Search Engine Project

CSCE 2203

Analysis and Design of Algorithms Lab

The American University in Cairo

Gehad Ahmed

900205068

Supervised under: Dr. Reem Haweel

1. Websites Indexing:

There is mainly on unordered map to store all the websites with their URLs as the key and a struct holding all the necessary information as the value

This is the structure of the data kept as the value of the map, it contains the number of impressions, click, the score calculated, the CTR, the normalized rank, along with a vector of all the keywords associated with this website:

```
struct link_data {
    int impressions;
    int clicks;
    double score;
    double CTR;
    double rank;
    vector<string> keywords;
};
```

There is also another unordered map to act as an adjacency list storing all the connections between the different websites (there exist a connected between website A and website B, implies that website A contains a link pointing at website B).

All these data are being read from 4 different files using a general “read_files” function that receives the file name and the purpose of reading from this file

Pseudocode:

```
// Purpose (1: reading the graph, 2: reading the keywords, 3: reading the
impressions, 4: reading the clicks)
read_from_file(filename, purpose) {
       using fstream open filename

    if (file not found)
        close the program

    while (not getting to the end of the file) {
        string line;
        Get the current line from the file
        Split the line by ‘,’ using (split_string) function declared in the code
        for (each word in this line)
            if (purpose == 1)
                Add the edge to the adjacency
            else if (purpose == 2)
                Add the keyword to the map
            else if (purpose == 3)
             // This will happen only one time (cur_line will be of length 2)
                Add the number of impressions
            else if (purpose == 4)
                Add the number of clicks
    }
}
```

Complexity Analysis:

The whole algorithm for reading from the files and storing the data and indexing the websites runs in **O(n)** where n is the number of characters presented in the files, as we are looping through each line to read it and then looping through each character to be able to split the lines. And there is not extra complexity for the indexing as unordered maps takes **O(1)** per operation (insertion or accessing a website). Therefore, the overall **time complexity** is **O(n)** and the **space complexity** is **O(n).**

2. Ranking Algorithm:

Some definitions:

- Number of impressions: the number of times a website appears from a search query.
- Number of Clicks: the number of times a website is being clicked by a user.
- CTR: Click through rate which is 1/number of impressions.
- Page Rank: a number defining the importance of a website based on the number of links pointing at it.

As demonstrated in the referenced video, the algorithm is a follows:

- Initialize the page rank to be (1/number of webpages).
- Going through some iterations and specifying the rank according to this equation

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

  Where **t** is the index of an iteration and $\mathbf{P_j}$ are all pages pointing at the current page ($\mathbf{P_i}$), and **C** is the outdegree of the page.
- The iterations go on till the difference between all numbers from the current iteration and the previous one is nearly zero.

Here, the algorithm is implemented by making use of the DFS (depth first search) traversal technique to pass by all nodes in the graph. So, there is the **DFS function** and **the page rank function**.

Pseudocode:

```
unordered_map<string, bool> visited;
unordered_map<string, double> previous_rank;
void dfs(current_page) {
    Mark current_page as visited by setting the value in the map to true;
    for (all adjacent_pages to current_page) {
      rank of the adjacent_page += previous_rank of current_page /
                                      outdegree of current_page

        if (!visited[adjacent_page]) dfs(adjacent_page);
    }
}

void create_page_rank() {
    Initialize the rank of all pages to be = 1/number of pages

    const int MAX_ITERATIONS = 1e5;
    const int EPSILON = 1e-5;

    for (0 -> MAX_ITERATIONS) {
        Clear the visited map;

        Copy the rank of to the previous_rank map
        Clear the current_rank to be able to update it

        // Traverse all node to update their page rank
        for (auto link : web_info)
            if (!visited[link.first])
                dfs(link.first);

         // Check if the difference between the updated version
              and the old one is nearly 0
        bool changed = false;
        for (auto link : web_info)
            if (abs(link.second.rank - prev_rank[link.first]) > EPSILON)
              changed = true;
        if (!changed) break;
    }
    Create a vector holding final rank and the link to sort based on ranks

    Updating the final rank of the link to be the index from the sorted ranks
    Normalize based on min_max normalization
    // new_value = (value - min) / (max - min)
}
```

Complexity Analysis:

The initialization part has a time complexity of **O(n)** where n is the number of pages.

The iterations loops **I** times where I is the number of iterations (constant) and in each iteration, we run a DFS on the whole graph which runs in **O(n + E)** where n is the number of pages and E is the number of connections presented in the graph, so an overall or **O(I * (n + E))** which is asymptotically to **O(I * n)**.

After that the sorting and normalizing part takes **O(n log n).**

So overall we have (n + I * n + n log n) and as I is constant we can get the **time complexity** of the whole algorithm to be **O(n log n)** and the **space complexity** to be **O(n).**

3. CTR (Click through rate) Calculations:

- The initial CTR is given to each website to be 1/number of impressions of the website.
- Then updating it with the same formula to adapt the change of the number of impressions.

Pseudocode:

```
void create_CTR() {
    for (all websites)
        website's CTR = 1.0 / website's impressions;
}
```

Complexity Analysis:

Calculating CTR is taking O(n) where n is the number of websites and for the space complexity, there is not extra memory used other than the original unordered map used previously.

4. Final Score (Main factor in ranking the result pages):

- It is calculated through the defined equation for the score as shown below in the pseudocode.

Pseudocode:

```
void create_score() {
    for (all websites)
        website's score = (0.4 * website's rank) + ((1 - (0.1 * website's
impressions / (1 + 0.1 * website's impressions))) * website's rank + (0.1 *
website's impressions / (1 + 0.1 * website's impressions)) * website's CTR) * 0.6;
}
```

Complexity Analysis:

Calculating the score is taking O(n) where n is the number of websites and for the space complexity, there is not extra memory used other than the original unordered map used previously.

5. Main Data Structures used:

1) Struct: (Used to having all data associated with each page

```
struct link_data {
    int impressions;
    int clicks;
    double score;
    double CTR;
    double rank;
    vector<string> keywords;
};
```

2) Unordered Maps:

```
// Storing the adjacency list of the graph
unordered_map<string, vector<string>> adjacency;
unordered_map<string, link_data> web_info; // Storing all data of the pages

// Used in the DFS function to know the visited pages
unordered_map<string, bool> visited;

// Storing the previous rank to help in calculating the next one in the page rank
algorithm
unordered_map<string, double> prev_rank;
```

3) Vectors:

```
vector<string> ans_websites; // Storing the answers of the search queries
```

6. Tradeoffs:

In most operations, unordered maps are used to avoid having regular maps of complexity O(log n) and get the benefit of the unordered maps acting as a hash table to hash the different web pages with operation complexity of O(1).
Lots of variables are declared globally to use them in deferent locations easily.

7. References:

- https://www.youtube.com/watch?v=P8Kt6Abq_rM
- https://en.wikipedia.org/wiki/Click-through_rate