered map "keywords". I chose this map specifically because its keys have all necessary websites. I got this unordered map from reading the "Keywords.csv" file.

I. Reading Files

I defined a function to read any CSV file only by passing its name. Then, according to it its name it reads the file.

Pseudo Code:

```
readFiles (name of the file)
{
    While(didn't reach the end of)
    {
            Get the first word before the the comma of each line and store it
            While (you reach the end of each line)
            {
                    Get the words between commas and store them
                    If (name of file == webgraph file)
                            Push back the first word to be the key, and the other
                            websites to be the key mapped value.
                    Else if (filename == keyword file)
                            Push back the first word to be the key, and the key
                            words to be the key mapped value.

                    Else if(file name == number of impressions file)

                            Turn the string except the first one to integers

                            Push back the website name to be the key, and the
                            integers to be the mapped value

            }

    }
}
```

Complexity:

Time:

To read the data it has complexity O(n^2). To retrieve from the unordered map whenever it is needed has a complexity O(1).

Space:

We push what we read in an unordered map. Then, the space complexity is O(n). To retrieve the websites from the unordered map "keyword" it has space complexity O(1).

II. String Query

First, I get it as input from the user and save it in a string. Then, I split this string to words by ignoring the spaces and save each word in a vector of strings. Then, I search about these words according to which case it is (And. Or, quotes”

Pseudo Code:

```
Search( )
{
    String query;
    Cin >> query;
    //check quotes
    If (query[0] == “ )
    For( i=1 , i-> query.size()-1)
            {
            Remove quotes
            }
            Query_case();
    Else
    {
            For( i=0 -> query size)
                    If (it has AND)
                            AND_case;
                    Else if( it has OR)
                            OR_case;
                    Else
                            OR case;
    }
}
```

Complexity:

Time:

I looped over the string to check which case is it and remove(AND , OR, “ ”) from any quotes.Then, the complexity of this algorithm is O(n) such that n is size of the query.

Space:

I pushed words of each string in a vector has the size n such that n is the number of words of the query without AND or OR. Then the space complexity is O(n).

2. Ranking Algorthims:

I. Page Ranking Algorithm

The rank of each website depending on some iterations such that I stopped these iterations when the difference between the value of the current page rank and previous page rank is less than 0.001. The first iteration each webpage rank is initialized to be $\frac{1}{\text{total number of vertices}}$. Then, in the following iterations I each webpage rank is updated according to the following equation:

pr(website) = $\sum_{0}^{n} \frac{pr(p)}{L(p)}$, where pr is the page rank, p is the parent of the website in the web graph and L is the number of out vertices of the parent of the website.

Pseudo code:

```
rank_webpage
{

for (i 0->n) // looping over the webpages
    pr(website) = 1/n;
while(stop when the difference exceed 0.001)
{for (i 0->n) // looping over parents
    for (j 0 -> m) //looping over the children of each webpage
            pr[website] += pr[parent] + number of the children of this website;
}
//Give the webpages ranks between 1, 2, 3, …., n
Great_prev = 10000.0;
for( i 0->n)// i is 1, 2, 3…
    for(website 0->n)
    if(pr(i) > pr(j) and page[j] < great_prev and pr(j) is not visited){
            pr(website) = i;
            mark this website as visited;
```

}

}

Complexity of this Algorithm:

Time:

First, it loops over the webpages to calculate the page rank value to initialize them to 1/n, so its complexity is O(n). Then, the rest of iteration it loops to get the webpage rank has the complexity O(c(n + e)) such that c is the number of iterations(constant), n is the number of websites(vertices) and e is the number of edges in the graph. Then, I looped over the page_rank map twice to give each website an integer rank (1, 2, 3, 4, …). This has complexity of O(n^2)Consequently, the total complexity of the algorithm = O(n^2) where n is the number of webpages.
Finally, the complexity of the whole algorithm is O(n^2).

Space:

I used an unordered map "page_rank" to store the page rank of each website and it has a spacecomplexity O(n). I used a visited map to mark what I visited, so it space complexity is O(n). Finally, the space complexity is O(n).

II. CTR

I used target_websites map, which includes the websites that have the keywords entered by the user, to calculate the score of each webpage according to the following equation:

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

Such that CTR = $\frac{number\ of\ clicks}{number\ of\ impressions} \times 100$

pseudo-code:

CTR

{

For (i : target_sites)

{

Get the number of impressions of i;

Get the number of clicks of i;

calculate the score using the previous equation;

}

for (i : target_sites)

push back the score to every site and they get arranged descending.

}

Complexity of the algorithm:

Time:

There two loops, each one of them has the complexity O(n) where m is the number of websites that include the target keyword, so O(2n). Finally, the complexity of this algorithm = O(n).

Space:

I used a map "target_website" to store the score of each website and the name site. I choose it specifically to arrange the sites according to the score. The space complexity of it is O(n).

3. main data structures

I. Unordered Map

```
//adjacency list to express the graph connecting between all websites.
unordered_map<string, vector<string>> web_graph;

//It is used to save the key words of each website such the key is URL and the mapped value is the keyword of this URL
unordered_map<string, vector<string>> keyword;

//It is used to save the number of impressions such that the key is URL and the mapped value is the number of impressions
unordered_map<string, vector<int>> number_of_impressions;

//It is used to save the rank of each page such that the key is the URL of the weppage and the mapped value is the rank
```

```
unordered_map<string, double> page_rank;
```

II. Ordered Map

```
//It is used to save the sites that includes the entered keywords in an ordered manner according to their score to display them to the user.

map<double, string, greater <double> > target_sites;
```

4. Design tradeoffs

1. I used most of the time unordered maps because the operation of unordered maps has O (1) complexity.
2. In the OR and AND cases I didn't use erase function in strings to remove them because it has O(n^2) complexity instead I looped over the string and pushed back the words in a temp vector except AND and OR by jumping by the index.