DFS and Strongly Connected Components for Directed Graphs

Consider DFS, recursive formulation, for a directed graph $G = (V, E)$.

procedure *DepthFirstSearch* (**Graph** $G = (V, E)$)
  for each $v \in V$ **do** $visited[v] \leftarrow$ false;
  for each $v \in V$ **do**
    if not $visited[v]$ **then** $DFS(v)$;

procedure DFS (**vertex** v)
  $visited[v] \leftarrow$ true
  $previsit(v)$ { some operation on v before visiting the neighbors }
  for all w with $(v, w) \in E$
    if not $visited[w]$ **then** $DFS(w)$
  $postvisit(v)$ { some other operation on v after visiting the neighbors }

DFS for directed graphs

Call $DFS(v)$ inspects all edges and vertices reachable from v

We can classify the edges in 4 classes, T, F, B, C:

- Tree edges T
  = edges that DFS follows by its recursive calls
- Forward edges F
  = edges (v, w) where w already visited and $v \longrightarrow_T^* w$
  (there is a path from v to w consisting only of tree edges)
- Backward edges B
  = edges (v, w) where w already visited and $w \longrightarrow_T^* v$
- Cross edges C
  = all other edges, i.e., w already visited and neither $v \longrightarrow_T^* w$ nor $w \longrightarrow_T^* v$

Extend DFS by conceptual computation of T, F, B, C

Add 2 numbers to each vertex v:
$dfsnum(v)$: order of recursive DFS calls (previsit order)
$compnum(v)$: order of completion of DFS calls (postvisit order)

procedure *DepthFirstSearch* (**Graph** $G = (V, E)$)
  $count1 \leftarrow 0$
  $count2 \leftarrow 0$
  { $T, F, B, C \leftarrow \emptyset$ }
  for each $v \in V$ **do** $visited[v] \leftarrow$ false;
  for each $v \in V$ **do**
    if not $visited[v]$ **then** $DFS(v)$;

Extend DFS by conceptual computation of T, F, B, C (cont.)

procedure DFS (**vertex** v)
  $visited[v] \leftarrow$ true
  $count1 \leftarrow count1+1$; $dfsnum[v] \leftarrow count1$
  for all w with $(v, w) \in E$
    if not $visited[w]$
    then
      { $T \leftarrow T \cup \{(v, w)\}$ }
      $DFS(w)$
    { **else** { w visited earlier }
      if $v \longrightarrow_T^* w$ **then** $F \leftarrow F \cup \{(v, w)\}$
      elseif $w \longrightarrow_T^* v$ **then** $B \leftarrow B \cup \{(v, w)\}$
      else $B \leftarrow B \cup \{(v, w)\}$ }
  $count2 \leftarrow count2+1$; $compnum[v] \leftarrow count2$

DFS Theorem

Lemma 1: DFS needs time $O(|V| + |E|)$.

Lemma 2: (Properties of the extended algorithm)

(a) T, F, B, C is a partition of E.

(b) T corresponds to the call tree of DFS.

(c) $v \longrightarrow_T^* w \;\Leftrightarrow\; dfsnum[v] \leq dfsnum[w]$ and $compnum[w] \leq compnum[v]$

(d) For all $(v, w) \in E$:
$(v, w) \in T \cup F \;\Leftrightarrow\; dfsnum[v] < dfsnum[w]$

(e) For all $(v, w) \in E$:
$(v, w) \in B \;\Leftrightarrow\; dfsnum[w] < dfsnum[v]$ and $compnum[w] > compnum[v]$

(f) For all $(v, w) \in E$:
$(v, w) \in C \;\Leftrightarrow\; dfsnum[w] < dfsnum[v]$ and $compnum[w] < compnum[v]$

DFS Theorem (cont.)

(g) For $v, w, z \in V$ with $v \longrightarrow_T^* w$, $(w, z) \in E$, and not $v \longrightarrow_T^* z$, holds:
- (i) $dfsnum[z] < dfsnum[v]$
- (ii) $(w, z) \in B \cup C$
- (iii) $compnum[z] > compnum[v] \Leftrightarrow (w, z) \in B$
- (iv) $compnum[z] < compnum[v] \Leftrightarrow (w, z) \in C$

Lemma 2 allows algorithmic classification from $dfsnum$ and $compnum$.

For acyclic graphs, $B = \emptyset$:
for all $(v, w) \in E$: $compnum[v] > compnum[w]$.

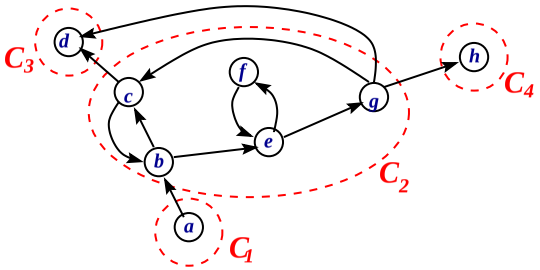
Thus, $num[v] = n + 1 - compnum[v]$ for all $v \in V$
yields a topological order $\rightarrow$ alternative to TOPSORT

Strongly Connected Components of a directed graph

An application of DFS.

A directed graph $G = (V, E)$ is strongly connected iff $\forall v, w \in V$: $v \longrightarrow^* w$.

The strongly connected components (SCC) of G are
the maximal (wrt. set inclusion) strongly connected subgraphs of G.



$C_1 = \{a\}$
$C_2 = \{b, c, e, f, g\}$
$C_3 = \{d\}$
$C_4 = \{h\}$

Application of SCCs:
Identifying loops in
low-level code

Computing the strongly connected components of a directed graph

The SCCs can be computed in time $\Theta(|V| + |E|)$ by an extension of DFS.

Use the transposed graph $G^T = (V, E^T)$ with $E^T = \{(v, u) : (u, v) \in E\}$.

G^T (adjacency list repres.) can be computed from G in time $O(|V| + |E|)$.

Note: G and G^T have the same SCCs.

1. $DepthFirstSearch(G)$ to compute $compnum[v]$ for each $v \in V$
2. compute G^T from G
3. $DepthFirstSearch(G^T)$, but in the main loop consider
the vertices v in order of decreasing $compnum[v]$
4. output the vertices of each tree in the DFS forest of step 3
as a separate SCC

Proof: see [Cormen/Leiserson/Rivest, chapter 23.5]