

Lab Project Report

By: Nada Badawi



November 27, 2021

UID: 900201364

1. Pseudo-code:

* Indexing:

Process\_Input(string query, string& res1, string& res2, condition& cond)

{

if (first and last characters of query are quotation marks)

{cond = none; res1 = query.substr(1, query.length() - 2); res2 = "";}

else if (there is a space character in query)

{

if (" AND " is found in query)

{cond = AND; res1 = query.substr(0, query.find(" AND "));

res2 = query.substr(query.find\_last\_of(" AND ") + 1, query.length() - query.find\_last\_of(" AND ") - 1);}

else if (" OR " is found in query)

{cond = OR; res1 = query.substr(0, query.find(" OR "));

res2 = query.substr(query.find\_last\_of(" OR ") + 1, query.length() - query.find\_last\_of(" OR ") - 1); }

else

{cond = OR; res1 = query.substr(0, query.find(" "));

res2 = query.substr(query.find(" ") + 1, query.length() - query.find(" ") - 1);}

}

else if (there is no space in the query) //one keyword only

{res1 = query; res2 = ""; cond = one;}

else

cond = error;

}

MatchStrings(string res1, string res2, condition c)

{

vector <string> results(webNum);

i = 0, j = 0

string test, website, space = " ";

ifstream keywords;

keywords open file

if (c == error)

{

resize results to 1.

results[0] = "No matching results found. Please check your spelling or try another search query.\n";

}

else

{

while (keywords >> website) //Did not reach the end of the file

{

Websitesj.SetName(website);

j++;

getline(keywords, test);

if (c == AND)

if (res1 is found in test && res2 is found in test)

{results­i = website; i++;}

else if (c == OR)

if (res1 is found in test || res2 is found in test)

{resultsi = website; i++;}

else if (c == none)

if (res1 is found in test)

{results­i = website; i++;}

else if (c == one)

if (res1 is found in test)

{resultsi = website; i++;}

}

}

keywords.close();

results.resize(i); //resize to i, the number of elements it holds

if (results is empty)

{

resize results to 1

results[0] = "Error\n";

}

return results;

}

void Search()

{

system("cls"); //Clear console

cout << "\n\t\t\tPlease Type Your Search Query\n"

string searchquery, res1, res2, dummy

condition cond

WebGraph G

cout << "\n\t\t\tYour Query: "

getline(cin, dummy)

getline(cin, searchquery)

Process\_Input(searchquery, res1, res2, cond)

vector <string> r = MatchStrings(res1, res2, cond)

if (r[0] == "Error\n")

{cout << "\n\t\t\t" << r[0] << endl; Menu();}

else

{

G.SetGraphImpressionsClicks(websites) //Sets impressions, clicks, and adjacency matrix

G.PageRank() //Calculates and sets the page rank for each website

G.FilterResults(r) //matches the website names to be displayed as results to those of type webpage in the vector websites

vector <Webpage> wp = G.GetWebPages() //returns vector webpages of search results

wp = G.UpdateImpressions() //Update number of impressions

DisplayRPage(wp, res1, res2) //Displays the results on console

}

}

WebGraph:: FilterResults (vector <string> words)

{

//Page rank, ctr, score, and name are all initialized to the vector of Webpage (results)

Find the max and min page rank and store it in the static attributes maxPR and minPR

for i = 0 to results.size( ) - 1 do

{

counter = 0

for j = 0 to words.size( ) – 1 do

{

if resultsi.name ≠ wordsj

counter++

if counter == words.size( )

{delete resultsi from vector of webpages. i--;}

}

}

}

* Page Rank

vector<vector<double>> MatrixH()

{

vector<double> counter //Size: n

vector<vector<double>> H //Size: n x n

for j = 0 to n – 1 do

for i = 0 to n – 1 do

if (Graphij == 1)

counterj++;

for j = 0 to n – 1 do

for i = 0 to n – 1 do

{

if (Graphij == 1)

Hij = 1.0 / counterj;

}

return H;

}

void PageRank()

{

vector<vector<double>> result //Size n x 1

vector<vector<double>> power =PowMatrix(MatrixH(), 2) //matrix H to the power 3

//PowMatrix was implemented in lab 5 task 3 🡪 same algorithm

vector<vector<double>> pagerankMatrix //Size: n x 1

for i = 0 to n - 1 do

pagerankMatrixi0 = pageranki;

//Multiply power matrix with the page rank matrix to get new page rank

result = MatrixMult(power, pagerankMatrix, Count, 1, Count);

for i = 0 to n - 1 do

{

if (webpagesi.clicks != 0) //Checks whether there is historical click data for webpage

pageranki = resulti0;

webpagesi.pagerank = pageranki;

}

}

1. Time and Space Complexity (Worst-case scenario)

* Indexing

MatchStrings Function: String::find() has a complexity O(N) where N is the number of characters in the new string (keyword found)

N = length of the keyword, n = number of websites.

FindMaxMinPR Function:

where k is a constant.

FilterResults Function:

Vector::erase() 🡪 O(n) linear complexity

where m is the number of websites that have keywords matched to the search query.

Total Complexity:

Space Complexity:

MatchStrings( ):

vector <string> results(webNum) 🡪 size: n

string test, website, space = " "; 🡪 size: 3n

FindMaxMinPR( ): O(1) It does not need any extra space to function properly.

vector <Webpage> results 🡪 size: n

Total Space Complexity: 3n + n + n + 1 = 5n + 1 = Ο(n).

* Ranking:

WebGraph::MatrixH( ): O(

WebGraph::PowMatrix( ): Complexity: Ο(log 3) = O(, according to the slides (Lab 5).

MatrixMult( ) : O(

PageRank( ): O(

Total Complexity: O (+ O (

Space Complexity: vector<vector<double>> result (Count, vector<double> (1)) 🡪 matrix nx1 So space complexity is Ο(n).

vector<vector<double>> power = PowMatrix (MatrixH(), 2) and vector<vector<double>> pagerankMatrix(Count, vector<double>(Count))

🡪 Each Size = n2. So, space complexity = n2 + n2 = 2n2 = Ο(n2).

Total space complexity: Ο(n) + Ο(n2) = Ο(n2)

1. Main data structures:

The code required using a graph represented as a 2d matrix (adjacency matrix) to show the edges between every two nodes or webpages. The matrix was implemented as a 2d static array since the size of the array does not change. This is due to the fixed number of web pages we have in the graph. A vector of vectors was used to calculate the page rank, which allows resizing. This, in a way, reduces any unnecessary space complexity. Vectors were also used in the code to store the websites in the file and the results to be displayed. Since the number of websites of the search results is changing depending on the search query, then a vector is the most suitable option for storing this kind of data. Similarly, the number of websites may change in the case of having a dynamic graph. Therefore, I opted for having a vector storing the list of web pages that the graph involves.

1. Design trade-offs:

The primary trade-off was not storing all of the keywords in a vector in the webpage class since not all of them are needed. This significantly reduces the space complexity because a vector of keywords of size m, for example, for each of the n webpages, will be occupied. So, the space complexity would be Ο(m\*n) instead of Ο(n).