MSIT F14-04630-B at Carnegie Mellon University

Timothy Kaboya

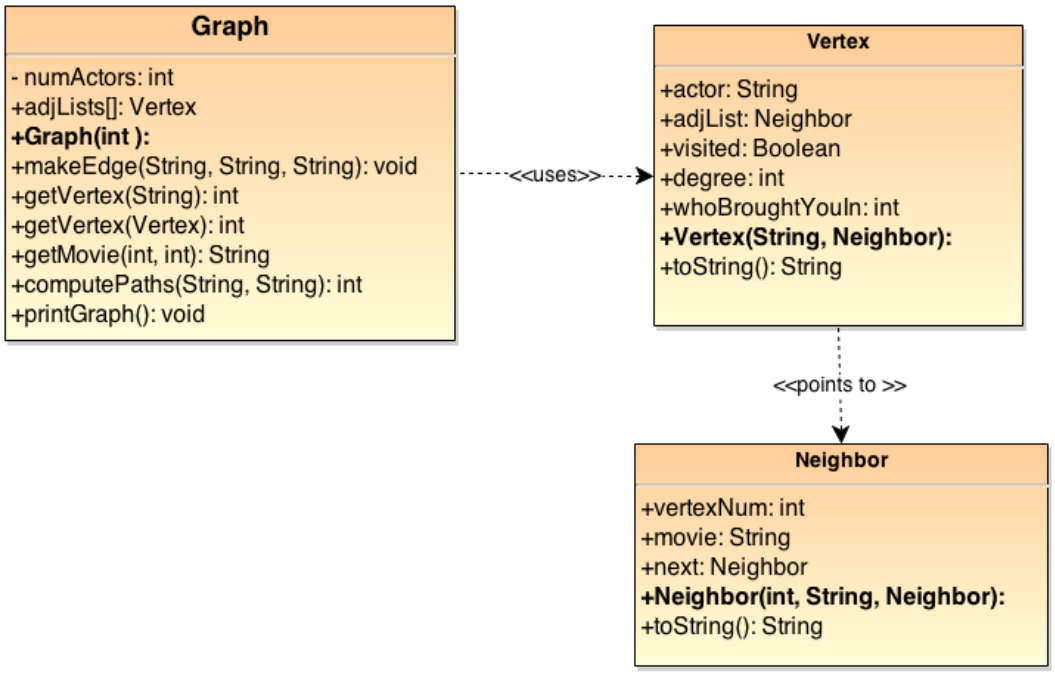
Assignment 5: 6 Degrees of Separation

Tips from Discussions with others

Antoine: When implementing your BFS, have a separation of concerns and first implement a bfs, before worrying about degrees of separation. .

Description of ADTs used

A UML Description of the Graph ADT



The main ADT in this program is the Graph Class which utilizes a custom LinkedList class based on a Vertex and Neighbor class. The Graph has an adjLists which is an array of Vertices that represents the entire graph. Operations on graph structure are made at this level.

Each Vertex is essentially an Actor with the name of the actor and a pointer to its neighbors. Each neighbor stores the movie information, an index into the array of Vertices(Actors) and a pointer to the next neighbor. Some variables are kept private for internal access only by the class (Ex numActors)

For expediency in access, some of the class attributes for the feeder classes (Vertex and Neighbor) are not given getter/setter properties. This is for convenience and easy access as they are only used by the Graph class.

Furthermore, the Graph implements a LinkedList implementation in lieu using a Matrix. A LinkedList uses less memory and computation.

A simple structure of the graph is represented below.

* Array of Linked Lists
* 0: Vertex0 → Neighbor1 → Neighbor2 → Neighbor3 → Neighbor6 → NULL
* 1:Vertex1 → Neighbor3 → Neighbor0 → NULL
* 2: Vertex2 → Neighbor0 → Neighbor4 → Neighbor1 → NULL
* 3: Vertex3 → Neighbor5 → Neighbor1 → Neighbor0 → NULL

Describe how to use your Program

The Program is fully described in the README.md that is under the docs folder.

Correctness and Complexity Analysis

A: Correctness

Partial Correctness

Assuming that program receives legal input for the two actors. Legal meaning that the actors are valid string and that both actors are already in our list of movie actors from the Data file. To be correct, the program must correctly compute the degree of separation between these two actors.

First, let us look at the correctness for the BFS algorithm in the computePaths graph. This is on assumption that the other trivial operations such as adding the movie-actor relationships to the graph worked out. We would have an infinite loop if we always added neighbors to the queue. However, we only add neighbors to the queue if they have not been visited yet and then we mark them visited. There is a finite number of neighbors and since for each iteration through the while loop, we pop a value off the queue. At some point the BFS will terminate with the value of the degree of separation if it exists.

Total Correctness

For total correctness, we must show that in addition to our BFS working correctly, the algorithm for the degree of correctness correctly computes the degree at each stage of the Breadth First Search. The degree for each neighbor is incremented based on that of the parent with the initial actor having a degree of -1. In this way we can see that the degree of separation increases to the desired one. The computePath algorithm also backtracks from the second actor to the first one.

B: Complexity Analysis

Running Time:

Our program running time is based upon the most complex runtime operations. Our two foremost complex operations are analyzed below.

Adding movies to file: For the large list given, we must iterate through each line adding the movie-actor relationships to our graph. This has a complexity of **O(n)** where n is the number of lines in the text file.

ComputePath Method: This method implementation is based on the BFS algorithm and its complexity should be similar. The time complexity is $O(|V| + |E|)$. Our graph is not sparse so $O(|E|)$ does not change the complexity.

Memory Usage

The main data structure used is the adjacency list.

Graph implementation is based on this adjacency list more specifically an array of Linked Lists. This utilizes $\Theta(|V| + |E|)$