INFORMATION PROCESSING LETTERS 2 (1974) 160–161. NORTH-HOLLAND PUBLISHING COMPANY

A NOTE ON FINDING THE BRIDGES OF A GRAPH*

R. Endre TARJAN
Computer Science Division, University of California, Berkeley, California, USA

Received 6 March 1974

algorithm bridge connectivity search spanning tree

Recently, algorithms have been developed which use depth-first search to test various connectivity properties of graphs efficiently. Examples include algorithms to find the connected, biconnected, and triconnected components of an undirected graph [1–3], and the strongly connected components [2] and dominators [4] of a directed graph. Depth-first search is not always necessary for testing connectivity properties like these efficiently, 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 $(v_1, v_2), (v_2, v_3), \cdots, (v_{n-1}, v_n)$ 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.

* This research was partially supported by the National Science Foundation, Contract Number NSF-GJ-35604X.

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 specified vertex called the 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 $v \to w$ and the existence of a path from v to w in a directed, rooted tree by $v \overset{*}{\to} w$. If $v \to w$, v is the *father* of w and w is a *son* of v. If $v \overset{*}{\to} w$, v is an *ancestor* of w and w is a *descendant* of v. Note that v is, by this convention, an ancestor of itself.

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 → w do POSTORDER(w);
    NUMBER (v) := i := i + 1;
  end;
```

```
    i := 0;
    POSTORDER(r); comment r is the root of T⃗;
end;
```

Hencefoth we shall refer to vertices by their number, so that v = NUMBER(v). For any vertex v, let ND(v) be the number of its descendants (including v itself). Furthermore let

$$S(v) = \{w | v \overset{*}{\rightarrow} w\} \cup \{w | \exists u (v \overset{*}{\rightarrow} u \text{ and } u - - w)\},$$

let $L(v)$ = minimum ($S(v)$), and let $H(v)$ = maximum ($S(v)$). The following lemmas are easy to prove:

Lemma 1: $v \overset{*}{\rightarrow} w$ iff $v - \text{ND}(v) < w \leq v$.

Lemma 2: $$\text{ND}(v) = 1 + \sum_{v \rightarrow w} \text{ND}(w).$$

Lemma 3: $L(v)$ = minimum ($\{v - \text{ND}(v) + 1\}$ $\cup \{L(w) | v \rightarrow w\} \cup \{w | v - - w\}$).

Lemma 4: $H(v)$ = maximum ($\{v\}$ $\cup \{H(w) | v \rightarrow w\} \cup \{w | v - - w\}$).

Our main result is:

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

Proof: Obviously, no non-tree edge is a bridge. Consider any tree edge $v \rightarrow w$. 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 $H(w)$. Q.E.D.

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 the theorem for each tree edge. The entire algorithm is:

```
for each connected component G1 of G do begin
    let G1 have V1 vertices;
    a: find a spanning tree T of G1;
    b: convert T to a directed, rooted tree T⃗;
    c: number the vertices of T⃗ in postorder;
    d: for v := 1 until V1 do begin
        ND(v) := 1 + Σ_{v→w} ND(w);
        L(v) := minimum ({v - ND(v) + 1}
            ∪ {L(w) | v → w} ∪ {w | v - - w});
        H(v) := maximum ({v} ∪ {H(w) | v → w}
            ∪ {w | v - - w});
        for w such that v → w do if H(w) ≤ w and L(w) >
            w - ND(w)
        then denote (v, w) a bridge;
    end;
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 $v \rightarrow w$ implies $v > w$ 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, Comm. A.C.M. 16 (1973) 372–378.

[2] R. Tarjan, SIAM J. Comput. 1 (1972) 146–160.

[3] J. Hopcroft and R. Tarjan, SIAM J. Comput. 2 (1973) 135–158.

[4] R. Tarjan, SIAM J. Comput., to appear.

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