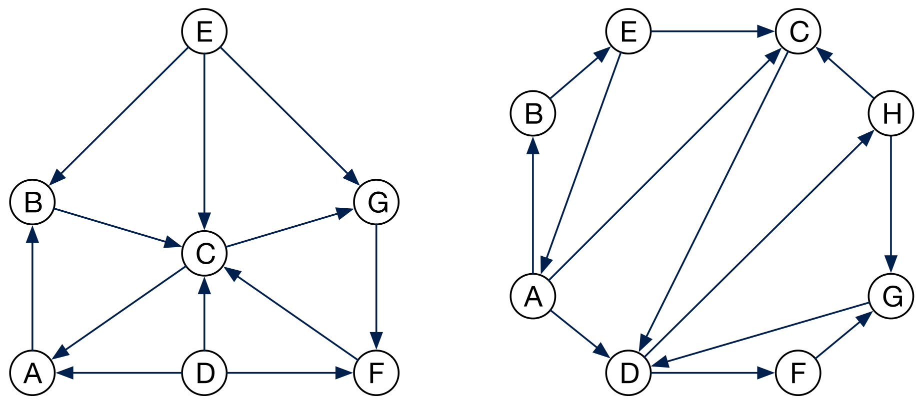
Assignment #8

Due: April 4, 2022 (11:59 pm)
(Total 40 marks)

CMPUT 204
Department of Computing Science
University of Alberta

Problem 1. (10 marks)

For the directed graphs given below, perform breath-first search (BFS) and depth-first search (DFS) on the graph using vertex A as a start vertex; whenever there is a choice of vertices, pick the one that is alphabetically first. That is, assume that each adjacency list is ordered alphabetically. Draw the resulting BFS/DFS tree, and classify each edge as a tree edge, forward edge, back edge, or cross edge. For DFS tree, you also need to show the discovery and finish time (i.e., $[dtime, ftime]$) of each node.



(a) (b)

Sol: The BFS and DFS forests of (a) and (b) are shown in figures 1-4.
(a):

- BFS: Back edges: (C, A), (F, C). Cross edges: $(D, A), (D, C), (D, F), (E, B), (E, C), (E, G)$.
- DFS: Same as BFS.

(b):

- BFS: Back edges: $(E, A), (G, D)$. Cross edges: $(E, C), (H, C), (H, G), (C, D)$.
- DFS: Forward edges: $(A, D), (A, C)$. Back edges: $(E, A), (G, D), (H, C)$. Cross edges: (H, G).

Problem 2. (10 marks)

Show how to find the strongly connected components (SCCs) of each graph given in Problem 1. Specifically, based on your DFS trees, do the following:

- Show the forest of $G^\intercal$ by traversing nodes in a decreasing $ftime$ in the main DFS loop. (Note that $G^\intercal$ denotes the graph after flipping the direction of each edge in the original graph G)
- Draw the component graph to illustrate the SCCs of the graph, and then given a topological sorting of the component graph.

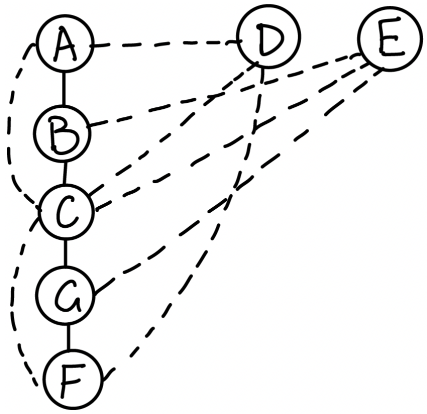


Figure 1: BFS forest of 1(a).

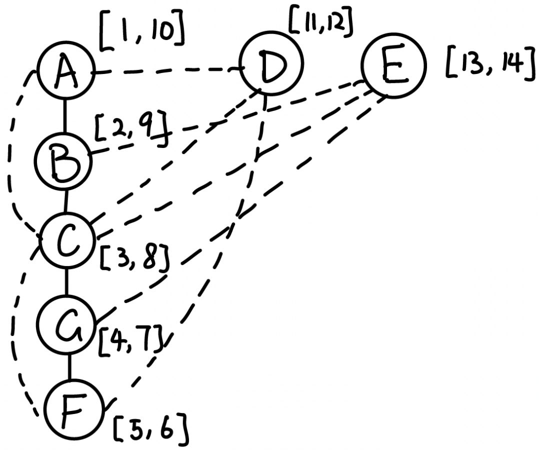


Figure 2: DFS forest of 1(a).

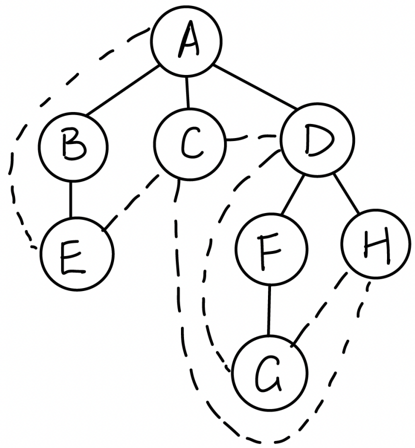


Figure 3: BFS forest of 1(b).

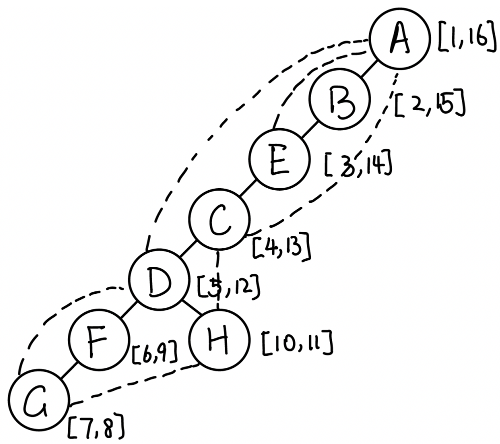


Figure 4: DFS forest of 1(b).

Sol: The forests of 1(a) and 1(b) are shown in figures 5-6.

The component graphs are shown in figures 7-8. The topological sorting of the first component graph is SCC(E), SCC(D), SCC(A); or, SCC(D), SCC(E), SCC(A)

Problem 3. (20 marks)

Cut Vertices: We define a cut vertex as a node that when removed causes a connected graph to become disconnected, namely, the number of connected components in the graph increases once a cut vertex is removed. For this problem, we will try to find the cut vertices in an undirected graph G with n nodes and m edges.

a. How can we efficiently check whether or not a graph is disconnected?

   Sol: Run BFS or DFS on any node in G (since G is undirected, any node will do). If the size of the set of visited nodes does not equal $|V|$, then the graph is disconnected.

b. Describe an algorithm that uses a brute force approach to find all the cut vertices in G in $O(n(n+m))$ time. You do not need to write the pseudo code.

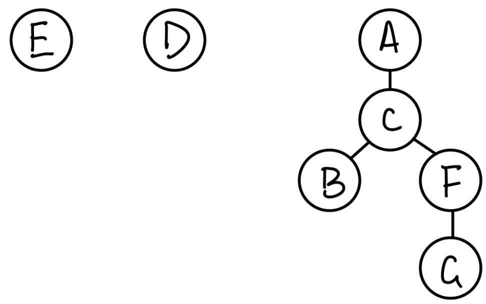


Figure 5: DFS Forest of 1(a).

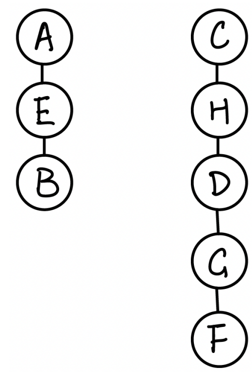


Figure 6: DFS forest of 1(b).

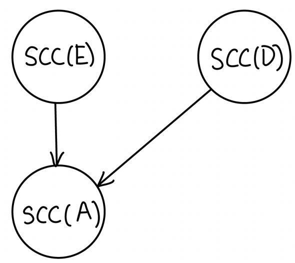


Figure 7: Component graph of 1(a).

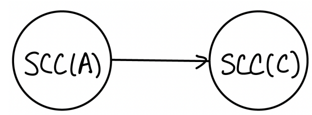


Figure 8: Component graph of 1(b).

Sol: For each vertex v in G, remove v and all the edges connected to v. Run the algorithm from part a. If the graph G is disconnected, then v is a cut vertex. Add v and all its edges back into G and repeat.

c. Draw two DFS trees starting from vertex C and F (again, whenever there is a choice of vertices, pick the one that is alphabetically first). Indicate non-tree edges using dotted links.

Sol: The DFS trees are shown in figures 10-11.

d. Based on the given DFS tree (starting from vertex A) in Figure 9(b) and the two DFS trees in part c (starting from vertex C, F), prove that i) the *root* of a DFS tree is a cut vertex if and only if it has at least two children, and ii) the *leaf* of a DFS tree is never a cut vertex.

Sol: The second claim is obvious. Here we prove the *if and only if* in both directions for the first claim.

First suppose the root r of DFS(G) is a cut vertex. Then the removal of r from G would cause the graph to disconnect, so r has at least two children in DFS(G).

Now suppose r has two children in DFS(G), we prove r must be a cut vertex. Since DFS has no cross edge in undirected graphs, there exists no edge from one subtree to the other. Therefore, removal of r will result in disconnecting the two subtrees rooted at the two children of r. The case with more than two children follows similarly.

e. In a DFS tree, other than the root and leaf nodes, we also have non-root, non-leaf vertices. We argue that any non-root, non-leaf vertex u is a cut vertex *if and only if* there exists a subtree rooted at a child of u that has no back edges to any *ancestor* of u. In other words, there exists some child y of u such that no back edge connects y (or any vertex in the subtree rooted at y, i.e., below y) to any node above u (i.e., ancestor of u). Note that the claim is an *if and only if* statement, so we need to

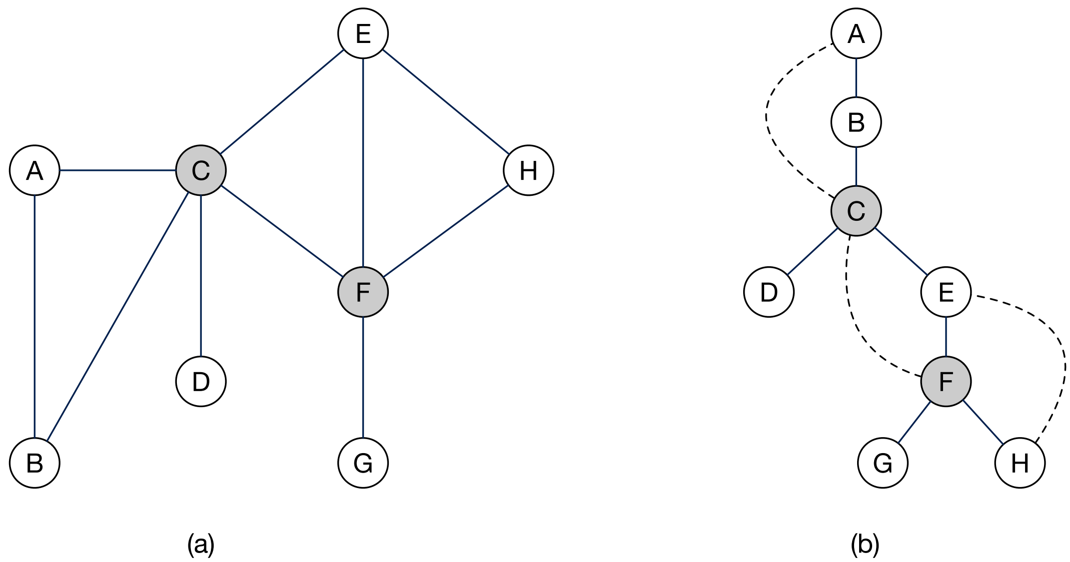


Figure 9: A cut vertex is any vertex whose removal will increases the number of connected components. In subfigure (a), the two gray nodes (i.e., C and F) are cut vertices. Subfigure (b) illustrates the DFS tree of the left graph starting from vertex A, where non-tree edges are shown as dotted lines.

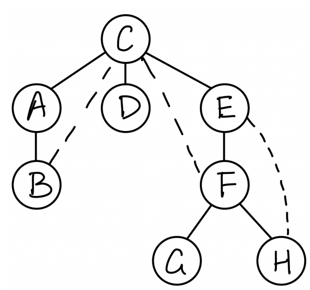


Figure 10: Component graph of 1(a).

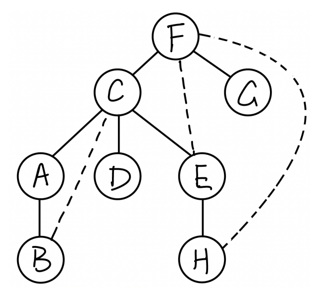


Figure 11: Component graph of 1(b).

prove it in both directions. Please follow the partial proof given below to complete the proof of both directions.

Claim 1: If u is a cut vertex, then there exists a subtree rooted at a child of u that has no back edges to any ancestor of u.

Proof of Claim 1: Assume for the sake of contradiction that all the subtrees have back edges to some ancestor of u. If we remove u ...

Sol: If we remove u, every subtree of u can still reach an ancestor of u through the back edge, meaning that the graph is still connected. This implies that u cannot be a cut vertex (by the definition of cut vertices), so our initial assumption is violated. Therefore, there must be a subtree with no back edge to any ancestor of u.

Claim 2: If there exists a subtree rooted at a child of u that has no back edges to any ancestor of u, then u is a cut vertex.

Proof of Claim 2: Note that a DFS on an undirected graph only produces tree edges and forward/back edges, no cross edges...

Sol: So there is no path from one subtree rooted at a child of u to another subtree rooted at a child of u, nor a path to a vertex that is neither an ancestor or descendent of u. Since the assumption in Claim 2 says there are no back edges either, then there must only be tree edges. If we remove u, then the subtree that has no back edges will be disconnected since we are effectively removing the connecting tree edge. Hence, u is a cut vertex.

Comment: The idea in part d and e can be extended to yield an $O(n + m)$ algorithm to find all cut vertices, improving upon the $O(n(n + m))$ brute force algorithm you proposed in part b. But we will not require you to design the algorithm for this assignment.