CSR Traversal vs HASH JOIN Challenge

The aim of this project was the implementation of simple query engine, able to perform the neighbor match in two ways: CSR and HASH table, making a comparative analysis.

Programming language

The project was developed in Python, one of the most diffused programming languages, used especially for machine learning and data analysis. The main library used for this project is Pandas, extremely useful for the handling of big data structure.

The implementation of the operators

Due to the fact to recourse as little as possible to existent libraries for the handling of the graph and the use of the neighbor match, the operators are constructed manually. The data of Pokec are download and stored in a DataFrame using Pandas.

HASH table

The HASH table is constructed with a for cycle, filling each cell with the neighbors of a node. Each node has its own cell and each cell has a specific position (row and column). The formulas used for identifying the cell of a generic node **i** (formulas used both in the construction phase of the table and in the consulting of the table) are the following:

$$Row\ Num = \left[\frac{i}{Row\ Length}\right]_{rounded\ up}$$

$$Col\ \ Num = \ i\ - (Row\ Number - 1) * Row\ Lenght$$

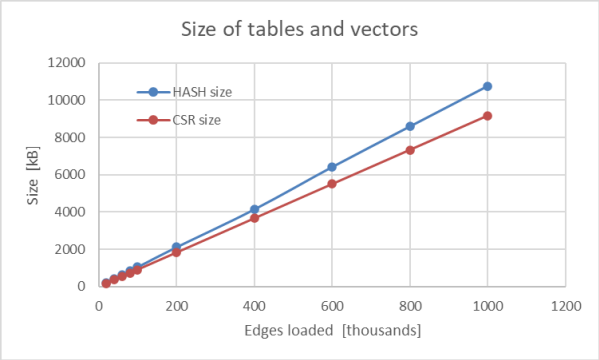
In a social network in which the edges are not oriented (as Facebook) the table of the destinations (in each cell there are the nodes that are linked with node **i** with the path i->j) and the table of the sources are equal (in each cell there are the nodes that are linked with node **i** with the path j->i) while in a social network in which the edges are not oriented (as Twitter) the tables differ. In this specific case it was construct only the HASH table for the destination.

CSR

The CSR method needs only two vectors: ptr and idx vectors. The ptr vector contains the cumulative degree of all the vertices. The idx vector is simply the vector of the destinations corresponding to the second column of the assigned data.

Loading the data

The data are store in a Pandas DataFrame and, due to the big size of the file, it is possible to load a subset of graph. The size of the subset is linked to the number of edges to load. For the various test the numbers of edges loaded has varied between one thousand edges and thousands of edges. In the figure below there is the representation of the size for the elements of the two methods (the size of the Hash table and the size of the two vectors for CSR method) as a function of the edges loaded. The figure reports only the dimension of elements, without considering the memory usage for the construction or for the use of the elements during the queries. It is evident how, with this implementation, the HASH table is always more expensive respect to the two vectors of CSR combined. The slope of the HASH method is around 10,18 kB for thousands of edges while CSR is around 9,16 kB/kEdges.



The second step of the project was the implementation of a basic queries engine. Some conditions were specified for the queries:

- Giving a vertex a, the engine has to return all the pair of nodes (a->b) in which a!=b
- In case of triplets (a->b->c), a!=b!=c
- All the variants up to 11 nodes, with all the nodes different between them

The code asks to the users some specifications before for the query:

- The number of edges to import
- The starting node (the code shows the upper limit for the node selection; choosing 5000 edges the maximum node is 24. With 10.000 edges is and so on)
- The length of "chain":
    - a->b length=1
    - a->b->c length=2
    - a->b->c->d length=3

- and so on up to length=11 that corresponds to 12 nodes linked together.

- The method: HASH, CSR or both

After receiving the specification, the script produced a txt file containing the results required. In the figure below a small example is reported. The figure shows all the triplets of nodes available selecting 14 as starting node and 10.000 edges.

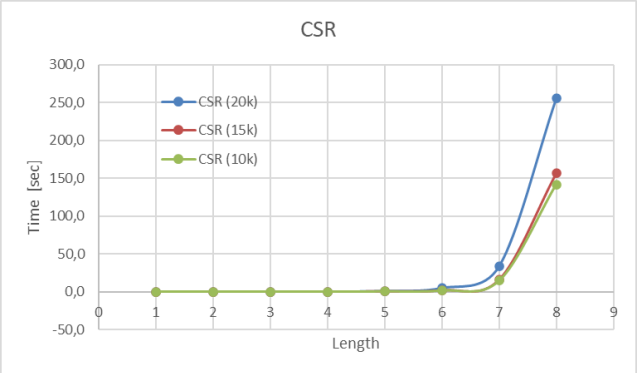
```
The selected starting node is 14
Depth: 2
14      1       13
14      1       11
14      1       6
14      1       3
14      1       4
14      1       5
14      1       15
14      1       7
14      1       8
14      1       12
14      1       9
14      1       10
14      1       16
```

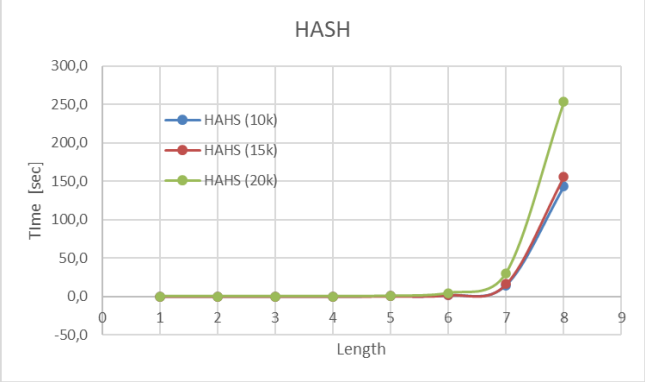
The node is not linked only to node 1 but all the other neighbors are outside of subset of data taken choosing 10 thousand edges. The highest node considering the first 10.000 edges is 189, while the other neighbors of 14 are equal or greater than 350 (except for node 1). The neighbors of nodes outside of the subset of data of course cannot be calculated.

The text files produced are allocated in the fame folder of the script, with the name of the method chosen (hash.txt or csr.txt)

Moreover, an excel file is created or updated each time a query is performed. The file contains some information such as the starting node, the length of the chain, the number of edges loaded, the time for the query (it refers only to the time due to the query excluding the time for the construction of hash table/csr vectors).

The test the engine some tests were performed. Selecting 10, 15, and 20 thousand nodes it was performed the neighbor match choosing node 14 as starting node. The data about the time used by the engine were stored into the excel file and then plotted. The length of the neighbor chain varies from 1 to 8. For 10k and 20k also the length of 9 was calculated. Longer chains or chains with a greater subset of data were not tested due to the enormous sizes of the files generated and the extremely long computational time. On the other columns there are reported the plots of the data about CSR and HASH calculation.





Conclusions

In this short report it was illustrated the function of the script developed. The script can load a subset of data from the Pokec file in a tabular form using the library of Pandas. The data are then elaborated to obtain the HASH table and the CSR vectors. A basic queries engine was implemented for the neighbor match operator, for a length from 1 to 10. The time used for the calculation and the size of the matrix/vectors are calculated. Unfortunately, due to a lack of time, I was not able to go deep into the analysis, extending the analysis also to the memory usage and making heuristic method for the minimization of the time.