CSCI 1933 Lab 12

Graphs

Lab Rules

You may work individually or with *one* partner in lab. We suggest that you have a TA evaluate your progress on each milestone before you move onto the next. The labs are designed to be completable by the end of lab. If you are unable to complete all of the milestones by the end of lab, you have **until the last office hours on Monday** to get those checked off by **any** of the course TAs. We suggest you get your milestones checked off as soon as you complete them since Monday office hours tend to become crowded. If you worked with a partner, both of you must be present when getting checked off to receive credit. You will only receive credit for the milestones you have checked off by a TA. There is nothing to submit to Moodle for this lab.

Attendance

Per the syllabus, students with more than 3 unexcused lab absences over the course of the semester will automatically fail the course. The TAs will take attendance during lab; it is expected that students will be actively working on the lab material. *Your physical presence in lab is not sufficient to be marked as present for the purposes of attendance.*

Introduction

This lab focuses on exploration of graphs. A graph contains vertices (vertex nodes that represent unit components of the graph) and edges (paths that connect vertices). There are some important characteristics of graphs:

- Graphs can be weighted, in which each edge has different cost values, or unweighted, in which each edge is simply binary
- Graphs can be directed (edges only travel in one direction, connecting node A to node B), or undirected (edges simply connect pairs of nodes with no particular direction)
- Graphs can be cyclic (the graph contains a cycle) or acyclic (the graph does not contain any cycle)

In this lab, we will be working with weighted, acyclic and undirected graphs. As depicted in Figure 1, an example graph case has been given to you. We have provided you with some basic Java implementations about graph:

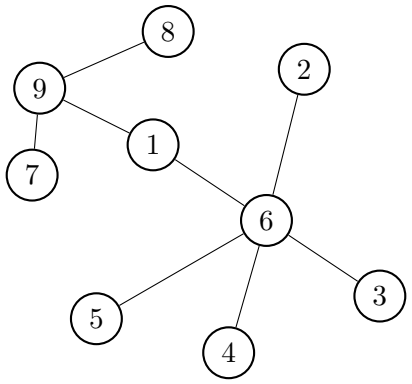

Figure 1: Sample Graph

- `GraphInterface`: This interface defines the methods you can use to interact with a graph abstract data type
- `SimpleGraph`: This class contains definition of Graph implementation based on the GraphInterface
- `HappyLabTwelve`: This class contains the main method to run your program, and sample graph initialization and test cases are already implemented for you. You will ONLY need to fill in `getBreadthFirstTraversal()`, `getDepthFirstTraversal()` and `measureDegrees()`
- `VertexInterface`: This interface defines the methods you can use to interact with a Vertex
- `Vertex`: This defines the vertex class that implements VertexInterface

Note: To simplify the lab, the implementation of basic graphs and test cases are already given to you. You will only need to implement three methods in `HappyLabTwelve` class and potentially some helper methods.

1 Breadth First Traversal

In this section, we will implement `getBreadthFirstTraversal()` in `HappyLabTwelve` class. The BFS (Breadth First Search) algorithm will start from an entry node in the graph and traverse through its immediate neighbors first. After, it will iteratively traverse through the neighbors of its immediate neighbors. An example BFS traversal is: 1-9-6-7-8-2-3-4-5 (Figure 2), 1-6-9-2-3-4-5-7-8, etc.

Please implement the `getBreadthFirstTraversal()` to accomplish the traversal of the graph and output the label values in the correct BFS order. Demonstrate that your BFS method works by performing a BFS from the vertex labeled 1 and 7 in the graph.

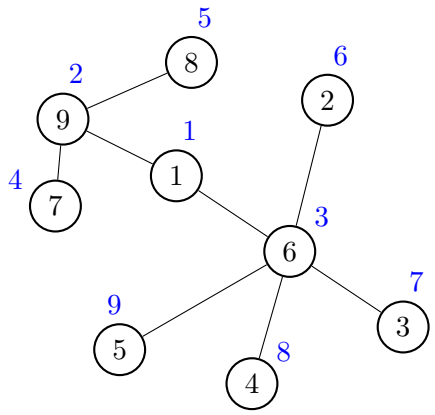

Figure 2: BFS Traversal Example

Milestone 1: Show your TA the output, your code, and briefly explain your algorithm and data structure.

2 Degree Measure

In this section, we will implement `measureDegrees()` in `HappyLabTwelve` class. The degree of a vertex of a graph is the number of edges incident to the vertex. For example, for the vertex labeled 1, the degree is 2. For the vertex labeled 6, the degree is 5.

Please implement the `measureDegrees()` to output degrees for all the vertices.

Milestone 2: Show your TA the output, your code, and briefly explain your algorithm and data structure.

3 Depth First Traversal (Optional)

In this section, we will implement `getDepthFirstTraversal()` in `HappyLabTwelve` class. DFS (Depth First Search) algorithm will start from an entry node in the graph and traverse as far as possible along the graph branch before it returns and traverses in other branches (backtracking). An example DFS traversal is: 1-9-7-8-6-2-3-4-5 (Figure 3), 1-6-2-3-4-5-9-7-8, etc.

Please implement the `getDepthFirstTraversal()` to accomplish the traversal of the graph and output the label values in the correct DFS order. Demonstrate that your DFS method works by performing a DFS from the vertex labeled 1 and 7 in the graph.

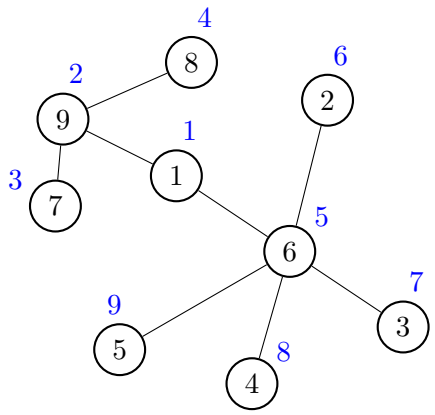

Figure 3: DFS Traversal Example

Milestone 3: (Optional) Show your TA the output, your code, and briefly explain your algorithm and data structure.