Programming project: Social Networks

Domantas Meidus

January 16, 2018

Contents

Abstract

Social Network is one of the biggest inventions in the internet which have huge influence in the people lives, habits and behavior in the cybernetics field. Social Network is a useful not only for users, but for programmers too, because in order to make Social Network working - a lot of computer science data structures and algorithms have to be used. For implementing this Social Network these resources was used: Reading data from the file and processing it; storing data in the data structures: Lists and Sets; storing all data to the Undirected Graph and implementing traversing Breath fist search algorithm and searching for the shortest path algorithm; User friendly console Menu, with different options of operations on Social Network. All these functions was implemented using Java programming language.

1 Introduction

The software application to be developed will have to manage a social network. This social network is formed by people that may be linked among each other if there is a friendship relationship among them. For each person the following items will be recorded:

- identifier (unique)
- name
- surname(s)
- birthdate
- birthplace
- home
- studiedat (he/she could have studied at many places)
- workedat (he/she could have worked at many places)
- movies
- groupcode

New people can be added to the network or deleted from it, besides other actions that may be of interest for the project.

2 Final version of the Project

Since this project was made by one person - Domantas Meidus, this project has one version. During this project implementation, no drastic changes haven't been made so there was no reason to split project into different parts. Project developing logs by performed tasks:

1. 2017.09.11 - 2017.10.25 : During this duration tasks 1,2,3,4,5,6,7 was implemented.
2. 2017.10.25 - 2017.11.22 : During this duration tasks 8,9,10 was implemented.
3. 2017.11.22 - 2018.01.05 : Task 11 was implemented.

2.1 Classes design

Social network program is based on three class: Main, Menu and Operations classes.
Main: The main Social Network class. This class have one function - invokes Menu class.
Menu: Collecting user keystrokes and displaying project Menu text. Menu class invokes Operations.
Operations: All Social Network project functions. All Social Network used classes are displayed in the project class diagram(Figure 1).
Graph: Class for creating Graph representation for all people in the social network.
Bag: Class for storing each person friends based on the relationship in friends files.
BreadthFirstPaths: Class for searching shortest friendship paths among all people in the social network.

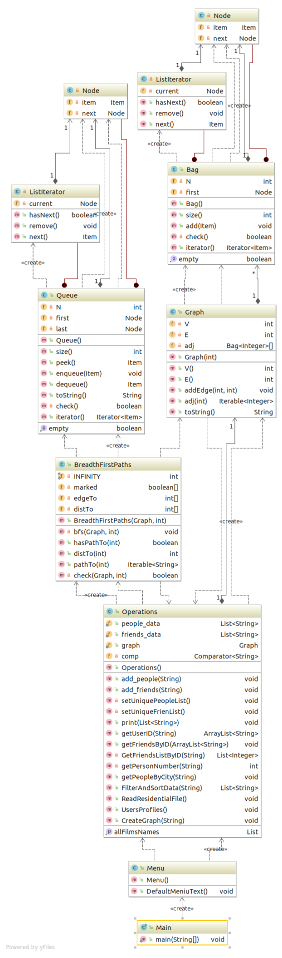

Figure 1: Project Classes diagram.

2.2 Description of the data structures used in the project

The main data structures used in the Social Network project are type of Lists: **people_data** and **friends_data**.

- **people_data**.

  Usage: List is used for storing all the people in the social network.

  Data type: String.

  Access modifiers: public static

  Class: Operations

  Implementation: **public static List<String> people_data = new ArrayList<String>();**

- **friends_data**.

  Usage: List is used for storing all the friends clique in the social network.

  Data type: String.

  Access modifiers: public static

  Class: Operations

  Implementation: **public static List<String> friends_data = new ArrayList<String>();**

Some Operations class functions have implemented lists data structures in order to store results or display information.

Additionally Sets data structure is used in this project for excluding repetitive data by converting Lists to Sets(sets have special feature to store an unique data) and after all data is unique convert again to the lists. For sets implementation in Java code was used HashSet data structure implementation. However, HashSet data structure in Java is not allowed iteration, so in the end sets converted to the List data structure.

2.3 Design and implementation of the methods

- **Menu class**:

  public Menu(): Menu class constructor which invokes Operations class in order to access Social Network functions. Constructor analyze user inputs in order to achieve Social Network functions. The diagram of all available functions which are available for user are displayed in the figure 2. Available operations according user input value:

  input = 0: Exit the Social Network program.

  input = 1: Take the data from a person and add him/her to the network. This function does not read any data from Console. This function receives the data as parameters.

  input = 2: Upload people's data (people.txt) from a file in our directory and add them to the social network. Use the function developed in the previous point.

  input = 3: Print out a listing to a text file of the people on the network.

  input = 4: The task is to implement an operation that relates them in the network, that is, the relationship has to be stored in the network.

  input = 5: On the social network: given a person surname, retrieve his/her friends.

  input = 6: On the social network: given a city, retrieve all people who were born there.

  input = 7: On the social network: retrieve the people who were born between dates D1 and D2, sorted by birthplace, surname, name. The order relationship will be implemented according to the lexicographic order (dictionary's order) of the strings used for the attributes.

  input = 8: Given a set of identifiers in a file named residential.txt, recover the values of the attributes name, surname, birthplace and studiedat of the people on the network whose birthplace matches the hometown of the people who are described in residential.txt. People whose birthplace/hometown is unknown do not affect the result of this operation. For example, if the file residential.txt contains the identifiers Mike222 and Mary123, your task is to retrieve the hometown of Mike222 and Mary123 people, and find all people who were born in those towns.

input = **9**: Two users have the same profile if they match the same collection of favorite movies. Task is to split the users into classes with the same profile and to build a list of those classes.

input = **10**: Six degrees of separation is the theory that everyone on Earth is six or fewer steps away, by way of introduction, from any other person in the world, so that a chain of "a friend of a friend" statements can be made to connect any two people in a maximum of six steps (or five intermediaries). On the social network: given two people of the network, your task is to retrieve the shortest chain that relates them.

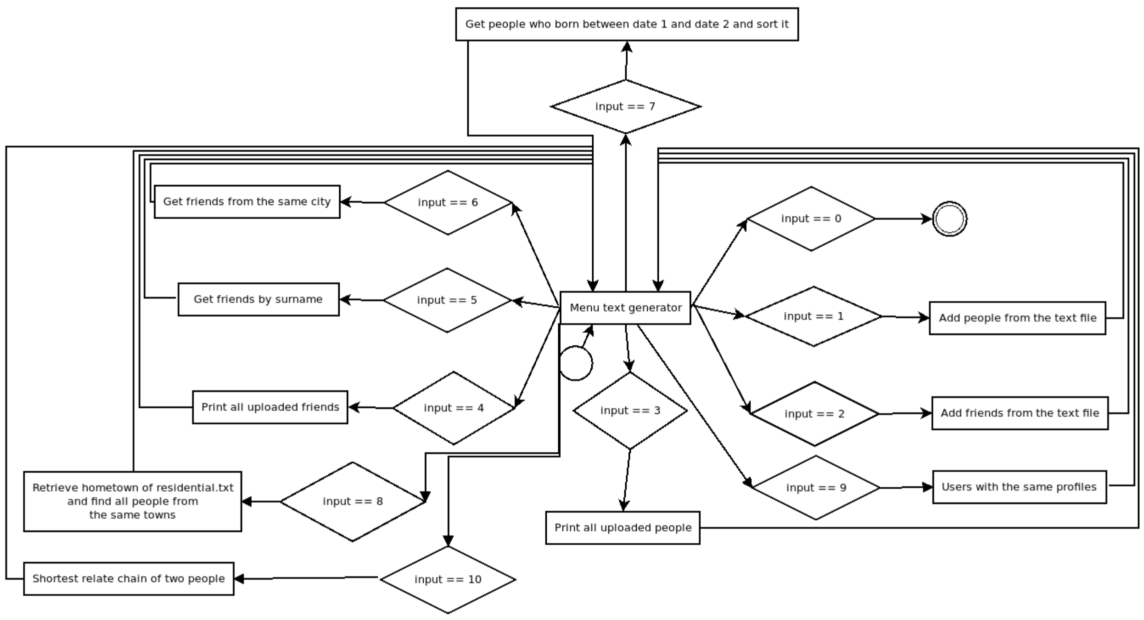
Get people who born between date 1 and date 2 and sort it
input == 7
Get friends from the same city
input == 6
input == 0
Get friends by surname
input == 5
Menu text generator
input == 1
Add people from the text file
Print all uploaded friends
input == 4
input == 3
input == 2
Add friends from the text file
Retrieve hometown of residential.txt and find all people from the same towns
input == 8
input == 9
Users with the same profiles
Print all uploaded people
Shortest relate chain of two people
input == 10

Figure 2: Menu operations scheme.

public void DefaultMeniuText(): This is a void methods which only display text of menu for the user. Menu text in the program can be seen in the figure 3.

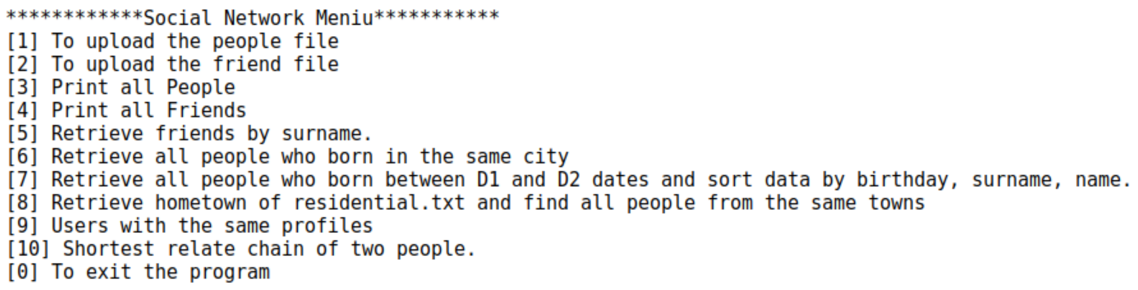
```
************Social Network Meniu***********
[1] To upload the people file
[2] To upload the friend file
[3] Print all People
[4] Print all Friends
[5] Retrieve friends by surname.
[6] Retrieve all people who born in the same city
[7] Retrieve all people who born between D1 and D2 dates and sort data by birthday, surname, name.
[8] Retrieve hometown of residential.txt and find all people from the same towns
[9] Users with the same profiles
[10] Shortest relate chain of two people.
[0] To exit the program
```

Figure 3: Displayed default Menu text in the Social Network program.

- **Operations class**:

public void addPeople(String path): Reading the data of people file using scanner and add the data to the list(friends_data) data structure. It has one argument: **String path** which is the path of the people text file in the local computer file system.

public void addFriends(String path): Reading the data of friends file using scanner and add the data to the list(friends_data) data structure. It has one argument: **String path** which is the path of the friends text file in the local computer file system.

private void setUniquePeopleList(): Creates an unique data set by converting people_data list to the HashSet and then converting back to the list. It helps to exclude repetitive data in the list, for example the same friendship relations among people in multiple times which may have an impact in other functions results.

private void setUniqueFriendsList(): Creates an unique data set by converting friends_data list to the HashSet and then converting back to the list. It helps to exclude repetitive data in the list, for example multiple people with the same person ID and other information which may have an impact in other functions results.

public void print(List<String> data): Prints the data in the terminal of the List. It has one argument: **List<String> data** - is a list of stored data. In program this method is used for displaying elements of **people_data**, **friends_data** or other lists which could be created during in operations implementation.

public ArrayList<String> getUserID(String surname): Method for getting list of UserID attributes values from people_data list. This method iterate between every element in people_data list and search for a surname which is passed from program User. It has one argument: **String surname** - which is an program user written input text value. This method can return multiple UserID because in social network multiple people can have the same surname.

public void getFriendsByID(ArrayList<String> userID): Method for displaying all friends of person which UserID attribute matches with given userID. It has one argument: **ArrayList<String> userID** - a list of all person unique ID who have certain Surname. At first in the list is stored all userID friends information: friend_id and friend_name. For displaying results, the output is printed fallowed this rule: **UserID: FriendID FriendSurname**. For example if searching surname would be "Garcia", the output result could be like this:

- Mike673, Marta059, Jon232, Maria001 <– All userID's which surname is equal to "Garcia"
- Marta059: Laura001 Gran <– Marta059 have friend Laura001. This ID surname is Gran.
- Mike673: Fran451 Zumalakarregui <– Mike673 have friend Fran451. This ID surname is Zumalakarregui.
- Jon232: Bizenta223 Juaristi <– Jon232 have friend Bizenta223. This ID surname is Juaristi.
- Jon232: HER20 SMITHWICK <– Jon232 have friend HER20. This ID surname is SMITHWICK. ...
  ...
  ...

private List<Integer> GetFriendsListByID(String user_id): Returns a list of a indexes of all userID friends in the people_data list. For getting an index value, getPersonNumber method is called in this method. This method is used for Social network Graph creation.

private int getPersonNumber(String user_id): Returns an index of the userID in the people_data list. This method is executed by GetFriendsListByID. This method is used for Social Network Graph creation.

public List<String> getPeopleByCity(String city): Method for displaying all people in the social network which birthday place matches with city text value. It has one argument: **String city** - which is the city of people birthday place. It returns a list of all person which birthday place text value matches with city text value.

public List<String> FilterAndSortData(String date): Method for creating new list for storing filtered and sorted people_data list data. It has one argument: **String date** - is written date range for filtering in the String data type written as an input from program User. Format of this method date should be like this: **date = "dd-mm-yyyy dd-mm-yyyy"**; for example: **date = "12-09-1982 15-04-1992"**. At first date should be less than the second date. Sorting operation is performed after data has been filtered.

Filtered data is sorted in ascending way according 3 people data attributes:

1. **Person Birthplace**
2. **Person surname**
3. **Person name**

ReadResidentialFile(): Reads **"residential.txt"** file and display all people who are born in the same place as written userID's in the "residential.txt" file. Residential.txt file should contains existing people ID from the current social network. The result is generated by displaying all people who birthplace matches with the userID hometown in the "residential.txt" file (Task 8 description). The output is displayed in these rules:

– Each UserID information in the "residential.txt" file hometown place.

– **PersonName PersonSurname PersonBirthplace PersonStudyAt** attributes is displayed those people who birthplace is matched with hometowns of UserID persons list.

public void UsersProfiles(): Displays profile and persons ID's who have the same films profile. In order to be film profile displayed, minimum two people have to have the same film profile. It's worth to mention, that films can be stored in the different ordering, for example: films profiles would be the same, if one person has "Avatar;Casablanca" films representation and other person has "Casablanca;Avatar" representation - these are the same film profiles just in different ordering. To prevent ordering problems - all films profiles are sorted in the *lexicographic order*.
For getting unique films profile list, getAllFilmsNames() method is called.

public List getAllFilmsNames(): Method for getting an unique film profiles with no repetition of the same profiles. To prevent ordering issues, all film profiles are sorted in *lexicographic order* and added to the Java set data structure HashSet. After the set with all unique data is created, this set is converted to the List data structure and returned.

public void CreateGraph(String user_ids): Undirected Graph representation of social network friends relationships in friends_data list. Each person ID is a vertex and all his friends are edges. The Graph is designed to store persons indexes in the people_data list.

- **Graph**: A class for Graph representation of social network relationship among people.

  public int V(): Return the number of vertices in the graph.

  public int E(): Return the number of edges in the graph.

  public int addEdge(): Add the undirected edge to graph.

  Subclasses for Graph implementation is **Queue and Bag**. These classes are needed for implementing Adjacency Lists.

- **BreadthFirstPaths**: Traversing Undirected graph of Social Network people using Breath depth search algorithm. BreadthFirstPaths class functions:

  private void bfs(Graph G, int s): Implementing Breadth fist search algorithm into Social Network Graph. Arguments: **Graph G** is a created graph of social network with Adjacency list implementation - each person is a vertex, and each person friend is an edge to the vertex; **int s** is a person index in the social network Graph which program search shortest path.

  public boolean hasPathTo(int v): searching if it is a path between s (or sources) and v.

  public int distTo(int v): length of shortest path between s (or sources) and v.

  public Iterable<String> pathTo(int v): Search for the shortest path between two persons in the Social Network undirected graph.

3 Conclusions

Current Social Network is not really big, so it's not comparable with now existing social networks, for example "Facebook" or "Twittter". With more data on this Social Network could lead to slower functioning, because implemented code is not optimized to the maximum time efficiency. For one of the solutions for improving performance it could be other Graphs algorithm implementation for storing and searching data in the Social Network, for instance Dijkstra's or A* search algorithms. However, the project goal was not to make perfect Social Network, the main idea was to implement and manipulate data structures and use Graph implementation with additional searching algorithms.

References

References

[1] John Lewis Joseph Chase *Java Software Structures (3rd edition).*

[2] Sedgewick and Wayne *Algorithms (4th edition).*

[3] Mark Allen Weiss *Data Structures and Algorithm Analysis in Java, 3rd Edition.*

[4] Mark Allen Weiss *Data Structures and Problem Solving Using Java, 4th Edition.*