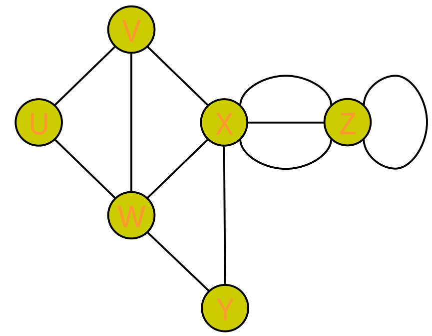
Algorithms

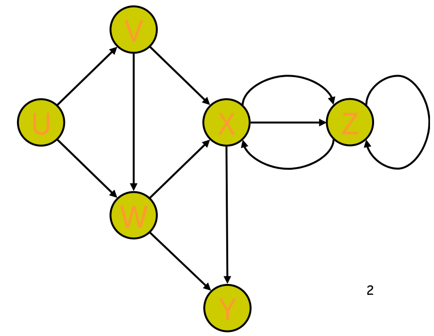
Dana Shapira

Lesson #5:
DFS,
Topological Sort
Strongly Connected Components

Graph

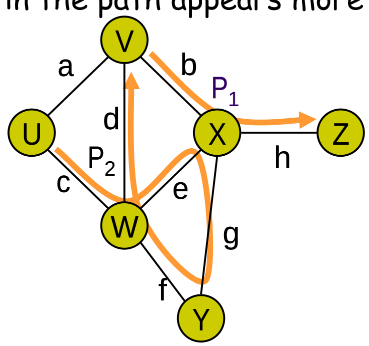
- A graph is a pair (*V*, *E*), where
  - *V* is a set of nodes, called vertices
  - *E* is a collection of pairs of vertices, called edges $\subseteq (V \times V)$
- If edge pairs are ordered, the graph is *directed*, otherwise *undirected*.





Paths

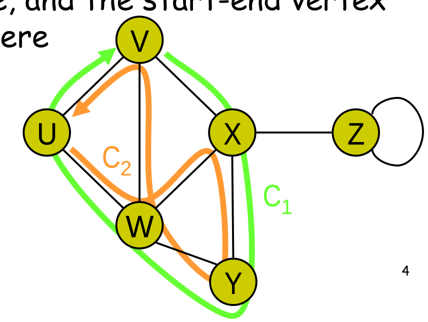
- **Path**
  - sequence of vertices $\{v_0, v_1, \ldots, v_p\}$ where $(v_i, v_{i+1}) \in E$
- **Simple path**
  - If no vertex in the path appears more than once



Cycles

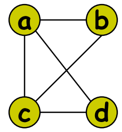
- **Circuit**
  - A path $\{v_0, v_1, \ldots, v_p\}$ where $v_0 = v_p$.
- **Simple circuit**
  - If no vertex, other than the start-end vertex, appears more than once, and the start-end vertex does not appear elsewhere

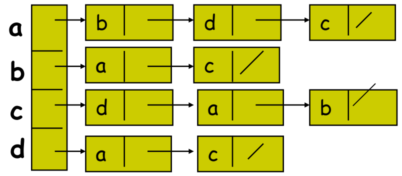
    in the circuit



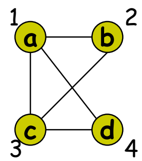
Graph Representations

- Adjacency Lists.





- Adjacency Matrix.



	1	2	3	4
1	0	1	1	1
2	1	0	1	0
3	1	1	0	1
4	1	0	1	0

Breadth-first Search

- **Input:** Graph $G = (V, E)$, directed or undirected, and ***source vertex*** $s \in V$.
- **Output:**
  for all $v \in V$
  - $d[v]$ = distance from s to v.
  - $\pi[v] = u$ such that (u, v) is the last edge on shortest path from s to v.
  - Builds ***breadth-first tree*** with root s that contains all reachable vertices.
- Colors the vertices to keep track of progress.
  - *White* – Undiscovered.
  - *Gray* – Discovered but not finished.
  - ***Black*** – Finished.

Depth-first Search

- *Input:* $G = (V, E)$, directed or undirected.
- *Output:*
  - for all $v \in V$.
    - $d[v]$ = ***discovery time*** (v turns from white to gray)
    - $f[v]$ = ***finishing time*** (v turns from gray to black)
  - $\pi[v]$: predecessor of $v = u$, such that v was discovered during the scan of u's adjacency list.
- Forest of *depth-first trees*:
  $G_\pi = (V, E_\pi)$ $\quad E_\pi = \{(\pi[v], v),\ v \in V \text{ and } \pi[v] \neq \textbf{null}\}$

DFS(G)

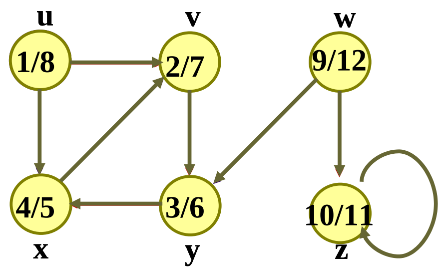
for each vertex $u \in V[G]$
    do $color[u] \leftarrow$ white
        $\pi[u] \leftarrow$ NULL
$time \leftarrow 0$
for each vertex $u \in V[G]$
    do if $color[u]$ = white
        then DFS-Visit(u)

Running time is $\theta(V+E)$

DFS-Visit(u)

$color[u] \leftarrow$ GRAY
$time \leftarrow time + 1$
$d[u] \leftarrow time$
for each $v \in Adj[u]$
    do if $color[v]$ = WHITE
        then $\pi[v] \leftarrow u$
            DFS-Visit(v)
$color[u] \leftarrow$ BLACK
$f[u] \leftarrow time \leftarrow time + 1$

Example (DFS)



Parenthesis Theorem

Theorem

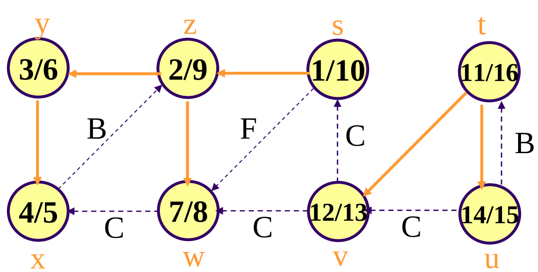
For all *u, v*, exactly one of the following holds:

1. $d[u] < f[u] < d[v] < f[v]$ or $d[v] < f[v] < d[u] < f[u]$ and neither *u* nor *v* is a descendant of the other.
2. $d[u] < d[v] < f[v] < f[u]$ and *v* is a descendant of *u*.
3. $d[v] < d[u] < f[u] < f[v]$ and *u* is a descendant of *v*.

- So $d[u] < d[v] < f[u] < f[v]$ *cannot* happen.
- *Corollary*

  v is a proper descendant of *u* if and only if $d[u] < d[v] < f[v] < f[u]$.

Example (Parenthesis Theorem)



(s (z (y (x x) y) (w w) z) s) (t (v v) (u u) t)

White-path Theorem

Theorem

v is a descendant of *u* if and only if at time *d*[*u*], there is a path $u \rightsquigarrow v$ consisting of only white vertices.

DFS: classification of edges

Theorem 3: In a depth-first search of an undirected graph $G=(V,E)$, every edge in E is either a tree edge or a back edge.

Proof: Let (u,v) be an an edge in E, and suppose that $d[u] < d[v]$. Then v must be discovered and finished before u is finished (*current*) since v is on u's adjacency list. If the edge (u,v) is explored in the direction $u \rightarrow v$, then u is *not_visited* until that time. If the edge is explored in the other direction, $u \leftarrow v$, then it is a back edge since u is still current at the time the edge is first explored.

Classification of Edges

- ***Tree edge:*** Edges in G_π. v was found by exploring (u, v).
- ***Back edge:*** (u, v), where u is a descendant of v in G_π .
- ***Forward edge:*** (u, v), where v is a descendant of u, but not a tree edge.
- ***Cross edge:*** any other edge. Can go between vertices in same depth-first tree or in different depth-first trees.

Identification of Edges

- Edge type for edge (u, v) can be identified when it is first explored by DFS.
- Identification is based on the color of v.
  - White – tree edge.
  - Gray – back edge.
  - Black – forward or cross edge.

Identification of Edges

Theorem:
In DFS of an undirected graph, we get only tree and back edges. No forward or cross edges.

Proof:

- Let $(u,v) \in E$. w.l.o.g let $d[u] < d[v]$.
  Then v must be discovered and finished before u is finished.
- If the edge (u,v) is explored first in the direction $u \rightarrow v$, then v is *white* until that time then it is a tree edge .
- If the edge is explored in the direction, $v \rightarrow u$, u is still gray at the time the edge is first explored, then it is a back edge.

Directed Acyclic Graph - DAG

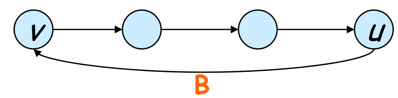
- partial order:
  - $a > b$ and $b > c \Rightarrow a > c$.
  - But may have a and b such that neither $a > b$ nor $b > a$.
- Can always make a total order (either $a > b$ or $b > a$ for all $a \neq b$) from a partial order.

Characterizing a DAG

Lemma
A directed graph G is acyclic iff a DFS of G yields no back edges.

Proof:

- $\Rightarrow$:
  - Suppose there is a back edge (u, v). Then v is an ancestor of u in depth-first forest.
  - Therefore, there is a path $v \rightsquigarrow u$, so $v \rightsquigarrow u \rightsquigarrow v$ is a cycle.

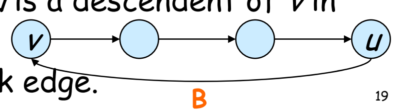


Characterizing a DAG

Lemma
A directed graph G is acyclic iff a DFS of G yields no back edges.

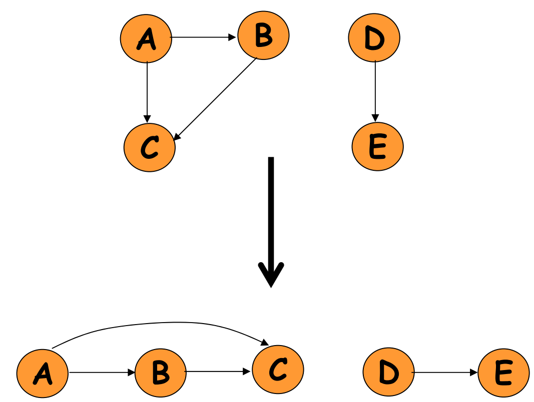
Proof (Cont.):

- $\Leftarrow$:
  - c : cycle in G, v : first vertex discovered in c, (u, v) : preceding edge in c.
  - At time $d[v]$, vertices of c form a white path $v \rightsquigarrow u$. Why?
  - By white-path theorem, u is a descendent of v in depth-first forest.
  - Therefore, (u, v) is a back edge.



Topological Sort

Want to "sort" a directed acyclic graph (DAG).



Topological Sort

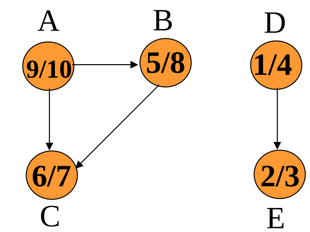
- Performed on a DAG.
- Linear ordering of the vertices of G such that if $(u, v) \in E$, then u appears somewhere before v.

<u>Topological-Sort (G)</u>

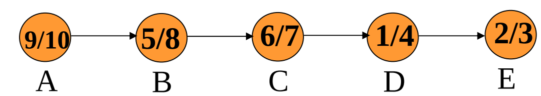
1. call DFS(G) to compute finishing times $f[v]$ for all $v \in V$
2. as each vertex is finished, insert it onto the front of a linked list
3. return the linked list of vertices

Running time is $\theta(V+E)$

Example



Linked List:

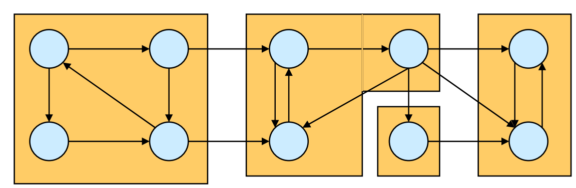


Correctness Proof

- Just need to show if $(u, v) \in E$, then $f[v] < f[u]$.
- When we explore (u, v), what are the colors of u and v?
  - u is gray.
  - Is v gray, too?
    - *No*, because then v would be an ancestor of u.
    - $\Rightarrow (u, v)$ is a back edge.
    - $\Rightarrow$ contradiction of Lemma (DAG has no back edges).
  - Is v white?
    - v is a descendant of u.
    - By parenthesis theorem, $d[u] < d[v] < \underline{f[v] < f[u]}$.
  - Is v black?
    - Then v is already finished.
    - Since we're exploring (u, v), we have not yet finished u.
    - $\Rightarrow f[v] < f[u]$.

Strongly Connected Components

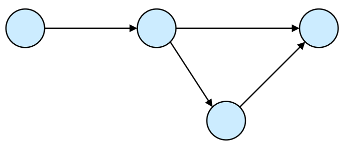
- G is strongly connected if every pair (u, v) of vertices in G is reachable from each other.
- A ***strongly connected component*** (***SCC***) of G is a maximal set of vertices $C \subseteq V$ such that for all $u, v \in C$, both $u \rightsquigarrow v$ and $v \rightsquigarrow u$ exist.



Component Graph

- $G^{SCC} = (V^{SCC}, E^{SCC})$.
- V^{SCC} has one vertex for each SCC in G.
- E^{SCC} has an edge if there is an edge between the corresponding SCC's in G.

G^{SCC} for the example considered:



G^{SCC} is a DAG

Lemma
Let C and C′ be distinct SCC's in G, let u, v ∈ C, u′, v′ ∈ C′, and suppose there is a path $u \rightsquigarrow u'$ in G. Then there cannot also be a path $v' \rightsquigarrow v$ in G.

Proof:

- Suppose there is a path $v' \rightsquigarrow v$ in G.
- Then there are paths $u \rightsquigarrow u' \rightsquigarrow v'$ and $v' \rightsquigarrow v \rightsquigarrow u$ in G.
- Therefore, u and v′ are reachable from each other, so they are not in separate SCC's.

Transpose of a Directed Graph

- G^T = transpose of directed G.
  - $G^T = (V, E^T)$, $E^T = \{(u, v) : (v, u) \in E\}$.
  - G^T is G with all edges reversed.
- Can create G^T in $\Theta(V + E)$ time if using adjacency lists.
- G and G^T have the same SCC's. (u and v are reachable from each other in G if and only if reachable from each other in G^T.)

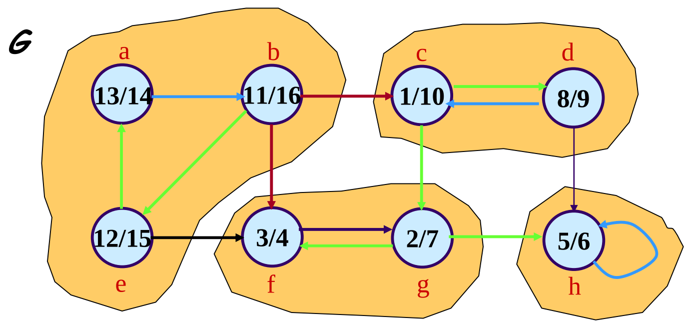
Algorithm to determine SCCs

SCC(*G*)

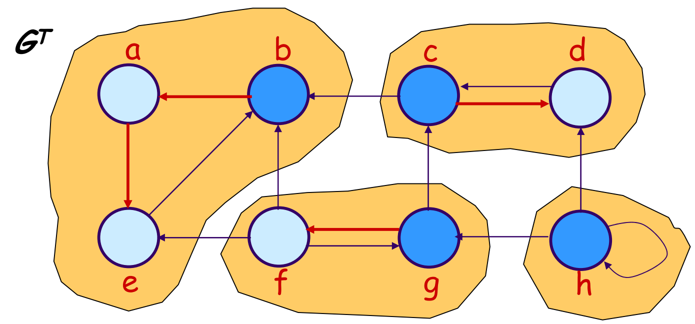
1. call DFS(*G*) to compute finishing times *f* [*u*] for all *u*
2. compute G^T
3. call DFS(G^T), but in the main loop, consider vertices in order of decreasing *f* [*u*] (as computed in first DFS)
4. output the vertices in each tree of the depth-first forest formed in second DFS as a separate SCC

Running time is $\theta(V+E)$

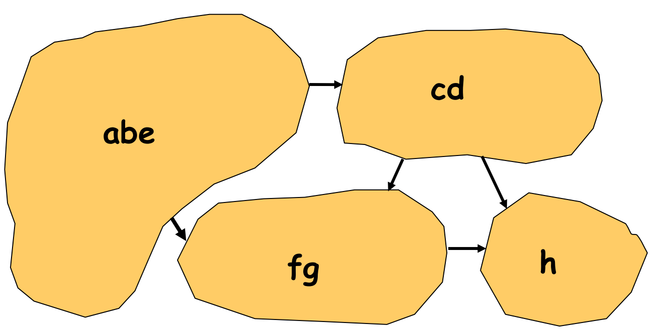
Example



Example



Example

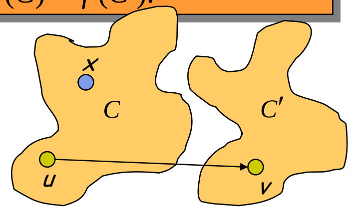


How does it work?

- **Idea:**
  - By considering vertices in second DFS in decreasing order of finishing times from first DFS, we are visiting vertices of the component graph in topologically sorted order.
  - Because we are running DFS on G^T, we will not be visiting any v from a u, where v and u are in different components.
- **Notation:**
  - $d[u]$ and $f[u]$ always refer to *first* DFS.
  - Extend notation for d and f to sets of vertices $U \subseteq V$:
    - $d(U) = \min_{u \in U}\{d[u]\}$ (earliest discovery time)
    - $f(U) = \max_{u \in U}\{f[u]\}$ (latest finishing time)

SCCs and DFS finishing times

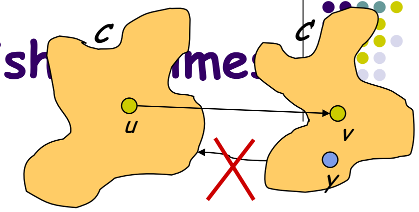
Lemma
Let C and C' be distinct SCC's in $G = (V, E)$. Suppose there is an edge $(u, v) \in E$ such that $u \in C$ and $v \in C'$. Then $f(C) > f(C')$.



Proof:

- Case 1: $d(C) < d(C')$
  - Let x be the first vertex discovered in C.
  - At time $d[x]$, all vertices in C and C' are white. Thus, there exist paths of white vertices from x to all vertices in C and C'.
  - By the white-path theorem, all vertices in C and C' are descendants of x in depth-first tree.
  - By the parenthesis theorem, $f[x] = f(C) > f(C')$.

SCCs and DFS finishing times



- Case 2: $d(C) > d(C')$
  - Let y be the first vertex discovered in C'.
  - At time $d[y]$, all vertices in C' are white and there is a white path from y to each vertex in C' $\Rightarrow$ all vertices in C' become descendants of y. Again, $f[y] = f(C')$.
  - At time $d[y]$, all vertices in C are also white.
  - By earlier lemma, since there is an edge (u, v), we cannot have a path from C to C'.
  - So no vertex in C is reachable from y.
  - Therefore, at time $f[y]$, all vertices in C are still white.
  - Therefore, for all $w \in C$, $f[w] > f[y]$, which implies that $f(C) > f(C')$.

SCCs and DFS finishing times

Corollary
Let C and C' be distinct SCC's in $G = (V, E)$. Suppose there is an edge
$(u, v) \in E^T$, where $u \in C$ and $v \in C'$. Then $f(C) < f(C')$.

Proof:

- $(u, v) \in E^T \Rightarrow (v, u) \in E$.
- Since SCC's of G and G^T are the same, $f(C') > f(C)$, by previous Lemma.

Correctness of SCC

- When we do the second DFS, on G^T, start with SCC C such that $f(C)$ is maximum.
  - The second DFS starts from some $x \in C$, and it visits all vertices in C.
  - The Corollary says that since $f(C) > f(C')$ for all $C \neq C'$, there are no edges from C to C' in G^T.
  - Therefore, DFS will visit *only* vertices in C.
  - Which means that the depth-first tree rooted at x contains *exactly* the vertices of C.

Correctness of SCC

- The next root chosen in the second DFS is in SCC C' such that $f(C')$ is maximum over all SCC's other than C.
  - DFS visits all vertices in C', but the only edges out of C' go to C, *which we've already visited*.
  - Therefore, the only tree edges will be to vertices in C'.
- We can continue the process.
- Each time we choose a root for the second DFS, it can reach only
  - vertices in its SCC—get tree edges to these,
  - vertices in SCC's *already visited* in second DFS—get *no* tree edges to these.