CSCI2270: Assignment 6 – Graphs

Objectives

1. Implement an unweighted, directed graph
2. Implement a breadth-first traversal of the graph
3. Use a breadth-first search to compute reachability

Instructions

The assignment submission will work the same as previous assignments. **Be sure to submit your code to both Canvas and GitHub**. Please read the entire document before writing any code. If you have any questions, ask a member of the course staff.

Problem Statement

For this assignment, you will write a program to do the following:

1. Read in a csv file of social media usernames and a set of profiles they follow
2. Represent the profiles as vertices in a graph, adding directed edges to represent followership as specified in the csv file
3. Use a breadth-first traversal to print out the usernames of profiles that are reachable from a specified starting vertex

Starter Code

For this assignment, we have provided you with some starter code for your convenience. We recommend reading through the entirety of the starter code before beginning this assignment. For a full description of the starter code files, please refer to the README on GitHub.

Questions

0. Review the header and implementation files for the `Graph` class. Your assignment will begin with the implementation of some methods within this class. Note that a few helper functions have been provided for you in the Graph.hpp file. While you do not need to implement these functions, you are encouraged to do so.

1. Implement the `Graph` member method `addVertex`. This method should accept one argument specifying the username of the profile to be added to the Graph. Create a new vertex in the graph representing the specified profile. If a vertex with the specified username already exists, this function should not create a new vertex.

   Sample declaration:

```
void Graph::addVertex(string new_user)
```

2. Implement the `Graph` member method `addEdge`. This method should accept two parameters. The first parameter will specify the username of the follower, and the second parameter will specify the username of the profile that is being followed. Add an edge to your graph to connect the two vertices appropriately. The vertex should start at the username of the follower and be directed toward the username

of the profile that is being followed. If either of the two profiles do not exist, or if there the desired edge already exists, your function should not create a new edge.
Hint: Remember that the graph is directed!

Sample declaration:

```
void Graph::addEdge(string v1, string v2)
```

3. Implement the `Graph` member method **`withinReach`**. This method should accept one parameter specifying the username associated with the starting vertex. The function should use a breadth-first traversal to print out the usernames of all of the profiles that are within reach of the starting vertex. A profile is considered "within reach" if there exists a path from the starting vertex to the other profile.
Note: This method should not print out the username of the starting profile.
Hint: You may choose how you wish to implement your Queue. You are free to use your own queue class (be sure to update the makefile), a vector, or the C++ standard library's queue implementation.

Sample declarations:

```
void Graph::withinReach(string start)
```

4. The following question will require you to write code in your `main.cpp` file. Using the `Graph` class you just created, write a complete program to do the following:
    a. The main function should take two command line arguments:
        1. the name of the input csv file containing profile usernames and a list of profiles being followed by each user.
        2. a string specifying a username to be the starting profile

        If there are too few or too many command line arguments, your program should print "Error: invalid command line arguments." and then your program should end.
        Hint: Remember that the name of the executable ("a.out") counts as a command line argument.

    b. Open the input csv file. If the file does not open, your main function should print "Error: could not read input file." and then your program should end. Make sure you close your input file when your program ends.

    c. Parse each line of the input csv file. Each line will start with a username representing a profile. Then, the rest of the line will contain the that are being followed by the first profile. Use your `addVertex` function to create profiles with the appropriate usernames. Then, use your `addEdge` function to add edges from the follower to the profiles that are being followed. Be sure to

check whether the profile username and the edge already exist before adding either to your Graph. Duplicate profile usernames and edges should be ignored.
Hint: Consider modifying your `splitLine` function from Assignment 2 to return a vector of strings.

d. When you have finished adding all of the profiles and edges to the graph, your program should display a list of all of the profiles that are reachable from a starting profile. The username of the starting profile is specified by the second command line argument. If the given username does not exist, your program should print "<username> does not exist." and then your program should end. Otherwise, your program should print "Profiles reachable from <username>:" and then call your `withinReach` function to list all the profiles that are within reach of the starting profile. The final command line argument from the user will specify the maximum path length for a profile to be considered within reach. After the function has finished, your program should end.
Hint: You can use the `stoi()` function to convert a string to an integer.

Note: In the instance where multiple edge cases occur at the same time, your program should only print one error message before terminating. The error message precedence should follow the order in which they are presented above. For example, suppose somebody used too many command line arguments AND an invalid file name. You program should only print "Error: invalid command line arguments." before quitting.

CSCI2270: Assignment 6 – Graphs

Sample Run 1:

Example run command: ./graph tests/profiles-short.csv FashionistaEmily

profiles.csv
SunshineSurfer,FashionistaEmily,UrbanExplorerJane
UrbanExplorerJane,TechWhiz2000,GourmetGuruEats,ArtisticVisions23
TechWhiz2000,UrbanExplorerJane,MusicMakerMike
GourmetGuruEats,SunshineSurfer,TechWhiz2000
AdventureSeekerSam,SunshineSurfer
ArtisticVisions23
BookwormBethany,MusicMakerMike,SunshineSurfer,UrbanExplorerJane
MusicMakerMike,BookwormBethany,FashionistaEmily,GourmetGuruEats
FashionistaEmily,BookwormBethany,UrbanExplorerJane

Expected Print Output:
Profiles reachable from FashionistaEmily:
GourmetEats
SunshineSurfer
TechWhiz2000
UrbanExplorerJane
MusicMakerMike
ArtisticVisions23
BookwormBethany

Sample Run 2:

Example run command: ./graph imaginaryfile.txt

Assume imaginaryfile.txt file does not exist

Expected Print Output:
Error: could not read input file.

Sample Run 3:

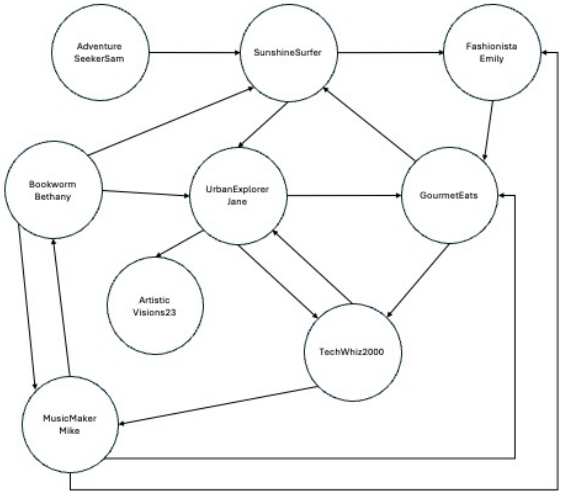
Example run command: ./graph

Expected Print Output:
Error: invalid command line arguments.

CSCI2270: Assignment 6 – Graphs

Visual Example:

profiles-short.csv
SunshineSurfer,FashionistaEmily,UrbanExplorerJane
UrbanExplorerJane,TechWhiz2000,GourmetEats,ArtisticVisions23
TechWhiz2000,UrbanExplorerJane,MusicMakerMike
GourmetEats,SunshineSurfer,TechWhiz2000
AdventureSeekerSam,SunshineSurfer
ArtisticVisions23
BookwormBethany,MusicMakerMike,SunshineSurfer,UrbanExplorerJane
MusicMakerMike,BookwormBethany,FashionistaEmily,GourmetEats
FashionistaEmily,GourmetEats



Visual Example of Reachability:
Profiles reachable from FashionistaEmily: