Lab 9:

Connected Graphs

Due: 12/ 1 @11PM

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.

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

Determine the connected components of an undirected graph

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.

Implement the following functions

- **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 conn_components (self):** 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.

SUBMISSION

Upload a file to PolyLearn, **my_graph.py** containing a class **MyGraph**, that contains your new graph class including functions that determine the connected components, **conn_components (self)**.

Examples:

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.conn_components() returns: [[1, 2, 3, 4, 5], [6, 7, 8, 9]]`

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.conn_components() returns: [[1, 2, 3], [4, 6, 7, 8] [5]]`