SIMPLE SEARCH ENGINE

PROJECT REPORT

Ibrahim Gohar

Student ID: 900203321

School of Sciences and Engineering

Department of Computer Science and Computer Engineering

The American University in Cairo

November 2021

Contents

1 Introduction

1.1 Abstract

The project is a simple search engine. It uses the connections between the websites to construct a web graph. Then, it starts calculating the rank of each page using page rank. Moreover, the projects reads the number of impressions (the number of times the web page was displayed as a search result) and clicks (the number of times the web page was clicked by the user) from CSV files to calculate the score of each page using CTR (click-through rate). It also reads the keywords of each website and map it. The project basically receives a search query from the user; it may contain 'AND', 'OR', or '"' (quotation marks). The project start searching for the keyword and return the websites associated with it. The next step is to sort the websites according to their scores and update both the impressions and clicks. At the end of each session, the program saves the new data into the CSV files so it can access them later on.

1.2 Requirements list

The following files are required:

clicks.csv: A CSV file containing the website associated with the number of clicks seperrated by [**,**]. **the format :** *website url***,***number of clicks*

impressions.csv: A CSV file containing the website associated with the number of impressions seperrated by [**,**]. **the format :** *website url***,***number of impressions*

key_words.csv: A CSV file containing the website associated with the number of impressions seperrated by [**,**]. **the format :** *website url***,***keyword 1***,***keyword 2***,...**

web_graph.csv: A CSV file containing the website associated with the website it is connected to and they are separated by [**,**]. **the format :** *website one url***,***website two url*

1.3 Project's Outline

The project could be divided into three main phases which are:

Reading phase: The program start reading the CSV files and construct the web graph and the trie (that stores the keywords and their associated web pages).

Ranking phase: The program start to calculate both the page rank and scores of each web page using CTR (click-through rate) and stores them for later usage.

Searching phase: The program takes a search query from the user and search for it in the trie then returns the associated websites.

Sorting phase: The program takes the resulted websites and and sort them using heap. Then, it returns the websites sorted.

Then, the user can either choose an web page to open or make a new search.

Saving phase: The program updates both the impressions and clicks for each web page in the CSV files for future usage of them.

i.e. the project is case sensitive. If the user searched without using the quotation marks ("), the program will display the websites associated with it with out considering the case of the search query. However, if the user used the quotation marks ("), the program will display only the websites associated with it in that specific case.

2 Classes

2.1 Trie

The class is designed to contain the keywords and the websites associated to it. It allows insertion and search for the keywords.

2.1.1 Attributes

`bool isEndOfWord` : A Boolean to indicate if the the keyword inserted to the Trie ended or not.

`vector<string> websites` : A Vector to store the websites associated with a specific keyword.

`Trie* children[]` : An array containing 62 (no. of English letters in upper and lower case) pointers to other tries that represents the children of the root.

2.1.2 Methods & Constructor

`Trie()` : A constructor to set `isEndOfWord` to `false` and initialize the `children` array.

`void insert(string, string, bool)` : Insert the keyword and its associated website to the trie in either case sensitive manner or not.

`bool remove(Trie*&, string)` : Returns `true` in case the function removed the keyword and `false` if it did not.

`vector<string> search(string, bool)` : Returns the resulted web pages associated with the keyword entered in either case sensitive manner or not.

`bool isEmpty(Trie const*)` : Returns `true` if the trie is empty and not otherwise.

2.2 Graph

The class is designed to contain the web graph representing the connections between the web pages and calculating the pages rank.

2.2.1 Attributes

`vector<vector<double>> adjMatrix` : A 2D Vector to store the the matrix representation of the web graph.

`vector<double> pr` : A Vector to store the rank calculated for each web page.

`vector<int> out` : A Vector to store the out going directed edges from each web page.

`int N` : Stores the number of web pages in the web graph.

`double dampingF` : Stores the damping factor used in the page rank algorithm.

`double stoppingF` : Stores the stopping factor used in the page rank algorithm.

2.2.2 Methods & Constructor

`Graph(vector<Edge> edges, int n, double df, double sf)` : A constructor to set the initial values of the web graph.

`void add_edge(int u, int v)` : Adds an edge to the web graph.

`bool is_zeros(int j)` : Returns `true` if the selected column is all zero and `false` otherwise.

`void convert(int j, double val)` : Convert the selected column to a specified value.

`bool done(vector<double> x, vector<double> y)` : Returns `true` if the page rank algorithm is done and `false` otherwise.

`void pagerank()` : Runs page rank algorithm and stores the page ranks into `vector<double> pr`.

`void printGraph()` : Prints the contents of the web graph.

3 Data structures

3.1 Edge

The edge represents an edge on the web graph which have source and destination. It has been used in the Graph class.

`int src` : It contains the index of the source node.

`int dest` : It contains the index of the destination node.

`Edge(int s, int d)` : A Constructor to initialize the edge data structure.

3.2 STL

3.2.1 Map

Stores the web pages and their scores indexed so the access process is easier.

3.2.2 Vector

Stores the clicks and impressions of each web page so it can be saved after the session is ended.

3.2.3 Heap

Stores the resulted web pages sorted so it can be displayed in order.

3.2.4 Unordered set

Removes the repeated results in OR case.

4 Psuedocode

4.1 Indexing

webpage url → webapge index

This is how strings are indexed to integers:

```
int c = 0;
while(getline(Keywords,line)) {
    stringstream iss(line);
    getline(iss, link, ',');
    webpages[link] = c;
}
```

Then Trie has been used to map the keywords and associate the web pages to every one:

```
while(getline(Keywords,line)) {
    stringstream iss(line);
    getline(iss, link, ',');
    for(int i = 0; i < MAX_LENGTH; i++){
        getline(iss,str,',');
        if (str.size()==0)
        {
            break;
        }
        keyword_trie->insert(str, link);
```

```
    }
}
```

4.2 Searching

The program first determine the type of search and starts the search:

```
Trie* root = this;

for (int i = 0; i < key.length(); i++) {
    int index = key[i] - 'A';
    if (root->children[index] == nullptr)
        break;

    root = root->children[index];
}

if (root != NULL && root->isEndOfWord)
{
    return root->websites;
}
```

4.3 Page Rank

The brute force approach of the page rank algorithm takes time complexity of $O(n^3)$. That is why the matrix representation of the the web graph reduces the complexity to around $O(n^2)$. However, the nodes that are not connected to any other nodes can not be accessed with that approach. The solution that has been found to be useful in such cases is using a predetermined damping factor. The idea behind the damping factor is to give 85 percent probability for the user to access a web page connected to the current one he/she is viewing and 15 percent probability for accessing a completely separated one.

First, it calculates the probabilities matrix :

```
for(int i = 0; i < N; i++)
{
    for(int j = 0; j < N; j++)
    {
        if(adjMatrix[i][j])
        {
            adjMatrix[i][j] = adjMatrix[i][j]/out[i];
        }
    }
}
```

Second, it calculates the stochastic matrix :

```
for(int j = 0; j < N; j++)
{
    if(is_zeros(j))
    {
        convert(j, 1/N);
    }
}
```

Third, it multiplies the matrix by (1 - damping factor) :

```
for(int i = 0; i < N; i++)
{
    for(int j = 0; j < N; j++)
    {
        adjMatrix[i][j] = adjMatrix[i][j] * dampingF;
    }
}
```

Then, it generate the transition matrix by summing the prior one with the multiplication of 1/n matrix and damping factor :

```
double rd = 1 - dampingF;
for(int i = 0; i < N; i++)
{
    for(int j = 0; j < N; j++)
    {
        adjMatrix[i][j] = adjMatrix[i][j] + (rd/N);
    }
}
```

After that, it generates the rank matrix :

```
for(int i = 0; i < N; i++)
{
    pr.push_back(1.0);
}

vector<double> pr_new = pr;
bool d = false;
for(int i = 0; i < 100; i++)
{
    if(done(pr, pr_new))
    {
        break;
    }
    for(int i = 0; i < N; i++)
    {
        double tmp = 0;
        for(int j = 0; j < N; j++)
        {
            tmp += adjMatrix[i][j] * pr[j];
        }
        pr_new[i] = tmp;
    }
    pr = pr_new;
}
pr_new.clear();
```

At the end, it normalize the results :

```
double sum = 0;
for (int i = 0; i < N; i++)
{
    sum += pr[i];
}
for (int i = 0; i < N; i++)
{
    pr[i] /= sum;
```

```
}
```

5 Complexity

5.1 Time Complexity

5.1.1 Indexing

- **Map indexing :** The program indexes the websites into integers while looping through them. If we have **n** webpages so the Time complexity is **O(n)**.
- **Trie construction :** The program accepts **K** keywords and construct them in a Trie so it loops in **K** keywords and each one has **L** length so the time complexity is **O(K*L)**.

5.1.2 Searching

The program searches through **K** keywords and each one has **L** length so the time complexity is **O(K*L)**.

5.1.3 Page Rank

First, it calculates the probabilities matrix with two **for** loops so it becomes n^2

```
for(int i = 0; i < N; i++)
    for(int j = 0; j < N; j++)
```

Second, it calculates the stochastic matrix using also two **for** loops so it becomes in the worst case scenario n^2

```
for(int j = 0; j < N; j++)
{
    if(is_zeros(j))
    {
        convert(j, 1/N);
    }
}
```

Third, it multiplies the matrix by (1 - damping factor) using also two **for** loops so it becomes n^2

```
for(int i = 0; i < N; i++)
{
    for(int j = 0; j < N; j++)
    {
        adjMatrix[i][j] = adjMatrix[i][j] * dampingF;
    }
}
```

Then, it generate the transition matrix by summing the prior one with the multiplication of 1/n matrix and damping factor using also two **for** loops so it becomes n^2

```
double rd = 1 - dampingF;
for(int i = 0; i < N; i++)
{
    for(int j = 0; j < N; j++)
    {
        adjMatrix[i][j] = adjMatrix[i][j] + (rd/N);
    }
}
```

After that, it generates the rank matrix using one **for** loop and two **for** loops for 100 time or less so it becomes in the worst case scenario $100n^2 + n$

```
for(int i = 0; i < N; i++)
{
    pr.push_back(1.0);
}

vector<double> pr_new = pr;
bool d = false;
for(int i = 0; i < 100; i++)
{
    if(done(pr, pr_new))
    {
        break;
    }
    for(int i = 0; i < N; i++)
    {
        double tmp = 0;
        for(int j = 0; j < N; j++)
        {
            tmp += adjMatrix[i][j] * pr[j];
        }
        pr_new[i] = tmp;
    }
    pr = pr_new;
}
pr_new.clear();
```

At the end, it normalize the results using one **for** loop twice so it becomes $2n$

```
double sum = 0;
for (int i = 0; i < N; i++)
{
    sum += pr[i];
}
for (int i = 0; i < N; i++)
{
    pr[i] /= sum;
}
```

So the total becomes $104n^2 + 3n$, and **the time complexity is** $O(n^2)$

5.2 Space Complexity

5.2.1 Trie

The trie uses average W websites and array of N nodes which are 62 and the keywords have an average L length. So the space complexity is $O(W * L)$.

5.2.2 Web graph

The web graph first requires **E** edges to construct a webgraph of **W** webpages. So, the space complexity is $O(W + E)$.

6 Design Trade-offs

6.1 Trie choosing

The keywords are stored under the websites and not the opposite to ensure that the searching process is faster where we will not have to iterate over each and every website.

6.2 Using maps

Maps were used to store the websites by hashing them to their indices where they are sorted ascendingly based on the key to facilitate accessibility.

6.3 Using unordered sets in the search function

The unordered set was used to remove any repetitions that may happen in the search results display where unordered sets do not store duplicate elements.

6.4 Matrix Representation

This trade off was for trading space with time. Using the matrix representation used more space but reduced the time complexity of the ranking algorithm greatly which is more efficient.