CS/ECE 374 FALL 2018
Homework 7 Problem 3

Zhe Zhang (zzhan157@illinois.edu)
Ray Ying (xinruiy2@illinois.edu)
Anqi Yao (anqiyao2@illinois.edu)

Given an *undirected* connected graph $G = (V, E)$ an edge (u, v) is called a cut edge or a bridge if removing it from G results in two connected components (which means that u is in one component and v in the other). The goal in this problem is to design an efficient algorithm to find *all* the cut-edges of a graph.

- What are the cut-edges in the graph shown in the figure?
- Given G and edge $e = (u, v)$ describe a linear-time algorithm that checks whether e is a cut-edge or not. What is the running time to find all cut-edges by trying your algorithm for each edge? No proofs necessary for this part.
- Consider *any* spanning tree T for G. Prove that every cut-edge must belong to T. Conclude that there can be at most $(n-1)$ cut-edges in a given graph. How does this information improve the algorithm to find all cut-edges from the one in the previous step?
- Suppose T is any spanning tree of G. Root it at some arbitrary node. Prove that an edge (u, v) in T where u is the parent of v is a cut-edge iff there is no edge in G, other than (u, v), with one end point in T_v (sub-tree of T rooted at v) and one end point outside T_v.
- Now consider the DFS tree T. Use the property in the preceding part to design a linear-time algorithm that outputs all the cut-edges of G. What additional information can you maintain while running DFS? Recall that there are no cross-edges in a DFS tree T. You don't have to prove the correctness of the algorithm.

Solution: (1) (e,g), (f,j) and (h,l) are the cut-edges in the graph.

(2) The edge(u,v) is called a cut edge or a bridge if removing it from the connected graph G results in two connected components. To check whether the edge e=(u,v) is a cut-edge, we could instead check whether u and v are disconnected after removing edge e. If u and v are disconnected then edge e=(u,v) is a cut-edge. Checking the connectivity starting at u via basic search requires O(n+m) and there are total m edges in the graph G. Thus the total running time is O(m(m+n)) = O(m^2).

(3) Let e=(u,v) be an cut-edge that does not belong to the spanning tree T. By the definition of the cut-edge, removing e from the connected graph G results in two connected components. In other words, u and v will be disconnected. However, there is always a path from u to v in the spanning tree. Since there is a contradiction, we've proved that every cut-edge must belong to T. By definition, a spanning tree of a graph on n vertices is a subset of n-1 edges that form a tree. Since we've proved that every cut-edge must belong to T and there are n-1 edges in T, there can be at most (n-1) cut-edges in a given graph. Knowing this information, we do not need to check all those m edges. We only need to check all these n-1 edges in the spanning tree. Thus the toal running time is O((n-1)(m+n)) = O(mn).

(4) Let A denotes an edge (u, v) in T where u is the parent of v is a cut-edge.

Let B denotes there is no edge in G, other than (u, v), with one end point in T_v (sub-tree of T rooted at v) and one end point outside T_v.
First we prove A ⇒ B and let's prove by contrapostive
Suppose there is a edge e in G, other than (u,v), with one end point in T_v and one end point outside T_v. Removing the edge (u,v) results in two connected components and we denotes these two components T_u and T_v where T_u contains u and T_v contains v. Adding the edge e results in one connected component and we denotes it as T'. Since T' is a spanning tree of G and every cut-edge must belong to T'(part 2), (u,v) cannot be a cut-edge.
Then we prove B ⇒ A
Since there is no edge other than (u,v), with one end point in T_v and one end point outside T_v. Adding the edge (u,v) results in one connected component and we denotes it as T'. Since T' is a spanning tree of G and (u,v) is on T', (u,v) is a cut-edge.
Since both A ⇒ B and B ⇒ A, we could conclude that an edge (u, v) in T where u is the parent of v is a cut-edge iff there is no edge in G, other than (u, v), with one end point in T_v (sub-tree of T rooted at v) and one end point outside T_v.

(5) The basic idea for question 5 is based on question 4. When giving a spanning tree, at every node, we want to find if their children connect to it through other path without the directed edge. Since, if you cut an edge in a tree, we disconnects the tree in two piece. So like preceding part, when we want to consider if an edge is a cut-edge in spanning tree T, we see if any node in children subtree have an edge in the original graph to the node parent subtree. Using DFS to traverse the spanning tree from the root, at each node, we go to their children nodes. At each step, if there is an edge in the original graph but not in the spanning tree that from the node to a node in its ancestor level(the level of the node it connects to are lower than the node itself(root is in level o and so on)), there won't be a cut-edge in the path from one node to the other, we can then put all edges in this path into a set S. Since when doing DFS, we will go through each node exactly once and for each node we only have to look at the connected edges in the original graph. The left edges that are not contains in S will be the cut-edge. The running time will be $O(|V| + |E|)$ where E is the number of edges in the original graph not the tree. ■