**Search Engine Using PageRank algorithm**

Kyrollos Zakaria Soliman - 900203309

Department of Computer Science

American University in Cairo

Dr. Reem Haweel

May 4, 2022

# I. indexing and ranking algorithms:

1. **Indexing Algorithm:**

It was implemented using Data Structure: **Map<int index, website website>. Website** is a user-defined data type, used to encapsulate each website attributes such as: impressions, clicks, PR(page rank), website name, and website index…etc.

**Indexing algorithm** was part of the **build\_web\_graph** function as follows:

build\_web\_graph(multimap<int, int>& TransposedwebGraph, vector<vector<int>> &TransposedVector)

{

Open webGraph file

Iterate through each line in the file till you reach eof

During each iteration:

Split the line at the comma

Find index of the website number and convert it to integer, then make it

website index as follows:



}

The advantage of using maps is that it eliminates duplicates, so if a website has more than edge it will not cause to be added twice to the map.

Then, I keep count of outgoing links of each website.

And Make Transposed adjacency list for page rank algorithm as follows

{



}

1. Time Complexity for indexing algorithm:

It iterates through the webgraph.csv file so it works in O(E), where E is the number of edges.

1. **Page Ranking Algorithm:**

**The algorithm and all numeric assumptions was according to:** [**http://www.ams.org/publicoutreach/feature-column/fcarc-pagerank**](http://www.ams.org/publicoutreach/feature-column/fcarc-pagerank)

It utilized **2D vector<int>** Transposed adjacency list of the web graph and the index-website map and two **vectors** to save the previous and current PR. The program can run a maximum of 100 iterations before reaching final PR; however it can terminates before the 100th iteration if the PR stabilized according to this function:



It takes number of websites as a parameter and compare between current and previous values of PR and if all values smaller than maximum error value (0.001), then there is no need to do more calculations.

During each iteration, PR was calculated as follows:



It resets the current PR to 0. Then for every website **j** pointing to the website we are calculating the PR for, we sum the value of the previous PR value of **j** over number of outgoing links form **j**  to the current PR. Then multiply by a dumping factor = 0.85 then add 0.15/ number of websites to the current value. And that to avoid errors if there is a website with no other websites pointing to it.

After terminating the loop, the program assigns each PR value to its corresponding website as follows:



The PR function is triggred from the main through a function called: PR\_spark() works as follows:



It calls the function on the first website and then all events keep going, it is like dominos falling!

1. PageRank Time complexity:

It runs 100 time at maximum and each time it runs through each website and their adjacency list so it works in O(100\*W\*E) = **O(WE)** , where W is the number of websites and E is the number of Edges.

# II. Main Data Structures used in the program:

1. Maps:
2. **map<int, website> websites**

A hash map to map each website to a certain index through the program.

1. **Map<double, website> resultWebsites**

A map to store the retrieved websites, the key value in this map is the score of each website, so it helps printing them in a descending order using reverse iterator.

1. Vectors:
2. **vector<vector<int>> &TransposedVector**

A 2D vector to store the transposed adjacency list for the web graph.

1. **vector<vector<string>> searchKeys(3);**

A 2D vector of type string and size three to store the significant key words of the seach query. Each row of it represent search type:

SearchKeys[0] 🡪 keyWords for AND search.

SearchKeys[1] 🡪 keyWords for Quotation search.

SearchKeys[2] 🡪 keyWords for OR search.(Default)

* Note: All data structures are passed to functions by reference to saved space.

# III. WebGraph and TestCase:

A picture containing diagram

Description automatically generated

This is the web graph I created to test my search engine. Each node represents a web site, for example node 7 represets: “www.test7.com”

To test PageRank function effectiveness. I made a file with all impressions equals 1 and run several searches to make sure of the ranks of each website according to this PageRank simulator:

Diagram

Description automatically generated

# IV. Design Trade-offs:

# put all keywords of a website in a single string to search for the increase the searching complexity from O(n) to O(log n).

# Make two functions called save\_screen() and print\_saved\_screen() to save the search results to avoid repeating search operations again each time the user opens a website and back to the results list.