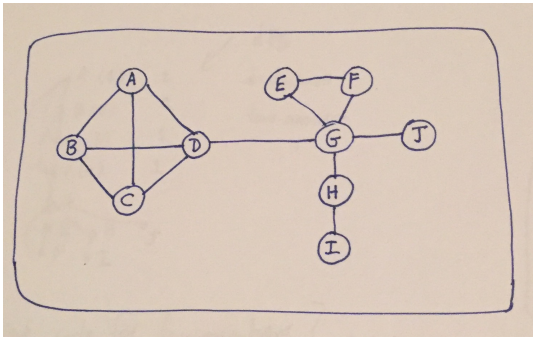
# CS:3330 Homework 4, Fall 2015

## Due in class on Tue, Nov 3

1. Let $G = (V, E)$ be a connected, undirected graph. An *articulation point* of $G$ is a vertex in $G$ whose removal disconnects $G$. For example, in the graph below, if we delete the vertex $D$, the graph that remains has two connected components, one with vertex set $\{A, B, C\}$ and the other with vertex set $\{E, F, G, H, I, J\}$.

   (a) Identify all articulation points in the following graph.




   (b) Let $T$ be a *depth-first search (DFS) tree* of $G$, rooted at a vertex $s \in V$. Prove that $s$ is an articulation point if and only if $s$ has at least two children in $T$.

   (c) Let $T$ be a DFS tree of $G$, rooted at a vertex $s \in V$. For any vertex $v \neq s$, prove that $v$ is an articulation point of $G$ if and only if $v$ has a child $w$ such that there is no non-tree edge from the subtree of $T$ rooted at $w$ to a proper ancestor of $v$.

   **Note:** To understand this claim take a look at the DFS tree in Fig 3.5(g) (Page 85) in the textbook. Vertex 3 has two children: vertex 5 and vertex 7. The subtree of the DFS tree rooted at vertex 5 contains vertices 4, 5, and 6. There are two non-tree edges from this subtree leading back to a proper ancestor of vertex 3, namely vertex 2. The subtree of the DFS tree rooted at vertex 7 contains vertices 7 and 8. There is a non-tree edge from this subtree to vertex 3, but not to a proper ancestor of vertex 3. Hence vertex 3 has a child (vertex 7) such that there is no non-tree edge from the subtree rooted at vertex 7 leading back to a proper ancestor of vertex 3. Therefore, according to the above claim, vertex 3 is an articulation point. This is easy to see because deletion of vertex 3 will separate vertices 7 and 8 from the rest of the graph.

2. The two claims stated in Problem 1, parts (b) and (c) above can be be used to obtain an efficient algorithm (based on DFS) for finding all articulation points in a graph. Consider the following recursive implementation of the DFS algorithm.

```
DFS(u):
  Explored[u] ← True
  time ← time + 1; D[u] ← time
  for each neighbor v of u do
      if not Explored[v] then
          DFS(v)
```

   This is essentially the pseudocode in the textbook (Page 84) except that I have added an array `D` that keeps track of the "time" at which each vertex is first visited by DFS.

Assume that the variable `time` that you see in the above pseudocode is a global variable that is initialized to 0 before DFS is called. Also assume that the array `Explored` has been initialized to all `False` values.

(a) For each vertex $v \in V$ define `low`$[v]$ as

$$\min\left(\{D[v]\} \cup \{D[w] \mid (u,w) \text{ is a non-tree edge for some descendent } u \text{ of } v\}\right).$$

For the graph provided for Problem 1, draw a DFS tree rooted at vertex $A$ (in the style of Fig 3.5(g)). For each vertex, show its `D`-value and `low`-value and also show all non-tree edges of $G$ with respect to this DFS tree.

(b) Modify the DFS function to compute the `low`-values for all vertices $v$ in $O(m)$ time. (Note that since $G$ is connected, $m \geq n-1$.)
**Hint:** The recursive structure of the DFS algorithm makes it easy to do this. Once control returns from all the recursive calls to `DFS(`$v$`)` at children $v$, then we can use the `low`-values computed for children $v$ to compute the `low`-value at $u$.

(c) Further modify the DFS function with `low`-value computation to show how to identify all articulation points in $G$ in $O(m)$ time.

3. Problem 11 in Chapter 3 (Pages 111-112). Make sure that you present an arument showing why your algorithm runs in $O(m+n)$ time.

4. Consider the problem of making change for $n$ cents (for positive integer $n$) using the fewest number of coins. Describe a *greedy* algorithm that uses quarters, dimes, nickels, and pennies to make change.

   Now comes the more interesting question: how do you *prove* that your algorithm yields an optimal solution? The next few parts of this problem will lead you through a proof.

   (a) Let $O$ be optimal change for $n$. In other words, $O$ is a smallest possible set of coins using quarters, dimes, nickels, and pennies, whose total value is $n$. We will first prove the following claim:

   > **Claim:** $O$ contains exactly as many quarters as produced by the greedy algorithm.

   To prove this claim, answer the following questions: what is the maximum number of pennies that $O$ can contain? what is the maximum number of nickels that $O$ can contain? what is the maximum number of dimes that $O$ can contain? what are all the possible combinations of dimes and nickels that $O$ can contain? Using the answers to these questions, determine the maximum value of dimes, nickels, and pennies in $O$. Use this value to prove the claim.

   (b) Use the above strategy to prove that $O$ contains exactly as many dimes as produced by the greedy algorithm and then exactly as many nickels as produced by the greedy algorithm.

5. Problem 6 in Chapter 4 (Page 191).

6. Problem 17 in Chapter 4 (Page 197).

7. Problem 18 in Chapter 4 (Pages 197-198).

---