UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA
Denkleiers • Leading Minds • Dikgopolo tša Dihlalefi

Department of Computer Science
Faculty of Engineering, Built Environment & IT
University of Pretoria

COS212 - Data structures and algorithms

Assignment 4 Specifications:
Graphs

Release Date: 28-05-2022 at 09:30

Fitch Fork Submissions open: 30-05-2022 at 06:00

Due Date: 10-06-2022 at 23:59

Total Marks: 36

Contents

1 General instructions:

- This assignment should be completed individually, no group effort is allowed.
- Be ready to upload your assignment well before the deadline as no extension will be granted.
- You may only import Java's **ArrayList** data structure. Importing any other library or data structure will result in a mark of zero. You may only make use of 1-dimensional native arrays where applicable. If you require additional data structures, you will have to implement them yourself.
- If your code does not compile, you will be awarded a mark of zero. Only the output of your program will be considered for marks, but your code may be inspected for the presence or absence of certain prescribed features.
- All submissions will be checked for plagiarism.
- Read the entire specification before you start coding.
- You will be afforded three upload opportunities.
- Submissions that result in a compilation failure or an run time error will also receive a mark of zero.

2 Plagiarism

The Department of Computer Science considers plagiarism as a serious offence. Disciplinary action will be taken against students who commit plagiarism. Plagiarism includes copying someone else's work without consent, copying a friend's work (even with consent) and copying material (such as text or program code) from the Internet. Copying will not be tolerated in this course. For a formal definition of plagiarism, the student is referred to http://www.library.up.ac.za/plagiarism/index.htm (from the main page of the University of Pretoria site, follow the Library quick link, and then choose the Plagiarism option under the Services menu). **If you have any form of question regarding this, please ask one of the lecturers, to avoid any misunderstanding.** Also note that the OOP principle of code re-use does not mean that you should copy and adapt code to suit your solution.

3 Outcomes

The primary aim of this assignment is to implement a minimum spanning tree algorithm and a graph colouring algorithm in the context of a graph based database.

4 Introduction

Complete the task below. Certain classes have been provided for you alongside this specification in the Student files folder. A very basic main has been provided. **Please note this main is not extensive and you will need to expand on it**. Remember to test boundary cases. Submission instructions are given at the end of this document.

5 Task 1: GraphDB

A Graph is a collection of vertices and connections between them. The connections are known as edges. Each edge connects to a pair of vertices. If the edges are bi-directional, the graph is known as a undirected graph. Each edge can be assigned a number that can represent values such as cost, distance, length or weight. Such a graph is then called a weighted undirected graph. A graph based database is a database that uses a graph to represent relationships between users. These graphs have gain popularity with social media sites as relationships between two users can be easily modeled. You will be implementing a primitive graph based database where users and relationships should be able to be added to the graph. As part of the analytics for the database, you will have to implement a graph colouring algorithm as well as a minimum spanning tree algorithm. You have been provided with the following files: User.java, Relationship.java and GraphDB.java. You may use the **ArrayList** data structure. You **will** have to add custom variables and functions to the provided classes.

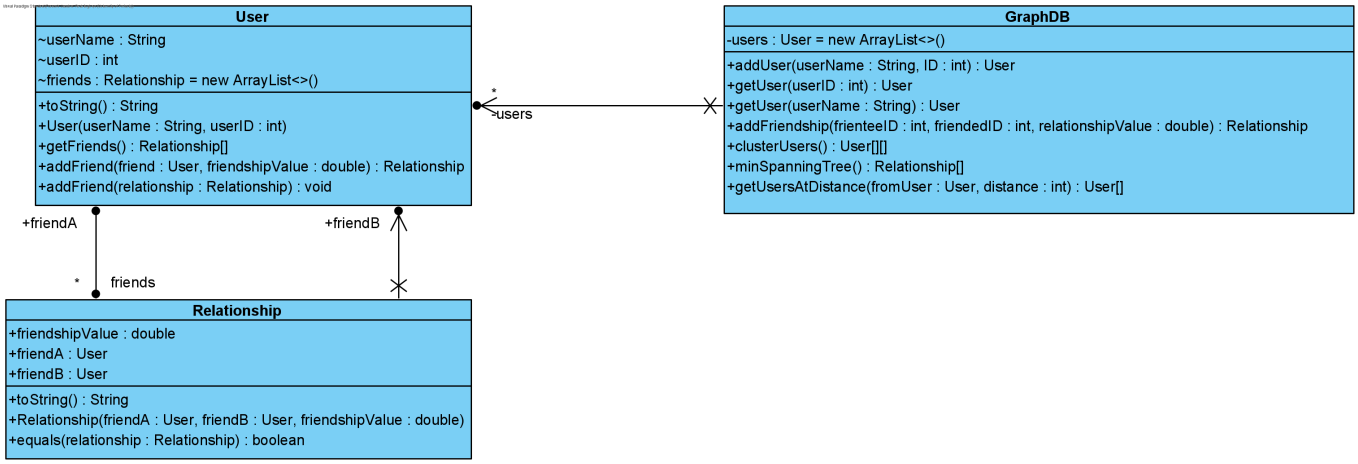

Figure 1: Complete class diagram

5.1 User

- Members:
  - userName: String
    - This is the userName of the user
  - userID: int
    - This is the unique ID for this user in the database.
  - friends: ArrayList<Relationship>
    - This is an array list of relationships representing other users which the current user is connected to.
- Functions:
  - toString: String
    - Do not modify this function
  - User(userName: String, userID: int)
    - This is the constructor for the User class. The userName and userID should be initialized accordingly.
  - getFriends(): Relationship[]
    - Do not modify this function
  - addFriend(friend: User, friendshipValue: double): Relationship
    - This function should add a relationship between the current user and the passed in friend.
    - The friendshipValue for the newly created relationship should be the passed in friendshipValue.
    - If friend is null, the function should return null.

        * If a relationship between the current user and the friend already exists, the function should return the already existing relationship.
        * If no relationship exists between the current user and the friend, the newly created relationship should be returned where friendA is the current user and friendB is the passed in friend.
        * The newly created relationship should also be added to the friends list.
        * *Hint: due to the undirected nature of the graph it is recommended to add the relationship to both the current user and the friend user's friends list.*
    - addFriend(relationship: Relationship): void
        * This is a helper function for the above addFriend function.
        * This function should add the passed in relationship to the current user's friends list.
        * *Note this function will not be tested*

5.2 Relationship

- Members:
    - friendshipValue: double
        * This is the value associated with the relationship.
        * This will serve as the weight for the edge.
    - friendA: User
        * This is one of the friends in the relationship.
    - friendB: User
        * This is the other friend in the relationship.
- Functions:
    - toString(): String
        * Do not change this function
    - Relationship(friendA: User, friendB: User, friendshipValue: double)
        * This is the constructor for the class.
        * Initialize the appropriate members with the appropriate passed in parameters.
    - equals(relationship: Relationship): boolean
        * Do not change this function
        * This function will determine if two relationships are equal by comparing the user IDs of the friends in the relationship.

5.3 GraphDB

- Members:
  - users: ArrayList<User>
    - This is a list of all the users in the database.
- Functions:
  - addUser(userName: String, ID: int): User
    - This function should attempt to add a user to the graph database.
    - If a user with the same ID already exists in the database the existing user should be returned else,
    - The newly created user should be initialized with the passed in parameters, added to the list and returned.
  - getUser(userID: int): User
    - This function should return the user with the same ID as the passed in parameter.
    - If no user exists with the corresponding ID then the function should return null.
  - getUser(userName: String): User
    - This function should return the first occurrence of a user with the same username as the passed in username.
    - If no user exists with the corresponding username then the function should return null.
  - addFriendship(frienteeID: int, friendedID, relationshipValue: double): Relationship
    - This function should attempt to add a relationship between the users with the corresponding IDs. In other words, a relationship between user with an ID equal to frienteeID and user with an ID equal to friendedID.
    - If a relationship already exists between the two users the existing relationship should be returned.
    - If no relationship exists, a new relationship should be created between the two users. Return the relationship where friendA has the ID of frienteeID and friendB the ID of friendedID.
    - If a relationship cannot be added between the two friends and no relationship exists between the two friends the function should return null.
  - clusterUsers(): User[][]
    - This function should cluster users together using the Brelaz graph colouring algorithm.
    - The function should then group all the users with the same colour into an index in the resulting 2d array.
    - Users should be sorted according to user IDs from smallest to largest.

    - The groups of users should be added to the resulting 2d array in accordance to colours assigned. The first colour assigned should be index 0. The second colour assigned should be index 1 etc.

- minSpanningTree(): Relationship[]
    - This function should create a minimum spanning tree of graph db.
    - The function should return all the relationships in the minimum spanning tree.
    - You do not have to sort the resulting array. The array will be sorted by the main.
    - Please see example for further clarity.
    - This function should not alter the graph.

- getUsersAtDistance(fromUser: User, distance: int): User[]
    - This function should return all the users that is a certain distance away from the starting user as an array.
    - The distance is defined as the number of users away from the starting user.
    - If no users are the required distance away from the staring user the function should return an empty array.
    - You do not have to sort the resulting array. The array will be sorted by the main.

6 Example

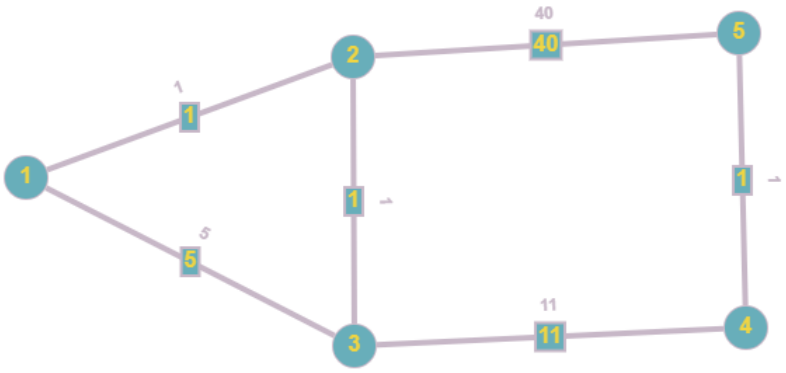

Figure 2: Example 1

If the above graph is used to represent the graph database where the numbers on the vertex represents the friendship values the following results would be obtained:

- clustering:
  - [0] = {2,4}
  - [1] = {3,5}
  - [2] = {1}
  - Note the above should be read as:
    [index] = {elements in array at index}
- minSpanningTree:
  - [0] = {1,2}
  - [1] = {2,3}
  - [2] = {3,4}
  - [3] = {4,5}
  - Note the above should be read as:
    [index] = {relationship with friendA, friendB}
- getUsersAtDistance(1, 2):
  - [0] = {4}
  - [1] = {5}
  - Note the above should be read as:
    [index] = {user at distance of 2}

7 Submission

You need to submit your source files on the Fitch Fork website (https://ff.cs.up.ac.za/).All methods need to be implemented (or at least stubbed) before submission. The following java files should at least be in a zip archive named uXXXXXXXX.zip where XXXXXXXX is your student number:

- GraphDB.java
- User.java
- Relationship.java

You may add any other custom classes that you created. Your code should be able to be compiled with the following command:

*make *.java*

You have 3 submissions and your best mark will be your final mark. Upload your archive to the Assignment 4 slot on the Fitch Fork website. Submit your work before the deadline. **No late submissions will be accepted!**