I.J. Education and Management Engineering **2012, 1,9-15**
Published Online January 2012 in MECS (http://www.mecs-press.net)
DOI: 10.5815/ijeme.2012.01.02

Available online at http://www.mecs-press.net/ijeme

Application of DFS in the Study of Edge-connected Graph

Cui-xia XU

Computer and Communication Engineering Weifang University Weifang261061, China

Abstract

In this paper a simple method is proposed to determine whether a graph is edge-connected. This method may calculate the minimum pre-order number of each vertex by back edge for the depth-first search spanning tree, and then find out the bridges in the graph. Finally, it may determine whether the graph is edge-connected. The best nature of method is to understand and hold the algorithm easily. It can help teaching improvement and practice application. It is also worth popularization.

Index Terms: spanning tree; bridge; edge-connected graph; DFS (depth-first search);back edge

1. Introduction

In the graph we sometimes wish there are multiple paths between each pair vertices, so that we can promptly deal with certain problems. For example, suppose that the main communication lines in integrated circuits, or the communication network is the edge-connected. If a wire or a link is broken, the rest of the circuit can work properly.

There are many similar problems. Such a kind of problems is called the edge-connectivity problem. This paper presents a method to determine whether a graph is edge-connected, which is based on the nature of the DFS tree.

2. basic concept

2.1. Graph, Connected Graph, Connected Component

A graph *G* consists of a finite set *V* of objects called vertices, a finite set *E* of objects called edges.

A graph *G* is called connected if there is if there is a path from any vertex to any other vertex in the graph G. Otherwise, the graph is disconnected. If the graph is disconnected, the various connected pieces are called the connected components [2] of the graph.

Corresponding author:
E-mail address: x115@sina.com

2.2. Tree, Rooted Tree, Subtree

Let *A* be a set, and let *T* be a relation on *A*. We say that *T* is a tree if there is a vertex v_0 in *A* with the property that there exists a unique path in *T* from v_0 to every other vertex in *A*, but no path from v_0 to v_0. The vertex v_0 is unique. It is often called the root of the tree T, and T is the referred to as rooted tree. We write (T, v_0) to denote a rooted tree T with root v_0.

If (*T*, v_0) is a rooted tree and $v \in T$, then *T* (*v*) is also a rooted tree with root *v*. We will say that *T* (*v*) is the subtree [3] of *T* beginning at *v*.

2.3. Spanning Tree, DFS Spanning Tree

If *R* is a symmetric, connected relation on a set *A*, we say that a tree *T* on *A* is a spanning tree for R if *T* is a tree with exactly the same vertices as *R* and which can be obtained from *R* by deleting some edges of *R*.

If a spanning tree [1] is used to describe the depth-first search process for a connected graph, it is called the depth-first search spanning tree or sometimes called DFS spanning tree of this graph.

2.4. Tree-Edge, Back edge

An edge (*v*, *w*) is a tree-edge in a connected graph if it is also in the DFS tree of this graph. When we are traveling in the edge (*v*, *w*) for the first time, the vertex *w* has not been traveled.

In the DFS tree, if the vertex *w* is the ancestor of vertex *v* (not its parent), and when we are traveling in the edge (*v*, *w*) for the first time, the vertex *w* has been traveled. Such an edge (*v*, *w*) is called a back edge.

2.5. Bridges

An edge is called a bridge [4] in a connected graph if deleting it would create a disconnected graph.

A connected graph is called edge-connected if it contains no bridge. Otherwise, the graph is edge-separable and the bridge is called separation edge.

2.6. Preorder Number of Vertex, Minimum Preorder Number Of Vertex

Suppose that we use an array *pre* to keep track of the preorder numbers of vertices and use an array *low* to keep track of the minimum preorder numbers of vertices.

The preorder number of vertex *v* is its number in the preorder sequence of the DFS tree, denoted by *pre* [*v*].

The minimum preorder number of vertex *v*, denoted by *low* [*v*], is got by a back edge in the subtree with root *v*, which points to a vertex at the lowest-numbered level of the DFS tree.

Let *low* [*v*] be the minimum value in the following three:

- *pre* [*v*]
- *low* [*w*], if edge (*v*, *w*) is a tree-edge.
- *pre* [*u*], if edge (*v*, *u*) is a back edge.

For example, in Fig. 2, *low* [9] has value 2, because in the subtree with root 9 there is a back edge pointing to vertex 4 (*pre* [4] = 2), but no other back edges pointing to the vertex at the lower-numbered level of the DFS tree.

For a vertex *w* in the DFS tree, if it has parent, that is, vertex *v*, and if *low* [*w*] = *pre* [*w*], then the tree-edge (*v*, *w*) is a bridge.

3. the property of bridge

Finding all bridges in a connected graph is actually the application of DFS, which need to use the basic properties of DFS trees for this graph. Note that a back edge can not be a bridge, because there is another path between its two vertices.

Property 1: In any DFS tree, a tree-edge (v, w) is a bridge if and only if there is no back edge to connect the ancestors and descendants of vertex w.

This property tells us that the only link between any vertex in the subtree with root w and vertex v that is not in the subtree, is the parent link. If and only if each path from any vertex in the subtree with root w to any other vertex not in the subtree with root w, which must include the edge (v, w), such an edge (v, w) is a bridge. In other words, if the edge (v, w) is removed from the graph, the new graph will not be connected.

4. algorithm of finding bridges

4. 1 the Basic Thought of Algorithm

Suppose that a connected graph G has n vertexes and m edges. Let $V = \{v_1, v_2, \ldots, v_n\}$ and let $E = \{e_1, e_2, \ldots, e_m\}$. The steps are as follows:

Step 1 For each vertex $v \in V$, let *pre*[v] = -1,that is, vertex v has not been traveled.

Step 2 Initialize variable *predfn* and variable *bcnt* with value 0. That is, let *predfn*=0 and let *bcnt*=0.

Step 3 Choose vertex v_1 as the first vertex. Let $v = v_1$.

Step 4 Call the procedure *Dfs_bridge* to this graph beginning at v, that is *Dfs_bridge*(G, EDGE(*e*.*v*=*v*, *e*.*w*=*v*)).

A procedure *Dfs_bridge*(G, EDGE(*e*.*v*, *e*.*w*)) consists of the following four steps:

Step 4.1 Replace *predfn* with *predfn*+1, and let $w = e.w$.

Step 4.2 Mark w traveled, that is, let *pre*[w] = *predfn*.

Step 4.3 Initialize *low*[w] with value *predfn*.

Step 4.4 For each edge (w, v) $\in E$, the specific operations are as follows:

If edge (w, v) is a tree-edge, then recursively call the procedure *Dfs_bridge* (G, EDGE(w, v)), and let *low*[w]= min{*low*[w], *low*[v]}, and if *low*[v]=*pre*[v], then let *bcnt* = *bcnt*+1 and print out the bridge (w, v).

If edge (w, v) is a back edge, then let *low*[w] = min{*low*[w], *pre*[v]}.

4. 2 Algorithm Implementation

```
int predfn, bcnt, pre[maxV], low[maxV];
void Graph_ bridge (Graph G)
{
int v;     link t;
for (v = 0; v < G->V; v++) pre[v] = -1;
predfn =0;
bcnt=0;
v=0;
Dfs_bridge (G, EDGE(v, v));
}
void Dfs_bridge (Graph G, Edge e)
{
 link t;   int v, w = e.w;
 predfn = predfn+1;
 pre[w] = predfn;
 low[w] = predfn;
 for (t = G->adj[w]; t != NULL; t = t->next)
```

```
if (pre[v = t->v] == -1)
{
Dfs_bridge(G, EDGE(w, v));
if (low[w] > low[v])
low[w] = low[v];
if (low[v] == pre[v])
{
bcnt++;
printf("%d-%d\n", w, v);
}
}
else if (v != e.v)
if (low[w] > pre[v])
low[w] = pre[v];
}
```

4. 3 Time Complexity of the Algorithm

Note that the procedure of above algorithm is actually a procedure of depth-first searching for the graph. Suppose that a connected graph *G* has *n* vertexes and *m* edges. Then, if the graph *G* is represented by adjacency-matrix, the algorithm has running time $O(n^2)$, and if the graph *G* is represented by adjacency-list, the algorithm has running time $O(n + m)$.

5. EXAMPLE

In the graph shown in Fig. 1, edges (0, 5), (6, 7) and (11, 12) are bridges, that is, all such edges are represented by thick lines. Here the graph has four edge-connected components, that is, respectively, {0,1,2,6}, {7,8,10}, {3,4,5,9,11} and {12}.

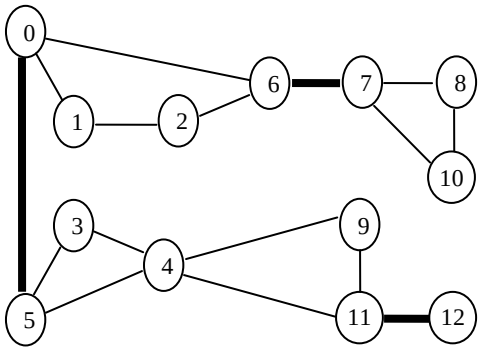

Figure 1. an edge-separable graph

We will represent the tree-edge as a solid line and represent the back edge as a dashed line. For the DFS tree each vertex *v* is labeled with the pair (*pre* [*v*], *low* [*v*]).

A depth-first search spanning tree for the graph in Fig. 1 may be found by using this algorithm beginning at any vertex. Fig. 2 shows a depth-first search spanning tree produced by beginning at vertex 0, and Fig. 3 shows a depth-first search spanning tree beginning at vertex 12.

Let us now apply above algorithm to the graph shown in Fig. 1 beginning at vertex 0. Start from vertex 0 to vertex 3, a back edge (3, 5) is found, let *low* [3] = *pre* [5]. Then return to vertex 4, let *low* [4] = *low* [3].

Proceeding as before from vertex 4 to vertex 12, as low [12] = pre [12], edge (11, 12) is a bridge. Now return to vertex 11, a back edge (11, 4) is found, let *low* [11] = *pre* [4]. Then return to vertex 9, let *low* [9] = *low* [11]. Then return to vertex 5, as low [5] = pre [5], edge (0, 1) is a bridge. At last return to vertex 0.

Proceeding as before from vertex 0 to vertex 8, a back edge (8, 7) is found, let *low* [8] = *pre* [7]. Now return to vertex 10, let *low* [10] = *low* [8]. Then return to vertex 7, as *low* [7] = *pre* [7], edge (6, 7) is a bridge. Then return to vertex 6, a back edge (6, 0) is found, let *low* [6] = *pre* [0]. Then return to vertices 2 and 1, let *low* [2] = *low* [6] and *low* [1] = *low* [6]. Finally, the search ends at the starting vertex. The result is shown in Fig.2.

In the DFS tree shown in Fig.2, vertices 5, 7, and 12 have the properties: no back edges to connect their descendants and their ancestors. And the other vertices do not have this nature. As shown, deleting any tree-edge between any of these vertices and its parent would make the subtree beginning at any such vertex and the rest of the graph disconnected. In other words, edges (0, 5), (11, 12) and (6, 7) are bridges. Here for the vertices 5, 7 and 12, their minimum preorder numbers are equal to their preorder numbers.

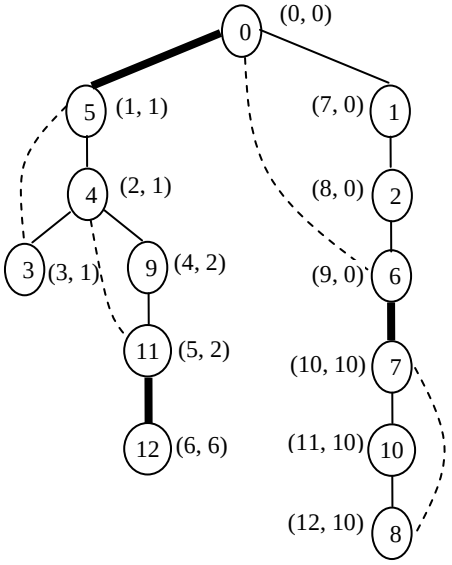

Figure 2. a DFS tree beginning at vertex 0

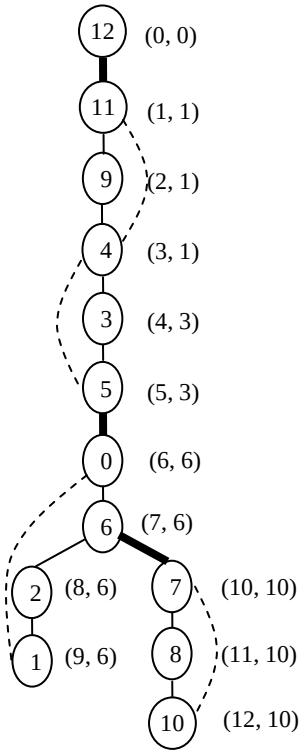

Figure 3. a DFS tree beginning at vertex 12

The DFS trees shown in Fig.2 and Fig.3, respectively, are not the same, but by them the same bridges are found. In Fig.3 for the vertices 11, 0 and 7, their minimum preorder numbers are equal to their preorder numbers.

But when we are searching the different DFS trees for the same graph, we find that search costs depend not only on the nature of that graph, but also on the natures of the DFS trees. For example, in Fig. 3, the stack for recursive calls requires a larger space.

6. CONCLUSION

As the example illustrates, spanning trees are not unique. If we choose different vertices as the starting vertices, and if we travel the vertices and edges in a different order, then we should produce the different DFS trees. Analysis showed that we should find the same bridges by either of the DFS trees.

On the one hand, the algorithm can find all the bridges in the graph, that is, it can determine whether a graph is edge- connected; on the other hand the algorithm is based on recursive function DFS, so it has a better effect upon the teaching.

REFERENCES

[1] [U.S.] Robert Sedgewick. “Algorithms in C(Third Edition)”. Beijing: POSTS & TELECOMMUNICATIONS PRESS, 2004. (in chinese)

[2] Sara baase, Allen Van Gelder.“Computer Algorithms:Introduction to Design and Analysis(Third Edition)”. Beijing: Higher Education Press, 2001.
[3] [Saudi Arabia] M.H. Alsuwaiyel. “Algorithms Design Techniques and Analysis”. Beijing: Electronic Industry Press, 2004.(in chinese)
[4] Bernard Kolman, Robert C. Busby, Sharon Cutler Ross.“Discrete Mathematical Structures”. Beijing: Higher Education Press, 2008.