A NOTE ON FINDING THE BRIDGES OF A GRAPH

by

R. Endre Tarjan

Memorandum No. ERL-M427

February 1974

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

A NOTE ON FINDING THE BRIDGES OF A GRAPH

R. Endre Tarjan

Department of Electrical Engineering and Computer Sciences
and the Electronics Research Laboratory
University of California, Berkeley, California 94720

February, 1974

ABSTRACT

Recently, algorithms have been developed which use depth-first search to efficiently test some connectivity properties of graphs. Depth-first search is not always necessary for efficiently testing such properties, however. This note presents an efficient algorithm which uses any search method to find all the bridges of a graph.

Keywords: algorithm, bridge, connectivity, search, spanning tree.

This research was partially supported by the National Science Foundation Grant GJ-35604X1.

A NOTE ON FINDING THE BRIDGES OF A GRAPH

R. Endre Tarjan

Recently, algorithms have been developed which use depth-first search to efficiently test various connectivity properties of graphs. Examples include algorithms to find the connected, biconnected, and triconnected components of an undirected graph [1,2,3] and the strongly connected components [2] and dominators [4] of a directed graph. Depth-first search is not always necessary for efficiently testing connectivity properties like these, however. This note presents an efficient algorithm which uses any search method to find all the bridges of a graph.

A graph $G = (\mathcal{V},\mathcal{E})$ is a set of vertices $\mathcal{V}$ and a set of edges $\mathcal{E}$. The edges are either unordered pairs (v,w) of distinct vertices (the graph is undirected) or ordered pairs (v,w) of distinct vertices (the graph is directed). We denote the number of vertices by V and the number of edges by E. Graph $G_1 = (\mathcal{V}_1,\mathcal{E}_1)$ is a subgraph of G if $\mathcal{V}_1 \subseteq \mathcal{V}$ and $\mathcal{E}_1 \subseteq \mathcal{E}$. A sequence of edges $(v_1, v_2), (v_2, v_3), \ldots (v_{n-1}, v_n)$ is a path from v_1 to v_n. A path is simple if all its vertices are distinct. There is a path of no edges from any vertex to itself. An undirected graph is connected if there is a path between every pair of vertices. If there is a path from a vertex v to a vertex w in G but every path from v to w contains edge e, then e is said to be a bridge of G. An undirected graph is bridge-connected if it is connected and has no bridges. The connected (bridge-connected) components of a graph are its maximal connected (bridge-connected) subgraphs.

A _tree_ is an undirected graph with exactly one simple path between every pair of distinct vertices. A _spanning tree_ of a graph is a subgraph which is a tree and which contains every vertex of the graph. A _directed, rooted tree_ is a directed graph with a unique root such that there is a unique path from the root to any other vertex in the tree. We denote the existence of an edge (v,w) in a directed, rooted tree by $v \rightarrow w$ and the existence of a path from v to w in a directed, rooted tree by $v \overset{*}{\rightarrow} w$. If $v \rightarrow w$, v is the _father_ of w and w is a _son_ of v. If $v \overset{*}{\rightarrow} w$, v is an _ancestor_ of w and w is a _descendant_ of v.

We wish to find all the bridges of an undirected graph G. Without loss of generality we may assume that G is connected; otherwise we can apply the procedure below to each connected component of G. Let T be any spanning tree of G. We can convert T into a directed, rooted tree $\vec{T}$ by choosing an arbitrary vertex r of T as root and, for every path from r to a vertex v in T, directing the edges on this path so that it is a directed path from r to v. We denote the existence of a non-tree edge (v,w) in G by v -- w. Number the vertices of $\vec{T}$ from 1 to V in postorder [5]. This ordering corresponds to applying the following algorithm to tree $\vec{T}$:

```
begin
   procedure POSTORDER(v); begin
      for w such that $v \rightarrow w$ do POSTORDER(w);
      NUMBER(v):=i:=i+1;
   end;
   i:=0;
   POSTORDER(r); comment r is the root of $\vec{T}$;
end;
```

Henceforth we shall refer to vertices by their number. For any vertex v, let ND(v) be the number of descendants of vertex v (including v itself).

Let $S(v) = \{w \mid v \overset{*}{\rightarrow} w\} \cup \{w \mid \exists u\ (v \overset{*}{\rightarrow} u \text{ and } u \text{ -- } w)\}$, let $L(v) = \text{minimum } (S(v))$, and let $H(v) = \text{maximum } (S(v))$.
The following lemmas are easy to prove:

<u>Lemma 1</u>:

$$v \overset{*}{\rightarrow} w \text{ iff } v - ND(v) < w \leq v.$$

<u>Lemma 2</u>:

$$ND(v) = 1 + \sum_{v \rightarrow w} ND(w).$$

<u>Lemma 3</u>:

$$L(v) = \min \{v-ND(v)\} \cup \{L(w) \mid v \rightarrow w\} \cup \{w \mid v \text{ -- } w\}$$

Lemma 4:

$$H(v) = \max \{v\} \cup \{H(w) \mid v \to w\} \cup \{w \mid v \text{ -- } w\}.$$

Our main result is:

Theorem 5:

Edge (v,w) is a bridge of G if and only if $v \to w$ in $\vec{T}$, $H(w) \leq w$, and $L(w) > w - ND(w)$.

Proof: Obviously, no non-tree edge is a bridge. Consider any tree edge $v \to w$. This edge is a bridge if and only if no descendant of w is joined by an edge to a non-descendant of w. This condition is equivalent to that stated in the theorem, by Lemma 1 and the definitions of $L(w)$ and $H(w)$.

To find all the bridges of G, we calculate $ND(v)$, $L(v)$, and $H(v)$ for all vertices v using Lemmas 2,3, and 4 and test the condition in Theorem 5 for each tree edge. The entire algorithm is:

```
for each connected component $G_1$ of G do begin
        let $G_1$ have $V_1$ vertices;
    a:  find a spanning tree $T$ of $G_1$;
    b:  convert $T$ to a directed, rooted tree $\vec{T}$;
    c:  number the vertices of $\vec{T}$ in postorder;
        for v:= 1 until $V_1$ do begin
           $ND(v) := 1 + \sum_{v \to w} ND(w)$;
           $L(v) := \min \{v-ND(v)\} \cup \{L(w) \mid v \to w\} \cup \{w \mid v \text{ -- } w\}$;
        $H(v) := \max \{v\} \cup \{H(w) \mid v \to w\} \cup \{w \mid v -- w\}$;
    end;
    for $v \to w$ do if $H(w) \leq w$ and $L(w) > w - ND(w)$ then
        denote $(v,w)$ a bridge;
    end;
```

Finding the connected components of G and carrying out steps a, b, and c on each component requires $O(V+E)$ time using any search method, if graph G is represented by a list structure [1, 2]. The computations of ND, L, and H are well defined since if $v \to w$, $v > w$ by the postorder numbering. These computations take $O(V+E)$ time. Thus, finding all the bridges of G requires $O(V+E)$ time with this algorithm. Knowing the bridges of G, it is an easy matter to find the bridge-connected components of G in $O(V+E)$ additional time.

REFERENCES

[1] J. Hopcroft and R. Tarjan, "Efficient algorithms for graph manipulation", Comm. A.C.M., Vol. 16, No. 6, (June 1973), 372-378.

[2] R. Tarjan, "Depth-first search and linear graph algorithms", SIAM J. Comput., Vol. 1, No. 2, (June 1972), 146-160.

[3] J. Hopcroft and R. Tarjan, "Dividing a graph into triconnected components", SIAM J. Comput., Vol. 2, No. 3, (September 1973), 135-158.

[4] R. Tarjan, "Finding dominators in directed graphs", SIAM J. Comput., to appear.

[5] D. Knuth, "The Art of Computer Programming, Vol. 1: Fundamental Algorithms, Addison-Wesley, Reading, Mass., 1968, 315-332.