Lecture Notes: Computing Cut Vertices, Bridges, and Biconnected Components using Depth First Search

Palash Dey
palash.dey@cse.iitkgp.ac.in
Indian Institute of Technology Kharagpur

June 6, 2026

In this article, we will discuss some applications of depth first search (DFS) in graphs. We will assume familiarity with DFS. In particular, we assume that the reader knows how the colors of the vertices change from white to gray to black during DFS, how the discovery time stamp $v.d$ and finish time stamp $v.f$ are computed for every vertex $v \in V[G]$ during DFS and the significance of these numbers.

1 Finding Cut Vertices or Articulation Points and Bridges

Let $G(V, E)$ be a connected undirected graph. If not mentioned otherwise, we will assume that it is represented as an adjacency list when given as input to an algorithm. Recall that, since G is an undirected graph, every edge of G is either a tree edge or a back edge during the DFS traversal of G. There cannot be any forward edge or cross edge.

A vertex $v \in V$ is a **cut vertex or an articulation point** of G if $G \setminus \{v\}$ is disconnected. We now discuss a DFS based algorithm for computing all the cut vertices in time $\mathcal{O}(m + n)$ time. Naively, we can remove every vertex and check if the graph remains connected after the removal of the vertex. This naive algorithm has worst-case time complexity $\mathcal{O}(n(m + n))$. The following lemma is easy to see and prove.

Lemma 1 *1. The root of a DFS tree is a cut vertex if and only if it has at least two children.*

2. A non-root node v in a DFS tree is a cut vertex if and only if it has a child u such that there is no back edge between u or any descendant of u and any proper ascendant of v.

In particular, no leaves of any DFS tree is a cut vertex.

To check the conditions of Lemma 1, we will compute a function that we call $u.low$ for every vertex u. The $u.low$ is defined to be the discovery time of the highest ancestor of u, say x such that there is a back edge between x and a descendant of u. If the vertex u does not have any descendant or if there is no back edge between an ancestor of u and any descendant of u, then $u.low$ is defined to be ∞. Hence, by Lemma 1, a vertex u is a cut vertex if and only if u has a descendant v such that $v.low \neq \infty$ and $v.low \geqslant u.d$.

Algorithmically, we can compute $low(v)$ for every vertex v during DFS traversal itself as follows. Before starting the DFS traversal, we initialize the low values of all the vertices to ∞. After we finish the recursive search from a child v of a node u, we update the low value of u as $u.low = \min\{u.low, v.low\}$. The vertex u is a cut vertex disconnecting v if $v.low \geqslant u.d$. Also, if u is a root vertex of any DFS tree, it is a cut vertex if u has a second child. During DFS, when

we encounter a back edge $\{x, y\}$ where x is an ancestor of y, then we update the low value of y as $y.low = \min\{y.low, x.d\}$.

An edge $\{u, v\}$ of G is called a **bridge** if $G \setminus \{u, v\}$ is disconnected. Observe that a bridge can never participate in a cycle and thus every bridge edge must be a tree edge of the DFS tree. The following lemma is easy to see and prove.

Lemma 2 *An edge $\{u, v\}$ is a bridge of G if and only if it is a tree edge, and if v is a child of u, then $v.low > u.d$.*

Hence, we can compute all the bridges of the graph by one DFS traversal by computing the low values of all the vertices and outputting an edge $\{u, v\}$ where u is the parent of v and $v.low > u.d$ when we finish the recursive search from v.

2 Finding Biconnected Components

A subgraph is called **biconnected** if it does not have any cut vertex. A **biconnected component** of an undirected graph G is a maximal subgraph H such that H does not have any cut vertex. Two biconnected components cannot have any common edge. Otherwise, the union of these two biconnected components will also be biconnected contradicting maximality of the initial two biconnected components. However, two biconnected components can have common vertices. These common vertices must necessarily be cut vertices otherwise some of the initial biconnected components will not be maximal.

Algorithmically, we can compute the edges of biconnected components as follows. We store the visited edges, both tree edges and back edges, in a stack. When the recursive search from a vertex v (irrespective of whether v is the root or not) returns to its parent u, then if u is a cut vertex separating v, then we output all the edges from the stack until the edge $\{u, v\}$. These edges form a biconnected component. Additionally, if v is the root of DFS tree, then after outputting all the biconnected components that involve v as per the above rule, we output all the remaining edges from the stack as a biconnected component.