

Forum:[Hack Forums](#)

Thread: [Social Network Data Structures Assignment \[\\$40 AUD PP\] if completed.](#)

Just Ice - May 16, 2018, 5:04 am

Hey there,

UPDATE: Willing to pay \$40 for it to be completed, pm me if interested.

I'm stuck with my data structures assignment where we have to read two text files that contain names and an index of connections we need to represent in a graph, the part I'm struggling with is extending the graph class we are provided with my SocialNetwork class and how I should create the SocialNetwork object with the graph class.

I have attached the pastebin links for the first 2 classes, I currently have it so that I can read the text files and the information is separated in two linkedlists.

Provided Graph Class <https://pastebin.com/GVuJb5zP>

My Driver Class <https://pastebin.com/DVvagkyX>

and my SocialNetwork class is below as I am unsure of how to proceed with it.

Code package com.socialnetwork;

```
public class SocialNetwork extends Graph{
```

```
    public SocialNetwork(int n) {  
        super(n);  
    }
```

```
}
```

Any help will be greatly appreciated!

GenericSkid - May 17, 2018, 1:14 am

Are there any specifications you need to follow? Where are the docs for the assignment aside from a few scattered comments?

Just Ice - May 17, 2018, 2:20 am

(May 17, 2018 01:14 AM)GenericSkid Wrote: [\[CITE\]](#) Are there any specifications you need to follow? Where are the docs for the assignment aside from a few scattered comments?

Yeah sure thing, PM me if you like <https://pastebin.com/qJrNMy68>

GenericSkid - May 17, 2018, 2:31 am

(May 17, 2018 02:20 AM)Just Ice Wrote: [\[CITE\]](#) Yeah sure thing, PM me if you like

<https://pastebin.com/qJrNMy68>

lol I already broke away from a few of the specified design choices. Nothing super that you won't be able to reverse yourself.

Anyways some complaints:

How are you to delete a member from the graph if it's a 2D array without remaking the network every time a person is removed? I mean really, the best you can do otherwise is removing edges.

They expect a name -> vertex but only store the values as vertex -> name. Great, now you gotta loop over everyone or keep track of the former yourself.

The way they load the text file requires it being on the classpath ??? Why not just handle it as an independent file?

Why do they require the file formats specifying the length of the file... when you can literally just compute the length yourself (list.size()), you literally have a list of the lines anyways)

Just Ice - May 17, 2018, 2:47 am

(May 17, 2018 02:31 AM)GenericSkid Wrote: [\[CITE\]](#) lol I already broke away from a few of the specified design choices. Nothing super that you won't be able to reverse yourself.

Anyways some complaints:

How are you to delete a member from the graph if it's a 2D array without remaking the network every time a person is removed? I mean really, the best you can do otherwise is removing edges.

They expect a name -> vertex but only store the values as vertex -> name. Great, now you gotta loop over everyone or keep track of the former yourself.

The way they load the text file requires it being on the classpath ??? Why not just handle it as an independent file?

Why do they require the file formats specifying the length of the file... when you can literally just compute the length yourself (list.size()), you literally have a list of the lines anyways)

Haha ok all good

For removing a user the only thing you can do is remove the edges, and from what I could gather you have to add a vertex twice like (x,y) and (y,x), we we're told to put the input files in the classpath and for the last point what file length do you mean?

I created the Driver class and I apologize for the mess that it is lol

Thanks

GenericSkid - May 17, 2018, 2:54 am

(May 17, 2018 02:47 AM)Just Ice Wrote: [\[CITE\]](#) Haha ok all good

For removing a user the only thing you can do is remove the edges, and from what I could gather you have to add a vertex twice like (x,y) and (y,x), we we're told to put the input files in the classpath and for the last point what file length do you mean?

I created the Driver class and I apologize for the mess that it is lol

Thanks

PHP Code

```
<?php
package com.socialnetwork;
```

```
import java.io.IOException;
import java.util.LinkedList;
import java.util.Scanner;
```

```
public class Driver {
    /**
     * Flag for showing the massive blurb assistance in the load task.
     */
    static boolean helpShown;
    /**
     * User input
     */
    static Scanner in = new Scanner(System.in);
```

```

/**
 * Name of friends file in classpath.
 */
static String fileFriend;
/**
 * Name of index file in classpath.
 */
static String fileIndex;
/**
 * Content of friends file, line by line.
 */
static LinkedList<String> listFriends = new LinkedList<String>();
/**
 * Content of index file, line by line.
 */
static LinkedList<String> listIndices = new LinkedList<String>();
/**
 * SocialNetwork instance.
 */
static SocialNetwork network;

public static void main(String[] args) throws IOException {
    // Program header
    System.out.println(
        "|_**_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_|Social Network|_**_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_|");
    System.out.println();
    // Processes the friend and index files directory provided via arguments
    if (args.length == 2) {
        fileFriend = args[0];
        fileIndex = args[1];
        readFiles();

    }
    // If the file names are not provided via arguments display a hint.
    if (fileFriend == null) {
        System.out.println(
            "|_**_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_|Hint|_**_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_|");
        System.out.println(
            "| You can provide the file locations via the program arguments, for example; |");
        System.out.println(
            "| >> javac Driver.class friend.txt index.txt |");
        System.out.println(
            "| The file must be stored within the program directory. |");
        System.out.println(
            "|_**_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_|Hint|_**_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_|");
        System.out.println();
    }
    // Main control loop
    while (true) {
        // print task menu, then handle user input.
        // for invalid input, alert user in default block.
        displayMenu();
        String choice = in.nextLine();

```

```

switch (choice) {
case "1":
    task1();
    break;
case "2":
    task2();
    break;
case "3":
    task3();
    break;
case "4":
    task4();
    break;
case "5":
    task5();
    break;
case "6":
    task6();
    break;
case "7":
    System.out.println("Exiting...");
    System.exit(0);
    break;
default:
    System.out.print(" is not a valid choice, please enter a choice from 1 to 7.");
    System.out.println();
    break;
}

}

}

/**
 * Read input files.
 */
private static void task1() {
    if (!helpShown) {
        System.out.println();
        System.out.println("The structure of the two files required to create a network is as follows; "
);
        System.out.println();
        System.out.println("Friend text file:");
        System.out.println(
"The friend text file consists of pairs of friends. The integers are references to friends.");
        System.out.println("defined in the index file.");
        System.out.println();
        System.out.println("Index text file:");
        System.out.println(
"Each line has one integer representing their vertex and then a space followed by their name with no
spaces.");
        System.out.println("The following is an example of friend.txt and index.txt");
        System.out.println(
"(they can be named differently, as long as they are provided in this order and are text files)");

```

```

System.out.println();
System.out.println("| friend.txt | | index.txt |");
System.out.println("| 0 3      | | 0 Gromit      |");
System.out.println("| 1 3      | | 1 Gwendolyn   |");
System.out.println("| 0 1      | | 2 Le-Spiderman |");
System.out.println("| 2 4      | | 3 Wallace     |");
System.out.println("| 1 5      | | 4 Batman       |");
System.out.println("|          | | 5 Superman     |");
System.out.println();
helpShown = true;
}

```

"Please enter a location for a friend text file and press enter and then do the same for an index text file."

```

System.out.println("Friend text file location (e.g. /friend.txt)");
System.out.print(">> ");
// Get user input for friend file location.
fileFriend = in.nextLine();
System.out.println();
System.out.println("Index text file location");
System.out.print(">> ");
// Get user input for index file location.
fileIndex = in.nextLine();
readFiles(); // Read files and store in linkedlist.
}

```

```

/**
 * Prompt for name, print friends and friends of friends
 */
private static void task2() {
    System.out.println("List friends & friends of friends of:");
    System.out.print(">> ");
    String name = in.nextLine();
    network.listFriendsAndMutals(name);
}

```

```

/**
 * Prompt for name, print friends
 */
private static void task3() {
    System.out.println("List friends of:");
    System.out.print(">> ");
    String name = in.nextLine();
    network.listFriends(name);
}

```

```

/**
 * Prompt for two names, list all friends known to the two people
 */
private static void task4() {
    System.out.println("List shared friends of:");
    System.out.print(">> ");
    String name1 = in.nextLine();
}

```

```

        System.out.print(">> ");
        String name2 = in.nextLine();
        network.listOverlap(name1, name2);
    }

    /**
     * Delete member from the network
     */
    private static void task5() {
        System.out.println("Delete account for user:");
        System.out.print(">> ");
        String name = in.nextLine();
        network.delete(name);
    }

    /**
     * List all members sorted by popularity, then by names
     */
    private static void task6() {
        network.sorted();
    }

    /**
     * Note that the information read from these two files represents a new
     * network. All information from the default or previous network will be
     * replaced by this new network.
     */
    public static void displayMenu() {
        System.out.println("Please enter the number representing your option and press enter;");
        System.out.print("1.  Prompt users for a friend filename and an index file name. ");
        // Checks if network already exists to inform user.
        if (network != null) {
            System.out.print("[Network Exists! Run to overwrite.]");
        }
        System.out.println();
        System.out.println(
"2.  Prompt users for a name and list all the friends and friends of the friends (task 2).");
        System.out.println("3.  Prompt users for a name and list all the friends (task 3).");
        System.out.println(
"4.  Prompt users for two names, and list all the friends known to these two people (task 4).");
        System.out.println("5.  Delete a member from the network (task 5).");
        System.out.println("6.  List all members sorted by popularity, then by names (task 6).");
        System.out.println("7.  Exit.");
        System.out.print("\n>> ");
    }

    /**
     * Parse content of the given file into a linked list.
     *
     * @param path
     *         Path to file in classpath.
     * @return Linked list containing file contents, line by line.
     */

```

```

private static LinkedList<String> getFileContents(String path) {
    LinkedList<String> filecontents = new LinkedList<String>();
    try (Scanner scan = new Scanner(Driver.class.getResourceAsStream(path))) {
        while (scan.hasNext()) {
            String i = scan.nextLine();
            filecontents.add(i);
        }
    } catch (Exception e) {
        System.err.println("File not found at: " + path);
    }
    return filecontents;
}

/**
 * Read stored file names into their respective lists, then create a network
 * based on the file contents.
 */
private static void readFiles() {
    if (fileFriend != null) {
        System.out.println("Reading friend file..");
        listFriends = getFileContents(fileFriend);
        // The original file specification declares the size in the first
        // line...
        // But updating that is easily forgotten so we'll just calculate the
        // number ourselves.
        if (listFriends.size() > 0 && !listFriends.get(0).trim().contains(" ")) {
            listFriends.remove(0);
        }
    } else {
        System.err.println("Friend file location not provided!");
    }
    if (fileIndex != null) {
        System.out.println("Reading index file..");
        listIndices = getFileContents(fileIndex);
        // The original file specification declares the size in the first
        // line...
        // But updating that is easily forgotten so we'll just calculate the
        // number ourselves.
        if (listIndices.size() > 0 && !listIndices.get(0).trim().contains(" ")) {
            listIndices.remove(0);
        }
    } else {
        System.err.println("Index file location not provided!");
    }
    // create the network
    network = new SocialNetwork(listIndices, listFriends);
}
}

```

PHP Code

```

<?php
package com.socialnetwork;

import java.util.*;

```

```

public class SocialNetwork extends Graph {
    /**
     * Map of names to vertex numbers.
     */
    private final Map<String, Integer> lookup = new HashMap<>();
    /**
     * Map of vertex numbers to names.
     */
    private final Map<Integer, String> lookupReverse = new HashMap<>();

    public SocialNetwork(LinkedList<String> index, LinkedList<String> friends) {
        // Size of the graph specified by the number of values in the index
        // list.
        super(index.size());
        // populate lookup table of friends and their vertex numbers
        index.forEach(line -> {
            // ensure line validity
            if (!line.contains(" ")) {
                System.err.println("Invalid index file!");
                System.exit(0);
            }
            // parse parts
            String[] parts = line.split(" ");
            int id = Integer.parseInt(parts[0]);
            String name = parts[1];
            System.err.println("==== " + id + " : " + name + "====");
            // add to lookup
            lookup.put(name, id);
            lookupReverse.put(id, name);
        });
        // add vertices and edges
        friends.forEach(line -> {
            // ensure line validity
            if (!line.contains(" ")) {
                System.err.println("Invalid friend file!");
                System.exit(0);
            }
            // parse parts
            String[] parts = line.split(" ");
            int source = Integer.parseInt(parts[0]);
            int target = Integer.parseInt(parts[1]);
            // add edges
            addEdge(source, target);
            addEdge(target, source);
        });
    }

    /**
     * @param name
     *      Name of person in network to list friends of.
     */
    public void listFriends(String name) {
        int vertex = getVertex(name);
    }
}

```

```

    if (vertex == -1) {
        System.out.println("No person in the network exists with the name '" + name + "'");
        return;
    }
    List<String> persons = new ArrayList<>();
    populateFriends(vertex, persons);
    System.out.println("Friends: ");
    print(persons);
}

/**
 * @param name
 *      Name of person in network to list friends of.
 */
public void listFriendsAndMutals(String name) {
    int vertex = getVertex(name);
    if (vertex == -1) {
        System.out.println("No person in the network exists with the name '" + name + "'");
        return;
    }
    List<String> persons = new ArrayList<>();
    populateFriends(vertex, persons);
    // copy for iteration without concurrent-modification exception
    List<String> iterationCopy = new ArrayList<>(persons);
    iterationCopy.forEach(p -> populateFriends(getVertex(p), persons));
    System.out.println("Friends + Mutuals: ");
    print(persons);
}

/**
 * List overlapping edges of two vertices <i>(persons)</i> in the network.
 *
 * @param name1
 *      Name of first person
 * @param name2
 *      Name of second person
 */
public void listOverlap(String name1, String name2) {
    int vertex1 = getVertex(name1);
    if (vertex1 == -1) {
        System.out.println("No person in the network exists with the name '" + name1 + "'");
        return;
    }
    int vertex2 = getVertex(name2);
    if (vertex2 == -1) {
        System.out.println("No person in the network exists with the name '" + name2 + "'");
        return;
    }
    List<String> persons1 = new ArrayList<>();
    List<String> persons2 = new ArrayList<>();
    populateFriends(vertex1, persons1);
    populateFriends(vertex2, persons2);
    // exclude non-overlapping entries

```

```

persons1.retainAll(persons2);
System.out.println("Overlap: ");
print(persons1);
}

/**
 * Remove all references to this member... Can't really delete them since
 * the array size can't be dynamically changed <i>(It's also private so its
 * not like they even want you to have raw access to it either)</i>.
 *
 * @param name
 *      Name of person to delete.
 */
public void delete(String name) {
    int vertex = getVertex(name);
    if (vertex == -1) {
        System.out.println("No person in the network exists with the name '" + name + "'");
        return;
    }
    System.out.println("Are you sure you with to delete '" + name + "'? Type 'Y' to confirm.");
    System.out.print(">> ");
    try (Scanner scan = new Scanner(System.in)) {
        if (scan.nextLine().equals("Y")) {
            // remove references
            for (int i = 0; i < getSize(); i++) {
                removeEdge(i, vertex);
                removeEdge(vertex, i);
            }
            System.out.println("User has been removed");

        } else {
            System.out.println("Deletion cancelled.");
        }
    } catch (Exception e) {}
}

public void sorted() {
    // get popularity of all vertices
    List<PopularityWrapper> persons = new ArrayList<>();
    for (int i = 0; i < getSize(); i++) {
        List<String> friends = new ArrayList<>();
        populateFriends(i, friends);
        persons.add(new PopularityWrapper(getName(i), friends.size()));
    }
    // sort & print
    Collections.sort(persons);
    persons.forEach(wrap -> System.out.println(wrap.name));
}

/**
 * Populate the persons list with edges linked to the given vertex.
 *

```

```

* @param vertex
*     Person.
* @param persons
*     List to add friends to.
*/
private void populateFriends(int vertex, List<String> persons) {
    for (int i = 0; i < getSize(); i++) {
        String name = getName(i);
        // check for:
        // * not self
        // * edge between other vertices
        // * result list does not already contain value
        if (vertex != i && isEdge(vertex, i) && !persons.contains(name)) {
            persons.add(name);
        }
    }
}

/**
* Print list.
*
* @param persons
*     List of names.
*/
private void print(List<String> persons) {
    persons.forEach(p -> System.out.println(p));
}

/**
* @param vertex
*     Vertex number
* @return Name of person associated with the vertex.
*/
public String getName(int vertex) {
    return lookupReverse.getDefault(vertex, "?Person?");
}

/**
* @param name
*     Name of person in network.
* @return Vertex number of person.
*/
public int getVertex(String name) {
    return lookup.getDefault(name, -1);
}

/**
* @return Size of network in persons.
*/
private int getSize() {
    return lookup.size();
}

```

```

/**
 * Class for wrapping a person's popularity / name. Allows for easy sorting via Collections.sort().
 */
private static class PopularityWrapper implements Comparable<PopularityWrapper> {
    private final String name;
    private final int popularity;

    PopularityWrapper(String name, int popularity) {
        this.name = name;
        this.popularity = popularity;
    }

    @Override
    public int compareTo(PopularityWrapper o) {
        int p = Integer.compare(o.popularity, popularity);
        if (p == 0) {
            p = name.compareToIgnoreCase(o.name);
        }
        return p;
    }
}

```

If this is an intro class, this'll probably have to be modified to remove "things outside of the scope of this class". My prof in our class let us do whatever the heck we wanted as long as the output was correct. I changed a few things here and there but most should be easy to revert back.

Just Ice - May 17, 2018, 3:10 am

Quote: If this is an intro class, this'll probably have to be modified to remove "things outside of the scope of this class". My prof in our class let us do whatever the heck we wanted as long as the output was correct. I changed a few things here and there but most should be easy to revert back.

Wow man I cannot believe this took you that long and I've been researching the best way to complete it all day, and its all so neat and clean!