From: Combinatorial Algorithms
by Reingold, Nievergelt & Deo

8.2.3 *Biconnectivity*

Sometimes it is not enough to know that a graph is connected; we may need to know how "well connected" a connected graph is. A connected graph, for example, may contain a vertex whose removal, along with its incident edges, disconnects the remaining vertices. Such a vertex is called an *articulation point* or a *cut-vertex*. A graph that contains an articulation point is called *separable*. Vertices b, f, and i in Figure 8.7(a), for example, are articulation points, and they are the only ones in the graph. A graph with no articulation points is called *biconnected* or *nonseparable*. A maximal biconnected subgraph of a graph is called a *biconnected component* or a *block*. Identification of the articulation points and biconnected components of a given graph is important in the study of the vulnerability of communication and transportation networks. It is also important in determining other properties, like planarity, of a graph G, since it is often advantageous to separate G into its biconnected components and examine each one individually (see Section 8.6).

A vertex v in an undirected connected graph is an articulation point if and only if there exist two other vertices x and y such that every path between x and y passes through v; in this case and only in this case does the deletion of v from G destroy all paths between x and y (i.e., disconnect G). This observation allows us to use depth-first search to find the articulation points and biconnected components of a graph in $O(|V| + |E|)$ operations.

The central idea can be understood by studying the example in Figure 8.8, which shows schematically a connected graph consisting of biconnected components G_i, $1 \leqq i \leqq 9$, and articulation points v_j, $1 \leqq j \leqq 5$. If we start the depth-first search at, say, the vertex s in G_9, we might, perhaps, leave G_9 to go into G_4 by

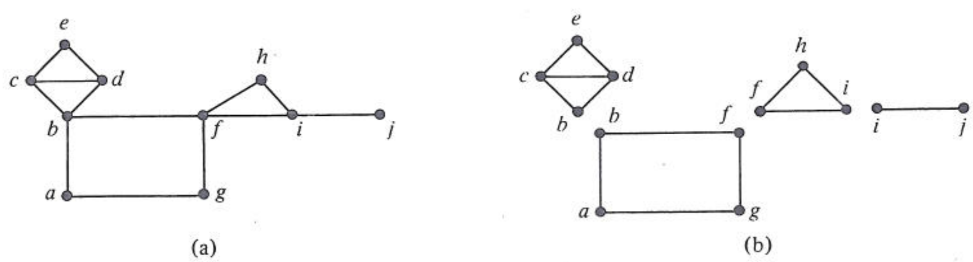

Figure 8.7 A separable graph (a) and its biconnected components (b). The articulation points are b, f, and i.

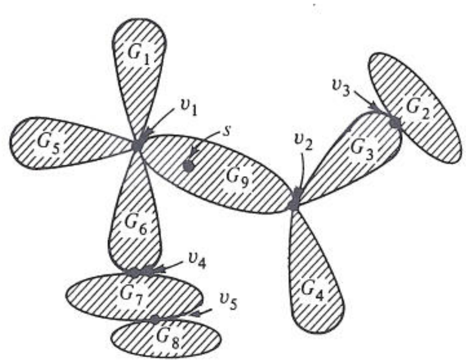

Figure 8.8 A schematic drawing of a graph with nine biconnected components and five articulation points.

passing through v_2. But by the depth-first nature of the search, all the edges in G_4 must be traversed before we back up to v_2; thus G_4 consists of exactly the edges traversed between visits to v_2. Matters are actually a little more complicated for the other biconnected components, since, for example, if we leave G_4 and go into G_3 and from there into G_2 through v_3, we would find ourselves in G_2, having traversed edges from G_9, G_3, and G_2. Fortunately, however, if we store the edges in a stack, by the time we pass through v_3 back into G_3 all the edges of G_2 will be on top of the stack. When they are removed, the edges on top of the stack will be from G_3, and we will once again be traversing G_3. Thus if we can recognize the articulation points, we can determine the biconnected components by applying depth-first search and storing the edges on a stack as they are traversed; the edges on top of the stack as we back up through an articulation point form a biconnected component.

In order to recognize an articulation point we need to compute, during the depth-first search, a new function *lowpt* (v) for every vertex v in the graph. We define *lowpt* (v) as the smallest value of *num* (x), where x is a vertex of the graph that can be reached from v by following a sequence of zero or more tree edges followed by at most one back edge. The function *lowpt* (v) is useful because of the following theorem.

Theorem 8.1

Let $G = (V, E)$ be a connected graph with a DFS-tree T and with back edges B. Then $a \in V$ is an articulation point if and only if there exist vertices $v, w \in V$ such that $(a, v) \in T$, w is not a descendant of v in T and *lowpt* $(v) \geqq$ *num* (a).

Proof: Suppose that such vertices v and w exist. Since $(a, v) \in T$ and *lowpt* $(v) \geqq$ *num* (a), any path starting at v that does not go through a must remain in the subtree with root v. Since w is not a descendant of v in T, such a path cannot contain w. Thus the only paths from v to w contain a and so a is an articulation point.

Conversely, suppose that a is an articulation point. If a is the root of T, then at least two edges of T start at a; otherwise there would be a path in G between every pair of vertices in $V - \{a\}$ that did not contain a. Let (a, v) and (a, w) be two of these edges; clearly, v and w satisfy the theorem. If a is not the root of T, then it has an ancestor w. One of the biconnected components containing a has all its nodes as descendants of a in T; in fact, they are all (except a) descendants of a vertex v; where (a, v) is an edge in T (why?). Clearly, v and w satisfy the theorem.

This theorem tells how to recognize articulation points if we have the values of *num* and *lowpt*: if we find a vertex v such that $(a, v) \in T$ and *lowpt* $(v) \geqq$ *num* (a), then a is either an articulation point or the root of T. This result follows from Theorem 8.1 by observing that a suitable w can be chosen among the ancestors of a if a is not the root. Furthermore, computing the *lowpt* values during depth-first search is simple because

$$lowpt\,(v) = \min(\{num\,(v)\} \cup \{lowpt\,(x) | (v, x) \in T\} \cup \{num\,(x) | (v, x) \in B\}).$$

Thus Algorithm 8.5 determines the biconnected components of a graph $G = (V, E)$. Since the algorithm is a depth-first search with a constant amount of extra work done as each edge is traversed, the time required is clearly $O(|V| + |E|)$. A proof that the algorithm works correctly is left as Exercise 20.

```
i ← 0
S ← empty stack
for x ∈ V do num (x) ← 0
for x ∈ V do if num (x) = 0 then BICON (x, 0)
procedure BICON (v, u)
    i ← i + 1
    num (v) ← i
    lowpt (v) ← i
    for w ∈ Adj (v) do if num (w) = 0 then {
            ⟦(v, w) is a tree edge⟧
            S ⇐ (v, w)
            BICON (w, v)
            lowpt (v) ← min (lowpt (v), lowpt (w))
            if lowpt (w) ≧ num (v) then {
                At this point v is either
                the root of the tree or it
                is an articulation point.
                Form a new biconnected
                component consisting of all
                the edges on the stack above
                and including (v, w). Remove
                these edges from the stack.
            }
        }
        else if num (w) < num (v) and w ≠ u then {
            ⟦(v, w) is a back edge⟧
            S ⇐ (v, w)
            lowpt (v) ← min (lowpt (v), num (w))
        }
    return
```

Algorithm 8.5 Determining the biconnected components of $G = (V, E)$.