Report INFORMATION RETRIEVAL AND WEB SEARCH

Alberto Carraro
855255

Matteo Baratella
856097

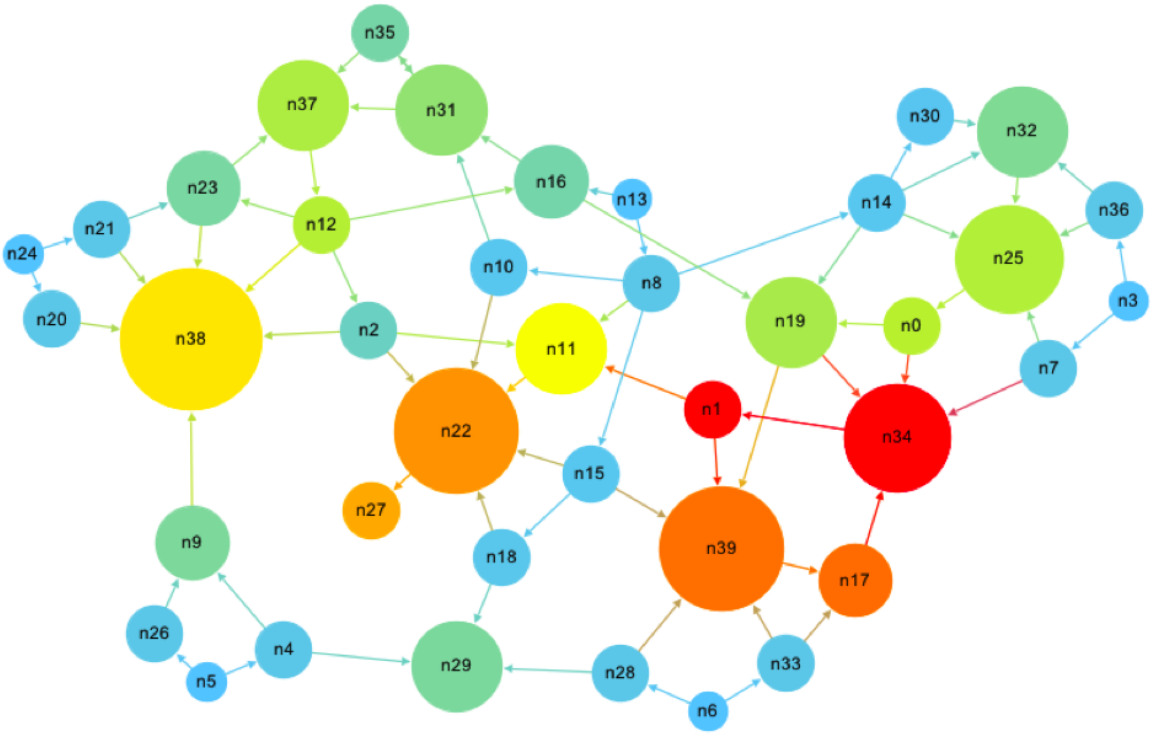
Sept 10, 2021

1. Introduction

In this report we are going to present our project developed in C++ regarding the implementation of three different link analysis algorithms studied during the course.

1.1. PageRank

PageRank is a link analysis algorithm used by Google Search to rank websites. Named after Larry Page, one of the founders of Google, PageRank produces an estimate of how important a certain website is by counting the number and quality of links directed to that page, with the assumption that the more important a website is the more likely it is to receive links from other pages. PageRank is based on the democratic nature of the Web since its vast link structure is used as an indicator of the quality of a page.
The more in-links that a page i receives, the more prestige the page i has. Pages that point to i also have their own prestige scores: a page of a higher prestige pointing to i is more important than a page of a lower prestige pointing to i.



PageRank formula is derived by the equation of the improved model:

$$P = (1-d)\frac{e}{n} + dA^T P$$

so, for each page i, the PageRank value is:

$$P(i) = (1-d) + d\sum_{j=1}^{n} A_{ji}P(j)$$

where

- A_{ji} represents the transition probability that the user in page i will navigate to page j
- d is called the damping factor, which can be set between 0 and 1 (we'll use 0.85)

PageRank is based on the **Random walk** model:
Random walk models a Web surfer randomly surfing the Web as state transition. We used O_i to denote the number of out-links of a node i, each transition probability is $1/O_i$ if we assume the Web surfer will click the hyperlinks in the page i uniformly at random, without using the back button of the browser or typing an URL, we can still have *dangling pages*.
Since a page is typically linked to a limited set of other pages the matrix A, even if very large, is usually sparse: for this reason we can store it using the so-called Compressed Sparse Row representation.
We'll also use the *mmap* library to maps files to memory regions, which are then accessed without the RAM

1.2. HITS

It stands for **Hypertext Induced Topic Search**. Unlike PageRank which is a static ranking algorithm, it is query dependent.
It produces two different rankings: **authority ranking**, like PageRank, and **hub ranking**, considering the page linked in the article.
The most important thing is: a good hub points to many good authorities and a good authority is pointed to by many good hubs.

It goes through four main points:

1. Send the query to a search engine.
2. Collect the t highest ranked pages. This set is called W.
3. It then grows W by including any page pointed to by a page in W and any page that points to a page in W.
4. Return W as S.

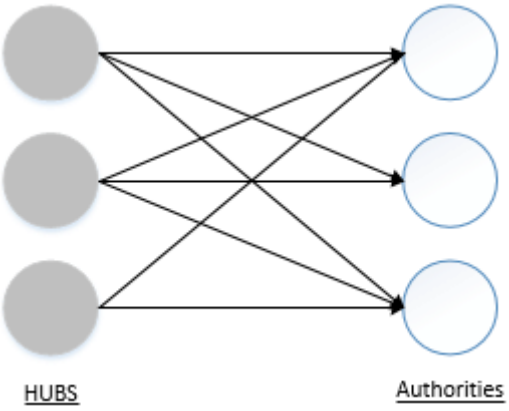
Now, it will assign two kind of scores:

- authority score of page i:

$$a(i) = \sum_{(j,i)\in E} h(j)$$

- hub score of page i:

$$h(i) = \sum_{(i,j)\in E} a(j)$$



Let a denote the column vector with all the **authority scores**

$$a = (a(1),\ a(2),\ ....,\ a(n))^T$$

Let h denote the column vector with all the **hub scores**

$$h = (h(1),\ h(2),\ ....,\ h(n))^T$$

Then:

$$a = L^T h$$
$$h = L\ \ a$$

L is the normal adjacency matrix of a graph (without dividing by the out-degree of the node as for PageRank).

$$L_{ij} = \begin{cases} 1 & \text{if } (i,j) \in E \\ 0 & \text{otherwise} \end{cases}$$

The computation of authority scores and hub scores is the same as the computation of the PageRank scores, using **power iteration**.

If we use a_k and h_k to denote authority and hub vectors at the k^{th} iteration, the iterations for generating the final solutions are:

$$a_k = L^T L\, a_{k-1}$$
$$h_k = L\, L^T\, h_{k-1}$$

starting with:

$$a_0 = h_0 = (1, 1, \ldots, 1)$$

1.3. InDegree

Social network analysis is the study of social entities, called actors, their interactions and relationships that could be represented with a graph.
There are two main aspect of social network analysis to study:

- **Centrality**: Important or prominent actors are those that are linked or involved with other actors extensively.
- **Prestige**: A prestigious actor is one who is the object of extensive ties as a recipient.

InDegree is the number of *edges* coming into a *vertex* in a directed graph.
The final score is normalized, and so we obtain, for each node, a score that is between 0 and 1.

2. Implementation

2.1. Data Structures

In order to implement our algorithms we needed some common utilities and data structures to define.
Once the implementation is completed, we need to track its execution times with some input sets, which are freely available at the Stanford Large Network Dataset Collection.

2.1.1. Compressed Sparse Row

The CSR (Compressed sparse row) is used for traversing matrices in row major order. We implemented a class, containing all methods getter and setter and the main computation of the CSR.

In this class there is a main methods we are focused with:

void CSR::compute()
This is the main function of the class, and it computes the CSR for the page rank.
First of all it creates two files in which will be stored all data retrieved by the main file:

- *column_index_file,* which stores the column indexes of the elements;
- *row_pointer_file*, which stores the position where the n-th row should start;

After that, it starts a cycle in order to compute these pointers from the main file and save them in the above files (with mmap library) in order to have them available later separately.
During the cycle it also stores the values of the non-zero elements of the matrix.

Using this strategy is useful for two reasons:

- as the dimension of the adjacency matrix grows, the sparseness of the matrix also increases: a sparse representation is therefore extremely convenient.
- working with "complete" matrices of large domains is unfeasible in practice: the memory requirements would be too high and the PageRank algorithm would not work.

2.1.2. MMap

Our implementation also uses the mmap library to map the file to a memory region in order to access it via pointers, just as we would access ordinary variables and objects.
In this way, the two arrays of the compressed representation col_index and row_pointer are written on external files :

- *column_index_file,* which stores the column indexes of the elements;
- *row_pointer_file*, which stores the position where the n-th row should start;

The two files are unmapped as soon as we stop using them.
To sum up, thanks to mmap it is possible to manage any matrix regardless of its size, without using physical RAM.

2.1.3 DAAT

In order to compute the top K nodes with the highest score we use the DAAT algorithm.

It uses a Min heap to store all the values computed before, and get the top K nodes with the highest score.

2.2. PageRank

The PageRank of a page is considered to be the steady-state probability that the surfer is visiting a particular page after a large number of click-throughs.
The algorithm we use to implement PageRank is the Power Iteration Method.
The termination condition is defined, as before, by the case in which two consecutive iterations of the algorithm produce two almost identical p-vectors.

At the end of the algorithm we obtain the nal PageRank values.

The computation of the page rank proceed as follows:

- initialize and compute the CSR
- perform the stochastization
- page rank main algorithm

2.2.1. Stochastization

This method allows us to have the main matrix stochastic that is one of the requisites of the page Rank algorithm.
In order to do that, we initialized the out_links at 0.
Then we init out_links from the mmap file, and we store only the values that were different from 0 (avoid dangling nodes).
The final step was the normalization, so divide each csr value for its out_link that is computed by two nested for loops, in which the deeper one computes the column new values and the other one only moves through rows.

2.2.2. Main algorithm

The main method that allow us to computer the PageRank is

void pageRank(const int *row_pointer, const int *col_index)

At this point we've already computed the CSR and done the stochastization.

After the initialization of some counters, and the initialization of a *p-vector* containing the number of nodes with an initial probability (given by 1 over the n° of nodes), the entire main code is contained in a loop.

This loop is our implementation of the Power Iteration method:

- we initialize the vector *p_new* from the values of the CSR;
- we obtain the final matrix after removing dangling nodes and adding teleportation to *p_new*;
- we check the termination condition and we update p;

2.3. HITS

As well as PageRank does, also HITS adopts the Power Iteration Method in order to compute the authority scores and the hub scores.
The termination condition is defined by the case in which two consecutive iterations of the algorithm produce two almost identical p-vectors.
At the end of the algorithm we obtain the HITS values.

The computation of the page rank proceed as follows:

- initialize and compute the CSR.
- choose the type of ranking to compute (authority / hub / both);
- hits main algorithm

2.3.1. Main algorithm

The main method that allow us to computer HITS is

int hits(bool isHubScore);

At this point we've already computed the CSR.
If we have to compute the authority score, we need first to generate the transposed CSR, and this is done at the beginning (on the main() function), if the file does not exist yet.

Then, if the value *isHubScore* is true, the program will compute the Power Iteration method as before with the CSR from the standard graph, otherwise it will use the CSR-transpose.

There are only a few differences in comparison to the previous implementation of the Power Iteration method: here we'll do a normalization of *p_new* after the teletransportation.

2.4. In Degree

The computation of the In Degree proceed as follows:

- initialize and compute the CSR-transposed.
- In Degree main algorithm

The main algorithm inDegree() simply computes the scores using a vector and by dividing each element by the total number of nodes.

3. Results

To analyze the behaviour of our implementation we have mainly used a is a raspberry[1], a Linux (Debian) arm32v7 machine as the operative system.
As for the graphs, we have selected the "web-NotreDame" data set to perform our results.

Notes - into the main() function is possible to change some parameter:

- **topK**: default to 30, limit the topK elements retrieved for each algorithm;
- **doAll(topK, true)**: the second parameter can be set to true, if you want the display of the full results, or can be set to false if you want just jaccard computation and a better performance.

The program will also display the elapsed time for the overall computation.

[1] raspberry is a series of small single-board computers.

3.1 PageRank results

```
COMPUTING CSR START
N° of Nodes: 325729, N° of Edges:: 1497134
COMPUTING CSR START END
COMPUTING PAGERANK START
- Stochastization Start
- Stochastization End
- PageRank Start
N° of iteration to converge: 43
- PageRank End
COMPUTING PAGERANK END
PageRank Top k:
1963: 0.00206908
0: 0.00206636
10336: 0.0018818
212843: 0.00143804
124802: 0.00121743
12129: 0.00102473
191267: 0.00100891
32830: 0.00100536
83606: 0.000904845
1973: 0.000872025
142732: 0.000861411
24823: 0.000821962
143218: 0.000791845
3451: 0.000760993
31331: 0.000690894
149039: 0.000663305
140170: 0.000535881
12838: 0.000516341
81878: 0.000511944
226950: 0.000456993
73859: 0.000402697
292009: 0.000399378
63364: 0.000365837
24944: 0.000365023
88448: 0.000353835
88118: 0.000348526
10335: 0.000336454
10331: 0.000323023
143082: 0.000309396
32833: 0.000306826
```

3.2. HITS Authority results

```
COMPUTING CSR START
N° of Nodes: 325729, N° of Edges:: 1497134
COMPUTING CSR START END
COMPUTING HITS START
- Authority Ranking Start
N° of iteration to converge: 69
- Authority Ranking End
COMPUTING HITS END
HITS Authority Top k:
12129: 0.00653518
199031: 0.00216739
235904: 0.00216739
151241: 0.00216254
193592: 0.00216254
198328: 0.00216254
155590: 0.00216254
199030: 0.000737596
199029: 0.000736429
199028: 0.000736429
151238: 0.000735189
151239: 0.000735189
155587: 0.000735189
155588: 0.000735185
151240: 0.000735126
155589: 0.000735125
236095: 0.000725708
236029: 0.000724474
235963: 0.000724474
193717: 0.000723263
193651: 0.000723263
193783: 0.000723262
198387: 0.000723262
198519: 0.000723262
198453: 0.000723261
260583: 0.000720791
260584: 0.000720791
260585: 0.000720791
260586: 0.000720791
260587: 0.000720791
```

3.3. In Degree Results

```
COMPUTING CSR START
N° of Nodes: 325729, N° of Edges:: 1497134
COMPUTING CSR START END
COPUTING IN-DEGREE START
COMPUTING IN-DEGREE END
InDegree Top k:
12129: 0.0329139
0: 0.0233906
124802: 0.0215701
31331: 0.0132012
140170: 0.0131367
199031: 0.0109815
235904: 0.0109785
151241: 0.0109662
193592: 0.0109631
155590: 0.0109385
198328: 0.0109355
191267: 0.00720538
12838: 0.00665584
81878: 0.00647164
1973: 0.00605411
142732: 0.00580544
143218: 0.00532344
46468: 0.00514231
24823: 0.00439015
3451: 0.00388667
212843: 0.00380378
212812: 0.00380071
73875: 0.00378536
307409: 0.00377615
73874: 0.00377615
307408: 0.00377001
73859: 0.00374544
292009: 0.0037393
199030: 0.0036779
199029: 0.00366255
```

3.4. Jaccard Distance

```
JACCARD RESULTS:
Jaccard PageRank - HITS Authority: 0.0169492
Jaccard PageRank - InDegree: 0.363636
Jaccard HITS Authority - InDegree: 0.176471

Time measured: 7.384 seconds.
```