CSCI 3901 FINAL PROJECT – EXTERNAL DOCUMENTATION

Instructor: Dr. Michael McAllister
Name: Nadipineni Hemanth Kumar
Banner ID: B00899473
Date: 2021-11-22

Overview:

The overview of this project is to develop a system that connects family tree information with a photo repository and the metadata associated with the photos. The system consists of a family tree database and a media archive. The system will store the information of individuals and their relations in the family tree database. The system will store the information of photos in the media archive. With all this information, we need to give output for the queries given such as a list of ancestors of person X for Z generations.

Files and External Data:

The program will contain three main classes: PersonIdentity class, FileIdentifier class, BiologicalRelation class. The program will also have a Genealogy.java class to manage the family tree, media archive, and reporting.

- In PersonIdentity class, it returns an identifier for the person whenever a person's name has been inserted into the database.
- In FileIdentifier class, it returns an identifier for the media file whenever a file has been inserted into the database.
- In BiologicalRelation class, it will calculate the cousinship and also the level of removal from a relation.
- In the Genealogy class, all the methods mentioned in the project will be there to record and report.

Data Structures and Their relations to each other:

The program will use multiple data structures as per requirement such as lists, sets, maps, Arrays…. etc. The program will use **Map** data structure for storing **attributes of a person**. The program will use **Map** data structure for storing **attributes of the media file**. The **people who appear in the media file** will be

stored as a **Map of Identifier (Key), a List of** persons [Map<String, List<String>>]. The **tree** can be stored as a node with parent (String) and child (ArrayList as a parent can have more than two children). So, the structure looks like (not a binary tree) a Node with one parent data and an Array list of child's data. This way, one parent can have more than 2 or 100 children. But this can also be implemented using HashMap. So, the children's data structure may change in the future**. PersonIdentity class():** The person identity class will store the data related to the person i.e, name as String, attributes as a HashMap, notes as String, and children as a node of array list so that one person can have many children. **fileIdentifier class ():** The file name will be stored as String, and attributes will be stored in a HashMap. **BiologicalRelation class():** This class will be responsible for trees and doing the calculations and stores the descendants in Set and ancestors in a set. Also, notes and references will be stored in a list. And list to store people in a media-related, vice versa i.e., media related to a person

Key Algorithms:

The algorithms that can be used on this are Depth-first search, Breadth-first search where the program starts from the root and goes towards the children. Then find the level at which they are present and by subtraction, this can report the level of separation and from the level of separation and cousinship, the relation of two persons can be found from the formulae mentioned in the breakdown analysis. For example, let's there is a person1 and he has three children person2, person3 and person4, and let's say only person4 has one child person5. Now if we want to find the relation or the generations gap between person1 and person5, we can start from the root(person1) and then go down (incrementing levelCounter) and check if it's person5, if not go down and check if person5 or if it's null go back to the previous level and continue the same with another child. This way we can perform DFS. Instead of going down after person2, we can go breadthwise. person1->person2->person3->person4->person5 (found). **Breadth-first search** will be more ideal in case the number of generations is way high. A depth-first search will be ideal if every parent has way more children. Instead of all this, we can directly store the

levels every time a person has been entered, this way it would be easy while fetching the data back. But the issue is whenever there is an addition of a person in between the tree or the top of the tree, all the levels of children below must be changed.

Database Design:

The database contains three tables (before normalization) to record data.

Table one: To record family tree database –

Identifier	Name	DOB	Location of Birth	Date of Death	Location of Death	Gender	Occupation	References	Notes
INT	CHAR	VARCHAR	CHAR	VARCHAR	CHAR	CHAR	CHAR	VARCHAR	VARCHAR

Table two: To record the media archive –

Identifier	File Name	Date	Location	Tags	Individuals in the picture
INT	CHAR	VARCHAR	CHAR	VARCHAR	VARCHAR

Table three: To record the relations

Parent/child relation	Partnering relation	Partnering dissolution
VARCHAR	VARCHAR	VARCHAR

The database can store the data as per the data types mentioned above but it may change according to the need.

Assumption:

The assumptions mentioned in the PDF were considered.

Limitations:

The program is limited only to biological family relations.

References:

The project statement PDF.

Test Cases:

Input validation (Tests on bad input data)

addPerson():

- Null value passed as name
- Empty string passed as name

recordAttributes():

- Map with null attributes passed
- Map with empty attributes passed

recordRefernece():

- Null value passed as reference
- Empty string passed as reference

recordNote():

recordPartnering():

recordDissolution():

addMediaFile():

- Null passed as fileLocation
- Empty string passed as fileLocation

recordMediaAttributes():

- Map with null attributes passed

- Map with empty attribute strings passed

peopleInMedia():

tagMedia():

- Null passed as tag
- Empty string passed as tag

findPeople():

- Null passed as name
- Empty string passed as name

findMediaFile():

- Null passed as name
- Empty string passed as name

findName():

findMediaFile():

findRelation():

descendents():

- Value of generations is negative

ancestores():

- Value of generations is negative

notesAndReferences():

findMediaByTag():

- Null passed as tag
- Empty string passed as tag
- Null passed as startDate
- Empty string passed as startDate
- Null passed as endDate
- Empty string passed as endDate

findMediaByLocation():

- Null passed as tag
- Empty string passed as tag
- Null passed as startDate
- Empty string passed as startDate
- Null passed as endDate
- Empty string passed as endDate

findIndividualsMedia():

- Null value passed as startDate

- Empty string passed as startDate
- Null value passed as endDate
- Empty string passed as endDate

findBiologicalFamilyMedia():

Boundary Cases (Tests at the edge of inputs)

addPerson():
- 1 character name passed

recordAttributes():
- 1 character attributes passed
- No attributes defined

recordReference():
- 1 character reference passed
- No reference defined

recordNote():
- 1 character note passed
- No note defined

recordChild():
- No child defined
- Only 1 child defined

recordPartnering():
- No partner defined
- 1 partner defined

recordDissolution():
- No dissolution defined
- 1 partner defined

addMediaFile():
- 1 character filename passed

recordMediaAttributes():
- 1 character attributes passed
- No attributes defined

peopleInMedia():
- 0 persons in media
- 1 person in media

tagMedia():
- 0 tags defined
- 1 tag defined
-

findPerson():
- No persons defined
- 1 person defined

findMediaFile():
- No media file defined
- 1 media file defined

findName():
- No persons defined
- 1 person defined

findMediaFile():
- No media file defined
- 1 media file defined

findRelation():
- No persons defined
- 2 persons defined

descendents():
- 0 descendents defined
- 1 descendent defined

ancestores():
- 0 ancestores defined
- 1 anscestor defined

notesAndReferences():
- 0 notes and 0 references defined
- 1 note and 1 reference defined

findMediaByTag():
- 0 tags in the given range
- 1 tag in the given range

findMediaByLocation():
- 0 media in the given location
- 1 media in the given location

findIndividualsMedia():
- 0 individuals in the given range
- 1 individual in the given range

findBiologicalFamilyMedia():
- 0 children for the given person
- 1 child for the given person with 0 media defined
- 1 child for the given person with 1 media defined

Control Flow Cases (Tests of the core operations)

addPerson():

- Duplicate name passed
- Add a person when there are no persons defined

recordAttributes():

- Duplicate attributes passed
- All attributes have only one line in them
- Many attributes recorded

recordReference():

- Duplicate attributes passed
- All references have only one line in them
- Many references present

recordNote():

- Duplicate note passed
- All notes have only one line in them
- Many notes are present

recordChild():

- Duplicate child passed
- Many children already recorded

recordPartnering():

- Duplicate partnering relation passed

recordDissolution():

- No relation to dissolute
- Duplicate dissolution passed

addMediaFile():

- Duplicate media file

  Many media files are present

recordMediaAttributes():

- Duplicate media attributes
- 1 line attributes
- Many lines attributes

peopleInMedia():

- Duplicate people in the media
- Many people in the media

tagMedia():

- 1 line tags in the media
- Duplicate tags in the media
- Many tags are added

findPerson():

- There are many persons defined

findMediaFile():
- There are many media files defined

findName():
- There are many persons defined

findMediaFile():
- There are many media files defined

findRelation():
- There is a relation between two partners
- There is no relation between two partners

descendents():
- There are many descendants
- There are no descendants

ancestores():
- There are many ancestors defined

notesAndReferences():
- There are many notes and references defined

findMediaByTag():
- There are many tags defined

findMediaByLocation():
- There are many media in this location

findIndividualsMedia():
- There are many individuals in this media

findBiologicalFamilyMedia():
- There are many media linked to the immediate children

Data Flow Cases (Tests around the order in which things are done)

recordPartnering():
- Record partnering after dissolution between the same persons

recordDissolution():
- Record dissolution before partnering them

findRelation():
- Find relation of two persons before defining a relation between them