CS 312: Project 3 - Articulation Points
Duane Johnson

1. Pseudo Code (Taken from HW 9 Key)

```
function findArticulationPoints(graph) {
      node start
      previsit(start);
      stack.push(start);
      while (!stack.empty()) {
            curr = stack.top();
            while ( exists unvisited edges adjacent to curr) {
                  adjacent = curr.next_neigbor();
                  if (adjacent.visited == false) {
                        //(curr, adjacent) is a tree edge
                        if (curr.parent == null) {
                              //Count the number of times we
                              // are at the root node
                              root_visits++;
                        }
                        previsit(adjacent, curr);
                        stack.push(adjacent)
                        curr = adjacent;
                  } else if (adjacent != curr.parent) {
                        //(curr, adjacent) is a back edge
                        curr.low = min(curr.low, adjacent.pre);
                  }
            }
            adjacent = stack.pop();
            curr = adjacent.parent; //Should be same as stack.top()
            postvisit(adjacent);
      }
}
function previsit(node, parent = null) {
      node.parent = parent;
      node.pre = node.low = count++;
      node.visited = true;
}
function postvisit(node) {
      if (node.parent != null) {
            node.parent.low = min(node.parent.low, node.low);
      }
}
```

```
function findBiconnectedComponents(graph, start)
{
      // stack setup
      set visited to empty
      add start to visited
      push start onto vertex-stack
      while vertex-stack is not empty
      {
            set parent to vertex-stack peek
            set child to parent's nextNeighbor
            if child is not null
            {
                  if child is not in visited
                  // we will be pushing on the child
                  {
                        add child to visited
                        if (parent is a Separating Vertex
                              and parent.pre <= child.low)
                              push a mark onto edge-stack
                        push child onto vertex-stack
                  }
                  remove parent from child's neighbor list
                  push edge parent-child onto edge-stack
                  increment parent's nextNeighbor
            }
            else // child was null so we are done with this parent
            {
                  set child to parent
                  pop from vertex-stack // this is popping the child
                  set parent to vertex-stack peek
                  if (parent is null // this was the end of the DFS
                        or (parent is a Separating Vertex
                        and parent.pre <= child.low))
                  {
                        create a new Biconnected Component list BCC
                        while edge-stack is not empty
                              and edge-stack peek is not a mark
                        {
                              pop edge-stack into BCC
                        }
                        if edge-stack is not empty
                        {
                              pop mark off edge-stack
                        }
                  }
            }
      }
}
```

2. How I Implemented the Core Search Algorithm

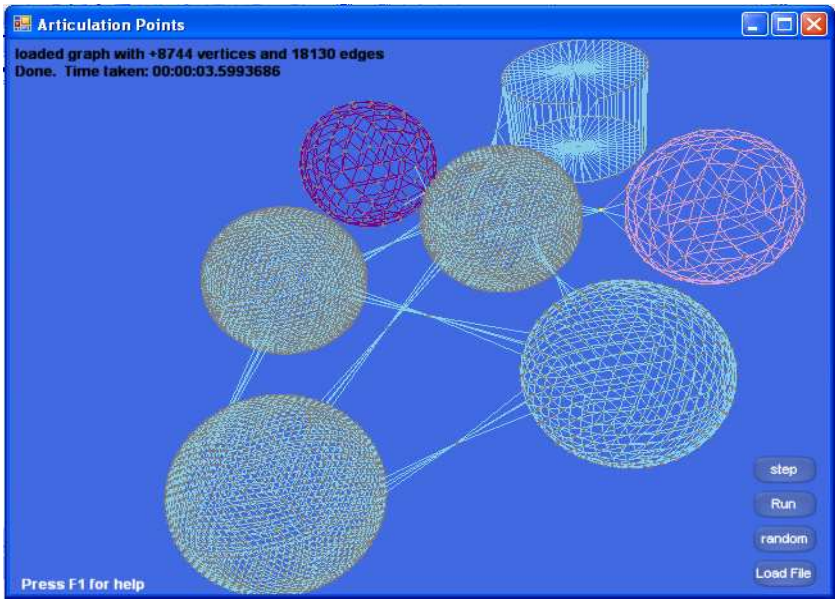
I used an iterative process, as suggested by the HW 9 key. One area that differs in my implementation is the use of a HashSet to record visited ("considered") nodes. This is a very fast way of keeping the information available inside the iterative loop without embedding it into the Node objects themselves. A HashSet lookup is an O(1) operation.

Using this "considered" HashSet, the code avoids expanding the same vertex more than once.

3. Using the following test cases, the algorithm I implemented was sufficiently fast for each requirement:

Test Case	Timing #1	Timing #2	Average Timing	Max. Timing
11.x3d	0.614	0.556	0.585	4
12.x3d	5.843	5.794	5.819	13
13.x3d	2.983	2.860	2.922	9
14.x3d	5.587	5.797	5.692	10
15.x3d	3.389	3.332	3.361	7
16.x3d	3.649	3.734	3.692	7

4. Screenshot



```
loaded graph with +8744 vertices and 18130 edges
Done.  Time taken: 00:00:03.5993686
```