Project

First Phase

January 20, 2016

1 Introduction

As we know, data bases have become so prevalent allowing us to keep a tight control on many operations in the daily life: Shipping information, credit card transactions, warehouse control inventory, etc. Many of those data bases are based in the relational algebra, which is the base of the well known SQL databases [4, 3]. However in a new century of Big Data applications, there is a need to break out from the old clichés in order to be able to process great amounts of data. Therefore, in this project we will attack the implementation of a Non-SQL database based in a graph structure [5, 1]. This will allow us to understand better the problems involved in building the data bases of the future.

2 Structure of the Data Base

Given the fact that the new data bases require to have a better representation of the world, it is natural to think about Graphs for the representation of data. For example, if we take in account the twitter information, a graph is most natural structure we can use for representation (Fig. 1)

Therefore, the designer of these databases needs to take in consideration the following attributes about the graph representation:

1. What type of information needs to stored at each node?
2. What information each edge should represent?

Therefore, it is necessary to think about effective representation of the graph data base. Thus, we need to look at the different representations of the database:

1. Adjacency Representation.
2. Triplet Representation.
3. etc.

Once, we have this representation, it is possible to start looking on how to represent information from our abstract graph (Fig. 2), and what kind of hurdles need to be overcome.

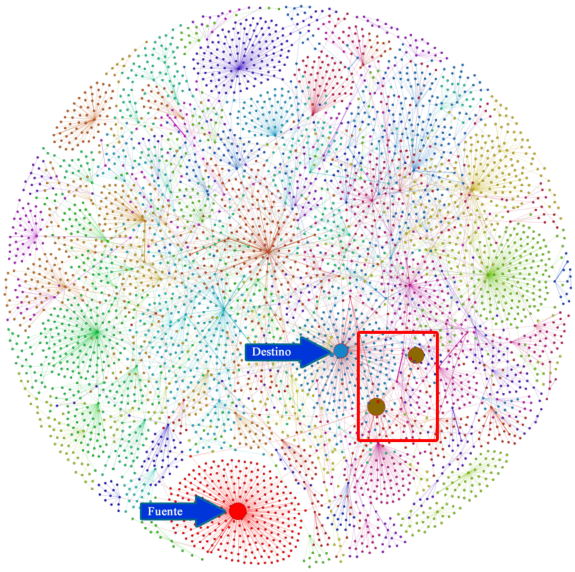

Figure 1: Twitter hashtags as links in a graph

2.1 Node Information

An example of node information can be seen in (Fig. 3). You need to decide where to stop based in what do you want to represent.

2.2 Relationships

Here is an example of a possible relation between two people bases in the distance between addresses in a city (Fig. 4). Clearly, multiple relations could exist between nodes. Thus, you need to take a decision about this specific problem for your database.

3 Sparse Representation of a Graph Database

Given the fact that graphs in the real life tend to be sparse, it is clear that using a matrix representation will be to onerous for the memory. However, while discussing what was better an array or a linked list with Zamora/Zavaleta (Class Fall 2015) about a embedded system, we reached the conclusion than a hybrid representation was better. This hybrid representation is using arrays when data is dense and linked when data is not so dense. Why we decide to use this representation? Because sooner or latter some swapping needs to be done, and it is better to move blocks of data in the limit of the block disk movement.

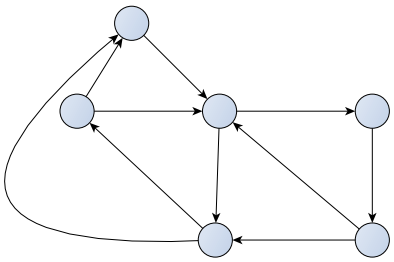

Figure 2: Abstract Graph

Figure 3: Example of a Node

Therefore, we need to store a triplet for each non-empty element: The row, the column and the value structure. Thus, we have something like in can be seen in (Fig. 5) where each array can represent the row non-empty elements. Therefore, there is a new series of field for a new node (Fig. 6). For more on this strategy, please take a look at the chapter on B-Trees at Cormen et al. [2].

4 Operations to be Supported

What type of data structures do we need to support? After all in a relational database it is possible to answer questions about

- Given that the field x has certain attribute 1 and field y has attribute 2 Can you return a table where both attributes are true?

Therefore, we need to support the following operations:

1. Search for similarity of a particular node.
2. Insertion for similarity of a particular node.
3. Deletion for similarity of a particular node.
4. Creation of lists of priorities of nodes based in the similarity to a certain node.

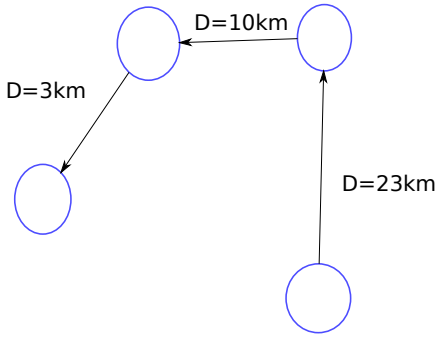

Figure 4: Distance relation between people in the same city

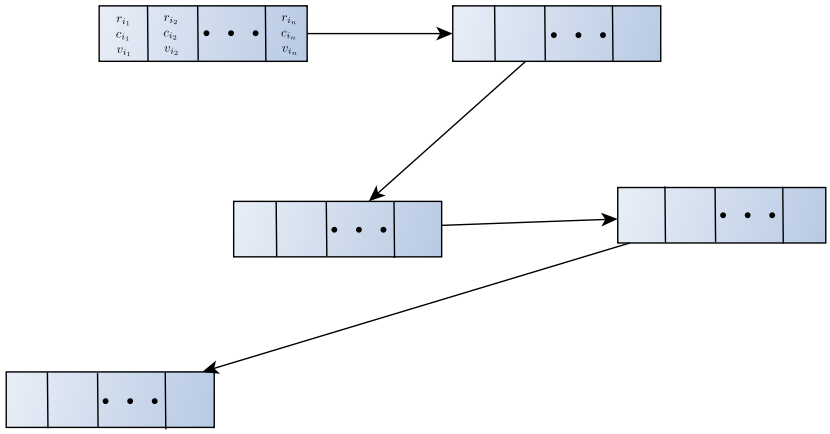

Figure 5: Sparse Hybrid Row-Column Representation of a Graph

Thus, it is necessary to support:

1. Fast access to node information.
2. Is there a relation of similarity between particular nodes?
3. Using this similarity create the "near by" selection operator.

In addition, I want you to support, given particular fields which the nodes posses the combination of the the following operations:

1. Select
2. and
3. or

Which are similar to the SQL based operations.

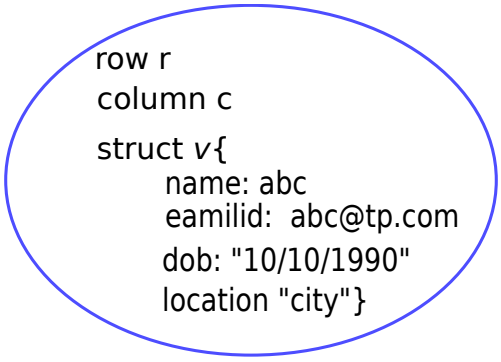

Figure 6: New Sparse Node

5 What do you need to have in your project

5.1 Specific Problem

You need to select some specific problem that can be modeled using a graph representation. This needs to be defined during the following phases:

1. Software Requirements Specification.
2. System Sequence Diagram.

5.2 Sparse Representation of the Graph

Given that the graphs are sparse, it is necessary to have a sparse compact representation of the weighted graph when using one of the three possible representations

1. Adjacency List Representation.
2. Matrix Representation.
3. Triplet Representation

5.3 Basic Operations

5.3.1 Search in the Graph Database

As a first approach for the structure, you can imagine the following node (Fig. 3). Here, I am expecting to see the following algorithms based in the similarity distance that you need to define:

1. Hash Tables for finding the nodes in memory using the similarity.
2. Disjoint Set Representation of the Nodes to group nodes in similar partitions.

3. Shortest path algorithms for handling the idea of near-by with respect to the similarity in the database.

4. Partition of the graph space for fast access when a select operation is issued.

5.3.2 Insertion in a Graph Database

Given the fact that we are using a sparse graph representation, we need to insert nodes in the structure (Fig. 5). Thus, we need to have an strategy for fast insertion into the graph structure, nevertheless if you think well about this you have that an insertion can be seen as a:

1. Search of the similar nodes

2. Insertion of the node in the partition of similar nodes.

Thus, it is possible to use a series the same algorithms than in search, but taking in account the insertion.

5.3.3 Deletion

Here, it is necessary to handle the deletion of similar nodes. This is quite similar to the search in the insertion part.

5.4 Priority Queues

Given that sometimes we like to rank our similarity so we can establish a threshold to binarize the graph space, it is necessary to use a priority queues to maintain the similarity ranks with respect to important nodes. Therefore, you need to implement this in your project.

5.5 Supporting "Select," "And" and "Or"

Here, we will simplify our life and we will avoid the implementation of a compiler by establishing simple operations to be supported. I will talk about this later on during the project.

References

[1] Renzo Angles and Claudio Gutierrez. Survey of graph database models. *ACM Comput. Surv.*, 40(1):1:1–1:39, February 2008.

[2] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Third Edition*. The MIT Press, 3rd edition, 2009.

[3] Hector Garcia-Molina, Jeffrey D. Ullman, and Jennifer Widom. *Database Systems: The Complete Book*. Prentice Hall Press, Upper Saddle River, NJ, USA, 2 edition, 2008.

[4] Jeffrey D. Ullman and Jennifer Widom. *First Course in Database Systems*. Prentice Hall Press, Upper Saddle River, NJ, USA, 3rd edition, 2007.

[5] Chad Vicknair, Michael Macias, Zhendong Zhao, Xiaofei Nan, Yixin Chen, and Dawn Wilkins. A comparison of a graph database and a relational database: A data provenance perspective. In *Proceedings of the 48th Annual Southeast Regional Conference*, ACM SE '10, pages 42:1–42:6, New York, NY, USA, 2010. ACM.