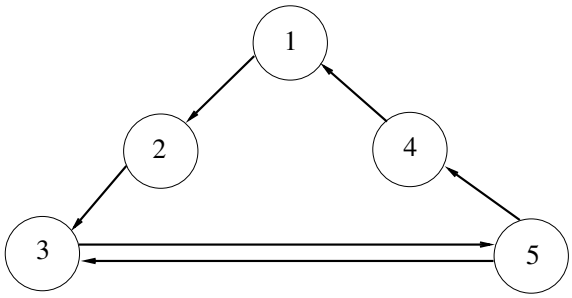
CSCI-1200 Data Structures — Spring 2018
Homework 7 — Spatially-Embedded Adjacency Lists

Overview

In this homework you will be taking on the role of an undergraduate student helping a graduate student with their research into social networks. They understand that you are currently in a 1000-level CS course, so you do not need to have a strong knowledge of social networks, graph algorithms, or graph theory.

A *social network* is simply a graph (like trees but they can have cycles) where each node (circle on the graph) represents a person, and each edge (connection between nodes) represents a social connection. In this assignment we will work with some synthetic datasets where each node is a user in a social network application, and each edge indicates a friendship. Edges are directed; this means that an edge from user 1 to 2 means 2 is a friend of 1, but does not mean 1 is a friend of 2. In order for a friendship to go both ways (sometimes called *reciprocated* friendship), there must be an edge in both directions, such as between 3 and 5 in the example below. The social network is loaded in by the `load-connections` command.

The number of friends a user has (i.e. the number of edges leaving a node) is sometimes called the *degree*.



A simple graph (not simple_connection.txt, but close)

In addition to the friendship network, some users choose to report their location. This means the data is actually a rare case of a *spatially embedded* data, meaning that there is a geospatial aspect to the social network. The location data is loaded in by the `load-locations` command.

Adjacency Lists and Matrixes

To represent a graph, a couple data structures are often used. The adjacency matrix is filled with 0s and 1s for no connection and connection respectively, where the row is the start of an edge and the column is the end of an edge.

$$\begin{bmatrix} & 1 & 2 & 3 & 4 & 5 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 2 & 0 & 0 & 1 & 0 & 0 \\ 3 & 0 & 0 & 0 & 0 & 1 \\ 4 & 1 & 0 & 0 & 0 & 0 \\ 5 & 0 & 0 & 1 & 1 & 0 \end{bmatrix}$$

Adjacency Matrix for the above graph

However, this wastes a lot of space with unnecessary 0s. So instead, sometimes adjacency lists are used. Each adjacency list corresponds to all edges starting from one node, and simply contains the IDs that the edges end at. In our context, this means each adjacency list is simply the list of friends that a user has. **NOTE:** In this assignment you will *only use adjacency lists* to represent the network.

Node ID	Friend IDs
1	2
2	3
3	5
4	1
5	3,4

Adjacency Lists for the above graph

Your Tasks

You are provided with a `main.cpp` file that is partially completed. There are four typedefs at the top of the provided code. They have not been filled in, so your first task is to replace ?????? with whatever type is appropriate in each case. It is possible to use a typedef in another typedef, so you should do so when appropriate. Once you have completed the four typedefs, the code should be able to compile and run. At least one of your typedefs should use an `STL map`. There is more than one correct answer for the typedefs, but you should make your choices based on efficiency and the program's functionality.

You can add additional #include statements, however do not write any of your own .h files. Do not change the foward declarations of functions near the top of the file, and do not change anything in main().

Below main() you will find a space to implement each of the functions; your implementations should go here. Since the graduate student you're helping is at a conference overseas and won't be reachable, they've left you detailed comments for each function, as well as a couple of sample networks, input files, and sample output. You will need to implement all of the functions in order to match the output.

The program takes a file with commands (e.g. `simple_test1.txt`) and an output file (e.g. `out_simple_test1.txt`). Code is already written in main to check arguments and read the command file.

Assumptions

You may assume the following:

- Every social network/location file is properly formatted
- There is no user ID larger than 4,000,000,000
- There are no duplicate edges in a network
- There is never an empty connections file.
- Each load command (*load-connections* and *load-locations*) is only called once per test, and the load commands are called before any other command.

Submission

Use good coding style when you design and implement your program. Don't forget to comment your code! Use the provided template `README.txt` file for notes you want the grader to read. Remember to provide a justification for your complexity analyses in the readme file. You must do this assignment on your own, as described in the "Collaboration Policy & Academic Integrity" handout. If you did discuss this assignment, problem solving techniques, or error messages, etc. with anyone, please list their names in your `README.txt` file.

If you earn at least 8 points on test cases 3, 4, and 5 by Wednesday March 28th 11:59:59 PM, you can submit on Friday without using a late day.