Lab 10 (Graphs)

CSC 172 (Data Structures and Algorithms)
Spring 2018
University of Rochester
Due Date: Sunday, April 22 @ 11:59 pm

Introduction

The labs in CSC172 will follow a pair programming paradigm. Every student is encouraged (but not strictly required) to have a lab partner. Every student must hand in their own work but also list the name of their lab partner (if any) on all labs.

Objectives

In this lab, you will explore Graph data structures. You will create an unweighted graph from a text file and then find the shortest path between two vertices.

Create `Graph` class

Create a class `Graph` consist of the following constructor and methods:

```
// Constructor
// Create a graph containing N vertices.
// Assume, all the vertices are numbered between 0 and N-1 for this lab
Graph(int N);

//  Add an undirected edge to the graph connecting Vertex i and Vertex j
void addEdge(int i, int j);

// Return a List of vertices which provides the shortest path between i and j.
//  Include both i and j if there is a valid path.
// If no such path exists, return an empty List.
// if i==j, return a list containing only i or j (but not both)
List<Integer> getShortestPath(int i, int j)
```

Creating Text files for Graphs

Define a file format for graphs as follows. The first line of the text file contains the number of nodes in the graph. Every line that follows represents an edge in the graph.

The content of a sample file may look like:

```
8
0 1
0 2
1 2
1 3
1 4
2 5
3 6
7 5
5 3
```

Create **three** (3) files following this structure representing the graphs you find interesting or challenging enough to demonstrate the power of your data structure.

Test your implementation

Test your implementation of the shortest path algorithm `getShortestPath` on the graphs you made. You can use the `main` method of `Graph.java` for this. For testing your implementation, read one of the text files to create and add edges to the graph. Once the graph is created pick any two vertices (You can safely assume, the vertices are valid) on the graph and print the path length and the shortest path. When you print the path, make sure it is in its natural order (i.e. the first node is printed first, last node last).

For example: for the given file, once you create the graph and add edges to it, running the following code `graph.getShortestPath(0, 7)` would return a list containing: {0, 2, 5, 7} where `graph` is an instance of `Graph`.

Traverse the list and print both, the path length and the path. Note that Path length is 1 less than the size of the array. If the two vertices are disconnected, print the path length as -1 and path as an empty string.
For our example, the main method should print the following two lines on the console.
Path Length: 3
Path: 0 2 5 7

Submission

Hand in `Graph.java`, the three .txt files that you have created representing your graphs and a README file at the appropriate location on the Blackboard system at `learn.rochester.edu`. You should hand in a single zip (compressed archive) `Lab10.zip`. The README file should state your and your team member's name and any other pertinent information.

Grading **(100 pts)**

- 10 pts deductions for not following the direction.
- 3 Text files: (**15** pts (5 pts each))
- `addEdge` method works correctly. (**15** pts)
- `getShortestPath` method works correctly: **50** pts
  (Must handle three cases:
  - A valid path where the two vertices are connected (40 pts).
  - A case where vertices are not connected (5pts)
  - A case where both the vertices are the same (5 pts)
- The `main` method used for testing. (Operations must perform: read file, create the graph, add edges, get the shortest path between any two vertices, print the shortest path)(**15** pts)
- README file (**5** pts)

READ THIS SECTION CAREFULLY:

- No Late Submission of any sort is allowed.
- The graph is **unweighted**.

- You can pick any data structure of your choice for storing the edges. Though we will suggest using an adjacency list.
- Feel free to add any number of member variables and helper functions to `Graph.java` as required.
- You can safely assume, the vertices used when calling `getShortestPath` are valid.
- TAs may use different test cases for testing your code. As long as the methods provided for Graph.java are correctly implemented, this should not cause any issue.