Name: Maya Hussein Date: Saturday, November 27th, 2021

UID: 900201198

CSCE 2203-04

Instructor: Dr. Mohamed El-Halaby

Search Engine Report

iSearch

Visual Graph Representation:

Diagram

Description automatically generated

Pseudo-Code for Indexing and Ranking Pages along with Time and Space Complexity:

1. Search Query()
   1. Pseudocode

if (keyword.find("AND") < keyword.length())

{

string keyword\_pt1 = keyword.substr(0, keyword.find("AND") - 1);

string keyword\_pt2 = keyword.substr(keyword.find\_last\_of("AND") + 2, keyword.length() - 1);

for (j = 0 🡪 web\_pages.size())

for (i = 0 🡪web\_pages[j].keywords.size())

{

if (web\_pages[j].keywords[i] == keyword\_pt1)

for (k = 0 🡪 web\_pages[j].keywords.size())

if (web\_pages[j].keywords[k] == keyword\_pt2)

{

searched\_websites.push\_back(web\_pages[j]);

m++;

}

}

searched\_websites.resize(m);

}

else if (keyword.find("OR") < keyword.length())

{

string keyword\_pt1 = keyword.substr(0, keyword.find("OR") - 1);

string keyword\_pt2 = keyword.substr(keyword.find\_last\_of("OR") + 1, keyword.length() - 1);

for (j = 0 🡪 web\_pages.size())

for (i = 0 🡪 web\_pages[j].keywords.size())

{

if (web\_pages[j].keywords[i] == keyword\_pt1 || web\_pages[j].keywords[i] == keyword\_pt2)

{

searched\_websites.push\_back(web\_pages[j]);

m++;

}

}

searched\_websites.resize(m);

}

else if (keyword.find("\"") == 0)

{

keyword.erase(0,1);

keyword.erase(keyword.size() - 1,keyword.size());

for (j = 0 🡪 web\_pages.size())

for (i = 0 🡪 web\_pages[j].keywords.size())

if (web\_pages[j].keywords[i] == keyword)

{

searched\_websites.push\_back(web\_pages[j]);

m++;

}

searched\_websites.resize(m);

}

else

{

cout << "No Results found. Error 404" << endl;

exit; }

return searched\_websites; }

b. Time and Space Complexity: Time: O(n3) / Ω(1) - Space: O(n)

1. setGraph()
   1. Pseudocode

void setGraph()

{

ifstream webGraphFile;

string website1, website2, comma;

open file

while (!webGraphFile.eof()) {

webGraphFile >> website1 >> comma >> website2;

if (website1 == web\_pages[j].name)

for (i = 0 🡪 web\_pages.size())

if (website2 == web\_pages[i].name)

{

graph[i][j] = 1;

edge\_counter[i]++;

break; }

else

j++; }

webGraphFile.close();

}

* 1. Time and Space Complexity O(n) – O(n2)

1. ReadFile()
   1. Pseudocode

Open files and check for failure

if fail print error opening file

else

{

while (!keywordsInFile.eof())

{

impressionsInFile >> dummy >> comma >> web\_pages[i].impressions;

keywordsInFile >> web\_pages[i].name;

keywordsInFile >> comma;

while( comma == ",")

{

keywordsInFile >> web\_pages[i].keywords[j];

keywordsInFile >> comma;

j++;

}

web\_pages[i].keywords.resize(j);

i++;

}

Close all files

* 1. Time and Space Complexity Time: O(n2) Space: O(n)

1. Page Rank
   1. Pseudocode

double\*\* pageRank()

{

double\*\* initial= new double\*[16];

for (i = 0 🡪 web\_pages.size())

{

initial[i] = new double[1];

initial[i][1] = 1 / web\_pages.size();

if (web\_pages[i].num\_of\_clicks == 0)

web\_pages[i].page\_rank = initial[i][1];

}

vector<double> counter(web\_pages.size());

double\*\* matrixH= new double\*[web\_pages.size()];

for (i = 0 🡪 web\_pages.size()) {

matrixH[i] = new double[web\_pages.size()]; }

for (int j = 0 🡪 web\_pages.size())

for (i = 0 🡪 web\_pages.size()) {

if (graph[i][j] == 1)

counter[j]++;

matrixH[i][j] = 0;

}

for (int j = 0 🡪 web\_pages.size())

for (i = 0 🡪 web\_pages.size())

if (graph[i][j] == 1)

matrixH[i][j] = 1.0 / counter[j];

double\*\* PowerMatrix = power\_matrix(matrixH, 3);

matrix\_multiply(PowerMatrix, initial, web\_pages.size(), 1, web\_pages.size());

for (i = 0 🡪 web\_pages.size()) {

for (int j = 0 🡪 web\_pages.size())

web\_pages[i].page\_rank = PowerMatrix[i][1];

}

return PowerMatrix;

}

* 1. Time and Space Complexity O(n3) – O(n2)

1. Page Score Sort
   1. Pseudocode

void web\_graph::sortPageScores()

{

sort(searched\_websites.begin(), searched\_websites.end(), [](web\_page website1, web\_page website2) {return website1.page\_score > website2.page\_score; });

}

* 1. Time and Space Complexity O(nlogn) - O(n)

1. Send Impressions
   1. Pseudocode

void web\_graph::sendImpressions()

{

Open file

int statement[16] = { false };

for (i = 0 🡪 web\_pages.size())

{

for (j = 0 🡪 searched\_websites.size())

if (web\_pages[i].name == searched\_websites[j].name)

{

impressionsUpdate << searched\_websites[j].name << " , " << searched\_websites[j].impressions << " \n";

statement[i] = true;

}

while (i < 16)

{

if (statement[i] == false)

{

impressionsUpdate << web\_pages[i].name << " , " << web\_pages[i].impressions << " \n";

statement[i] = true;

}

i++;

}

* 1. Time and Space Complexity O(n2) – O(n)

1. Send Clicks
   1. Pseudocode

void web\_graph::sendClicks()

{

bool statement[16] = { false };

open file

for (i = 0 🡪 web\_pages.size())

{

for (j = 0 🡪 searched\_websites.size())

if (web\_pages[i].name == searched\_websites[j].name)

{

clicksUpdate << searched\_websites[j].name << " , " << searched\_websites[j].num\_of\_clicks << " \n";

statement[i] = true;

}

}

int i = 0;

while (i < 16)

{

if (statement[i]==false)

{

clicksUpdate << web\_pages[i].name << " , " << web\_pages[i].num\_of\_clicks << " \n";

statement[i] = true;

}

i++;

}

}

* 1. Time and Space Complexity O(n2) – O(n)

Main Data Structures:

A variety of data structures were used, including but not limited to: vectors, arrays, strings, and stacks.

Vectors were used in declaring web pages and web graph because they are more flexible than other data structures, and since the number of websites is easily interchangeable a dynamic data structure (vector) was more approachable especially since these vectors were of type classes. Some examples include vectors of keywords, links (connected edges), web graph, web page, and many more.

Arrays were essentially utilized in setting the graph and its edges. Since the website characteristics are now stored in class web\_pages as an attribute; its quite easy and definite to identify their size. Hence, the usage of an array is more affordable in terms of utilities and efficiency. Also, note that the arrays were only used in basic functional requirements such as matrix multiply, power matrix, and graph setting because the only requirement was to access certain indices of the array, which is very suitable from the array perspective.

The sorting mechanism in the sorting function covered by c++ library uses binary tree to sort, which uses complexity O(nlogn), which is more efficient instead of the usage of an array saving the time complexity of O(n2), and the space complexity of an adjacency list.

Lastly, strings were used in countless situations, but they were primarily used to store the desired keyword, and was consequently used to read from and to files.

TradeOffs:

1. A vector of keywords and another vector of links were used to save time complexity and read the data only once instead of constant repetition; however, the space complexity slightly increased to O(n), which is reasonable.
2. An array of connected edges of type int was used to store data of edges instead of recreating them every time; however, as a compensation, the edges were stored in strings and compared with every iteration to balance between the time and space complexity.
3. Page Score sorting was of complexity O(nlogn), where data are stored in a binary tree saving around more than half the standard complexity of O(n2), without affecting the space complexity.