CS CAPSTONE TECHNOLOGY REVIEW

NOVEMBER 22, 2017

ANCESTRY DATA VIEWER

PREPARED FOR

ASHLEY MCGRATH

PREPARED BY

GROUP 22
TEAM ANCESTRY DATA VIEWER(ADVR)
YONGPING LI

Abstract

The objective of our project is to develop a software to generate the family tree with different views from a GEDCOM file. The software also has the functionality of finding the common ancestor between two people. Our team decided to split the project into nine subsections including the parser, database, data structure, 2D visualization, 3D visualization, VR hardware, 2D UI, 3D UI, and display algorithm. I will be reviewing tools for the parser, database, and data structure. The parser is for capturing data; we need to be able to determine the pieces of information that we need in the GEDCOM file and capture it. The database is for storing data before we use it because we don't want to capture the information for the GEDCOM file every time we need to use the data. The data structure is used to make access to data easier.

CONTENTS

1 Parser

1.1 Overview

The GEDCOM file given has a lot of information on the fur and most of the information isn't needed for our project. The format of the GEDCOM file is complicated, but there are patterns to it. We will need to find a pattern in the information and develop an algorithm to capture the data that we need. Although the capturing and generating the data in a certain format can be performed by a tool call parser after we have the algorithm, we also have the option of implementing our own parser.

1.2 Criteria

The only real criteria to consider in this part is performed. We need this process to be done at a fast speed because we everything else happens after we got the data we needed to use. The user will have to wait during the process so making them wait too long is not good.

1.3 Potential choices

The potential choices for parser are Nail, log parser and creating our own parser.

1.4 Choice One: Nail

Nail is a parser that is heavily focused on security. It is very secure compared to other parser and will reduce the effort needed for programmers. Nail can use the protocol grammar to define the data format and the internal object model of the data. Therefore, it can identify the different, unique objects in the data and information related to the object. Nail use other existing parsers to eliminate the notion of semantic actions, so it hides what the data is to be captured from the file with the cost of its performance. There's an authoritative DNS server that can outperform the BIND9 DNS server. This tool is free to share and adapt, but it can't be use for commercial purposes.[1]

1.5 Choice Two: Log parser

Log parser is a powerful tool that can perform universal query access to text-based data. These text-based data files include XML files, CSV files, and even file such as event log for your windows operating system. This tool accepts many input formats, and it's possible to create your own custom input format and use it with the log parser engine. The results of the query can also be custom-formatted in text-based output.[2]

1.6 Choice Three: Creating our own parser

Of course, we have the option of not using existing parser and implement one ourselves. By analyzing the GEDCOM file and finding the pattern of the data. We could use this pattern and build an algorithm that captures the needed data which shouldn't be difficult after knowing the pattern.

1.7 Compare and Contrast

Creating our own parser will cost us time and the performance depends on the language that we are using and the algorithm. We could assume that the existing parser will have better performance than the parser we implement. The best part about creating our own parser is to use the data right away after capturing it, so it doesn't have to output to a file, and then we have to access the file for data. Nail performs slower than log parser due capturing the data in a more secure method, but our software doesn't have to be done over the Internet and security isn't something we should worry about. Log parser is the tool with the greatest performance in these three options and it allows us to customize input and output.

1.8 Decision

I would choose to use Log parser because of its performance. It doesn't really deal with security unlike Nail but security shouldn't be a problem. We shouldn't spend the time to implement our own parser which isn't better for performance. Therefore, as long as log parser is able to capture the data we need from the GEDCOM file, we shouldn't implement our own parser.

2 DATABASE

2.1 Overview

The database is for storing of the data. We need the data to be stored so we can use it whenever we need and don't have to go back to the GEDCOM file every time. Going through and parsing the GEDCOM file will take time and we don't know how long it would take, but we don't want to spend much time on this process.

2.2 Criteria

The criteria are to store in a correct manner to ease the access and use of the data for our software. The data store has to be able to depict the relationships between each person in our data. It should also to able to perform the find common ancestor functionality.

2.3 Potential choices

The potential choices for storing the data are MySQL database, Neo4j database, and not using a database but implement our own storage.

2.4 Choice One: MySQL

MySQL is a relational database management system. The tool is based on the relational model of data which depicts the relationship between the entities in the data. The data is stored in rows and each with a unique id. The columns will be categories of information about this unique data. In our case, each person will be linked to a unique id and the columns indicate the relationship between the people on our data. For example, we could have a column name "Mother of this person", and inside that column is that person's mother's unique id. It can depict many relationships in just a row of data. It can perform query search for specific data in the database.[3][4]

2.5 Choice Two: Neo4j

Neo4j is a graph database. It stores every unique item as a node and the relationship between the nodes. Each person's name will be a node and the relationship between people will be defined in the data. Neo4j takes its input from CSV files to create all the different nodes and relationships. It's able to load up to 10 million nodes so it can deal with large data. This tool can perform queries to search the data that we need which will be useful for the find common ancestor function. It's a lot faster than MySQL due to its queries requiring 10 to 100 times less code. It's the specialized tool for storing relational data and displaying it as graphs.[5]

2.6 Choice Three: Not using a database

Not using a database is also an acceptable option for our project. We could create our own way of storing the data locally and doesn't have to get into a database for accessing the data.

2.7 Compare and Contrast

Both the Neo4j and MySQL can depict the relational data. They can perform query search for specific data in the database. Neo4j will take inputs from CSV files which we can create but it's will take us time to do so. MySQL is easier for inputting the data into the database. MySQL is able to use online but requires users to log in. Not using a database would be easy for storing the data, we just need to format the data and store it.

2.8 Decision

I think it's best for us to create our own way of storing the data. The data can be easily stored, and we don't have to go through other means to store the data in a database. Storing the data in a database will cost them for manipulating the data and making it cope with the format for the database. There isn't a good reason for doing extra things and make the software function slower.

3 Data Structure

3.1 Overview

The data structures are for making access to data more easily. There are many existing search algorithms and functions that could be used for the different data structures.

3.2 Criteria

The criteria are the data structure needs to make access easy. It also needs to be able to illustrate the different relationships between the people, including parents and children, husband and wife, ex-husband and ex-wife, etc.

3.3 Potential choices

The potential choices for data structure are linked-list, tree, and graph.

3.4 Choice One: Linked-List

A linked list is a linear data structure, so items in the linked list are actually sequenced. We could implement this like a graph and make one person connect with multiple people. Linked list contains a data field to store data and a link field. In the link field, we can create multiple links to make it into a graph. The links can be named differently in order to determine the relationships. The links are actually pointers pointing from current address to address the next item. The address of items in this data structure is not sequential, that's why it uses links. The benefit of using this structure is that it is easy to add items to the linked-list. Also, there aren't limitations on the number of items to be in the linked-list, unlike arrays, you have to define its size. [6]

3.5 Choice Two: Tree

Tree is a type of nonlinear data structure. Every item in the tree is called a node. There is one distinguished node being the root of the tree. The other nodes will have a parent node and child nodes. Every node can have a corresponding height and depth determining their place on the tree. Nodes that shared the same parent node are siblings. The binary tree is well known to the programmers. It's a specialized tree that each parent node could only have up to two child nodes. This won't be helpful for our project because people can have more than two children. The general tree is what we needed, but a problem we will have with this data structure is illustrated marital status. We could put the parents on one node, but this makes access to be more difficult.[7]

3.6 Choice Three: Graph

The graph is another type of nonlinear data structure. The graph has two things to it and that is vertices and edges. The vertices represent the objects and edges are the links between the objects. The edges don't only store the link between the two vertices, but it also can store an integer. This integer has a variety of uses, it can be used as the distance between two locations if locations are the vertices of the graph. We can use this integer to determine the relationships between the nodes.[8][9]

3.7 Compare and Contrast

The linked-list uses pointers so we need to change the address of the pointer in order to access the data. It's really a mess if we used linked-list in our project. The tree data structure has problems illustrating the marital status. If we put parents into one node, we would have to make more effort in retrieving the information from these nodes. The graph is able to illustrate all the different relationships between people through the integer that can be stored in its edges.

3.8 Decision

I would choose graph because it's able to depict all the different kinds of relationships. The linked - list is also able to do the job, but it's very hard to get access to the data. The tree can have only one parent node, so we need to put the parents into one node in order to use this data structure. We will have to put more effort into accessing the node/data. Therefore, the data store in the graph is easier to access and it's able to depict all the relationships easier than other data structures.

4 Bibliography

1 https://www.usenix.org/system/files/conference/osdi14/osdi14-paper-bangert.pdf

2 https://technet.microsoft.com/en-us/scriptcenter/dd919274.aspx

3 https://dev.mysql.com/doc/refman/5.5/en/introduction.html

4 https://www.w3schools.com/php/php_mysql_intro.asp

5 https://www.slideshare.net/thobe/an-overview-of-neo4j-internals

6 https://www.cs.cmu.edu/~adamchik/15-121/lectures/Linked%20Lists/linked%20lists.html

7 http://www.cs.cmu.edu/~clo/www/CMU/DataStructures/Lessons/lesson4_1.htm

8 http://www.geeksforgeeks.org/greedy-algorithms-set-9-boruvkas-algorithm/

9 https://www.tutorialspoint.com/data_structures_algorithms/graph_data_structure.htm