CPE 202 Winter 2020

Lab 9: Connected Graphs

Finding connected components Finding connected components and graph coloring are important problems in computer science. Connected components in an undirected graph give information about sets of vertices such that each pair of vertices has a path between them.

You only need to do either this lab or Lab 10. In case you do both lab 9 and lab 10, you can earn up to 100 points as extra credit toward the lab category.

A **coloring** is an assignment of a color to each vertex in the graph so that no two adjacent vertices have the same color. Colorings are important in modeling problems where a set of objects must satisfy certain constraints. More information on bicoloring and bipartite graphs can be found here: https://en.wikipedia.org/wiki/Bipartite_graph. A graph is bipartite or bicolorable if the vertices of the graph can be assigned labels, say red and black, such that no two adjacent vertices have the same label. (An equivalent definition is that the vertex set can be divided into two disjoint subsets where all the edges go from a vertex in one subset to a vertex in another subset.) You may assume that the graph is **undirected and does not contain self-loops (**edges from a vertex to itself). A good place to start is the graph class given in the lecture. Your goal is to have a correct algorithm that is simple and clear as possible.

Implement algorithms based on **Depth First Search** to:

1. Determine the connected components of an undirected graph
2. Determine if it is a bipartite graph (or in different words, 2 colorable.)

Implement an algorithm based on Breadth First Search as well. YOU MAY NOT USE BUILTIN LIST POP FUNCTION NOR DICTIONARY. USE ONE OF YOUR QUEUES FROM LAB3 AS THE QUEUE.

An input file consists of one graph for testing:

• Each graph will represent an undirected graph.

• The first line contains a **positive integer n**, that is the number of vertices in the graph to be tested, where 1<n<200. Each vertex is represented by a number from 1 to n

• The second line is the number of edges, call this number **e**.

• This is followed by **e** lines each containing a pair of integers each integer between 1 and n

separated by a space that represents an edge between the vertices represented by the numbers.

Create a file my_graph.py, and implement the following functions in the file:

- **def __init__ (self, filename**):
    - reads in the specification of a graph and creates a graph using an adjacency list representation. You may assume the graph is not empty and is a correct specification. E.g. each edge is represented by a pair of vertices between 1 and n. Note that the graph is **not directed** so each edge specified in the input file should appear on the adjacency list of each vertex of the two vertices associated with the edge.
- **def get_conn_components (self, is_dfs=True):**
    - returns a list of lists. For example, if there are three connected components then you will return a list of three lists. The order of the sub-lists is not important. However each sub-list will contain the vertices (in ascending order) in the connected component represented by that list. Each vertex is represented by an integer from 1 to n. If a vertex has no edges it will be in a connected component containing only itself. This function shall call either dfs or bfs method depending on the value of the argument **is_dfs**: i.e. do DFS if is_dfs=True, otherwise do BFS.
- **def init_vertices(self):**
    - initializes vertices.
- **def dfs(self, vertex, component):**
    - does dfs, finds connected components, and determines if the graph is bicolorable or not. If it is bicolorable, it is going to set the is_bicolor member variable to True, otherwise False. Takes two arguments, vertex (Vertex) and component (list). Returns a connected component as a list of vertex keys.
- **def bfs(self, vertex, component):**
    - does bfs, finds connected components, and determines if the graph is bicolorable or not. If it is bicolorable, it is going to set the is_bicolor member variable to True, otherwise False. Takes two arguments, vertex (Vertex) and component (list). Returns a connected component as a list of vertex keys.
- **def bicolor (self):**
    - returns True if the graph is bicolorable and false otherwise. i.e. It returns the value of is_bicolor.

SUBMISSION Upload a zip file containing **my_graph.py, which in turn** contains a

class **MyGraph, and your tests to the grader, then to polylearn.** MyGraph should contain your new graph class including functions that determine the connected components, **get_conn_components (self), which returns a list of lists of vertex keys**, and if the graph is bipartite, **bicolor (self), which simply returns a boolean value**.

Examples:
Test input files are available on polylearn.
Input:
9
8
1 2
1 3
1 4
1 5
6 7
7 9
9 8
8 6

9 is number of vertices and 8 is number of edges.
If the graph created is g1, then g1.get_conn_components() returns: [[1, 2, 3, 4, 5], [6, 7, 8, 9]], and g1.bicolor() returns: True.

Input:
8
6
1 2
2 3
3 1
4 6
4 7
4 8

8 is number of vertices and 6 is number of edges.
In this graph based on 8 vertices 5 is not connected to any other vertices and has no edges, therefore it is a sublist contains itself.
If the graph created is g2, then
g2.get_conn_components() returns: [[1, 2, 3], [4, 6, 7, 8], [5]], and g2.bicolor() returns: False