Depth First Search and Biconnectivity

May 20, 2002

Deep and Wide

Breadth-First Search

Depth-First Search

Explore across before down

Explore down before across

Depth-First Search

NumberDFS(Graph G, Vertex *root)
{
    for each (v in G) {
        Encountered(v) = false;
        Number(v) = -1;
    }
    int num = 1;
    RecursiveDFS(root, &num);
}

RecursiveDFS(Vertex *v, int *pn)
{
    Number(v) = (*pn)++; 
    for each (w in v->Neighbors())
        if (!Encountered(w))
            RecursiveDFS(w, pn);
}
If we lose Wenatchee, can Seattle still talk to Spokane?
Biconnectivity

Two Equivalent Definitions

Two vertex-disjoint paths between any two distinct vertices

Connected and no cut vertices

• v a cut vertex if \((G - v)\) is disconnected

Testing Biconnectivity

• Is \(G\) connected? How do you know?

• Is \(G\) biconnected? How do you know?

Looking at DFS

If the root has more than one child, it is a cut vertex!
If we remove $r$, then there is no path between $u$ and $v$.

Why?

Red edges are followed; Black edges are skipped.

Which node is the cutvertex here?
The Lemma

\[ v \text{ is a cutvertex } \iff \text{ some child of } v \text{ has no skipped edges to an ancestor of } v \]

The Lemma, Right to Left

\[ \text{some child of } v \text{ has no skipped edges to an ancestor of } v \]

\[ \implies \]

\[ v \text{ is a cutvertex} \]

The Lemma, Left to Right

\[ \text{all children of } v \text{ have skipped edges to an ancestor of } v \]

\[ \implies \]

\[ v \text{ is not a cutvertex} \]
What We Need to Know

Depth = 0
Depth = 1
Depth = 2
Depth = 3
Depth = 4

\* v is an ancestor of w iff \[\text{Depth}(v) < \text{Depth}(w)\]
\* w is an descendant of v iff \[\text{Depth}(w) > \text{Depth}(v)\]
\* For those falling asleep: this DFS tree is impossible. Why?

Computing Depth

```
DepthDFS(Graph G, Vertex *root)
{
  for each (v in G) Depth(v) = -1;
  Depth(root) = 0;
  Recurse(root);
}

Recurse(Vertex *v)
{
  // Recurse function implementation
}
```

Checking Skipped Edges

```
int MinDepthSeen(Vertex *v)
{
  // MinDepthSeen function implementation
}

bool IsCutVertex(Vertex *v)
{
  // IsCutVertex function implementation
}
```
Doing Everything at the Same Time

```c
FindCutVerticies(Graph G, Vertex *root)
{
    for each (v in G) Depth(v) = -1;
    Depth(root) = 0;
    Recurse(root);
    if (#(depth 1 nbrs of root) > 1)
        CutVertex(root) = true;
}

int Recurse(Vertex *v)
{
    int min_depth = Depth(v);
    for each (nbr w of v) {
        if (Depth(w) == -1) {
            Depth(w) = Depth(v) + 1;
            int d = Recurse(w);
            min_depth = min(min_depth, d);
            if (d >= Depth(v) and v != root)
                CutVertex(v) = true;
        } else
            min_depth = min(min_depth, Depth(w));
    }
    return min_depth;
}
```

Running Time?

```c
FindCutVerticies(Graph G, Vertex *root)
{
    for each (v in G) Depth(v) = -1;
    Depth(root) = 0;
    Recurse(root);
    if (#(depth 1 nbrs of root) > 1)
        CutVertex(root) = true;
}

int Recurse(Vertex *v)
{
    int min_depth = Depth(v);
    for each (nbr w of v) {
        if (Depth(w) == -1) {
            Depth(w) = Depth(v) + 1;
            int d = Recurse(w);
            min_depth = min(min_depth, d);
            if (d >= Depth(v) and v != root)
                CutVertex(v) = true;
        } else
            min_depth = min(min_depth, Depth(w));
    }
    return min_depth;
}
```