PAGERANK, HITS AND INDEGREE

Giacomo Zanatta - 859156

IRWS Project - Ca' Foscari University, Venice

`https://github.com/giacomozanatta/pagerank-hits`

1 Introduction

The project [4] will show the computation of **PageRank**[3] and **HITS**[2] ranking algorithm and how these rankings are similiar.

The implementation was written in **C language** with extensibility in mind: effort was made in order to write a library prototype that can be easily expanded adding for example other distance functions or different ranking algorithms.

Since we are dealing with large datasets, the underlying adjancency matrices are represented in a **Compressed Row Storage** (from now, CSR) form in addition to the **mmap** support.

This report is subdivided in 3 main chapter. In this section we will provide a brief introduction about the implemented algorithms, giving an high level overview.
In the next chapter we discuss about the implementation dissecting the library file and we show an example of utilization.
On the last chapter we present some results, comparing the generated ranking using the Jaccard similarity.

1.1 Algorithms

Here we will explain the main algorithms used for the project.

1.1.1 PageRank

PageRank is an algorithm used by Google Search Engine to measure the quality and importance of a page. It assigns a weight on every pages of a graph. The PageRank algorithm outputs a probability distribution used to represent the likelihood that a person randomly clicking on links will arrive at any particular page. Weights are expressed in percentage, for example a weight of 0.1 on page A means that a surfer has 10% probability

of reaching the page A by clicking a random link. An hyperlink from page A to B can be seen as a vote by page A to page B. Vote casted by important pages weight more heavily and help to make other pages more important. A page has a prestige level based on how many inlinks it has. The PageRank score of page i is defined by:

$P(i) = \sum_{(j,i)\in E} \frac{P(j)}{O_j}$

where $P(j)$ is the PageRank score of page j and O_j is the number of out links of j.

Page Rank can be computed using the Power iteration method: we start assigning at each page a score P_0 of 1/(number of nodes). We define A, the transition matrix of the graph, as:

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

Since many pages have no out-links (some rows have all zeros), we can remove it from the computation. Note that the matrix is aperiodic. We can fix this by adding a dumping factor d: with probability d, a random surfer randomly chooses an out-link to follow. With probability 1-d instead, it will jump to a random page (teleportation). An improved PageRank equation that take this into account is:

$P = ((1-d)E/n + dA^T)P$

Where E is a n*n square matrix with every entry setted to 1/n (n is the number of nodes). Using power iteration, the PageRank vector at iteration k+1 is computed as:

$P_{k+1} = (1-d)e/n + dA^T P_k$

Where P_0 is a vector of all 1's. The value of d is 0.85. This is used in the PageRank paper.

1.1.2 HITS

HITS (Hypertext Induced Topic Search) is a query-dependent ranking and it is based on the idea of Authority and Hub pages: an authority is a page with many inlinks. An hub is a page with many outlinks. Hubs and Authority has a mutual reinforcement relationships: an hub is good if it has a lot of outgoing links to good authority. On the other hand, a good authority wants in-links from good hubs. HITS assigns two score at every page: authority score and hub score.

The adjacency matrix of the graph is defined by the matrix L:

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

The authority of a page i, a(i), is: $a(i) = \sum_{(j,i)\in E} h(j)$.

The hub of a page i, h(i), is: $h(i) = \sum_{(i,j)\in E} a(j)$.

In matrix notation, we can write:

$a = L^T h$

$h = La$

where a, h are the column vectors that contains, respectively, the authority and hub scores of each page. Using the Power iteration method and starting with $a_0 = h_0 = (1, 1, 1, 1, ..., 1)$, the final solutions are:

$a_k = L^T L a_{k-1}$

$h_k = L L^T h_{k-1}$

1.1.3 In-Degree

The indegree ranking simply counts how many ingoing link a page has. For computing indegree score for a page, we can take the transpose of the adjacency matrix and count for a page X how many indegree it has, then we divide the sum by the number of total nodes.

1.1.4 Jaccard Similarity

Jaccard similarity between two ranks A and B can be computed by this formula:

$J(A, B) = \frac{|A \cap B|}{|A \cup B|}$

That is in other words the number of different elements in A and B (intersection) divided by the total number of elements in the two set. Since the two sets have the same size, this is calculated as 2*(size of set).

2 Implementation

2.1 Input data

A ranking algorithms works with a set of pages. Every page could have a set of outgoing links. An outgoing links represents a link from a page to another one.

We can model this by using a directed graph: a page is a node of the graph and an directed edge from page x to page y means that page x has an outgoing link to page y.

Network graph is taken from a text file, we assume that a file is structured as a follows:

1. **Must starts with an Header**. Header lines begins with the `#` character. Header contains useful information like the number of nodes and the number of edges of the graph stored in a special line structured like this:

   ```
   # Nodes: X Edges: Y
   ```

   Where X and Y are two integers that represents respectively the number of nodes and the number of edges. File must have only one Header section.

2. **Must contains a number of edges that matches the number specified in the header**. After the section header, file must contains Y lines (where Y is the number of edges). An edge of the graph is defined by a line in the file. A line after the header must be in the next format:

   ```
   A B
   ```

   Where A and B are two integers. A line in the file says that page identified by ID A has an outgoing link to the page identified by id B. For better understandability, we call A FROMNODEID and B TONODEID

3. **Edges entry must be ordered**. The entry of the file must be sorted first by the first integer of the line and then by the second one.

Here we provide a simple example of a valid file.

```
# Test graph
# Nodes: 8 Edges: 14
# FromNodeId ToNodeId
0 3
1 2
1 4
2 0
3 1
4 1
4 2
4 3
4 5
5 2
5 7
6 0
6 2
7 0
```

2.2 Matrix representation

The previous example corresponds to the next adajcency matrix:

$$\begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Storing this matrix in a standard matrix representation requires $8 * 8 = 64$ entries.

Problem arises when we have to work with huge dataset: some network has **millions of nodes** and this requires a lot of space.

For example, if we store entries as a float (4 bytes) a network with 50.000 nodes has 2.500.000.000 entries and we need 4*2.500.000.000 bytes (10 gigabytes) for store them.

Since large matrix are very sparse (we assume that a page contains only a few outgoing links) we can store the matrix in a CSR format.

2.2.1 CSR format

Matrices in CSR format are expressed by three one-dimensional array [1]: one array (val) contains the non-zero values of the matrix. Another array (col_index) is used to store the column indexes of the elements in the val vector. The last vector (row_ptr) stores the locations in the val vector that start a row. In a more specific say:

1. The arrays val and col_index are of length N (number of total non-zero values in matrix), and contain the non-zero values and the column indices of those values respectively.
2. The row_pointer array is of length (m + 1) (m is the number of row of the matrix) and is defined recursively as:
   - row_pointer[0] = 0
   - row_pointer[i] = row_pointer[$i-1$]+number of non-zero elements in the (i-1) th row of the matrix.

For example, the previous defined matrix is represented in CSR format as:

- val = $[1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0]$
- col_index = $[3, 2, 4, 0, 1, 1, 2, 3, 5, 2, 7, 0, 2, 0]$
- row_pointer = $[0, 1, 3, 4, 5, 9, 11, 13, 14]$

On the next subsection we analize how the CSR is created from the dataset and we give an overview of the library.

2.3 Library Structure

The library provides some header files that contains some function and data structures used for ranking computation. The header files are:

1. **dataset.h**: provides function for parsing the file and to generate a Dataset struct.
2. **csr.h**: provides function to generate a csr struct from a dataset and the stochastization method used for page rank.

3. **ranking.h**: provides ranking functions and similarity computations between ranking.
4. **utils.h**: provides utils stand-alone functions.
5. **constants.h**: in this header some constants are defined.

The header files and implementations are discussed in the next subsection.

2.3.1 Dataset

The dataset struct contains this attributes:

1. int** DATA: this represents the set of entries of the dataset. An entry is a single line of the file and consists of an array with length 2: the first one is the FROMNODEID, the second one is the TONODEID. This is not always the case: if we read the transpose dataset, the two elements of the array are swapped.
2. int n_nodes: number of nodes in the dataset.
3. int n_edges: number of edges in the dataset.
4. char* name: the name of the dataset. It is generated from the file name.

The Dataset header provides the next functions:

1. int read_dataset_from_file(char *file_path, DATASET* dataset, int order): reads the dataset from the given file. Order parameter can be used to obtain a transposed dataset: order can be 0 (standard order), or 1 (transposed order). User can use the defined costants for the order: TO_NODE_ID_FIRST and FROM_NODE_ID_FIRST. This function will populate the n_nodes and n_edges from the header of the file and it will copy the dataset entries inside the DATA array, allocating memory.
2. void sort_dataset(DATASET *dataset): this function will sort the dataset using quicksort.
3. void destroy_dataset(DATASET* dataset): it will free the memory used by the dataset.
4. void print_dataset(DATASET dataset): debug function that print out the dataset in standard output.

2.3.2 Csr

The CSR struct contains this attributes:

1. float* val: the val array, of size n_cols.

2. int *col_index: the col_index array, of size n_cols.
3. int *row_ptr: the row_ptr array, of size n_rows + 1.
4. int n_rows: number of rows in the matrix.
5. int n_cols: number of columns in the matrix.

The CSR header provides the next functions:

1. int csr_from_dataset(DATASET dataset, CSR* csr): creates a CSR from a dataset, allocating the memory for the arrays.
2. int make_stochastic(CSR* csr): make the CSR matrix stochastic. This is used for PageRank computation. It will transforms every values in the val array with 1/(number of elements in the row of the entry). For example, the value array defined in section 2 (CSR format) after calling the make_stochastic function it will be transformed into:

   val = $[1.0, 1/2, 1/2, 1.0, 1.0, 1/4, 1/4, 1/4, 1/4, 1/2, 1/2, 1/2, 1/2, 1.0]$ Note that the sum of the entries of every row is equal to 1.
3. int destroy_csr(CSR* csr): frees the memory used by the CSR matrix.

About mmap The mmap feature is implemented using compile vars. It was done this way for not change too much the implementation and to permits a standard usage of the library, without having functions redecleration that will make the library utilization more complex. For enabling the mmap on CSR data structured, user must compile the source with the -DMMAP flag. Please note that mmap will creates the next files inside the folder of the dataset file:

1. {fileName}Scol: for the col_index array.
2. {fileName}Srow: for the row_pointer array.
3. {fileName}Sval: for the val array.
4. {fileName}Tcol: for the col_index array of a transposed csr.
5. {fileName}Trow: for the row_pointer array of a transposed csr.
6. {fileName}Tval: for the val array of a transposed csr.

where {fileName} is the name of the file.

2.3.3 Ranking

The Ranking type is defined as an array of RankEntry. A RankEntry is defined as a struct with two attributes:

1. int ID: the ID of the page
2. float value: the ranking of the page

The ranking header provides the next functions:

1. int pagerank(CSR csr, Ranking* P, int* n_iter): computes the PageRank from the given CSR. It will store the ranking in the P variable, and the number of iterations to converge is returned inside the n_iter variable.
2. int hits(CSR csr, CSR csr_transpose, Ranking* A, Ranking* H, int *n_iter): computes the HITS hub and authority ranking. These ranking are returned in the A and H variable respectively. It needs the CSR and the transposed CSR.
3. int indegree(CSR csr, Ranking* I): computes the Indegree (returned inside I) of the given CSR.
4. void sort_ranking(Ranking R, int n): it will sorts the RankEntries of the Ranking variable by value, decreasing order.
5. double jaccard_score(Ranking A, Ranking B, int k): it will compute the Jaccard Score of two ranking limited to position K.

2.3.4 Constants and Utils

The constants and utils header provides some defined constants intended to use for communicating with the library and some common functions used by the library.

3 Utilisation of the library

The library is shipped out with a simple main that shows how the library is intended to use.

Firstly, an user needs to load the dataset. He must create a variable of type Dataset and pass the pointer of it to the function read_dataset_from_file specifying also the file name and the order: TO_NODE_ID_FIRST order is the standard one. FROM_NODE_ID_FIRST is used for load the dataset transpose.

After initializing the dataset struct, he can create the CSR matrix by calling the csr_from_dataset function passing the dataset and a pointer to a CSR struct, for example the next lines of code:

```
Dataset dataset;
CSR csr;
char* file_name ="data/web-NotreDame.txt";
// Read from file
if (read_dataset_from_file(file_name, &dataset, FROM_NODE_ID_FIRST) == STATUS_ERR) {
  printf("[ERR] Error reading file.\n");
  return STATUS_ERR;
}
if (csr_from_dataset(dataset, &csr) == STATUS_ERR) {
  printf("[ERR] CSR Creation error.\n");
  return STATUS_ERR;
}
```

will load the dataset from file data/web-NotreDame.txt into the dataset variable and it will create the CSR struct from the given dataset.

After that, user can use the CSR for compute a rank. For example, if user wants to compute PageRank he can firstly make the CSR stochastic with the appropriate function and then call the page_rank function.

```
Ranking page_rank;
int n_iter;

if (make_stochastic(&csr) == STATUS_ERR) {
    printf("[ERR] Matrix Stochastization Error.\n");
    free(&csr);
    return STATUS_ERR;
}
if (pagerank(csr, &page_rank, &n_iter) == STATUS_ERR) {
    printf("[ERR] PageRank calculation error.\n");
    free(&csr);
    free(page_rank);
    return STATUS_ERR;
}
printf("No. of iterations: %d\n", n_iter);
sort_ranking(*page_rank, csr.n_rows-1);
// print top 30:
for (i = 0; i < 30; i++) {
    printf("%d) ID: %d RANK: %f\n", i+1, R[i].ID, R[i].value);
}
```

The previous image shows how to use to print out the top k pages: user must sort the ranking and then iterate over the Ranking array. The Jaccard Similarity between two Ranking can be computed by calling:

```
// compute jaccard between page rank and hits auth
int jaccard = jaccard_score(page_rank,hits_authority,30));
printf("Page Rank - Hits authority similarity at k=30: %d", jaccard);
```

3.1 The main file

3.2 Building the demo

The demo program can be builded by launching the make command from the command line. User can open a new terminal window from the project directory and launch the next command:

```
make
```

for building the project. This command reads the MakeFile and executes the default rule.

For using the mmap feature, user must call from the terminal

```
make mmap
```

The two commands mentioned before generates, respectively, two executable files:

1. **ranking:** the demo program with standard feature.
2. **ranking-mmap:** the demo program with mmap feature.

3.3 Executing the demo

User must launch the program passing two parameters as arguments: the first one is the path of the input file and the second one is an integer K used for print out the K-top ranking and to compute similarity, for example:

```
./ranking data/web-NotreDame.txt 30
```

it will launch the default version of program (without mmap) using the dataset data/web-NotreDame.txt and with the top 30 ranking printed out.

4 Some results

Now we analyze the dataset web-NotreDame.
With error threshold at 10e-8 and d=0.85 PageRank converges at iteration 66.
With error threshold at 10e-8 Hits converges at iteration 93.
The top 30 pages returned by page rank are:

```
1) ID: 1963 RANK: 0.002069
2) ID: 0 RANK: 0.002066
3) ID: 10336 RANK: 0.001882
```

```
4) ID: 212843 RANK: 0.001438
5) ID: 124802 RANK: 0.001217
6) ID: 12129 RANK: 0.001025
7) ID: 191267 RANK: 0.001009
8) ID: 32830 RANK: 0.001005
9) ID: 83606 RANK: 0.000905
10) ID: 1973 RANK: 0.000872
11) ID: 142732 RANK: 0.000861
12) ID: 24823 RANK: 0.000822
13) ID: 143218 RANK: 0.000792
14) ID: 3451 RANK: 0.000761
15) ID: 31331 RANK: 0.000691
16) ID: 149039 RANK: 0.000663
17) ID: 140170 RANK: 0.000536
18) ID: 12838 RANK: 0.000516
19) ID: 81878 RANK: 0.000512
20) ID: 226950 RANK: 0.000457
21) ID: 73859 RANK: 0.000403
22) ID: 292009 RANK: 0.000399
23) ID: 63364 RANK: 0.000366
24) ID: 24944 RANK: 0.000365
25) ID: 88448 RANK: 0.000354
26) ID: 88118 RANK: 0.000349
27) ID: 10335 RANK: 0.000336
28) ID: 10331 RANK: 0.000323
29) ID: 143082 RANK: 0.000309
30) ID: 32833 RANK: 0.000307
```

The top 30 pages returned by HITS are:

```
HITS AUTHORITY:
1) ID: 12129 RANK: 0.011848
2) ID: 199031 RANK: 0.003976
3) ID: 235904 RANK: 0.003975
4) ID: 151241 RANK: 0.003940
5) ID: 193592 RANK: 0.003939
6) ID: 155590 RANK: 0.003928
7) ID: 198328 RANK: 0.003927
8) ID: 199030 RANK: 0.001340
9) ID: 199029 RANK: 0.001321
10) ID: 199028 RANK: 0.001321
11) ID: 151240 RANK: 0.001315
12) ID: 151238 RANK: 0.001315
13) ID: 151239 RANK: 0.001315
14) ID: 155589 RANK: 0.001313
15) ID: 155587 RANK: 0.001313
```

```
16) ID: 155588 RANK: 0.001306
17) ID: 236095 RANK: 0.000551
18) ID: 260644 RANK: 0.000550
19) ID: 260645 RANK: 0.000550
20) ID: 260646 RANK: 0.000550
21) ID: 260647 RANK: 0.000550
22) ID: 260648 RANK: 0.000550
23) ID: 260649 RANK: 0.000550
24) ID: 260650 RANK: 0.000550
25) ID: 260651 RANK: 0.000550
26) ID: 260652 RANK: 0.000550
27) ID: 260653 RANK: 0.000550
28) ID: 260654 RANK: 0.000550
29) ID: 260655 RANK: 0.000550
30) ID: 260656 RANK: 0.000550

HITS HUB:
1) ID: 260552 RANK: 0.000304
2) ID: 236095 RANK: 0.000304
3) ID: 260634 RANK: 0.000304
4) ID: 260635 RANK: 0.000304
5) ID: 260636 RANK: 0.000304
6) ID: 260637 RANK: 0.000304
7) ID: 260638 RANK: 0.000304
8) ID: 260639 RANK: 0.000304
9) ID: 260640 RANK: 0.000304
10) ID: 260641 RANK: 0.000304
11) ID: 260642 RANK: 0.000304
12) ID: 260643 RANK: 0.000304
13) ID: 260644 RANK: 0.000304
14) ID: 260588 RANK: 0.000304
15) ID: 260589 RANK: 0.000304
16) ID: 260590 RANK: 0.000304
17) ID: 260591 RANK: 0.000304
18) ID: 260592 RANK: 0.000304
19) ID: 260593 RANK: 0.000304
20) ID: 260594 RANK: 0.000304
21) ID: 260595 RANK: 0.000304
22) ID: 260596 RANK: 0.000304
23) ID: 260597 RANK: 0.000304
24) ID: 260598 RANK: 0.000304
25) ID: 260599 RANK: 0.000304
26) ID: 260645 RANK: 0.000304
27) ID: 260646 RANK: 0.000304
28) ID: 260647 RANK: 0.000304
29) ID: 260600 RANK: 0.000304
```

```
30) ID: 260648 RANK: 0.000304
```

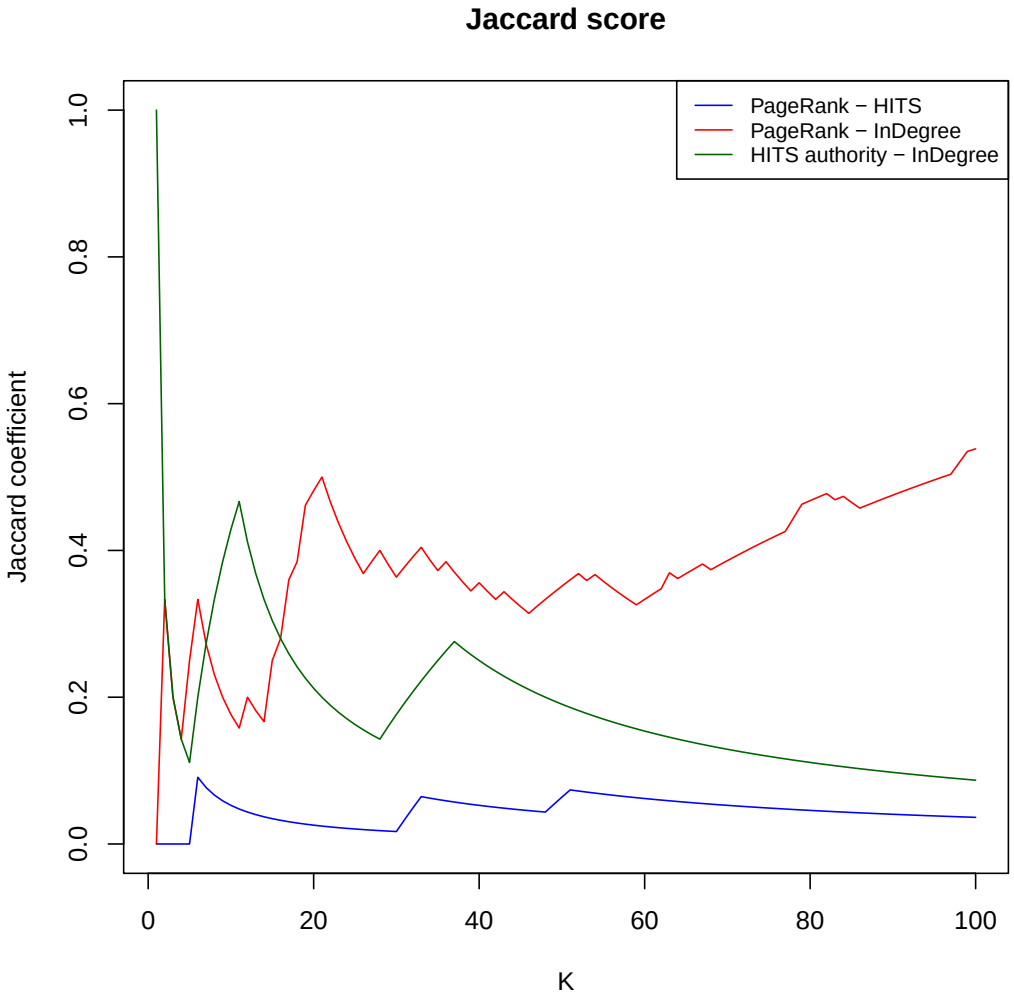
The top 30 pages returned by InDegree are:

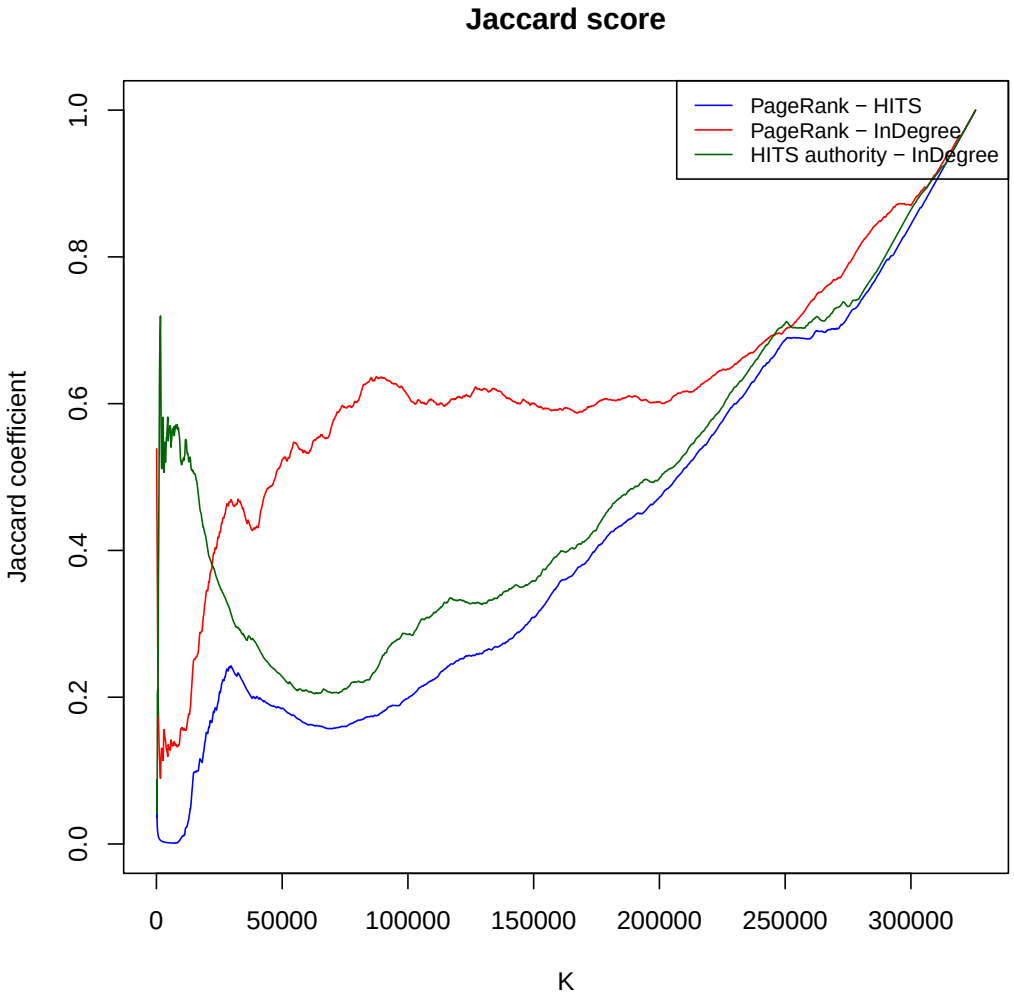
```
1) ID: 12129 RANK: 0.032914
2) ID: 0 RANK: 0.023391
3) ID: 124802 RANK: 0.021570
4) ID: 31331 RANK: 0.013201
5) ID: 140170 RANK: 0.013137
6) ID: 199031 RANK: 0.010982
7) ID: 235904 RANK: 0.010978
8) ID: 151241 RANK: 0.010966
9) ID: 193592 RANK: 0.010963
10) ID: 155590 RANK: 0.010939
11) ID: 198328 RANK: 0.010935
12) ID: 191267 RANK: 0.007205
13) ID: 12838 RANK: 0.006656
14) ID: 81878 RANK: 0.006472
15) ID: 1973 RANK: 0.006054
16) ID: 142732 RANK: 0.005805
17) ID: 143218 RANK: 0.005323
18) ID: 46468 RANK: 0.005142
19) ID: 24823 RANK: 0.004390
20) ID: 3451 RANK: 0.003887
21) ID: 212843 RANK: 0.003804
22) ID: 212812 RANK: 0.003801
23) ID: 73875 RANK: 0.003785
24) ID: 307409 RANK: 0.003776
25) ID: 73874 RANK: 0.003776
26) ID: 307408 RANK: 0.003770
27) ID: 73859 RANK: 0.003745
28) ID: 292009 RANK: 0.003739
29) ID: 199030 RANK: 0.003678
30) ID: 199029 RANK: 0.003663
```

The Jaccard similarity between ranking are:

K	PageRank - HITS	PageRank - InDegree	HITS - InDegree
10	0.052632	0.176471	0.428571
20	0.025641	0.481481	0.212121
30	0.016949	0.363636	0.176471

The next two images will show how the jaccard score change when K change: the first one shows the value of the jaccard score for k from 1 to 100. The second image shows how jaccard score behaves for k from 1 to the number of nodes in the dataset.


Jaccard score
PageRank − HITS
PageRank − InDegree
HITS authority − InDegree
Jaccard coefficient
0.0
0.2
0.4
0.6
0.8
1.0
0
20
40
60
80
100
K



5 Conclusion

We have seen an efficient implementation of PageRank and HITS in C language, using the mmap capabilities.
We have computed the jaccard similarity between the ranking, and we seen that HITS and InDegree are more similiar for lower k. When K becomes bigger, this similarity decreases and the similarity between PageRank and InDegree increase.
HITS and PageRank, instead, are not very similiar.
Talking about performance, from the output of the program we can see that the PageRank computation is faster than HITS (). This because HITS output 2 different

ranking and the computation is a bit more complex.
Here the logging of the program:

```
[INFO] Done reading dataset. Time: 0.371658
[INFO] Reading transposed dataset...
[INFO] Done reading transpose dataset. Time: 0.554761
[INFO] Reading dataset...
[INFO] Generating CSR...
[INFO] Done generating CSR. Time: 0.014086
[INFO] Generating transposed CSR...
[INFO] Done generating transpose CSR. Time: 0.024232
[INFO] Generating transposed CSR...
[INFO] Done generating transpose CSR. Time: 0.016360
[INFO] Stochastization...
[INFO] Done stochastization. Time: 0.006181
[INFO] Computing Pagerank...
[INFO] PAGERANK: Computing...
[INFO] PAGERANK: Done computing. Time: 0.208020
[INFO] PAGERANK: No. of iterations: 66
[INFO] PAGERANK: sorting rank entries...
[INFO] PAGERANK: done sorting rank entries. Time: 0.025548
[INFO] Done computing Pagerank. Total time spent: 0.233604
[INFO] Computing HITS...
[INFO] HITS: Computing...
[INFO] HITS: Done computing. Time: 0.800050
[INFO] HITS: No. of iterations: 93
[INFO] HITS: sorting rank entries...
[INFO] HITS: done sorting rank entries. Time: 0.035991
[INFO] Done computing HITS. Total time spent: 0.836071
[INFO] Computing INDEGREE...
[INFO] INDEGREE: Computing...
[INFO] INDEGREE: Done computing. Time: 0.001482
[INFO] INDEGREE: sorting rank entries...
[INFO] INDEGREE: done sorting rank entries. Time: 0.005886
[INFO] Done computing INDEGREE. Total time spent: 0.007383
```

We notice also that PageRank converges faster than HITS. This because HITS needs that both the errors of hubs and authority are under the threshold.

6 Future works and improvement

A nice to have feature could be the possibility to parameterize some variables, such as:

- dumping factor

- error threshold
- error function

This permits more freedom from user perspective.

References

[1] J. Dongarra, P. Koev, X. Li, J. Demmel, and H. van der Vorst. *10. Common Issues*, pages 315–336. SIAM, 2000.

[2] J. M. Kleinberg. Hubs, authorities, and communities. *ACM Computing Surveys*, 31(4es):5, Dec. 1999.

[3] L. Page, S. Brin, R. Motwani, and T. Winograd. The pagerank citation ranking: Bringing order to the web. In *Proceedings of the 7th International World Wide Web Conference*, pages 161–172, Brisbane, Australia, 1998.

[4] G. Zanatta. Page Rank and HITS in C. `https://github.com/giacomozanatta/pagerank-hits`, 2022.