15-451: Algorithms

Lecture 16: BCCs and SCCs (notes KS)

1 Classifying Edges by DFS

We use counters to assign time stamps to every node during DFS.

- $\delta(u)$: the DFS number of u, and
- $\gamma(u)$: the completion number of u.

```
dfs( vertex x )
{
    mark x;
    delta[x] = ++c1;
    forall (x,y) in E do
        if( y not marked ) dfs( y ); // explore edge
    gamma[x] = ++c2;
}

// usage
c1 = c2 = 0;
forall x in V do
  if( x not marked ) dfs(x);
```

Upon completion, the explored edges (x, y) form a spanning forest T of G. For any set U of edges write $x \longrightarrow_U y$ to indicate the existence of a path from x to y whose edges all belong to U. Classify the edges as follows:

- T: tree edges
- $F = \{ (x,y) \mid x \longrightarrow_T y \}$: forward edges
- $B = \{ (x,y) \mid y \longrightarrow_T x \}$: backward edges
- $C = E - (T \cup F \cup B)$: cross edges

Note that this decomposition of E depends not just on the graph but on the actual adjacency lists. However, regardless of the order of elements on the adjacency lists, the time stamps $\delta(x)$ and $\gamma(x)$ can be nested but they are non-overlapping in the sense that we cannot have

$$\delta(x) < \delta(y) \text{ and } \gamma(x) < \gamma(y).$$

Nested intervals correspond to nested calls:

$$x \longrightarrow_T y \iff \delta(x) \leq \delta(y) \wedge \gamma(y) \leq \gamma(x)$$

Lemma 1.1 *Let (x, y) be an edge. Then*

$$\begin{aligned} (x,y) \in T \cup F &\iff \delta(x) < \delta(y) \\ (x,y) \in B \cup C &\iff \delta(x) > \delta(y) \end{aligned}$$

Proof. It suffices to prove the first equivalence (we are not dealing with self-loops). Direction $\Rightarrow$ is obvious from the definition. For the opposite direction consider the time when y is first encountered in DFS. If this happens because edge (x, y) has been explored, we're done. Otherwise, some edge (u, y) has been explored. Since $\delta(x) < \delta(y)$ this must happen inside the call to x, so $x \longrightarrow_T u$. Hence $(x, y) \in F$, done. □

2 DFS in Undirected Graphs

In this section, $G = \langle V, E \rangle$ is always assumed to be undirected (a symmtric relation).

Lemma 2.1 *DFS on an undirected graph G produces no cross edges.*

Proof. Assume otherwise and let (w, z) be the first edge to be placed into C. Then $\delta(z) < \delta(w)$ and $\gamma(z) < \gamma(w)$. But during the call to z, edge (z, w) must have been explored, hence the call to w is nested inside the call to z, contradiction. □

This implies in particular that $(v, w) \in B$ iff $(w, v) \in T \cup F$.

Lemma 2.2 *Suppose G is connected. Then DFS produces a spanning tree of G.*

Proof. Assume otherwise, say, T has a component T' and there are vertices u, v such that $u \in T'$, $v \notin T'$, but $\{u, v\} \in E$. Suppose u is first encountered in DFS. Since (u, v) is not a cross edge and also not in T, it is either a back or forward edge. Hence $v \in T'$, contradiction. □

3 Biconnected Components

An *articulation point* of a connected graph G is a vertex v such that $G - v$ is disconnected (i.e., the deletion of v, together with all edges incident on v, produces a graph that has at least two connected components). A *biconnected graph* is a connected graph that has no articulation points.

In many graph applications such as a communication network, articulation points are undesirable. Note that any graph other than K_2 is biconnected iff for any two distinct nodes u and v there are two vertex-disjoint paths from u to v.

A *biconnected component (BCC)* of a connected undirected graph is a maximal biconnected subgraph. Note that two biconnected components have at most one vertex in common. Biconnected components form a partition of the edges (but not the vertices).

Lemma 3.1 *Biconnected components are edge-disjoint.*

Proof. Assume edge (u, v) belongs to two BCCs A and B. Since $A \cup B$ is no longer biconnected, there must be an articulation point w whose removal leaves $A \cup B$ disconnected. Suppose a and b are points in two components of $(A \cup B) - w$. Since A and B are biconnected, without loss of generality $a \in A$ and $b \in B$. We may assume $u \neq w$. But then there is a path from a to u and from u to b, contradiction. □

We can calculate the biconnected components in linear time with an augmented version of DFS. The augmented algorithm computes DFS numbers as well as "back" numbers that allow us to identify the entry points of the search in BCCs.

```
DFSbiconn(v) {
  delta[v] = ++dfsnum;
  back[v] = delta[v];

  foreach (v,w) in E do
    if ( delta[w] < delta[v] )
      push (v,w) on EdgeStack;

    if( delta[w] == 0 ) {
      DFSbiconn(w);

      if( back[w] >= delta[v] )
        pop EdgeStack until (v,w) is popped; // get BCC
      else
        back[v] = min(back[v], back(w));
    }
    else
      back[v] = min( back[v], delta[w] );
}
```

4 Strongly Connected Components

A subset $C \subseteq V$ is a *strongly connected* of G if $\forall x, y \in C\,(x \longrightarrow_C y \wedge y \longrightarrow_C x)$. A subset $C \subseteq V$ is a *strongly connected component (SCC)* of G if it is maximally strongly connected: no proper superset of C is strongly connected.

Thus, any two vertices in a SCC lie on a cycle. Clearly, the SCCs form a partition of the vertex set. Also note that the SCC of a vertex v is trivially contained in the WCC of v.

SCC algorithms

- Brute force: late 50's (compute all WCCs)
- Warshall's algorithm: 1962
- Tarjan's algorithm: 1972
- Kosaraju's algorithm: 1978

Motivation for Tarjan's algorithm.

Suppose x, y lie in some SCC C. In the easiest case we have, say, $x \longrightarrow_T y$ after DFS and there is a back edge $(y, x) \in B$. Then we know that the points belong to the same component. However, in general we have to cope with two problems:

- $x \longrightarrow_T y$ may not hold (and neither $y \longrightarrow_T x$)
- instead of a back edge there may be a path of consisting of tree, back and cross edges.

To cope with these problems, define the *root* of a SCC C to be the vertex $v \in C$ such that

$$\forall\, x \in C\, (\delta(v) \leq \delta(x))$$

Write $\mathsf{root}(C)$ for the root of C, and $\mathsf{root}(x)$ for the root of the SCC containing x.

Observations. Suppose C is a SCC with root v.

- $\forall\, x \in C\, (v \longrightarrow_T x)$
- v has maximal completion number γ in C.

Lemma 4.1 *Suppose C is a SCC with root v. Then x belongs to C iff $v \longrightarrow_T x$ and that path does not encounter any other roots.*

Proof. First suppose $x \in C$, so that $v \rightarrow_T z \rightarrow_T x \rightarrow v$ where z is any intermediate vertex. Then $z \in C$, and in particular cannot be another root.

For the opposite direction, suppose $v \longrightarrow_T x$ and let $v' = \mathsf{root}(x)$, so that $v' \longrightarrow_T x$. By our assumption, v' fails to lie on the path from v to x, so we must have $v' \rightarrow_T v \rightarrow_T x \rightarrow v'$. But then $v = v'$, done. □

The last lemma is crucial for the algorithm: we can generate the SCC bottom-up (with respect to the DFS tree). To this end, we modify DFS so that at the end of a call to any vertex x we will be able to determine whether x is a root. If so, we associate x with all the vertices that have already been found, but are not associated with any other root: that produces the SCC of x.

To identify roots, we would like to compute something like the *high number* $\eta(x)$ of any vertex x:

$$\eta(x) = \min\big(\, \delta(z) \;\big|\; z \in \text{ SCC of } x \,\big).$$

Thus, $\eta(x)$ indicates the highest vertex in T that can be reached from x. Unfortunately, it is not quite possible to compute $\eta(x)$ as stated. Instead, we use a slightly different definition. First, let

$$S_x = \{\, z \;\big|\; \gamma(x) \leq \gamma(\mathsf{root}(z))\,\}.$$

Then define

$$\eta(x) = \min \begin{cases} \delta(x) & \\ \delta(z) & (x, z) \in B \cup C, z \in S_x \\ \eta(z) & (x, z) \in T. \end{cases}$$

Note that trivially $\eta(x) \leq \delta(x)$. We will shortly show that $\eta(x) = \delta(x)$ iff x is a root. For the proof we will need a little proposition.

Proposition 4.1 *Let v, w, z be vertices such that $v \longrightarrow_T w$, $(w, z) \in B \cup C$, but not $v \longrightarrow_T z$. Then $\delta(z) < \delta(v)$.*

Proof. Assume that $\delta(v) < \delta(z)$. Since (w, z) is a back or cross edge we have $\delta(z) < \delta(w)$. But then

$$\delta(v) < \delta(z) < \delta(w) \qquad \gamma(w) < \gamma(z) < \gamma(v).$$

This contradicts our assumption that there is no T-path from v to z. □

Lemma 4.2 *$\eta(x) = \delta(x)$ iff x is a root.*

Proof. It suffices to establish the following two claims.

Claim 1: $\eta(v) \geq \delta(\mathsf{root}(v))$.

Claim 2: If $v \neq \mathsf{root}(v)$ then $\eta(v) < \delta(v)$.

Proof of the first claim is by induction on $\gamma(v)$. There are three cases depending on the value of $\eta(v)$.

$\eta(v) = \delta(v)$. By the definition of root, $\delta(v) \geq \delta(\mathsf{root}(v))$.

$\eta(v) = \delta(z) < \delta(v)$ where $(x, z) \in B \cup C, \gamma(v) \leq \gamma(\mathsf{root}(z))$.

Let $r = \mathsf{root}(z)$, so $r \longrightarrow_T z$ and thus $\delta(r) \leq \delta(z) < \delta(v)$. By assumption $\gamma(r) \geq \gamma(v)$. Hence the call to v is nested inside the call to r, $r \longrightarrow_T v$, and we have $r \to_T v \to z \to r$. Therefore, $r = \mathsf{root}(v)$, done.

$\eta(v) = \eta(u) < \delta(v)$ where $(v, u) \in T$. Note that $\gamma(v) > \gamma(u)$, so the IH applies and we have $\eta(u) \geq \delta(\mathsf{root}(u))$. Note that since $(v, u) \in T$ and we must have $\mathsf{root}(u) = \mathsf{root}(v)$ or $\mathsf{root}(u) = u$. In the second case $\eta(u) = \delta(u) > \delta(v)$ since $(v, u) \in T$, contradicting our assumption. Hence $\mathsf{root}(u) = \mathsf{root}(v)$ and thus $\delta(\mathsf{root}(v)) = \delta(\mathsf{root}(u)) \leq \eta(u)$.

For the second claim, suppose v is not a root and let $r = \mathsf{root}(v)$. Then for some vertex z we must have

$$r \to_T v \to_T w \to_{B \cup C} z \to r$$

and there is no T-path from z to r. By the last proposition we have $\delta(z) < \delta(v)$.

Now $\gamma(w) < \gamma(r)$, and r is the root of w, whence z lies in S_w. Hence $\eta(w) \leq \delta(z)$. But then

$$\eta(v) \leq \eta(w) \leq \delta(z) < \delta(v).$$

□

For the implementation, we actually do not need the completion numbers $\gamma(x)$, they are relevant only to the proof.

```
scc( vertex v )
{
   eta[v] = delta[x];
   forall (v,w) in E  do
   {
      if( w not marked )
         mark w, add w to S;
         delta[w] = ++c;
         scc(w);
```

```
            eta[v] = min( eta[v], eta[w] );
        if( delta[w] < delta[v] )
            eta[v] = min( eta[v], delta[w] );
    }
    if( delta[v] == eta[v] )
        remove all w s.t. delta[w] >= delta[v] from S;
}
```

To implement S one can use a stack (elements enter in increasing order with respect to δ), in conjunction with a bitvector for fast membership queries. Hence we have established the following theorem.

Theorem 4.1 *The strongly connected components of a directed graph can be computed in time $O(n+e)$.*

The *(directed acyclic) collapse* of a directed graph $G = \langle V, E \rangle$ is the quotient graph $H = \langle U, E' \rangle$ where U is the collection of SCCs of V, and $(C_1, C_2) \in E'$ iff $\exists\, x \in C_1, y \in C_2\, ((x, y) \in E)$.

Theorem 4.2 *The collapse of a directed graph can be computed in time $O(n + e)$.*

4.1 Application

Satisfiability for Boolean formulae in 2-CNF can be tested in polynomial time.

To see this, consider a formula φ and the directed graph $G_\varphi = \langle V, E \rangle$ whose vertices are the literals in the formula, and whose edges are defined by

$$(x, y) \in E \iff \{\overline{x}, y\} \text{ is a clause.}$$

Claim: The formula is satisfiable iff no SCC of G_φ contains both x and $\overline{x}$.

To see this, note that if a truth assignment α satisfies φ, and $x_1, \dots, x_r$ is a path in G_φ such that $\alpha(x_1) = 1$, then $\alpha(x_r) = 1$.

Exercise 4.1 Show how to use the collapse of G_φ to determine a satisfying truth assignment if one exists. Conclude that a satisfying truth assignment can be constructed in linear time.