

# Depth-First Search

## CS 624 — Analysis of Algorithms

November 27, 2024



# Depth-First Search (DFS)

- ▶ DFS scans a graph (**directed** or **undirected**)  $G = (V, E)$ .  
Unlike BFS, no source vertex given!
- ▶ Adds 2 “**timestamps**” to each vertex, integers in  $[1 .. 2|V|]$ .
  - ▶  $d[v]$  = discovery time ( $v$  turns from White to Gray)
  - ▶  $f[v]$  = finishing time ( $v$  turns from Gray to Black)
- ▶  $\pi[v]$  = predecessor of  $v$ ; a vertex  $u$  such that  $v$  was discovered during the scan of  $u$ 's adjacency list
- ▶ Uses the same coloring scheme for vertices as BFS.
- ▶ The key mechanisms in this algorithm are the **timestamps** and a **stack** (implicit in the structure of recursive calls).

# The DFS Algorithm

---

## Algorithm 1 DFS( $G$ )

---

```
1: for each  $u \in V[G]$  do
2:    $color[u] \leftarrow \text{White}$ 
3:    $\pi[u] \leftarrow \text{NIL}$ 
4: end for
5:  $time \leftarrow 0$ 
6: for each  $u \in V[G]$  do
7:   if  $color[u] = \text{White}$  then
8:     DFS-Visit( $u$ )
9:   end if
10: end for
```

---

---

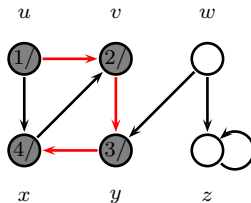
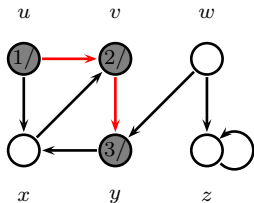
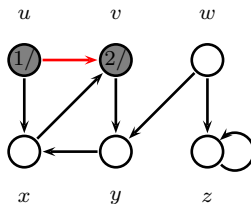
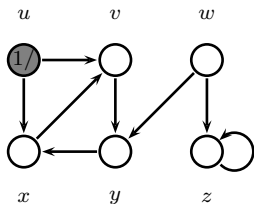
## Algorithm 2 DFS-Visit( $u$ )

---

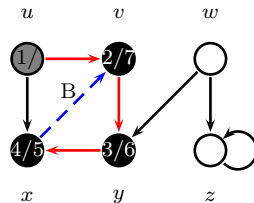
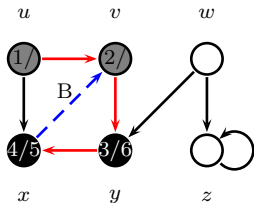
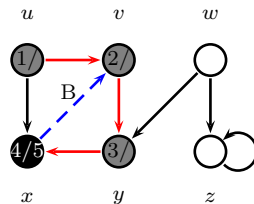
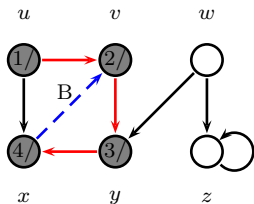
```
1:  $color[u] \leftarrow \text{Gray}$ 
2:  $time \leftarrow time + 1$ 
3:  $d[u] \leftarrow time$ 
4: for each  $v \in Adj[u]$  do
5:   if  $color[v] = \text{White}$  then
6:      $\pi[v] \leftarrow u$ 
7:     DFS-Visit( $v$ )
8:   end if
9: end for
10:  $color[u] \leftarrow \text{Black}$ 
11:  $time \leftarrow time + 1$ 
12:  $f[u].time \leftarrow time$ 
```

---

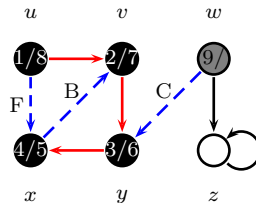
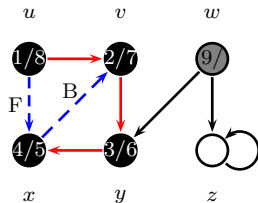
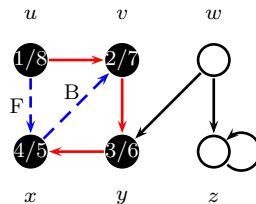
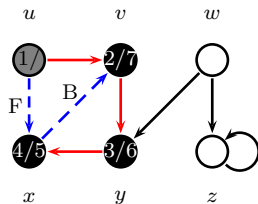
# The DFS Algorithm – Example



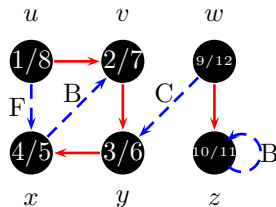
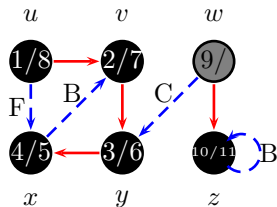
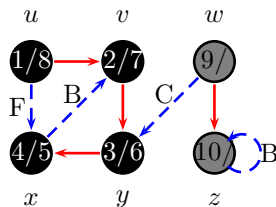
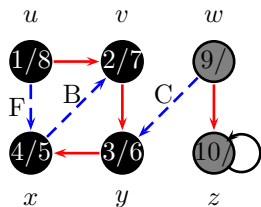
# The DFS Algorithm – Example



# The DFS Algorithm – Example



# The DFS Algorithm – Example



# The DFS Algorithm – Runtime and Properties

- ▶ The loops on lines 1-2 & 5-7 take  $\Theta(V)$  time, excluding time to execute DFS-Visit.
- ▶ DFS-Visit is called once for each white vertex  $v \in V$  when it's painted gray the first time.
- ▶ Lines 3-6 of DFS-Visit is executed  $|Adj[v]|$  times. The total cost of executing DFS-Visit is  $\sum_{v \in V} |Adj[v]| = \Theta(E)$ .
- ▶ Total running time of DFS is  $\Theta(V + E)$ .



# The Parenthesis Theorem

## Theorem (Parenthesis Theorem, Alternative Version)

Let  $G = (V, E)$  and let  $u, v \in V$  and suppose  $d[u] < d[v]$  after DFS.

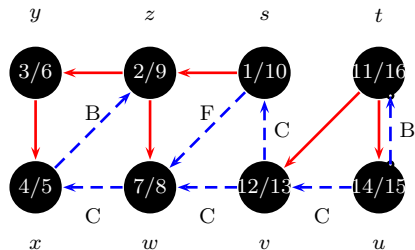
Then exactly one of the following cases holds:

1.  $d[u] < f[u] < d[v] < f[v]$ , 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  $u$

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

- ▶ OK:  $()[]([)][()]$
- ▶ Not OK:  $([)][([)]$

# The Parenthesis Theorem – Example



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

# The 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  $v$ .*

## Corollary

*$v$  is a proper descendant of  $u$  iff  $d[u] < d[v] < f[v] < f[u]$ .*

# The Parenthesis Theorem

## Proof.

- ▶ If  $start[x] < start[y] < finish[x]$  then  $x$  is on the stack when  $y$  is first reached.
- ▶ Therefore the processing of  $y$  starts while  $x$  is on the stack, and so it also must finish while  $x$  is on the stack:
- ▶ we have  $start[x] < start[y] < finish[y] < finish[x]$ .
- ▶ The case when  $start[y] < start[x] < finish[y]$  is handled in the same way.



- ▶ Another way to state the parenthesis nesting property is that given any two nodes  $x$  and  $y$ , the intervals  $[start[x], finish[x]]$  and  $[start[y], finish[y]]$  must be either nested or disjoint.

# Depth First Trees

- ▶ Predecessor subgraph defined slightly different from that of BFS.
- ▶ The predecessor subgraph of DFS is  $G_\pi = (V, E_\pi)$  where  $E_\pi = \{(\pi[v], v) : v \in V \text{ and } \pi[v] \neq \text{NIL}\}$ .
- ▶ How does it differ from that of BFS?
- ▶ The predecessor subgraph  $G_\pi$  forms a depth-first forest composed of several depth-first trees.
- ▶ The edges in  $E_\pi$  are called tree edges.

## Definition (Forest)

An acyclic graph  $G$  that may be disconnected.

# White Path Theorem

## Theorem

*$v$  is a tree descendant of  $u$  if and only if at time  $d[u]$ , there is a path  $u \rightsquigarrow v$  consisting of only white vertices (except for  $u$ , which was just colored gray).*

## Proof.

**One direction:** (if  $v$  is a tree descendant of  $u$  then there is a white path  $u \rightsquigarrow v$  at time  $d[u]$ ) is obvious from the definition of a tree descendant (see the parenthesis theorem). □

# White Path Theorem

## Cont. – Reverse Direction.

- ▶ Is it possible that  $v$  is not a descendant of  $u$  in the DFS forest?
- ▶ By induction on all the vertices along the path: Of course  $u$  is a descendant of itself.
- ▶ Let us pick any vertex  $p$  on the path other than the first vertex  $u$ , and let  $q$  be the previous vertex on the path (so it can be that  $q = u$ ).
- ▶ We assume that all vertices along the path from  $u$  to  $q$  inclusive are descendants of  $u$  (inductive hypothesis).
- ▶ We will argue that  $p$  is also a descendant of  $u$ .



# White Path Theorem

## Cont. – Reverse Direction.

- ▶ At time  $d[u]$  vertex  $p$  is white [by assumption about the white path], so  $d[u] < d[p]$ .
- ▶ But there is an edge from  $q$  to  $p$ , so  $q$  must explore this edge before finishing.
- ▶ At the time when the edge is explored,  $p$  can be:
- ▶ **WHITE**, then  $p$  becomes a descendant of  $q$ , and so of  $u$ .
- ▶ **BLACK**, then  $f[p] < f[q]$  [because  $f[p]$  must have already been assigned by that time, while  $f[q]$  will get assigned later].
- ▶ But since  $q$  is a descendant of  $u$  [not necessarily proper],  $f[q] \leq f[u]$ , we have  $d[u] < d[p] < f[p] < f[q] \leq f[u]$ , and we can use the Parenthesis Theorem to conclude that  $p$  is a descendant of  $u$ .



# White Path Theorem

## Cont. – Reverse Direction.

- ▶ **GRAY**, then  $p$  is already discovered, while  $q$  is not yet finished, so  $d[p] < f[q]$ .
- ▶ Since  $q$  is a descendant of  $u$  [not necessarily proper], by the Parenthesis Theorem,  $f[q] \leq f[u]$ .
- ▶ Hence  $d[u] < d[p] < f[q] \leq f[u]$ . So  $d[p]$  belongs to the set  $\{d[u], \dots, f[u]\}$ , and so we can use the the Parenthesis Theorem again to conclude that  $p$  must be a descendant of  $u$ .
- ▶ The conclusion thus far is that  $p$  is a descendant of  $u$ . Now, as long as there is a vertex on the remainder of the path from  $p$  to  $v$ , we can repeatedly apply the inductive argument, and finally conclude that the vertex  $v$  is a descendant of  $u$ , too.



# Classification of Edges

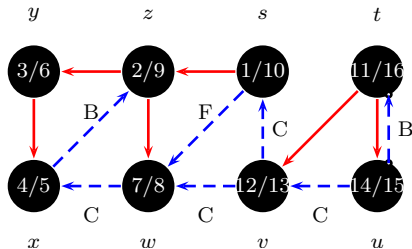
- ▶ **Tree edge:** in the depth-first forest. Found by exploring  $(u, v)$ .
- ▶ **Back edge:**  $(u, v)$ , where  $u$  is a descendant of  $v$  (in the depth-first tree).
- ▶ **Forward edge:**  $(u, v)$ , where  $v$  is a descendant of  $u$ , but  $(u, v)$  is not a tree edge.
- ▶ **Cross edge:** any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

Edge type for edge  $(u, v)$  can be identified when it is first explored by DFS based on the color of  $v$ .

White  $\rightarrow$  tree edge. Gray  $\rightarrow$  back edge.

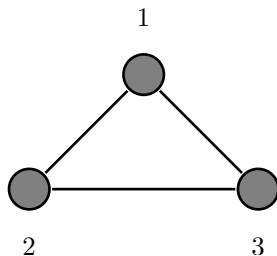
Black  $\rightarrow$  forward or cross edge.

# Classification of Edges



The edge  $x \rightarrow z$  is discovered when exploring  $z$ , so it is a back edge.

# Classification of Edges



## Theorem

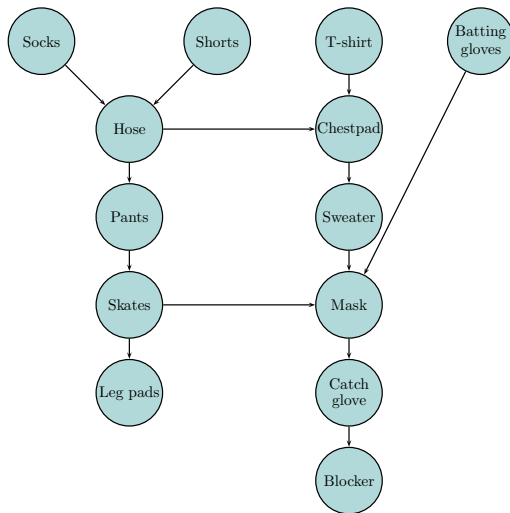
*In DFS of an undirected graph, we get only tree and back edges. No forward or cross edges.*

Starting from 1, either 2 discovers 3 or vice versa, therefore one of them is the other's descendant, Hence no cross edges.

# Directed Acyclic Graph (DAG)

- ▶ DAG – Directed graph with no cycles.
- ▶ Good for modeling processes and structures that have a 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.

# Directed Acyclic Graph (DAG) – Example



# Characterizing a DAG

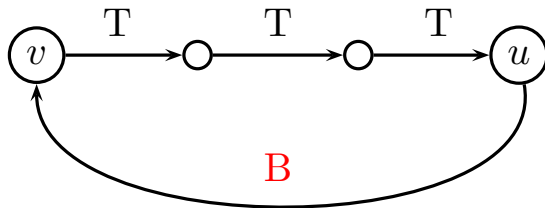
## Lemma

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

## Proof.

$\Rightarrow$  Show that back edge  $\rightarrow$  cycle:

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

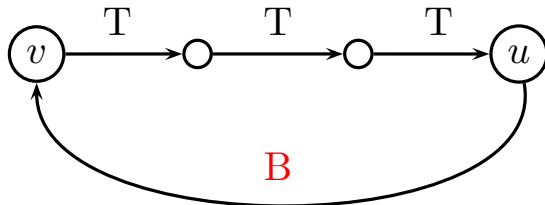


# Characterizing a DAG

## Proof.

$\Rightarrow$ : Show that a cycle implies a back edge.

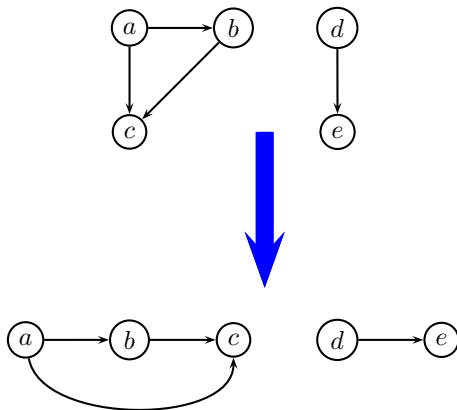
- ▶  $c$  : cycle in  $G$ ,  $u$  : first vertex discovered in  $c$ ,  $(v, u)$  : preceding edge in  $c$ .
- ▶ At time  $d[v]$ , vertices of  $c$  form a white path  $u \rightsquigarrow v$ . Why?
- ▶ By white-path theorem,  $v$  is a descendent of  $u$  in depth-first forest.
- ▶ Therefore,  $(v, u)$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





# Topological Sorting

- ▶ We want to “sort” a DAG.
- ▶ Think of original DAG as a partial order.
- ▶ We want a total order that extends this partial order.



# Topological Sorting

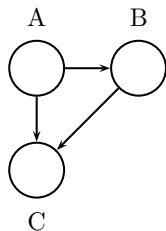
- ▶ Performed on a DAG.
- ▶ Linear ordering of the vertices of  $G$  such that if  $(u, v) \in E$ , then  $u$  appears somewhere before  $v$ .

## TopologicalSort( $G$ )

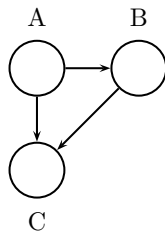
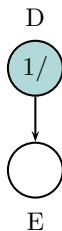
1. call DFS( $G$ ) to compute finishing times  $f[v]$  for all  $v \in V$
2. as each vertex is finished, insert it onto the front of a linked list
3. return the linked list of vertices (with *decreasing* finish times)

Runtime –  $\Theta(V + E)$

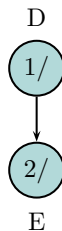
# Topological Sorting – Example



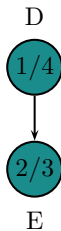
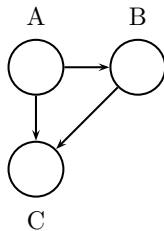
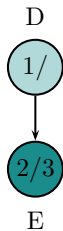
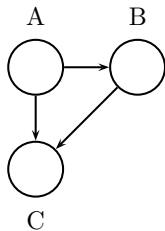
**Linked list:**



**Linked list:**



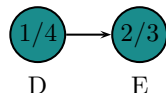
# Topological Sorting – Example



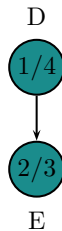
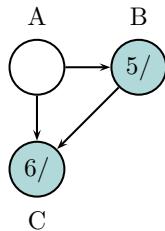
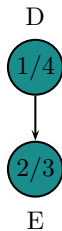
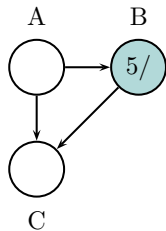
Linked list:



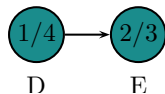
Linked list:



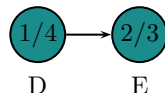
# Topological Sorting – Example



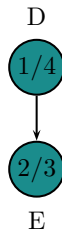
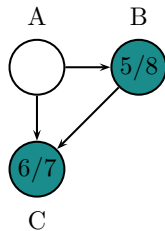
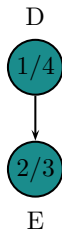
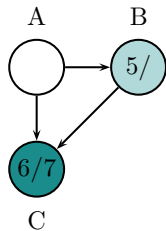
Linked list:



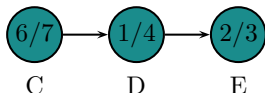
Linked list:



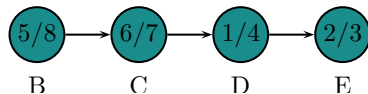
# Topological Sorting – Example



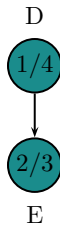
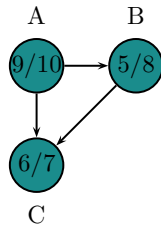
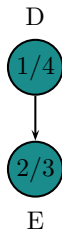
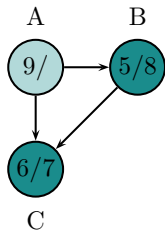
Linked list:



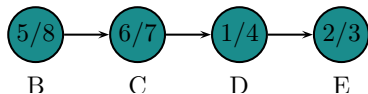
Linked list:



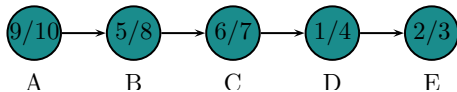
# Topological Sorting – Example



Linked list:



Linked list:



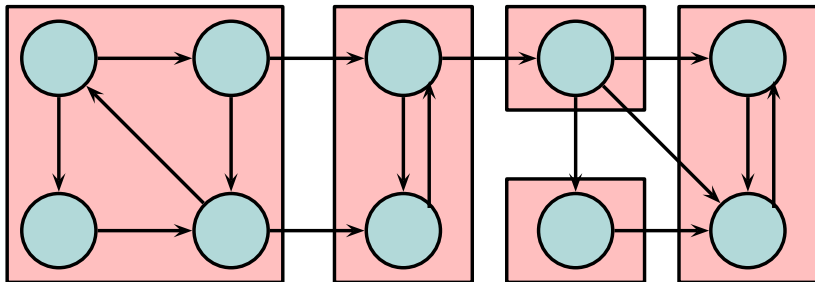
# Topological Sorting – Proof of Correctness

- ▶ Just need to show if  $(u, v) \in E$ , then  $f[v] < f[u]$ .
- ▶ When we explore  $(u, v)$  then  $u$  is gray. What is the color of  $v$ ?
- ▶ Is  $v$  **gray**?
- ▶ No, because then  $v$  would be ancestor of  $u$ .  $\Rightarrow (u, v)$  is a back edge, which contradicts the fact that A DAG has no back edges.
- ▶ Is  $v$  **white**?
- ▶ Then becomes descendant of  $u$ .
- ▶ By parenthesis theorem,  $d[u] < d[v] < f[v] < f[u]$ .
- ▶ Is  $v$  **black**?
- ▶ Then  $v$  is already finished.
- ▶ Since we're exploring  $(u, v)$ , we have not yet finished  $u$ .
- ▶ Therefore,  $f[v] < f[u]$ .



# Strongly Connected Components

- ▶  $G$  is strongly connected if every pair  $(u, v)$  of vertices in  $G$  is reachable from one another.
- ▶ A strongly connected component (SCC) of  $G$  is a maximal set of vertices  $C \subseteq V$  such that for all  $u, v \in C$ , there is a path from  $u$  to  $v$  and from  $v$  to  $u$ .



## Theorem

*Let  $C$  and  $C'$  be distinct SCC's in  $G$ , let  $u, v \in C, u', v' \in C'$ , 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 from  $v'$  to  $v$  in  $G$ .
- ▶ Then there are paths from  $u$  to  $u'$  to  $v'$  and from  $v'$  to  $v$  to  $u$  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  $G^T$ ).

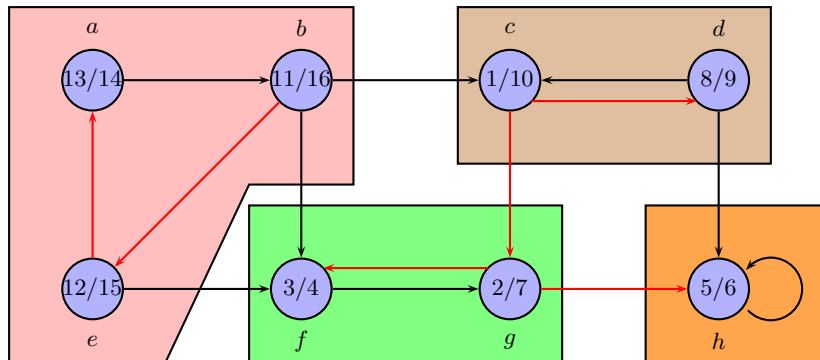
# Algorithm to Determine SCC

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

Runtime –  $\Theta(V + E)$

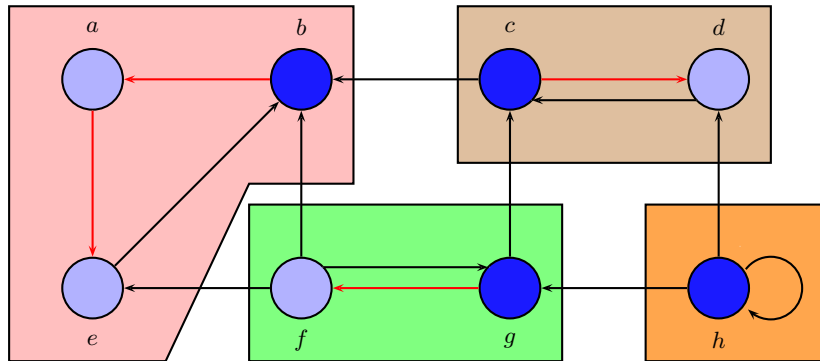
# Example

$G$

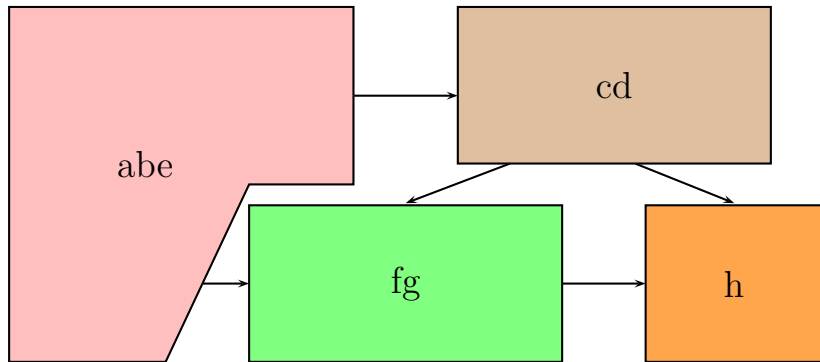


# Example

$G^T$



# 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 GT, 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)



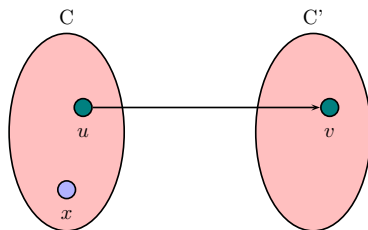
# SCCs and DFS finishing times

## 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  $f(C) > f(C')$ .

### 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 unvisited. Thus, there exist paths of unvisited vertices from  $x$  to all vertices in  $C$  and  $C'$ .
- ▶ All vertices in  $C$  and  $C'$  are descendants of  $x$  in depth-first tree.
- ▶ Therefore,  $f[x] = f(C) > f(C')$ .



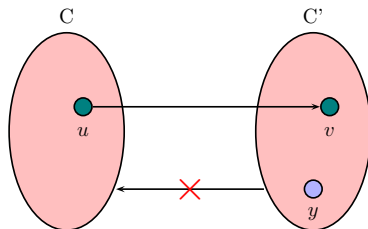
# SCCs and DFS finishing times

## 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  $f(C) > f(C')$ .

### Case 2: $d(C) > d(C')$ .

- ▶ Let  $y$  be the first vertex discovered in  $C'$ .
- ▶ At time  $d[y]$ , all vertices in  $C'$  are unvisited and there is an unvisited path from  $y$  to each vertex in  $C'$  all vertices in  $C'$  become descendants of  $y$ . Again,  $f[y] = f(C')$ .
- ▶ At time  $d[y]$ , all vertices in  $C$  are also unvisited.



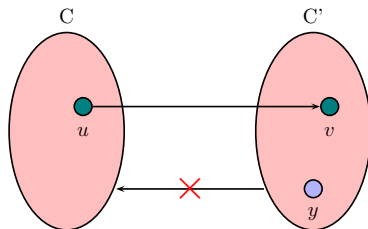
# SCCs and DFS finishing times

## 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  $f(C) > f(C')$ .

## Case 2: $d(C) > d(C')$ .

- ▶ 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 former 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  $C$ .
- ▶ Corollary above 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.