f(v)=finishing time after DFS(G,v) terminates

McGill University COMP251 Fall 2009 **Instructor: Phuong Nguyen**

Correctness of the algorithm for finding strongly connected components

Recall the key lemma:

Key lemma: Let C be a strongly connected component of G, and v be a vertex not in C. Suppose that there is a path from C to v (i.e., there is a path from some vertex in C to v). Then $max\{f[u] : u \in C\} > f[v]$.

S[u]<S[v]<f[v]<f[u]

From this we obtain:

Corollary 1: Let C_1, C_2 be two strongly connected components of G, and suppose that there is a path from (some vertex in) C_1 to (some vertex in) C_2. Then $max\{f[u] : u \in C_1\} > max\{f[v] : v \in C_2\}$

S[C1]<S[C2]<f[C2]<f[C1]

And from this we obtain:

Corollary 2: Let C_1, C_2 be two strongly connected components of G, and suppose that $max\{f[u] : u \in C_1\} > max\{f[v] : v \in C_2\}$. Then there is no path in G from C_2 to C_1.

C1→C2

From Corollary 2 we obtain the following corollary, which is crucial for our algorithm:

Corollary 3: Let C_1, C_2 be two strongly connected components of G, and suppose that $max\{f[u] : u \in C_1\} > max\{f[v] : v \in C_2\}$. Then there is no path in G^T from C_1 to C_2.

C1←C2

(Recall that G^T is the transpose of G which is obtained from G by reversing all the edges of G.)

Recall that the idea of the algorithm is as follows:

1. call DFS(G) to compute the finishing time $f[v]$ for every vertex v, sort the vertices of G in decreasing order of their finishing time;
2. compute the transpose G^T of G;
3. Perform another DFS on G, this time in the main for-loop we go through the vertices of G in the decreasing order of $f[v]$;

f[u]=m...f[v]=1

4. output the vertices of each tree in the DFS forest (formed by the second DFS) as a separate strongly connected component.

To prove the correctness of the SCC algorithm, we prove the following theorem. Recall that we use the colors White, Gray, Black for the first DFS, and Black, Blue, Red for the second DFS.

Theorem: Let $C_1, C_2, \ldots, C_k$ be all strongly connected components of G, such that

$$max\{f[u] : u \in C_1\} > max\{f[u] : u \in C_2\} > \ldots > max\{f[u] : u \in C_k\}$$

Then by the time the i-th Black vertex is seen in the main for-loop of the second DFS, all $C_1, C_2, \ldots, C_{i-1}$ have been visited, and $C_i, \ldots, C_k$ have not been visited.

Proof: We prove the theorem by induction on i.

Base case: $i = 1$: the first Black vertex seen in the main for-loop of DFS 2 is the vertex u with maximum $f[u]$. So $u \in C_1$. When is it first seen in the for-loop, none of the components has been visited.

Induction step: Assume that the theorem is true for some i, $1 \leq i < k$. We prove it for $i + 1$. By the induction hypothesis, the i-th Black vertex that is seen in the main for-loop is the vertex u such that $f[u]$ is maximum among all vertices in $C_i, C_{i+1}, \ldots, C_k$. So u is in C_i. By Corollary

3, there are no path in G^T from $C_{i+1}, C_{i+2}, \ldots, C_k$ to C_i. Hence Visit2 starting at u visits only (and all) vertices in C_i. Thus by the time the $(i+1)$-st Black vertex is seen in the main for-loop of DFS2, we have visited all $C_1, C_2, \ldots, C_{i-1}, C_i$ but none of $C_{i+1}, C_{i+2}, \ldots, C_k$. QED.

It follows from the theorem that the i-th DFS tree produced by the second DFS consists of all vertices of C_i.