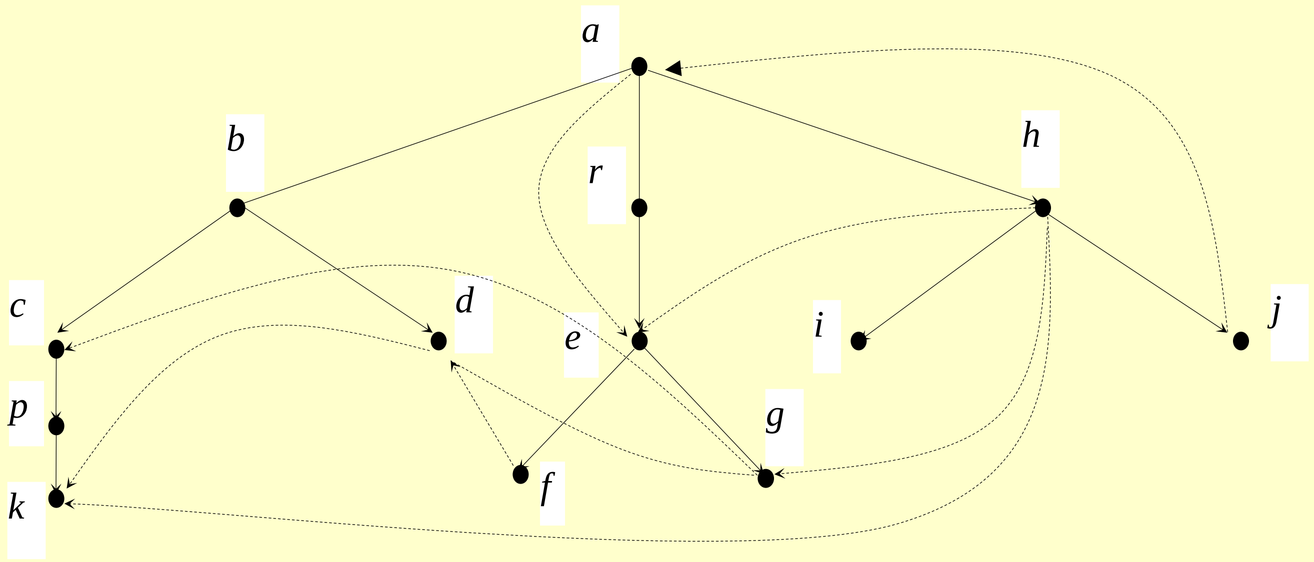
Topological Order and SCC

- Edge classification
- Topological order
- Recognition of strongly connected components

Classification of Edges

- It is well known that the preorder (depth-first) traversal of $G(V, E)$ introduces a spanning tree (forest) T. With respect to T, $E(G)$ can be classified into four groups:
- *tree edges* (E_{treet}): edges appearing in T.
- *cross edges* (E_{cross}): any edge $u \rightarrow v$ such that u and v are not on the same path in T.
- *forward edges* ($E_{forward}$): any edge $u \rightarrow v$ not appearing in T, but there exists a path from u to v in T
- *back edges* (E_{back}): any edge $u \rightarrow v$ not appearing in T, but there exists a path from v to u in T.

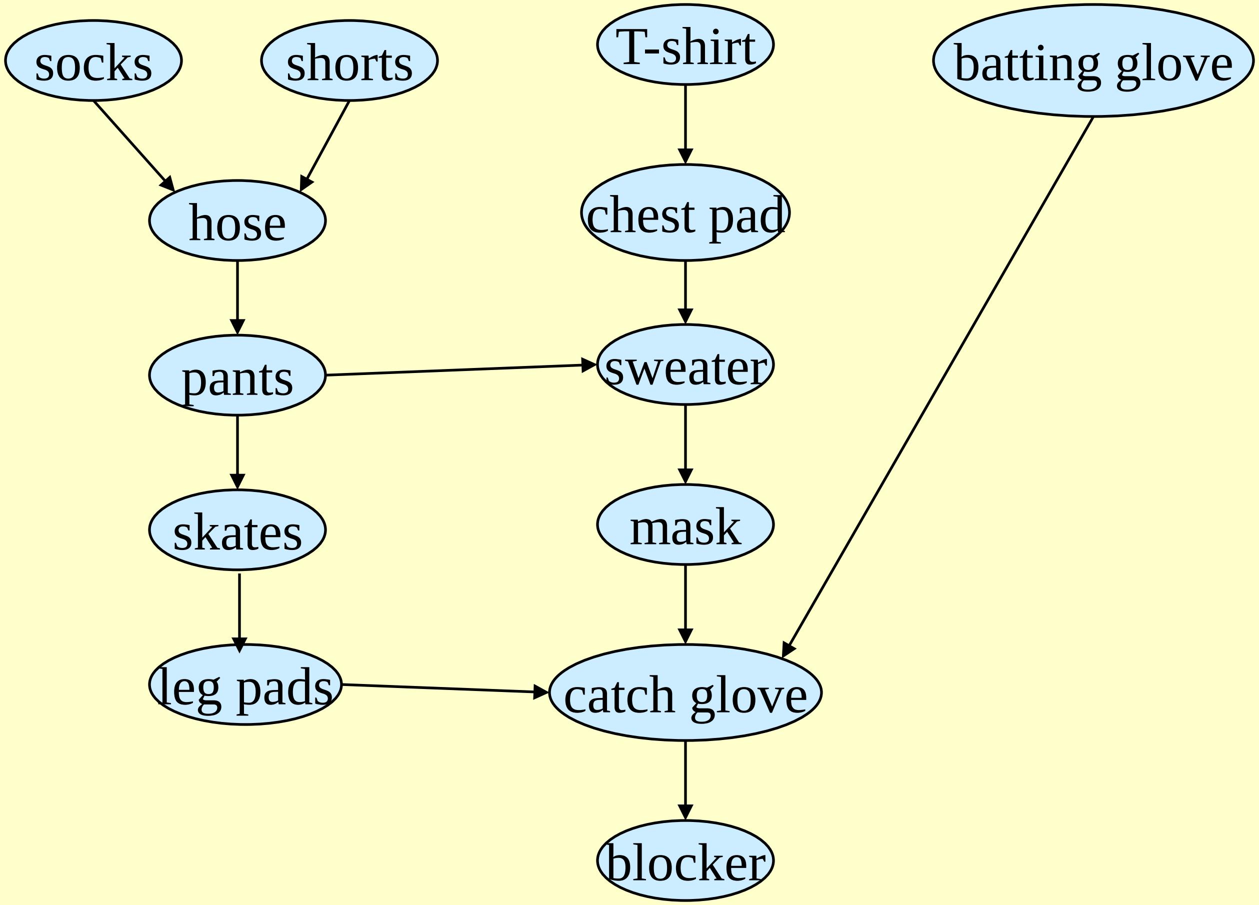

a
b
r
h
c
d
e
i
j
p
g
f
k

Directed Acyclic Graph

- DAG – **D**irected **A**cyclic **G**raph (directed graph with no cycles)
- Used for modeling processes and structures that have a **partial order:**
  - » Let a, b, c be three elements in a set U.
  - » $a > b$ and $b > c \Rightarrow a > c$.
  - » But may have a and b such that neither $a > b$ nor $b > a$.
- We can always make a **total order** (either $a > b$ or $b > a$ for all $a \neq b$) from a partial order.

Example

DAG of dependencies for putting on goalie equipment.

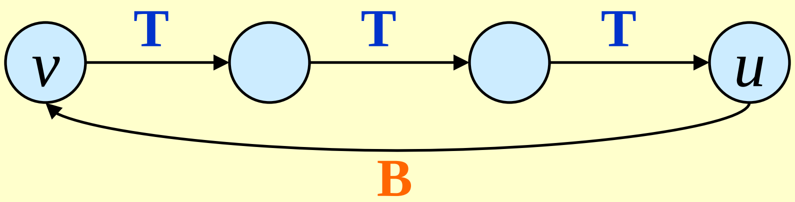


Characterizing a DAG

Lemma 22.11
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 $v \rightsquigarrow u \rightarrow v$ is a cycle.



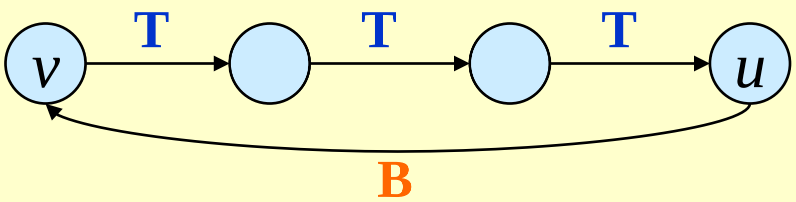
Characterizing a DAG

Lemma 22.11

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

Proof (Contd.):

- $\Leftarrow$: Show that a cycle implies a back edge.
  - » 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.



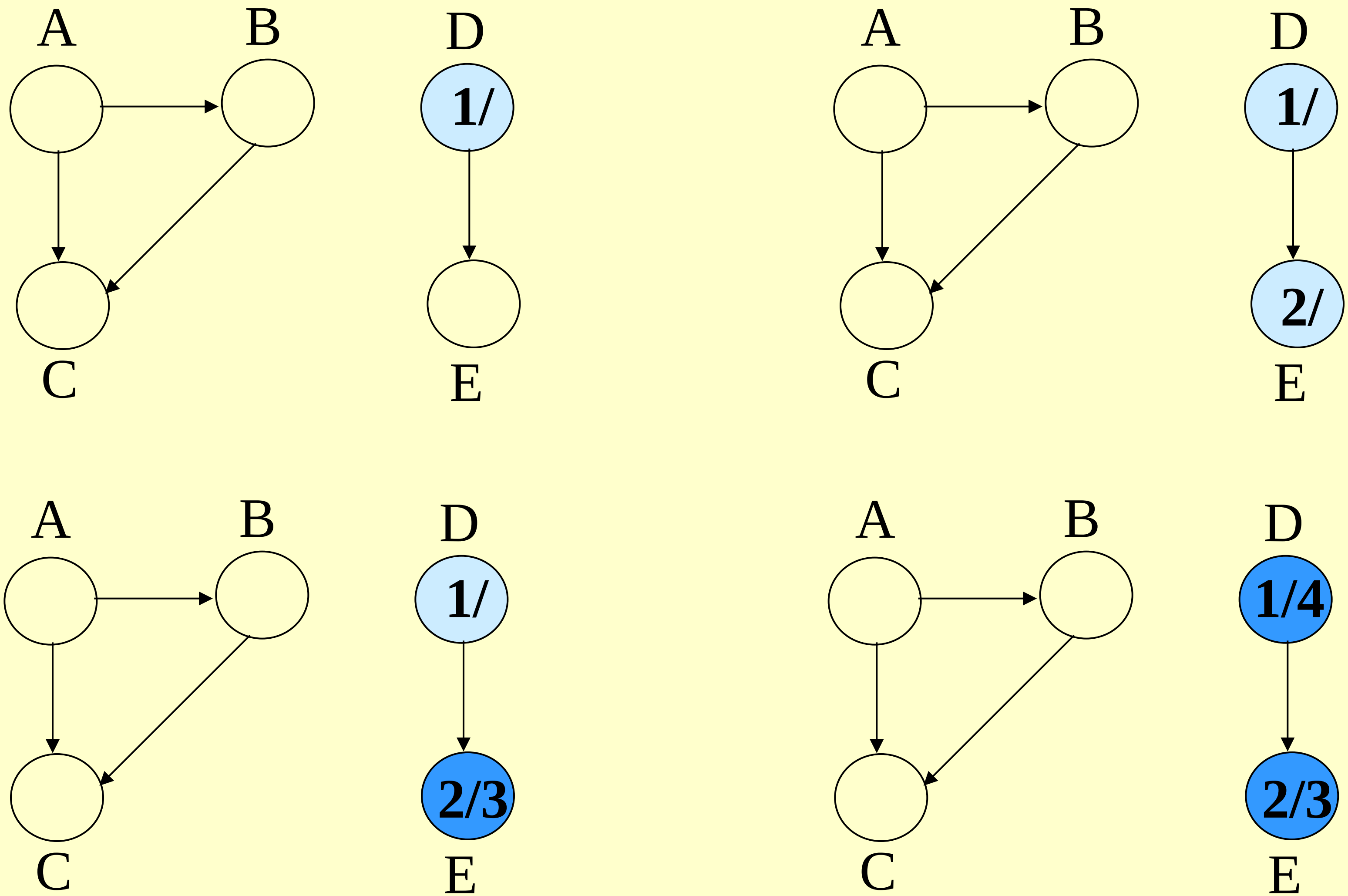
Depth-first Search

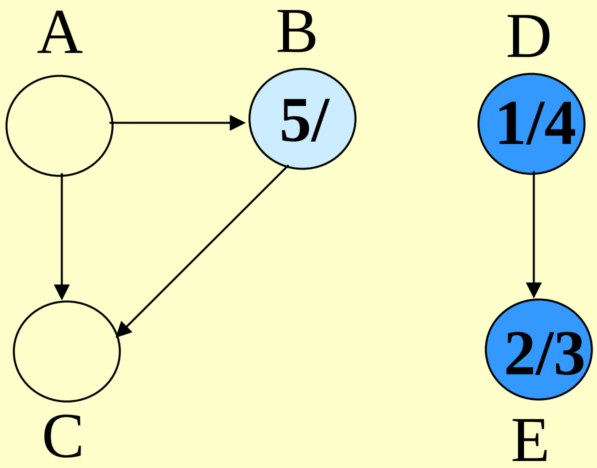
- **Input:** $G = (V, E)$, directed or undirected. No source vertex given!
- **Output:**

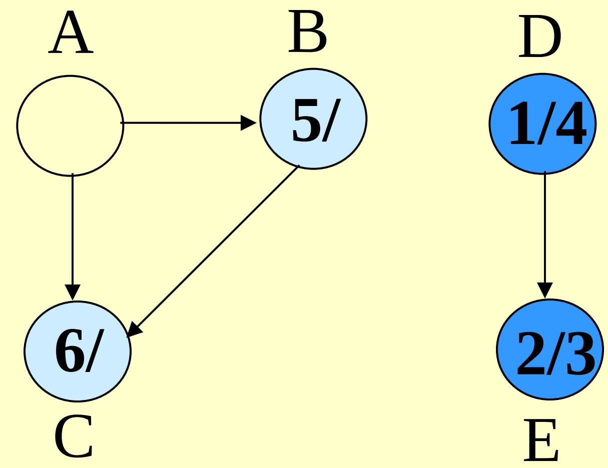
2 **timestamps** on each vertex. Integers between 1 and $2|V|$.

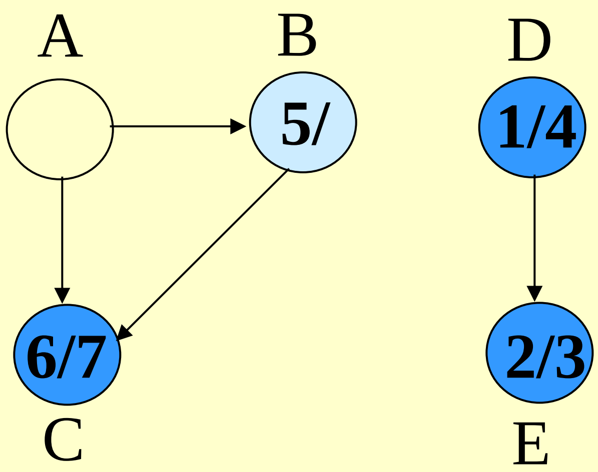
  - $d[v]$ = ***discovery time***
  - $f[v]$ = ***finishing time***

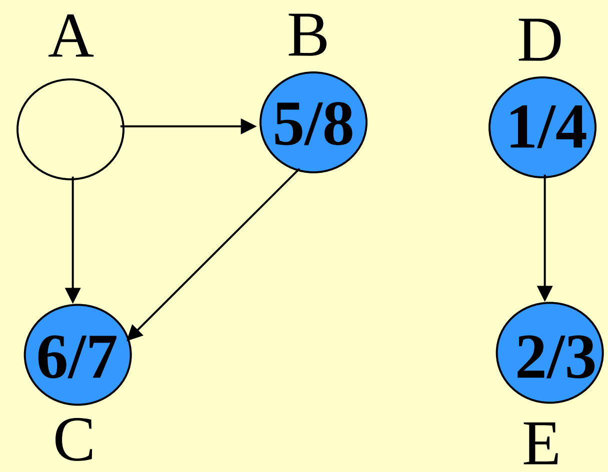
Discovery time - the first time it is encountered during the search.
Finishing time - A vertex is "finished" if it is a leaf node or all vertices adjacent to it have been finished.

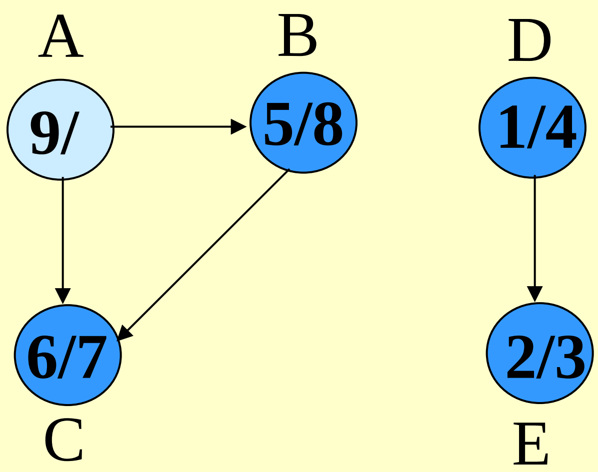

A
B
D
1/
C
E
A
B
D
1/
2/
C
E
A
B
D
1/
2/3
C
E
A
B
D
1/4
2/3
C
E

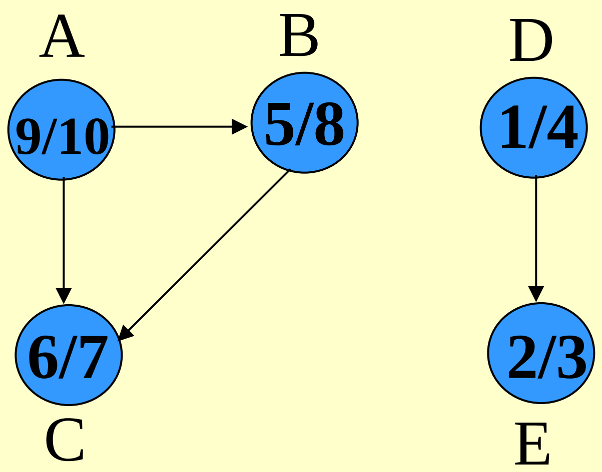

A
B
5/
D
1/4
2/3
C
E


A
B
5/
D
1/4
6/
2/3
C
E


A
B
5/
D
1/4
6/7
2/3
C
E

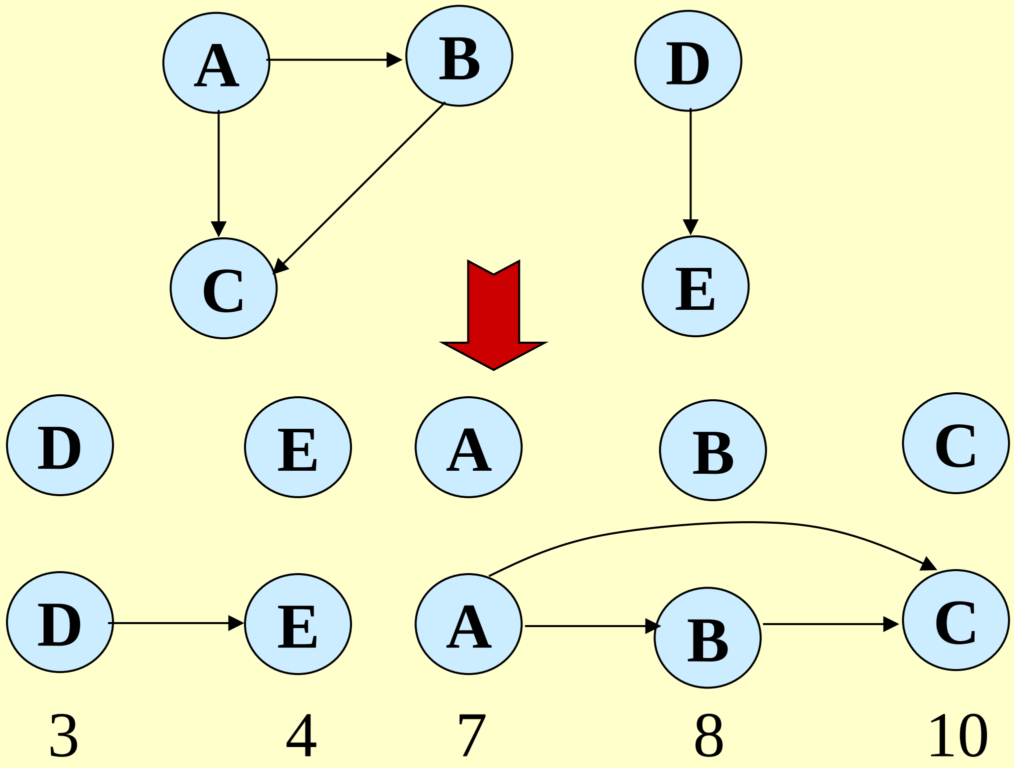

A
B
5/8
D
1/4
6/7
2/3
C
E


A
B
D
9/
5/8
1/4
6/7
2/3
C
E


A
B
D
9/10
5/8
1/4
6/7
2/3
C
E

Topological Sort

Sort a directed acyclic graph (DAG) by the nodes' finishing times.



Think of original DAG as a **partial order**.

By sorting, we get a **total order** that extends this partial order.

Topological Sort

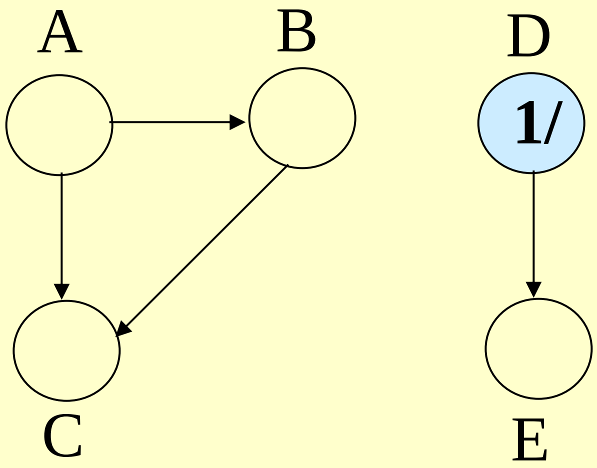
- Performed on a DAG.
- Linear ordering of the vertices of G such that if $(u, v) \in E$, then u appears somewhere before v.

Topological-Sort (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

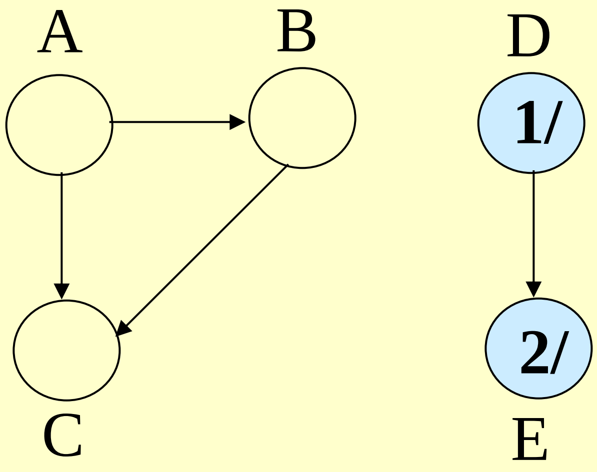
Time: $O(|V| + |E|)$.

Example



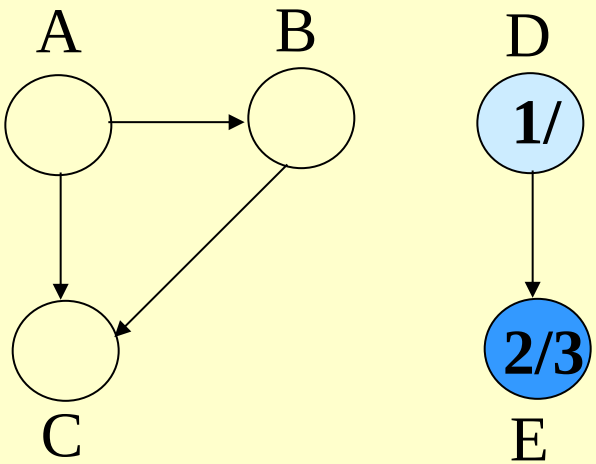
Linked List:

Example

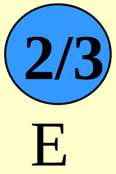


Linked List:

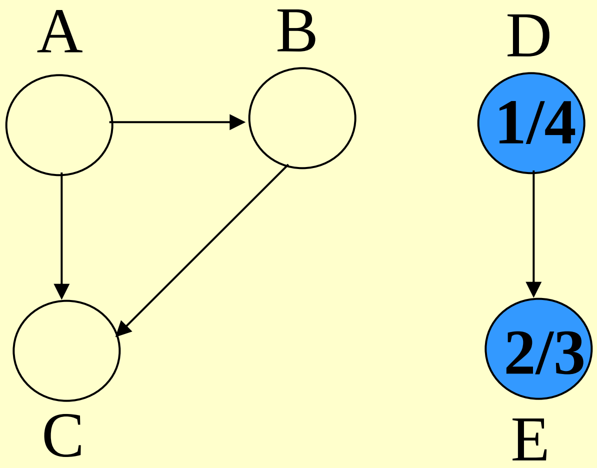
Example



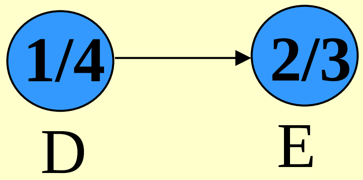
Linked List:



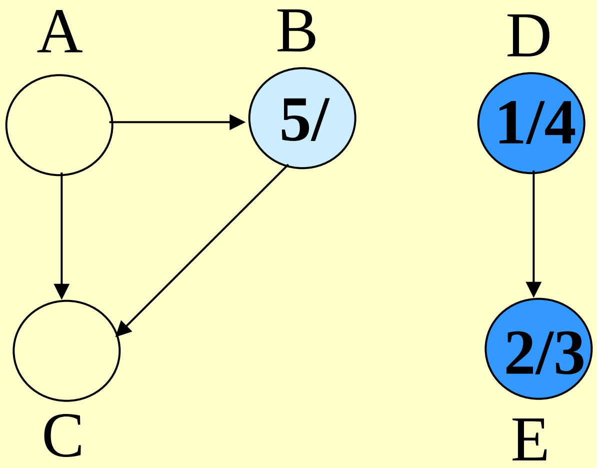
Example



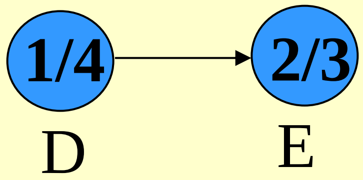
Linked List:



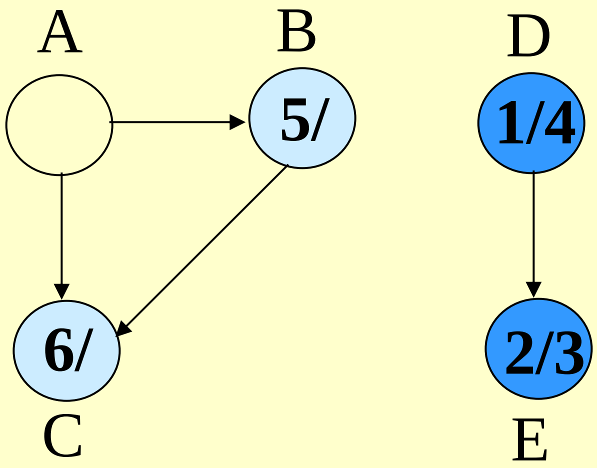
Example



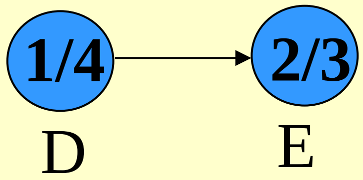
Linked List:



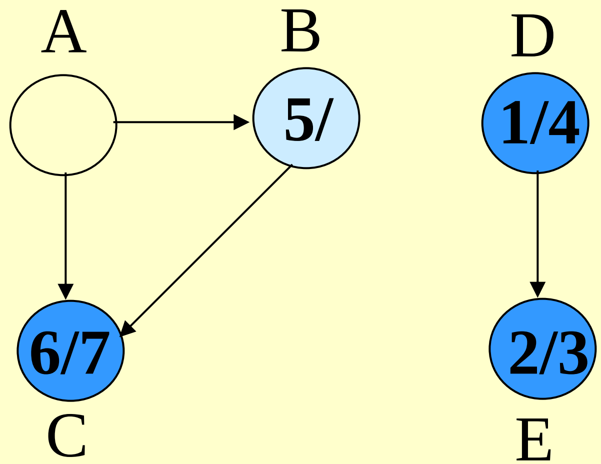
Example



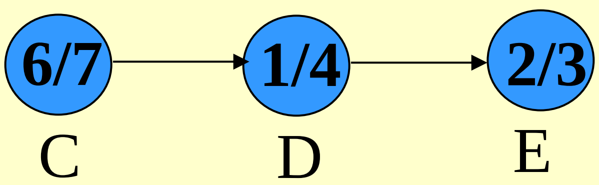
Linked List:



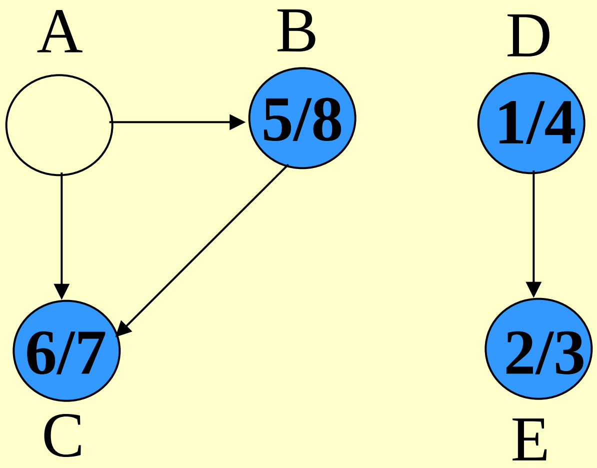
Example



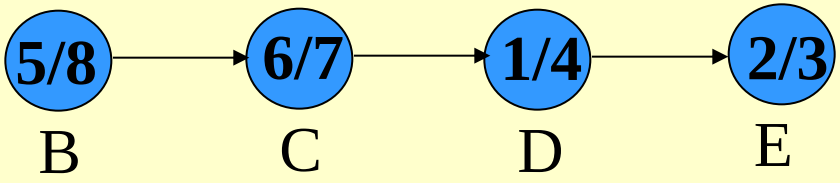
Linked List:



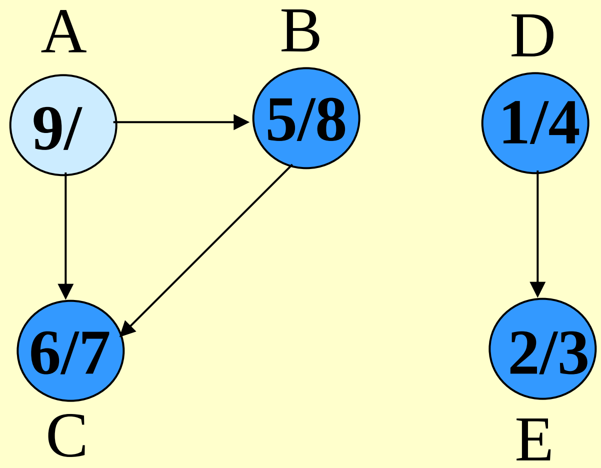
Example



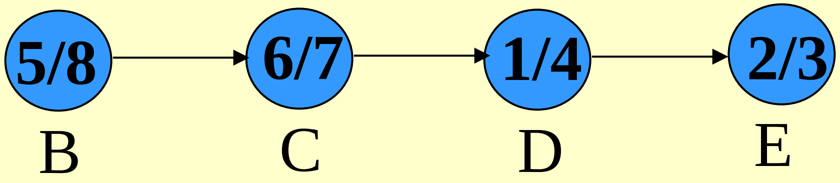
Linked List:



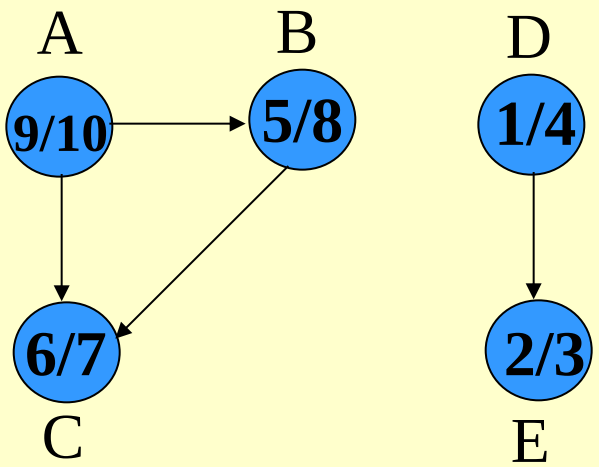
Example



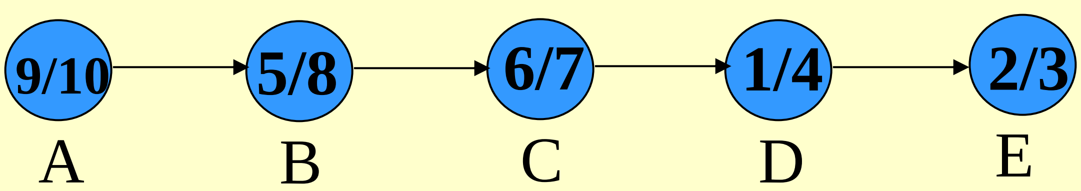
Linked List:



Example



Linked List:

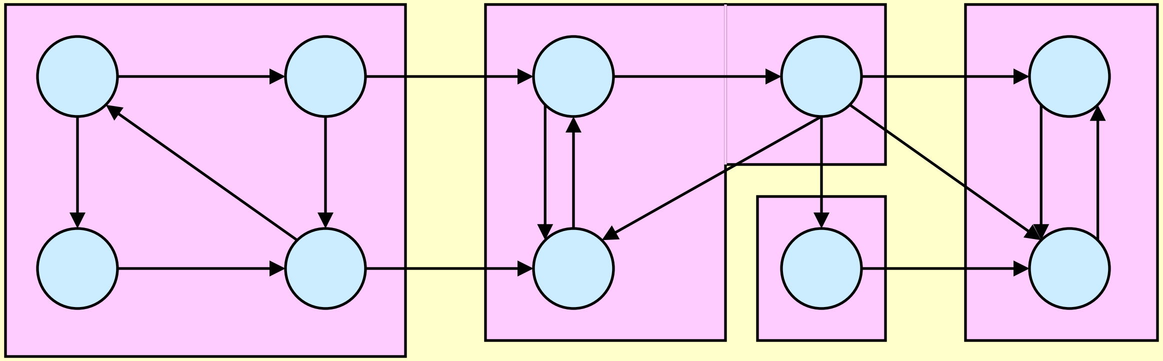


Correctness Proof

- 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 ancestor of $u \Rightarrow (u, v)$ is a back edge.
    - $\Rightarrow$ contradiction of Lemma 22.11 (dag has no back edges).
  - » Is v white?
    - Then becomes 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.
    - Therefore, $f[v] < f[u]$.

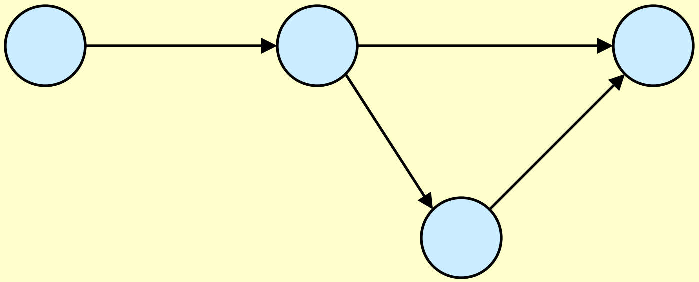
Strongly Connected Components

- 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 $u, v \in C$, both $u \leadsto v$ and $v \leadsto u$ 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's an edge between the corresponding SCC's in G.
- G^{SCC} for the example considered:



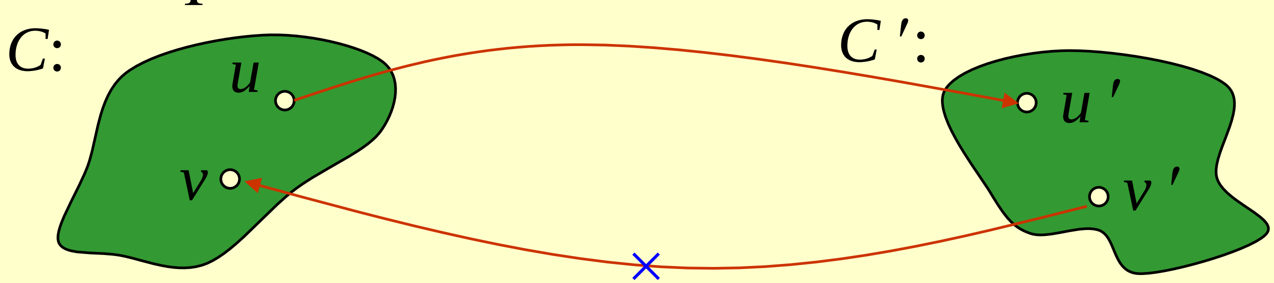
G^{SCC} is a DAG

Lemma 22.13

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 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 $O(|V| + |E|)$ 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.)

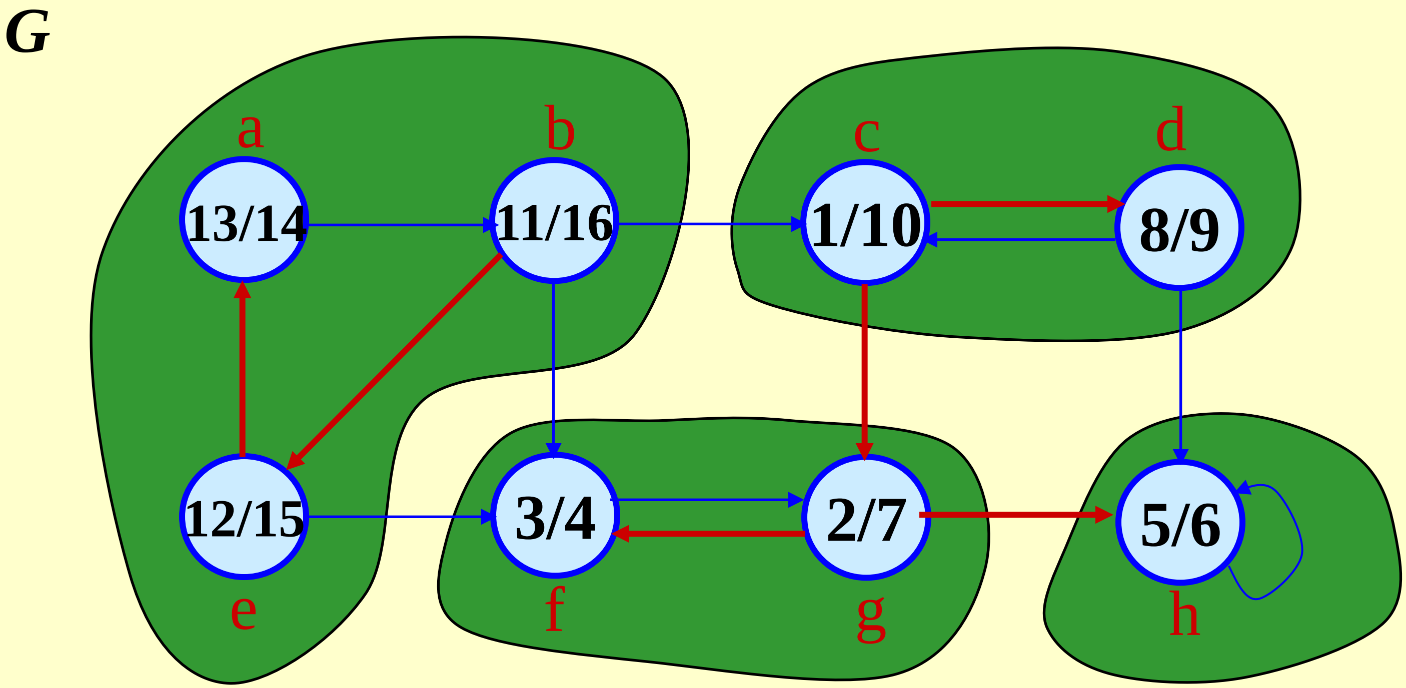
Algorithm to determine SCCs

SCC(G)

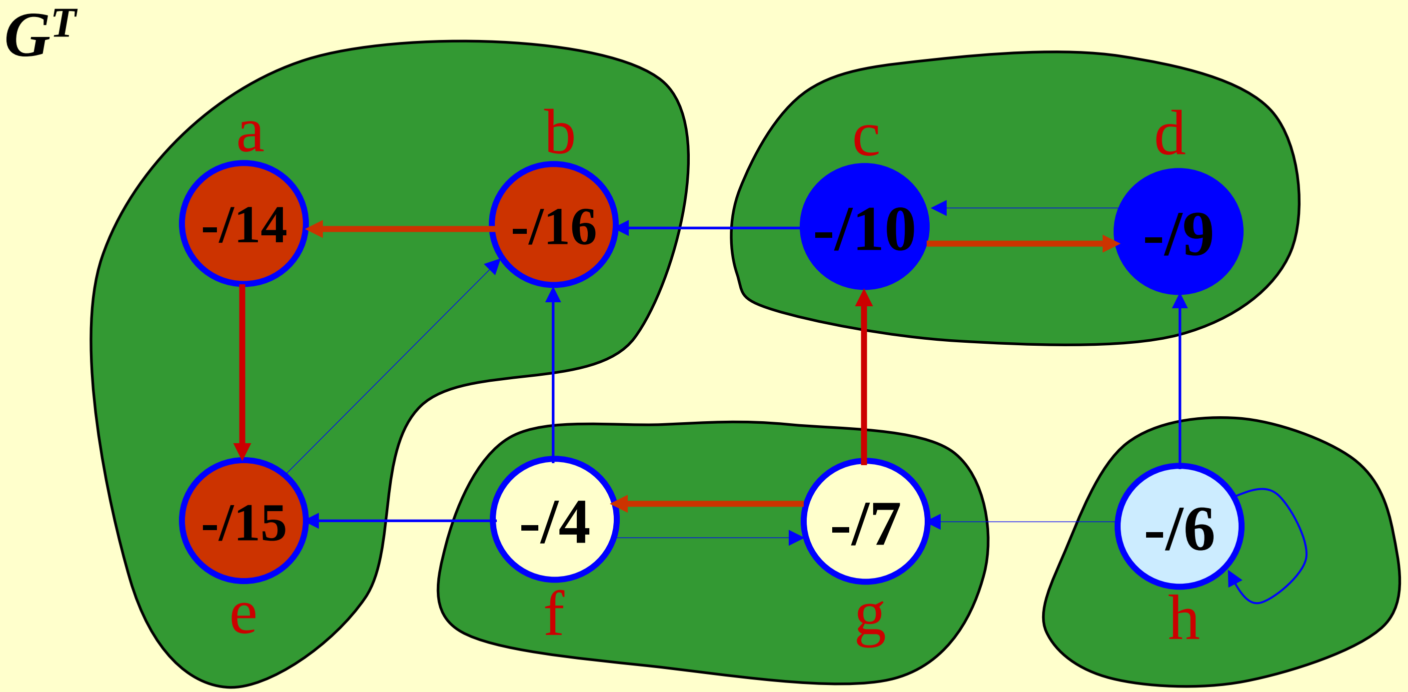
1. call DFS(G) to compute finishing times $f[u]$ for all u
2. compute G^T
3. call DFS(G^T), but in the main loop, consider vertices in order of decreasing $f[u]$ (as computed in first DFS)
4. output the vertices in each tree of the depth-first forest formed in second DFS as a separate SCC

Time: $O(|V| + |E|)$.

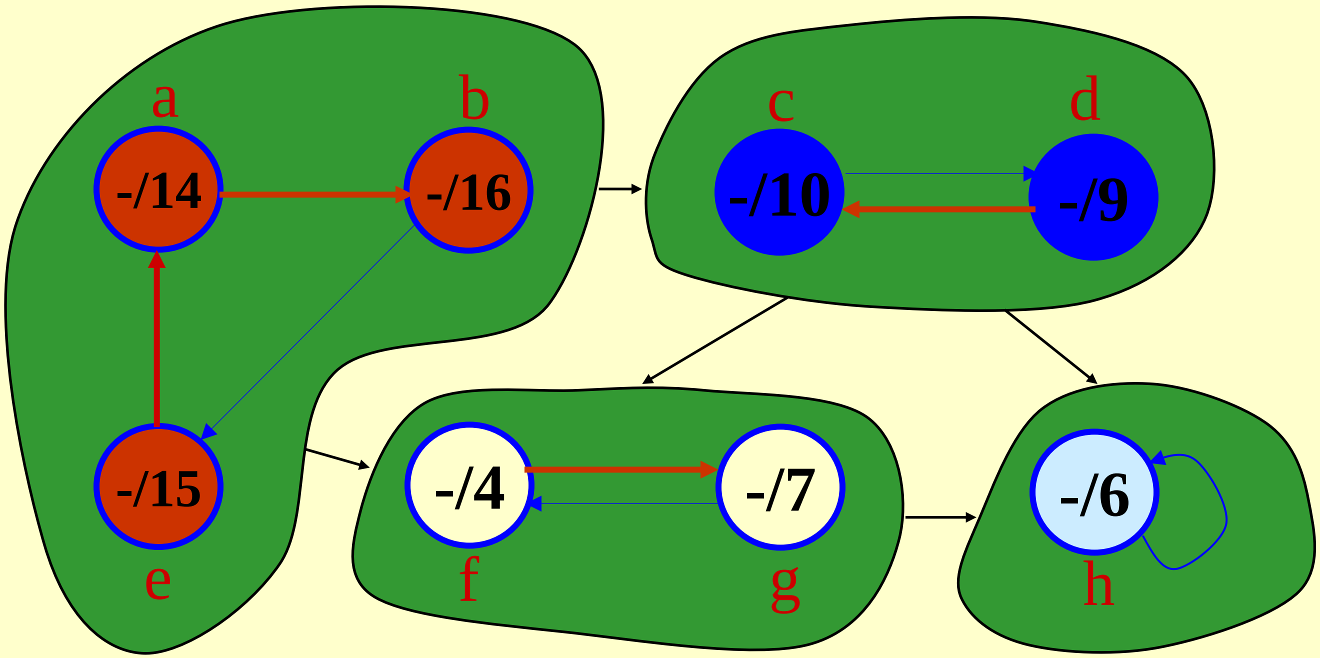
Example



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

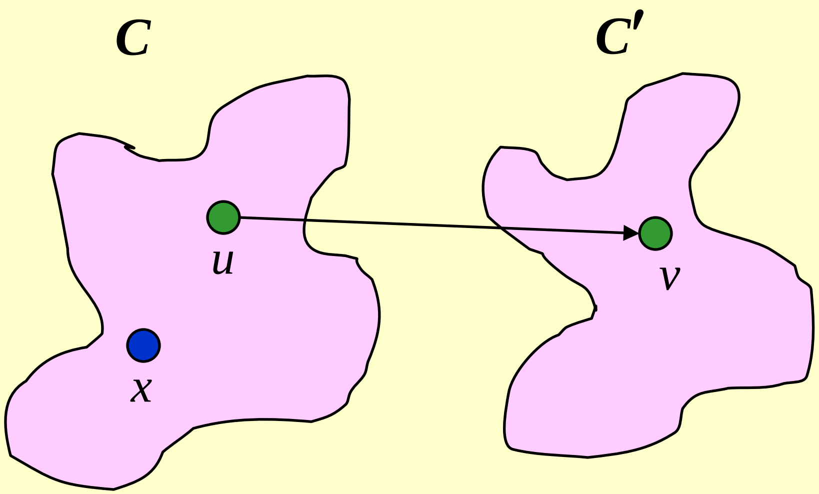
Lemma 22.14

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, $f[x] = f(C) > f(C')$.

$d(x) < d(v) < f(v) < f(x)$



$d(C) = \min_{u \in C}\{d[u]\})$

$f(C) = \max_{u \in C}\{f[u]\}$

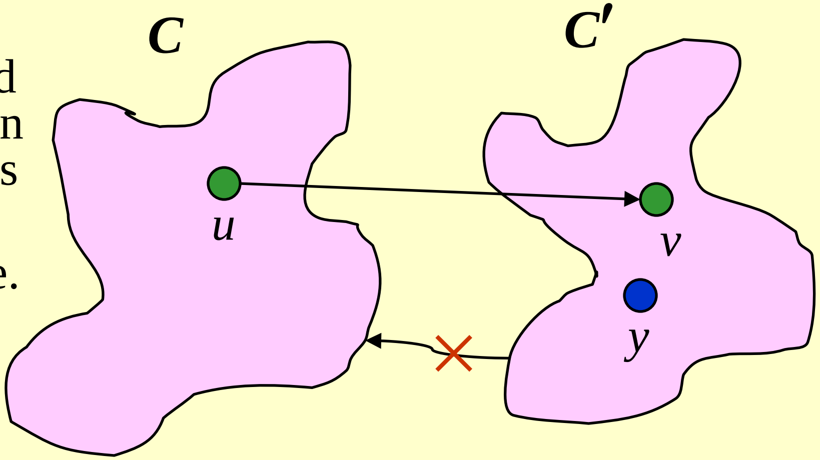
SCCs and DFS finishing times

Lemma 22.14

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 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 $v \in C$, $f[v] > f[y]$, which implies that $f(C) > f(C')$.



SCCs and DFS finishing times

Corollary 22.15
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 Lemma 22.14.

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 22.15 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.