

Goals:

1. To become proficient with a graph data structure and graph algorithms
2. To implement a program to find articulation points and biconnected components

What to do:

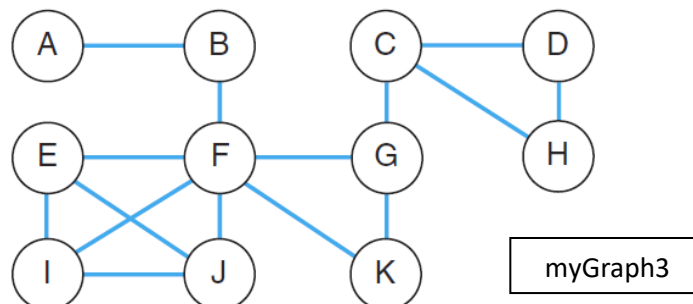
In Java, implement an **$O(V+E)$ algorithm** for finding the **articulation points** and **biconnected components** of a **connected undirected graph**, where V is the number of vertices (nodes) and E is the number of edges. You must print out the following information as you find the articulation points and biconnected components:

- 1) The **depth first index** of each node: this is the location where you first process (i.e. expand) a node in a depth first search tree. Use a depth of 1 to indicate the root. **Do not explicitly build the depth first search tree.**
- 2) The **back link** of each node: this helps you determine whether a node is an articulation point. Do not print the back link if it is just to the parent in the search tree (there will be lots of these). Do print all other back links, as the node name to which they point.
- 3) A **leaf**, whenever one is found.
- 4) An **articulation point**, as it is found. If it is also a root, indicate this in your discovery message by saying that it is an **articulation point at a root**. **Do not print articulation points multiple times (even if they are “discovered” again).**
- 5) All biconnected components, as they are found, specified as a list of nodes.

I have set up a testing file for you to use: `RHgraphTest.java`. It is not thorough, but it will be sufficient for this assignment. When submitting your assignment for grading, **also submit (as a file on eClass) the results of running your program with my test data.**

You may not **import** any Java libraries, except **LinkedList, ArrayList, and ArrayDeque**. Other libraries might be fine, but check with the instructor before including them. Note that ArrayDeque will give you a stack (Java's Stack library is deprecated.)

Here is an example of what your program should do. Suppose that we have this graph¹:



¹ From McConnell, Analysis of Algorithms (as are other 2 graphs on last page)

This graph will be created in Java as:

```
Graph2017 myGraph3 = new Graph2017(11); //constructor takes number of nodes
myGraph3.addEdge(0, 1);
myGraph3.addEdge(1, 5);
myGraph3.addEdge(2, 3);
myGraph3.addEdge(2, 6);
myGraph3.addEdge(2, 7);
myGraph3.addEdge(3, 7);
myGraph3.addEdge(4, 5);
myGraph3.addEdge(4, 8);
myGraph3.addEdge(4, 9);
myGraph3.addEdge(5, 6);
myGraph3.addEdge(5, 8);
myGraph3.addEdge(5, 9);
myGraph3.addEdge(5, 10);
myGraph3.addEdge(6, 10);
myGraph3.addEdge(8, 9);

myGraph3.setLabel(0, "A");
myGraph3.setLabel(1, "B");
myGraph3.setLabel(2, "C");
myGraph3.setLabel(3, "D");
myGraph3.setLabel(4, "E");
myGraph3.setLabel(5, "F");
myGraph3.setLabel(6, "G");
myGraph3.setLabel(7, "H");
myGraph3.setLabel(8, "I");
myGraph3.setLabel(9, "J");
myGraph3.setLabel(10, "K");
```



When you run `myGraph3.biconnect(6)`, this is the output that is expected:


```
=====
====Biconnected components starting at G
Depth First Index of G is 1
Depth First Index of C is 2
Depth First Index of D is 3
Depth First Index of H is 4
    Set backlink of H to C
    Found a leaf: H

APAPAPAPAPAPAPAPAPAPAPAPAPAPAPAPAPAPAPAPAPAPAPAP
AP    Found articulation point: C
APAPAPAPAPAPAPAPAPAPAPAPAPAPAPAPAPAPAPAPAPAPAPAP

BCBCBCBCBCBCBCBCBCBCBCBCBCBCBCBCBCBCBCBCBCBCBCBC
BC    Found bi component: H D C
BCBCBCBCBCBCBCBCBCBCBCBCBCBCBCBCBCBCBCBCBCBCBCBC

APAPAPAPAPAPAPAPAPAPAPAPAPAPAPAPAPAPAPAPAPAPAPAP
AP    Found articulation point at ROOT: G
APAPAPAPAPAPAPAPAPAPAPAPAPAPAPAPAPAPAPAPAPAPAPAP
```


2) General Outline

- a) Set up a graph (either use adjacency matrix or list – either is equally good for this assignment). You must call your graph class **“Graph2017”**. The constructor must take the number of nodes in the graph as the only parameter. You must have a method to add an edge, called **“addEdge”**, which takes two integer parameters and creates an edge between the two nodes corresponding to these integers. Furthermore, you must be able to associate a label with each vertex. This method must be called **“setLabel”**, and is used as follows: **setLabel(0, "A")** associates the label “A” with node 0. Your method to find the articulation points and biconnected components must be part of this graph class, and it must be called **“biConnect”**. This method takes a single node number as the starting point for the search.
 - b) Also consider having a separate (or nested private) node class. 
 - c) Figure out how to do depth first search on this graph. Print out the depth index.
 - d) Draw a diagram (by hand, on paper) of the depth first search, along with the back edges as they would appear when the first articulation point is discovered.
 - e) Add in the back links (for back edges) to your code.
 - f) Determine what characteristic between a node’s depth first index and its neighbors’ back edges make an articulation point. Add this to your code. You will need special cases for leaves, roots, and nodes which have not been traversed before.
 - g) Probably use a stack to be able to print the biconnected components.
- 3) Do not assume that anything you read on the topic of articulation points and biconnected components is correct (including what is in your text ☺). To prepare for this topic, I checked three textbooks plus many internet sites, and they all seemed to have some errors.
- 4) **Start today**. This is the big culminating project for AUCSC 310. I think about 150 – 200 lines of code, with documentation being extra.

Hand in:

- 1) Hard copy of your program (with documentation). If you added onto someone else’s code, **make it very clear what portion you coded**, and properly credit the author of the other portion. Make sure that almost all of the code is your own!
- 2) Your testing results, when run on my testing file, on eClass – file only (no hard copy).
- 3) Your program on eClass.

I will mark all of the following:

- 1) Java Program Approach and Completeness (30%) Is your code relatively minimal, with no unnecessary stuff? Does your code clearly show what is happening? Have you considered all of the cases? Did you submit testing results? Do you divide the problem up into reasonable steps/methods? Did you create

complete data types (e.g. node or graph)? If your program contains run-time or compile-time errors, the maximum mark possible is 50%.

- 2) Documentation (30%) You NEED to have file headers (containing your name, date, and summary of how the file is organized), method headers (describing the purpose of each method and what each parameter is), descriptive variable names, proper indentation, parenthesis marking over blocks, and inline comments that add to the program (no inline comments that just repeat code, please; rather explain “why” you do something). Your documentation must show that you understand what you are doing, at an “English-level.” Do not make any lines in your program so long, that they wrap to the next line without proper indentation. If your program is too hard to read, the maximum mark possible for this assignment is 50%.
- 3) Correctness (40%). If your code runs in a higher order of magnitude (i.e. not in $O(V+E)$), the maximum mark possible for this assignment is 60%.

Here are diagrams of the other two graphs found in the testing file:

