Class Notes: Depth-First Search

Assume that during a depth-first search the vertices of a graph are numbered from 1 to n in both preorder (the order of first visitation, called **discovery order** in the book) and in postorder (the order of unstacking, called **finishing order** in the book). Denote by $pre(v)$ and $post(v)$ the preorder and postorder number of v, respectively, and by $nd(v)$ the number of descendants of v, including v itself.

Descendants Lemma. The following four conditions are equivalent:

(i) Vertex v is an ancestor of vertex w;

(ii) $pre\ (v)\ \leq\ pre(w)\ <\ pre(v)\ +\ nd\,(v)$;

(iii) $post(v)\ -\ nd(v)\ <\ post(w)\ \leq\ post(v)$;

(iv) $pre(v) \leq pre(w)$ and $post(v) \geq post(w)$.

Proof: The proper descendants of v are exactly those vertices added to the stack while v is on the stack. These vertices receive consecutively increasing preorder numbers starting with $pre(v) + 1$, which implies that conditions (i) and (ii) are equivalent. These vertices also receive consecutively increasing postorder numbers ending with $post(v) - 1$, which implies that (i) and (iii) are equivalent. Since (i) implies (ii) and (iii), (i) also implies (iv). Suppose (iv) is true with $v \neq w$. Then $pre(v) < pre(w)$, which implies that w is added to the stack after v, but $post(v) > post(w)$, which implies that w is popped from the stack before v. Then w must be added before v is popped, and w must be an ancestor of v, i.e., (iv) implies (i). □

Lemma 1 (Edge Lemma) *An edge (v, w) is of one of the following four kinds.*

(i) w is a child of v: (v, w) is a **tree edge**.

(ii) w is a descendent but not a child of v: (v, w) is a **forward edge**.

(iii) w is an ancestor of v: (v, w) is a **cycle edge** *(or* **backedge***).*

(iv) w and v are unrelated and $v >_{pre} w$: (v, w) is a **cross edge***.*

Proof: Suppose (v, w) is an edge and v and w are unrelated. We must show that $v >_{pre} w$. Suppose by way of contradiction that $v <_{pre} w$. Then w is unvisited when v is stacked. Before v is unstacked, edge (v, w) will be traversed, which means w will be stacked before v is unstacked, i.e., w is a descendant of v. This is a contradiction. □

Lemma 2 (Path Lemma) *Any path from v to w with $v <_{pre} w$ contains a common ancestor of v and w.*

Proof: Suppose v and w are unrelated; otherwise the lemma is immediate. Let u be the nearest common ancestor of v and w, and let x be the child of u that is an ancestor of v. (If v and w have no common ancestor because they are in separate dfs trees, let x be the root of the tree containing v; we shall see later that this case is impossible.) Finally, let S be the set of vertices less than or equal to x in postorder. Since v and w are unrelated by assumption, $v <_{post} w$. But this means that $x <_{post} w$ and hence that $w \notin S$. Consider the first edge (y, z) the path from v to w that leaves set S. This edge has $y <_{post} z$, which means it is a cycle edge; that is, y is a descendant of z. Since $y \in S, y \leq_{post} x$, and since $z \notin S$, $x <_{post} z$. But $y \leq_{post} x <_{post} z$ and y a descendant of z simply that x is a descendent of z, and hence that z is an ancestor of u and hence of both v and w. (In particular, u must exist.) □

Lemma 3 (Component Lemma) *The vertices of a strong component form of a subtree of a dfs tree.*

Proof: Let S be the vertex set of a strong component. Let r be the smallest vertex in S in postorder. If $x \in S$, there must be a path in S from x to r, which by the path lemma must contain a common ancestor of x and r. This ancestor must be r. Hence every vertex in $x \in S$ must be a descendant of r. If y is a descendent of r and an ancestor of x, the existence of the tree path from r through y to x implies that y is in the component S. Hence S defines a subtree. □

Call the smallest vertex in a component (in preorder) the **root** of the component.

One-Way Depth-First-Search Strong Components Algorithm

Initialize $num = 0$, stack = empty.

Perform a depth-first search.

- When visiting a vertex v in preorder:
  - set $low(v) = pre(v) = num + 1$;
  - push v onto stack.
- When retreating along an edge (v, w):
  - set $low(v) = \min\{low(v), low(w)\}$.
- When visiting a vertex v in postorder:
  - if $low(v) = pre(v)$ then pop vertices off the stack down to and including v, and for each
    - popped vertex w set $low(w) = \infty$. (Vertex v is the root of a strong component containing all the just-popped vertices.)

Proof of correctness of the strong components algorithm

Call a vertex v **tagged** if $low(v) = pre(v)$ when v is visited in postorder. When a tagged vertex is visited in postorder, it and all its descendants that are still on the stack are popped. Thus the set of popped vertices is exactly the set of descendants of v that have no tagged ancestor other than v. It follows that if the tagged vertices are exactly the component roots, then the algorithm correctly identifies the components. Thus all that remains is to prove that a vertex v is tagged if and only if it is a component root. We prove this by induction on the postorder number of v.

For vertex v, let F_v be the set of edges (x, y) such that x is a descendant of v but y is not. F_v contains only cycle edges and cross edges. Each edge $(x, y) \in F_v$ has $y <_{pre} v$.

Suppose v is not a component root. Then there must be an edge $(x, y) \in F_v$ such that y is in the same strong component as v, because the root of the component containing v is not a descendant of v and some edge on the path from v to this root must be in F_v. Let (x, y) be such an edge, and let u be the nearest common ancestor of y and v. No vertex on the tree path from u to y, except possibly u itself, can be a component root. But all vertices on this tree path except u are smaller than v in postorder, and hence are untagged, by the induction hypothesis. Hence y is on the stack when the retreat along edge (x, y) occurs. This means that $low(x) \leq pre(y)$ after this retreat, and after the retreats along all the tree edges from v to x occur, $low(v) \leq \mathrm{pre}(y)$. Since $pre(y) < pre(v)$, vertex v is untagged.

Conversely, suppose v is a component root. then for every edge $(x, y) \in F_v$, y must be in a different strong component than v. Consider an edge $(x, y) \in F_v$. Let u be the nearest common ancestor of y and v. Vertices y and u cannot be in the same strong component, for then v would be in the same component as well. Thus there must be a component root that is not u on the tree path from u to v. By the induction hypothesis u is tagged, which means that $low(y) = \infty$ when the retreat along (x, y) occurs. A proof by induction on the order of retreats shows that for every descendant z of v, $low(z) \geq pre(v)$. In particular, $low(v) \geq pre(v)$ when v is visited in postorder, which implies $low(v) = pre(v)$, and v is tagged. $\square$

Strong Components Consider a directed graph G. Define an equivalence relation $\sim$ on the vertices by $x \sim y$ iff there is a path from x to y and a path from y to x. It is easy to verify that $\sim$ is indeed an equivalence relation, i.e. that

(i) $x \sim x$ for all x,

(ii) $x \sim y$ implies $y \sim x$ for all x and y, and

(iii) $x \sim y$ and $y \sim z$ implies $x \sim z$ for all x, y, and z. (Exercise: verify this.)

The subgraphs defined by the equivalence classes of $\sim$ are called the *strong components* of G. We wish to find the strong components of G in $\mathrm{O}(n + m)$ time.

Method 1 (Two-pass method).

Step 1. Perform a depth-first search of G, numbering the vertices from 1 to n in depth-first postorder.

Step 2. Unmark all vertices. Process the vertices in *decreasing* order with respect to the numbering computed in Step 1. To process a vertex, test whether v is unmarked. If so, find all unmarked vertices from which v is reachable by a path of unmarked vertices, declare these vertices to define a strong component, and mark them all. The unmarked vertices from which an unmarked vertex v is reachable can be found by a backward search from v that immediately retreats from marked vertices. (This search need not be depth-first.)

Exercise: Verify that this algorithm runs in $\mathrm{O}(n+m)$ time, if appropriately implemented.

Proof of correctness. Let X be a set of vertices declared to be a strong component by the algorithm, resulting from a backward search from a vertex v. X contains only vertices numbered no more than v in postorder. (All higher numbered vertices are marked before the search from v.) Let $w \in X$.

Claim: *w is a descendant of v.*

Proof of Claim: Since w is numbered no greater than v and there is a path from w to v, the path of unmarked vertices from w to v must contain a common ancestor of w and v, which must be v itself. □

The claim implies that every vertex in X is reachable from v by a path in the depth-first spanning tree. All vertices on this path must be unmarked when the search from v is performed. (Exercise: verify this.)
It follows that X must define a strong component. (Exercise: verify this, by induction on the number of components so far identified.)

Exercise: Show that if v is the largest-numbered vertex in a strong component, the component consists exactly of all vertices numbered no greater than v from which v is reachable by a path of vertices numbered no greater than v.

Problem: Devise a clearer, more succinct proof of correctness of this algorithm.

Method 1 is the method given in the text in Section 23.5. The components are generated in a topological order. (Exercise: verify this.)

<u>Method 2</u> (One-pass method).

This method performs a single depth-first search, maintaining a stack of vertices and two numberings: $pre(v)$, the preorder number of v, and $L(v)$, used to help compute components. The method consists of a depth-first search of the graph, with the vertex stack initialized to empty, with the following additions:

- When first visiting a vertex v, assign v its preorder number $pre(v)$, push v on the stack, and initialize $L(v) = pre(v)$.
- When retreating along an edge (v, w), set $L(v) = \min\{L(v), L(w)\}$.
- When last visiting a vertex v (just before returning to its parent), test whether $L(v) = pre(v)$. If so, pop v and all vertices on top of it off of the stack, declare these vertices to form a component, and set $L(v) = \infty$ for each vertex popped.

Note: For a tree edge (v, w), we retreat along (v, w) just after last visiting w. For a nontree edge (v, w), we retreat along (v, w) just after advancing along it.

Proof of correctness (sketch): By induction on the number of strong components identified. The components are identified in increasing postorder number of their roots, thus in a reverse topological order. Consider the root v of a component, and assume that all components rooted at vertices less than v in postorder are correctly identified. Let X be the component containing v. Let $w \in X$, $w \neq v$. There is a path from w to v within X. Furthermore X contains only descendants of v. Let (x, y) be the first edge along the path such that y is not a descendant of w. Then y is less than w in postorder. Since $y \in X$, y has not yet been placed in a component, which means that $L(y) \leq pre(y)$. When the search retreats along (x, y), it will update $L(x)$ to be no greater than $pre(y)$. The search will eventually retreat along the tree path from w to x, updating $L(w)$ as well to be no greater than $pre(y) < pre(w)$. Thus w will fail the test to be a component root.

Since all edges from descendants of v to nondescendants of v must lead to vertices in components other than X, which are identified before any vertex in X is visited for the first time, $L(v) = \mathit{pre}(v)$ until v is last visited, when v will be identified as a component root. At this time all of X will be sitting on top of v in the stack; the remainder of the stack contains only vertices less than v in preorder. □

Exercise: Fill in the gaps in this proof.

Problem: Devise a cleaner, more succinct proof of correctness.

Blocks, Ear Decompositions, Bipolar Numberings (See Problem 23-2, pages 495–496).

Consider an undirected graph G. We define the *blocks* (*biconnected components*) of G to be the subgraphs of G induced by the equivalence relation $\equiv$ on edges defined by $(x, y) \equiv (w, z)$ iff $(x, y) = (w, z)$ or (x, y) and (w, z) are on a common simple cycle. (Exercise: prove that this is an equivalence relation.)

The blocks of G are also the maximal subgraphs of G not containing a cut vertex. (Exercise: prove this.)

An *ear decomposition* of G is a partition of the edges of G into simple paths $p_1, p_2, \ldots p_k$, such that for $i \geq 2$, p_i is vertex-disjoint from $p_1, p_2, \ldots, p_{i-1}$, except that for the two ends of p_i, each of which is on some p_j for $j < i$. A *bipolar numbering* of G is a numbering of the vertices of G from 1 to n such that, each vertex except those numbered 1 and n is adjacent both to a lower-numbered and to a higher-numbered vertex.

Theorem: *The following three conditions are equivalent:*

(i) *G with the addition of the edge (s, t) is biconnected;*

(ii) *G has an open ear decomposition such that p_1 has ends s and t;*

(iii) *G has a bipolar numbering with s number 1 and t number n.*

Exercise: Prove that (ii) $\Leftrightarrow$ (iii) $\Rightarrow$ (i).

We shall prove that (i) $\Rightarrow$ (ii),(iii) via linear-time algorithms to compute an ear decomposition and a bipolar numbering. First we give a linear-time algorithm to find blocks. This algorithm resembles Method 2 for finding strong components, but it uses an edge stack in place of a vertex stack and the L-values it computes do not depend upon the manipulation of the stack.

Finding Blocks

Let $pre(v)$ for a vertex v be the preorder number assigned to v by a depth-first search. Let $L(v)$ for a vertex v be defined by

$$L(v) \quad = \quad \min\{pre(w) \mid \text{there is a path consisting of zero or more tree edges followed by at most one cycle edge from } v \text{ to } w\}.$$

Lemma: *A vertex v is a cut vertex iff either*

(i) *v is the root of a depth-first spanning tree and v has at least two children; or*

(ii) *v is not a root but has a child w such that $L(w) \geq pre(v)$.*

Lemma: *An edge (v, w) is a cut edge iff it is a tree edge with v the parent of w and $L(w) \geq w$.*

Exercise: Prove these lemmas.

Block-finding algorithm.

Begin with an empty edge stack and perform a depth-first search. When first visiting a vertex v, compute $pre(v)$ and initialize $L(v) = pre(v)$. When advancing along an edge (v, w), add (v, w) to the stack. When retreating along an edge (v, w), if (v, w) is a cycle edge, replace $L(v)$ by $\min\{L(v), pre(w)\}$. Otherwise, i.e. (v, w) is a tree edge, replace $L(v)$ by $\min\{L(v), L(w)\}$ and, if $L(w) \geq pre(v)$, pop all edges down to and including (v, w) on the stack and declare these edges to define a block.

Exercise: Verify the correctness and $O(n + m)$ running time of this algorithm.

Finding an ear decomposition.

The ear decomposition algorithm consists of three passes. The first pass performs a depth-first search of G with edge (s, t) added, starting at vertex s and first advancing along edge (s, t). During the search, preorder numbers and L-values are computed as in the block-finding algorithm. The second pass rearranges the incidence lists so that, for each vertex v, the first edge (v, w) is either a cycle edge with $pre(w) = L(v)$ or a tree edge with $L(w) = L(v)$. The third pass is another depth-first search of G beginning with edge (s, t) that constructs a sequence of the edges other than (s, t) in the order it advances along them. This edge sequence is partitioned into paths, one per cycle edge, making each cycle edge last on its path. (Each path consists of zero or more tree edges followed by a cycle edge.) This gives an open ear decomposition.

Exercise: Verify the correctness and $O(n + m)$ running time of this algorithm.

Hint: Use the lemma characterizing cut vertices. You must show that each edge sequence in the partition actually forms a path, starting and ending at vertices on previous paths, that the last edge advanced along is a cycle edge, and that each path is simple.

Finding a bipolar numbering.

The bipolar numbering algorithm consists of two passes. The first pass is a depth-first search of G with edge (s, t) added, beginning at vertex s and first advancing along edge (s, t). During the search, preorder numbers and L-values are computed, as well as the parent $p(v)$ of each vertex v in the depth-first spanning tree. The first pass can if necessary test that G with edge (s, t) added is biconnected. The second pass constructs a list X of the vertices, such that if the vertices are numbered in the order they occur in X, a bipolar numbering results. The second pass processes the vertices in preorder. During this pass, each vertex u that is a proper ancestor of the current vertex v has a *sign* that is minus if u precedes v in X and plus if u follows v in X. Initially $X = [s, t]$ and s has sign minus. The second pass consists of repeating the following step for each vertex $v \notin \{s, t\}$ in preorder:

- If $sign(L((v)) = plus$, insert v after $p(v)$ in L and set $sign(p(v)) = minus$;
- If $sign(L(v)) = minus$, insert v before $p(v)$ in L and set $sign(p(v)) = plus$.

Exercise: Verify the correctness and $O(n+m)$ running time of this algorithm. You must show that (i) the signs assigned to vertices that are proper ancestors of the current vertex have the claimed meaning, and (ii) if vertices are numbered in the order they occur in L, a bipolar numbering results.

Exercise: Determine the relationship between the ear decomposition algorithm and the bipolar numbering algorithm.