1. Important notes:

* Required files for the program to work: **impressions.csv, clicks.csv, keywords.csv, webgraph.csv**
* Key words are all separated by commas and **NOT** by spaces

1. Indexing:

* **Indexing the websites:**
* Created a function that counts all available websites
* Created a 2D dynamic array to store the information of all available websites
* Each website has a vector of keywords, where keywords are added to each corresponding website. Keywords are extracted from keywords.csv
* Each site has a vertex number that represents the site the graph
* Each time the program is opened: number of impressions, number of clicks and keywords are all set in the corresponding variables for each site.
* Then, CTR (Click Through Rate) is calculated based on the formula:
* Then the score of each site is calculated **after** calculating the PageRank of each site
* Space Complexity: O(n)
* Time Complexity: O(n2), since there is a while loop inside a for loop that aims to split each line into keywords by commas
* Detailed Analysis:
* Complexity : T(n) = n+n2
* Thus, worst complexity is: O(n2)

**PSEUDOCODE:**

Class website

{

int vertexNumber;

    string link = "";

    vector<string> keywords;

    double pageRank=0;

    double numberOfClicks=0;

    double numOfImpressions=0;

    double ctr=0;

    double score=0;

}

Void setData(website \*sites, int numberOfSites)

{

For int i to numberOfSites {

Open initialization input files;

Get each keyword line - >keywords vector; then push the vector inside the it’s corresponding website;

Repeat for the impressions and the number of clicks;

}

}

* **Indexing: (Reading the query) & Searching:**
* There are three Boolean that work as indicators: AND, OR, and quotation
* Thus, there are 4 possibilities: Either one of them is true, or none of them
* Keywords are extracted from the query, while not taking words: AND, OR, and the quotation. If there exists a quotation, the quotation marks are removed, and the search starts for the whole block of word inside the quotation marks.
* In case of AND is true: all keywords **MUST** be found in the same website
* In case of OR is true: all websites that have **AT LEAST ONE** of the keywords are added in a vector of sites
* In case of quotation is true: it searches for the word as it is inside the quotation in all available websites, and websites resulted are added in a vector of sites
* Then, all websites are sorted in a **DESCENDING** order using Merge-Sort algorithm according to their scores, **after** that PageRank was already calculated for each site.
* Then, each time a website appears in the search results, number of impressions is incremented by 1
* If the user chooses a specific website, number of clicks gets incremented
* Then, immediately, the data are saved in the original 2D dynamic array of websites, and in the files: **impressions.csv, clicks.csv**
* Then, the CTR, and score components of each website is being calculated once again after the update in the variables.
* Space Complexity: O(n)
* Time Complexity of searching: O(n3)
* Detailed Analysis:
* Complexity : T(n) = n+n2+n log(n) + n3
* Thus, worst complexity is: O(n3)

**PSEUDOCODE:**

void search(website \*sites, int numOfSites){

Bool AND, OR, quotation //they work as indicators

Index=0

String temp;

While(query splitting did not end)

{

If it contains “OR”: OR = true

If it contains “AND”: AND = true

If if contains “”: quotation = true

}

Int scores[]; //array that stores the scores of the sites that have the same entered keyword

If(AND = false and OR = false and quotation=false) //basic case: when all of the Booleans are false

{

For(int i=0 to numberOfSites)

{

Vector<string>::iterator it starts from beginning of adjacency list of site[i]

While(iterates over adjacency list of each site)

{

If(keyword found in site[i])

{

scores[index] = score of site[i]

Index++;

}

Iterator++

}

}

removeDuplicates(scores, 0, index); //to remove any score duplicated just in case any duplications occurs

mergeSort(scores, 0, index) //to sort all scores in a descending order

vector<website> availableSites; //stores the sites that have the scores in array of scores, and they will be added in a descending order since scores were first sorted.

for(int i=0 to index)

                {

                    for(int j=0 to numberOfAllSites)

                    {

                        if(scores[i]==sites[j].score)

                        {

                            availableSites.push\_back(sites[j]);

                        }

                    }

                }

for(int i=0 to availableSites.size()) //to show results in a sorted way

                {

                    cout<<i+1<<"- "<<availableSites[i].link<<endl;

                    availableSites[i].numOfImpressions = availableSites[i].numOfImpressions + 1; //increment impressions of each site since they appeared in the search results

                }

for(int i=0 to numberOfTotalSites)

{

for(int j=0 to )

{

if(sites[i].link==availableSites[j].link) //these two loops aim to update the original array of websites called sites with latest values after the number of impressions is being increased after each search result in the vector of availableSites

{

sites[i].numOfImpressions = availableSites[j].numOfImpressions;

sites[i].numberOfClicks = availableSites[j].numberOfClicks;

}

}

}

Then, saveData(sites,numOfSites); //aims to update the files: impressions.csv and clicks.csv with all latest values.

Then, if user chooses websites, numbers of clicks is being increased and the two above functions are re-called again to keep all values in variables and files up to date.

}

If(AND = **true** and OR = false and quotation=false) //AND CASE

{

stringstream split(query);

string data;

vector<string> keyWords; //store all keywords except the word “AND”

while (query splitting did not end)

{

string data;

getline(split, data,' ');

if(data!=”AND”)

{

Keywords.push\_back(data)

}

}

Then, For(int i=0 to numberOfSites)

{

Vector<string>::iterator it starts from beginning of adjacency list of site[i]

While(iterates over adjacency list of site[i])

{

If(keyword found in site[i])

{

scores[index] = score of site[i]

Index++;

}

Iterator++

}

}

removeDuplicates(scores, 0, index); //to remove any score duplicated just in case any duplications occurs

mergeSort(scores, 0, index) //to sort all scores in a descending order

vector<website> availableSites; //stores the sites that have the scores in array of scores, and they will be added in a descending order since scores were first sorted.

for(int i=0 to index)

                {

                    for(int j=0 to numberOfAllSites)

                    {

                        if(scores[i]==sites[j].score)

                        {

                            availableSites.push\_back(sites[j]);

                        }

                    }

                }

for(int i=0 to availableSites.size()) //to show results in a sorted way

                {

                    cout<<i+1<<"- "<<availableSites[i].link<<endl;

                    availableSites[i].numOfImpressions = availableSites[i].numOfImpressions + 1; //increment impressions of each site since they appeared in the search results

                }

for(int i=0 to numberOfTotalSites)

{

for(int j=0 to )

{

if(sites[i].link==availableSites[j].link) //these two loops aim to updates the original array of websites calles sites with latest values after the number of impressions is being increased after each search result

{

sites[i].numOfImpressions = availableSites[j].numOfImpressions;

sites[i].numberOfClicks = availableSites[j].numberOfClicks;

}

}

}

Then, saveData(sites,numOfSites); //aims to update the files: impressions.csv and clicks.csv with all latest values.

Then, if user chooses websites, numbers of clicks is being increased and the two above functions are re-called again to keep all values in variables and files up to date.

}

}

If(AND = false and OR = **true** and quotation=false) //OR CASE

{

stringstream split(query);

                string data;

                vector<string> keyWords;

while (query splitting did not end)

{

string data;

getline(split, data,' ');

if(data!=”OR”)

{

Keywords.push\_back(data)

}

}

Then, For(int i=0 to numberOfSites)

{

Vector<string>::iterator it starts from beginning of adjacency list of site[i]

While(iterates over adjacency list of site[i])

{

If(keyword found in site[i])

{

scores[index] = score of site[i]

Index++;

}

Iterator++

}

}

removeDuplicates(scores, 0, index); //to remove any score duplicated just in case any duplications occurs

mergeSort(scores, 0, index) //to sort all scores in a descending order

vector<website> availableSites; //stores the sites that have the scores in array of scores, and they will be added in a descending order since scores were first sorted.

for(int i=0 to index)

                {

                    for(int j=0 to numberOfAllSites)

                    {

                        if(scores[i]==sites[j].score)

                        {

                            availableSites.push\_back(sites[j]);

                        }

                    }

                }

for(int i=0 to availableSites.size()) //to show results in a sorted way

                {

                    cout<<i+1<<"- "<<availableSites[i].link<<endl;

                    availableSites[i].numOfImpressions = availableSites[i].numOfImpressions + 1; //increment impressions of each site since they appeared in the search results

                }

for(int i=0 to numberOfTotalSites)

{

for(int j=0 to )

{

if(sites[i].link==availableSites[j].link) //these two loops aim to updates the original array of websites calles sites with latest values after the number of impressions is being increased after each search result

{

sites[i].numOfImpressions = availableSites[j].numOfImpressions;

sites[i].numberOfClicks = availableSites[j].numberOfClicks;

}

}

}

Then, saveData(sites,numOfSites); //aims to update the files: impressions.csv and clicks.csv with all latest values.

Then, if user chooses websites, numbers of clicks is being increased and the two above functions are re-called again to keep all values in variables and files up to date.

}

}

}

Final case, If(AND = false and OR = false and quotation=**true**) //Quotation Case

{

query=temp; //temp = query before being split in first time to check if it contains AND, OR, or “”

              query = query.substr(1, query.size() - 2); //Quotation marks are removed

Then, For(int i=0 to numberOfSites)

{

Vector<string>::iterator it starts from beginning of adjacency list of site[i]

While(iterates over adjacency list of site[i])

{

If(keyword found in site[i])

{

scores[index] = score of site[i]

Index++;

}

Iterator++

}

}

removeDuplicates(scores, 0, index); //to remove any score duplicated just in case any duplications occurs

mergeSort(scores, 0, index) //to sort all scores in a descending order

vector<website> availableSites; //stores the sites that have the scores in array of scores, and they will be added in a descending order since scores were first sorted.

for(int i=0 to index)

                {

                    for(int j=0 to numberOfAllSites)

                    {

                        if(scores[i]==sites[j].score)

                        {

                            availableSites.push\_back(sites[j]);

                        }

                    }

                }

for(int i=0 to availableSites.size()) //to show results in a sorted way

                {

                    cout<<i+1<<"- "<<availableSites[i].link<<endl;

                    availableSites[i].numOfImpressions = availableSites[i].numOfImpressions + 1; //increment impressions of each site since they appeared in the search results

                }

for(int i=0 to numberOfTotalSites)

{

for(int j=0 to )

{

if(sites[i].link==availableSites[j].link) //these two loops aim to updates the original array of websites calles sites with latest values after the number of impressions is being increased after each search result

{

sites[i].numOfImpressions = availableSites[j].numOfImpressions;

sites[i].numberOfClicks = availableSites[j].numberOfClicks;

}

}

}

Then, saveData(sites,numOfSites); //aims to update the files: impressions.csv and clicks.csv with all latest values.

Then, if user chooses websites, numbers of clicks is being increased and the two above functions are re-called again to keep all values in variables and files up to date.

}

}

}

}

1. PageRank Algorithm

* Initially, all nodes have initial page rank =
* I used the matrix representation of PageRank, and calculated it using the method: Matrix-Vector Multiplication
* My reference is: [PageRank Algorithm - Matrix Representation - YouTube](https://www.youtube.com/watch?v=kSmQbVxqOJc)
* The only part that was not in this video, is the case when there is a dangling node (a node that has no outgoing edges), after searching and understand it, the solution to this problem is to assume that this dangling node is connected to all other nodes and itself so that PageRank calculation becomes logical and valid.
* So, the initial vector is a vector containing initial Page Ranks which is equal to:
* Then, I do matrix-vector multiplication which results in giving the PageRank of each node in the graph
* Then, score is calculated based on values of number of impressions, CTRs, and PageRank
* Space Complexity: O(n)
* Time Complexity of calculating PageRank: O(n3)
* Detailed Analysis:
* Complexity : T(n) = n+n2+n3
* Thus, worst complexity is: O(n3)

**PSEUDOCODE:**

Vector<double> initial\_page\_rank;

For(int i=0 to totalNumberOfSites)

{

Initial\_page\_rank.push\_back(1/totalNumberOfSites)

}

Then,

* C=total number of sites
* Matrix: is a matrix that contains all probabilities as mentioned by the video.
* In case of dangling node (a node that has no outgoing edges), I must assume it is connected to all nodes and itself.
* **There is a vector called result, which hold the result of matrix-vector multiplication**

double result[c];

        double\*\* resultMatrix = new double\*[c]; //2d matrix of probabilities

        for (int i = 0; i < c; i++)

        {

            resultMatrix[i] = new double[c];

        }

        for (int i = 0; i < c; i++)

        {

            for (int j = 0; j < c; j++)

            {

                resultMatrix[i][j] = 0; //zeroing all elements of matrix

            }

        }

        for (int i = 0 to c) //matrix multiplication: matrix2

        {

            for (int j = 0 to c)

            {

                resultMatrix[i][j] = 0;

                for (int k = 0; k < c; k++)

                {

                    resultMatrix[i][j] += matrix[i][k] \* matrix[k][j];

                }

            }

        }

        for (int i = 0 to c)

        {

            result[i]=0;

            for (int j = 0 to c)

            {

                result[i] += (resultMatrix[i][j]\*initialPRs[j]);

            }

            sites[i].pageRank=result[i]; //set the PageRank of each site

        }

Then, from the main function, the score is

1. Main data structures used:

* Dynamic Arrays (1D & 2D) to form the matrices and the dynamic array of websites
* Graph to create the webgraph
* Vectors to store keywords of each site, initial page ranks, final page ranks, keywords to search for, and to store all websites that have keywords from the query