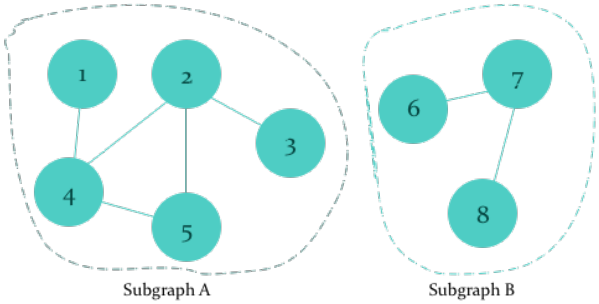
Project 5

Finding connected components and testing if a graph is bipartite

Finding connected components and graph coloring are important problems in computer science.

Knowing the **connected components** in an undirected graph provides information about sets of vertices such that each pair of vertices has a path between them. It is a list of the sub-graphs within the larger graph.

For the graph to the right, we have two sub-graphs within the larger graph.



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. In this assignment, we will determine if a graph is bicolorable (also referred to as bipartite). If a graph is bicolorable, it means that each vertex can be assigned one of two colors such that no two adjacent vertices (i.e. two vertices with an edge connecting them) have the same color.

More information on bicoloring and bipartite graphs can be found here:
https://en.wikipedia.org/wiki/Bipartite_graph .

For this assignment, you will implement algorithms based on Breadth and Depth First Searches to:

1.) Determine the connected components of an undirected graph, using **Depth First Search**.

2.) Determine if the graph is bipartite (bicolorable), using **Breadth First Search**.

As noted above, a graph is bipartite or bicolorable if the vertices of the graph can be assigned labels (e.g. red and black), such that no two adjacent vertices have the same label. Note that if any of the subgraphs (you will need to check each subgraph separately) is not bipartite, then the overall graph is not bipartite.

You may assume that the graph is **undirected and does not contain self-loops (**edges from a vertex to itself).

Implement the following functions in a Graph class, contained in a file named graph.py

- **def __init__ (self, filename**): Creates and returns a reference to a Graph object
- **def add_vertex (self, id)**: Add vertex with "id" to graph. Can assume id not in graph, and that the id will be an integer.
- **def add_edge (self, v1, v2):** v1 and v2 are vertex id's. As this is an undirected graph, add an edge from v1 to v2 and an edge from v2 to v1.
- **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. Each sub list will contain the vertices (in ascending order) in the connected component represented by that list. If a vertex has no edges, it will be in a connected component containing only itself. For testing, you can assume that the vertices will be labelled with integers from 1 to n, where n is the number of vertices. See examples below.
- **def bicolor (self):** returns True if the graph is bicolorable and false otherwise.

You will want additional functions in your Graph and Vertex classes in order to implement the requirements, but you must provide the above functions as specified for testing purposes. Many of the functions you developed for Lab 9 will be useful in this assignment.

SUBMISSION

Upload to PolyLearn, all your .py files containing a class **Graph** (can also contain a Vertex class), that provides the functions described above, and test_cases.

Example 1:

Input file:

```
1 2
1 3
1 4
1 5
6 7
7 9
9 8
8 6
```

```
g1.conn_components() returns: [[1, 2, 3, 4, 5], [6, 7, 8, 9]]
g1.bicolor() returns: True
```

Example 2:

Input:

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

```
g2.conn_components() returns: [[1, 2, 3], [4, 6, 7, 8]]
g2.bicolor() returns: False
```