Search Engine Project

CSCE 2203

Analysis and Design of Algorithms Lab

The American University in Cairo

Nermien Elassy

900196006

Supervised under: Dr. Mohamed Elhalaby

1. Websites Indexing:

I have three unordered maps that store all my websites with their URLs as their keys. The one that I am counting on is the "webInfo" unordered map. This one has the URLs of the websites as keys and the value of each key contains all the information concerning the rating of this URL (Impressions, Clicks, Page rank, CTR, and Score). I store my URLs while reading through the function called "read_impressions". Thus, throughout my whole code, I use this unordered map to index the websites.

Pseudocode:

```
void read_impressions()
{
  string line, fword
  fstream ImpFile
  open ImpFile

  while (not end ImpFile)
  {
    store targeted line in ImpFile in line.
    stringstream s(line)
    store website name in fword
    loop on each line to get the impressions of each website
    push it with its value (Number of impressions) to unordered
    map "webInfo".
   }
  }
```

Complexity Analysis:

The whole algorithm for reading the file and storing the data is taking O(n) where n is the number of lines of my read file. I did not add extra complexity to my code for the indexing and made use of the pre-existing function. Thus, the **time complexity of indexing** is just **O(1)** (Insertion complexity in my unordered map) and **space complexity** is **O(n).**

2. Ranking Algorithm:

Definitions:

Number of Impressions: The number of times a webpage appears in a search result.
Number of Clicks: The number of times the user clicked on that webpage.
CTR: Click-Through-Rate.
Page rank.

(According to the value of the above parameters, the score (rank) of the webpage is given.
As illustrated in the referenced video, **The algorithm goes as follows:**

1. Page rank algorithm:

- In the first iteration, all the webpages have the same rate (1/number of webpages).
- In the rest of the iterations, the website is getting its rank according to the following equation: PR(Website) = sum (PR(parent)/V(Website)) from 0 to n where n is the number of websites, PR is the PageRank, the parent is the website parent, and V is the number of outgoing edges of the website's parent.
- The iterations are looping till the difference between the current website rank and the previous one is 1/100000 which is nearly zero difference (error).
- At last, the websites are given ranks as 1,2,3,...,n where n is the number of websites then normalized according to min-max normalization.

Note: I have two functions for calculating the page rank. The first, one is to just give the initial page rank in the zero iteration, and the second one is to calculate the page rank according to the proposed algorithm.

pseudocode:

```
void Initial_PageRank( )
{
        for(i in webInfo)
                website rank = 1/size of webInfo
}

void Update_PageRank()
{
        while(difference does not exceed 0.00001){
          {
              for(i 0 to n) // n is number of websites
              for(j 0 to  m) // m are the children of each website
                    PR[website] += PR[parent] / number of the children of this website's parent
          }
          prev_max = DBL_MAX
          for( i n to 1)  // n is number of websites
                  for(website 0 to n) // n is number of websites
                     if( PR(website) >=  PR(last_max_rank), and PR(website) < prev_max, and
                         website is not visited){
                                  PR(website) = i;
```

```
                                mark current website as visited;
                                }
//Normalizing page rank according to min-max noromalization
        for(i 0 to n) // n is the number of websites
                PR(i) = [PR(i)-1]/ n-1

}
```

Complexity Analysis:

The initial iteration takes O(n) where n is the number of websites. The rest of iterations takes O(I*n*m) where I is the number of iterations (constant), n is the number of websites and m is number of children of each website + O(n^2) (The required complexity to give each website its correct ranking)+O(n) (Normalizing code snippet). Thus, the **time complexity** of the whole algorithm is **O(n^2)** where n is the number of websites and the space complexity is O(2n) (prev_rank unordered map) + O(2n) (visited map). Thus, **space complexity** is **O(n).**

2. CTR (Click-Through-Rate Algorithm):

- The initial CTR given to each website is 1/number of impressions of this website.
- I update CTR then as follows: Website's CTR = number of clicks on this website/number of impressions of the website.

Note: I have two functions for calculating the CTR. The first, one is to just give the initial CTR and the second one is to update the CTR according to the proposed algorithm.

pseudocode:

```
void Initial_CTR( )
{
        for(i 0 in n) // n is number of websites
             websites' CTR = 1/number of impressions of current website
}

void Update_CTR( )
{
        for(i 0 in n) // n is number of websites
            websites' CTR = number of websites' clicks/number of impressions of current website
}
```

Complexity Analysis:

The **time complexity** of the two functions is taking **O(n)** where n is the number of websites and for the **space complexity,** I did not use any extra memory space other than the original unordered map I created while reading the files.

3. Final Score (Ranking):

pseudocode:

```
void Update_Score( )
{
for(i 0 in n) // n is number of websites
websites'score = (0.4 * websites'page rank) + ((1 - (0.1 * websites'impressions / 1 + 0.1 * websites'impressions)) * websites'page rank+ (0.1 *  websites'impressions  / 1 + 0.1 * websites'impressions ) * websites'CTR) * 0.6
}
```

Complexity Analysis:

For the **time complexity** of the function, it is **O(n)** where n is the number of websites and for the **space complexity,** I did not use any extra memory space other than the original unordered map I created while reading the files.

3. Main Data Structures used:

1. Unordered Maps:
    - unordered_map<string, vector<string>> webGraph; // **To store the webGraph (key: website, value: vector of strings of outgoing edges**)
    - unordered_map<string, vector<string>> keyWord; // **To store keywords of each website (Key: website, value: vector of strings of keywords)**
    - unordered_map<string, web_data> webInfo; // **To store ranking information of each website (Key: website, value: struct with (number of impressions and clicks, page rank, CTR, and final score)**
    - unordered_map<string, double> prev_rank; // A temp map to store previous rank while calculating the page rank **(key: website, value: website's page rank)**

2. Vectors:
    - vector<string> finalScore; // **A temp vector to store the targeted website**
    - vector<string> words; // **A vector to store users' search query**

- vector<string> websites; // **A temp vector to display search results according to search query.**
- vector<string> web_names; // **A temp vector to store seach results**

3. Maps:
    - map<string, int> visited; // **A map to check for the visited websites while calculating the page rank**.
    - map<int, string> scoresMap; // **A map to display the websites descendingly according to their final score.**

4. Priority Queues:
    - priority_queue<int, vector<int>, less<int>> queue; // **A priority queue to store the final scores descendingly.**

5. Sets:
    - set<string> webs; // A temp set used to store websites in (OR Case) during search.

4. Tradeoffs:

Most of the operations, I am using unordered maps because they are simply a hash table with operations complexity of O(1) unlike maps which have the complexity of O(logn). I declared most of the unordered maps as global to avoid creating many local vectors and unordered maps which saves space complexity.

5. References:

1. https://www.geeksforgeeks.org/analysis-of-time-and-space-complexity-of-stl-containers/
2. https://iq.opengenus.org/time-complexity-of-hash-table/
3. https://www.youtube.com/watch?v=P8Kt6Abq_rM
4. https://en.wikipedia.org/wiki/Click-through_rate