CS 201: Data Structure and Algorithm

Strongly Connected Component

Last Class's Topic

- DFS
- Topological Sort
- Problems:
  - Detect cycle in an undirected graph
  - Detect cycle in a directed graph
  - How many paths are there from "s" to "t" in a directed acyclic graph?

Connectivity

- Connected Graph
  - In an **undirected graph** G, two vertices u and v are called connected if G contains a path from u to v. Otherwise, they are called disconnected.
  - A **directed graph** is called connected if every pair of distinct vertices in the graph is connected.
- Connected Components
  - A connected component is a maximal connected subgraph of G. Each vertex belongs to exactly one connected component, as does each edge.

Connectivity (cont.)

- Weakly Connected Graph
  - A directed graph is called **weakly connected** if replacing all of its directed edges with undirected edges produces a connected (undirected) graph.

- Strongly Connected Graph
  - It is strongly connected or strong if it contains a directed path from u to v for every pair of vertices u, v. The strong components are the maximal strongly connected subgraphs

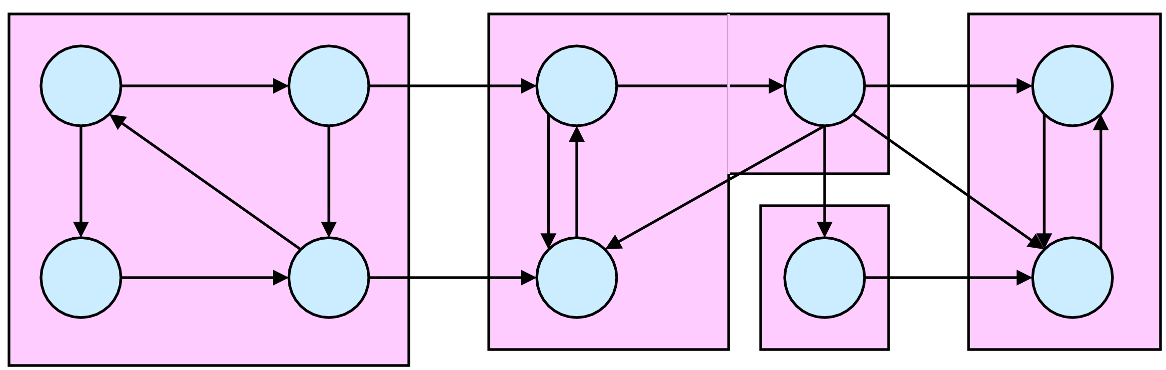
Connected Components

- Strongly connected graph
  - A directed graph is called *strongly connected* if for every pair of vertices u and v there is a path from u to v and a path from v to u.

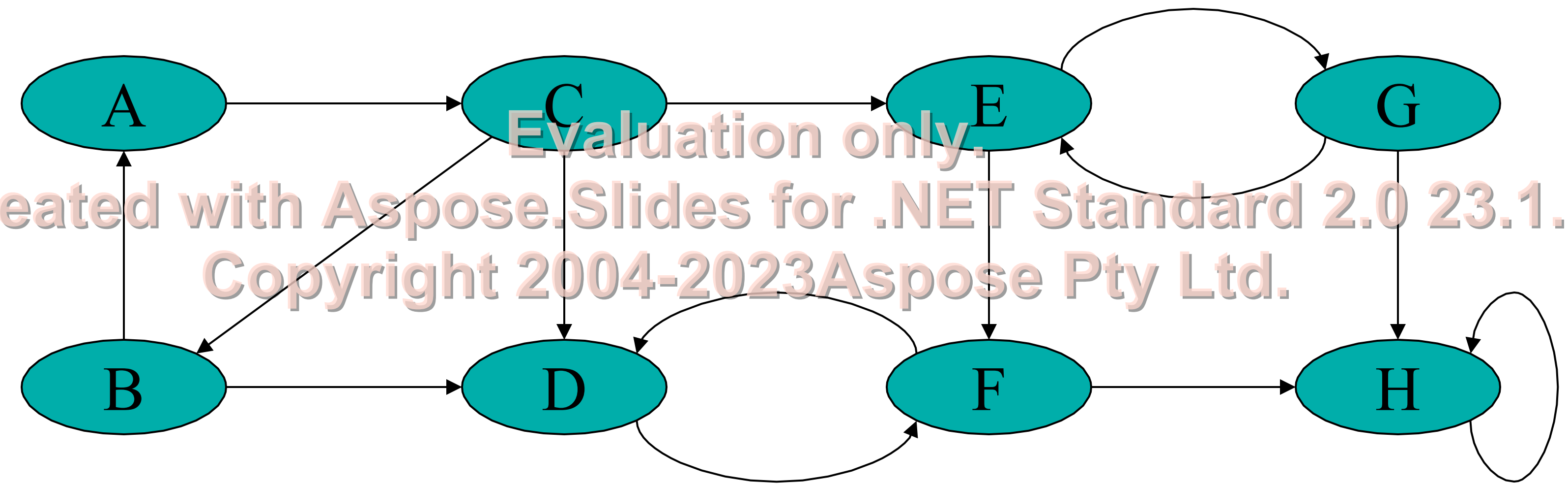
- Strongly Connected Components (SCC)
  - The **strongly connected components** (**SCC**) of a directed graph are its maximal strongly connected subgraphs.
- Here, we work with
  - Directed unweighted graph

Strongly Connected Components

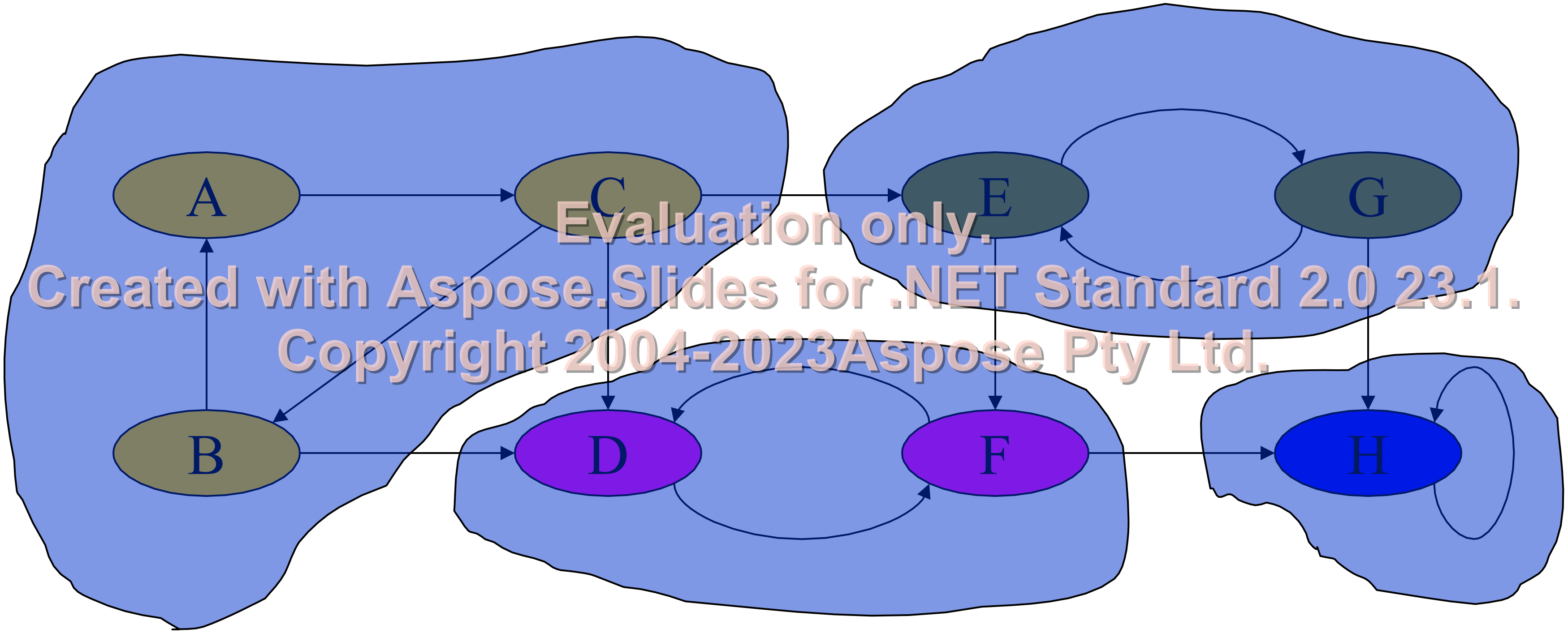
- G is strongly connected if every pair (u, v) of vertices in G is reachable from one another.
- A **strongly connected component** (***SCC***) of G is a maximal set of vertices $C \subseteq V$ such that for all $u, v \in C$, both $u \leadsto v$ and $v \leadsto u$ exist.



DFS - Strongly Connected Components

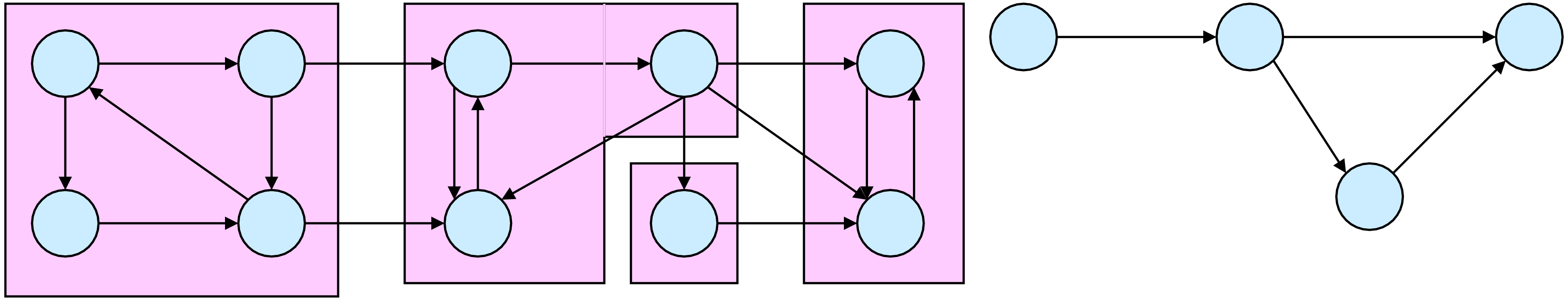


DFS - Strongly Connected Components



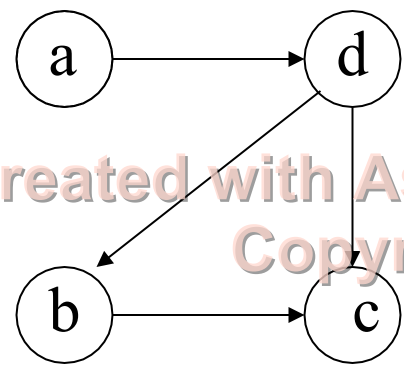
Component Graph

- $G^{\text{SCC}} = (V^{\text{SCC}}, E^{\text{SCC}})$.
- 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:

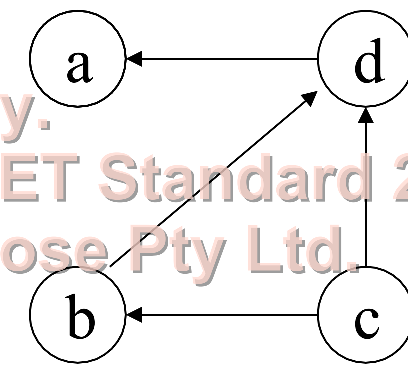


Strongly Connected Components

The **transpose** M^T of an NxN matrix M is the matrix obtained when the rows become columns and the column become rows:



G



G^T

Edges have reverse direction!

M

	a	b	c	d
a				1
b			1	
c				
d		1	1	

M^T

	a	b	c	d
a				
b				1
c		1		1
d	1			

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.)

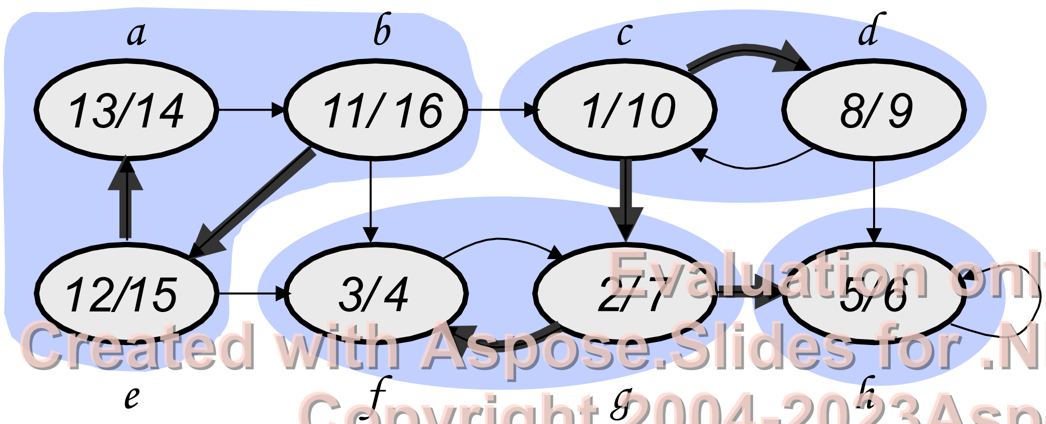
Algorithm to determine SCCs

<u>SCC(G)</u>

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

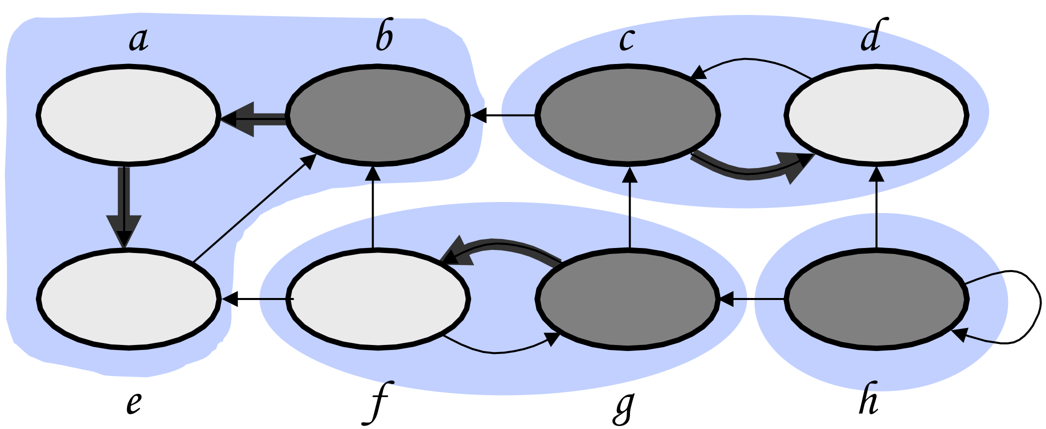
Time: $\Theta(V + E)$.

Example



DFS on the initial graph G

b	*e*	*a*	*c*	*d*	*g*	*h*	*f*
16	*15*	*14*	*10*	*9*	*7*	*6*	*4*

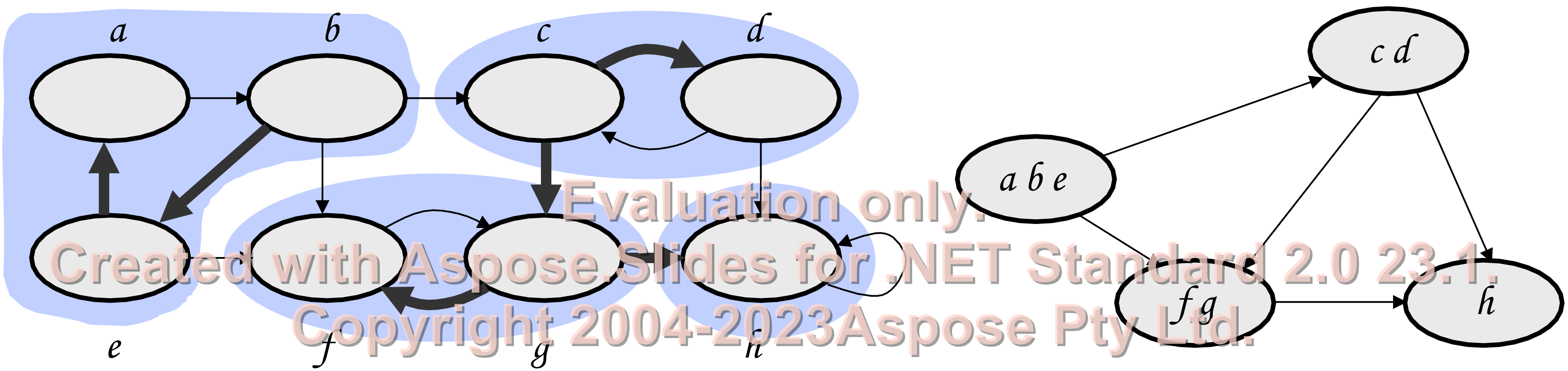


DFS on G^{T}:

- *start at b: visit a, e*
- *start at c: visit d*
- *start at g: visit f*
- *start at h*

Strongly connected components: C_1 = {a, b, e}, C_2 = {c, d}, C_3 = {f, g}, C_4 = {h}

Component Graph



- The **component graph** $G^{SCC} = (V^{SCC}, E^{SCC})$:
  - $V^{SCC} = \{v_1, v_2, \ldots, v_k\}$, where v_i corresponds to each strongly connected component C_i
  - There is an edge $(v_i, v_j) \in E^{SCC}$ if G contains a directed edge (x, y) for some $x \in C_i$ and $y \in C_j$
- The component graph is a DAG

Lemma 1

Let C and C' be distinct SCCs in G

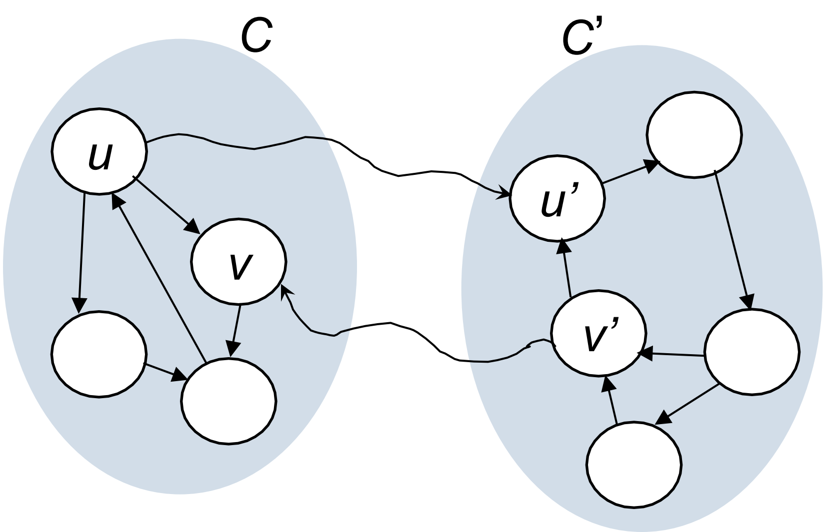
Let u, v ∈ C, and u′, v′ ∈ C′

Suppose there is a path u ⇨ u′ in G

Then there cannot also be a path v′ ⇨ v in G.

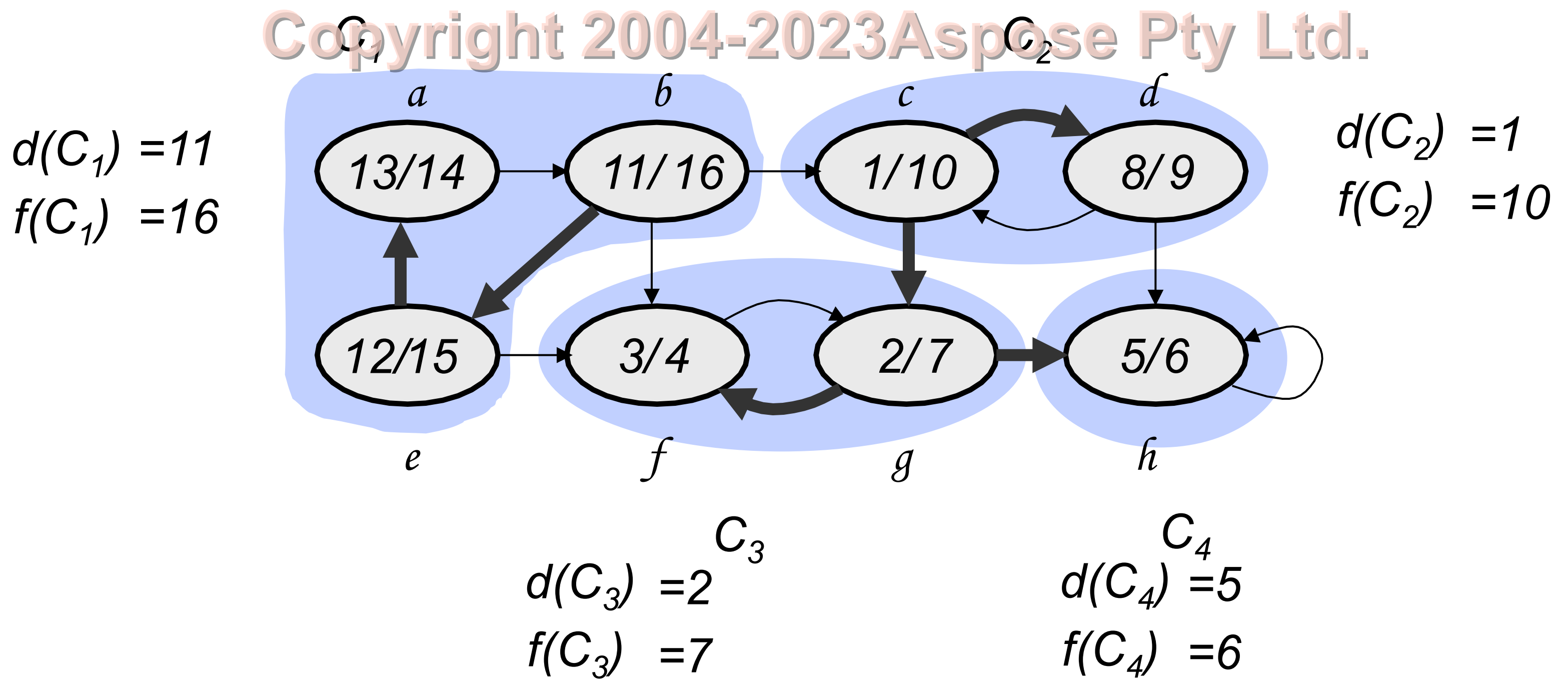
Proof

- *Suppose there is a path v′ ⇨ v*
- *There exists u ⇨ u′ ⇨ v′*
- *There exists v′ ⇨ v ⇨ u*
- *u and v′ are reachable from each other, so they are not in separate SCC's: contradiction!*



Notations

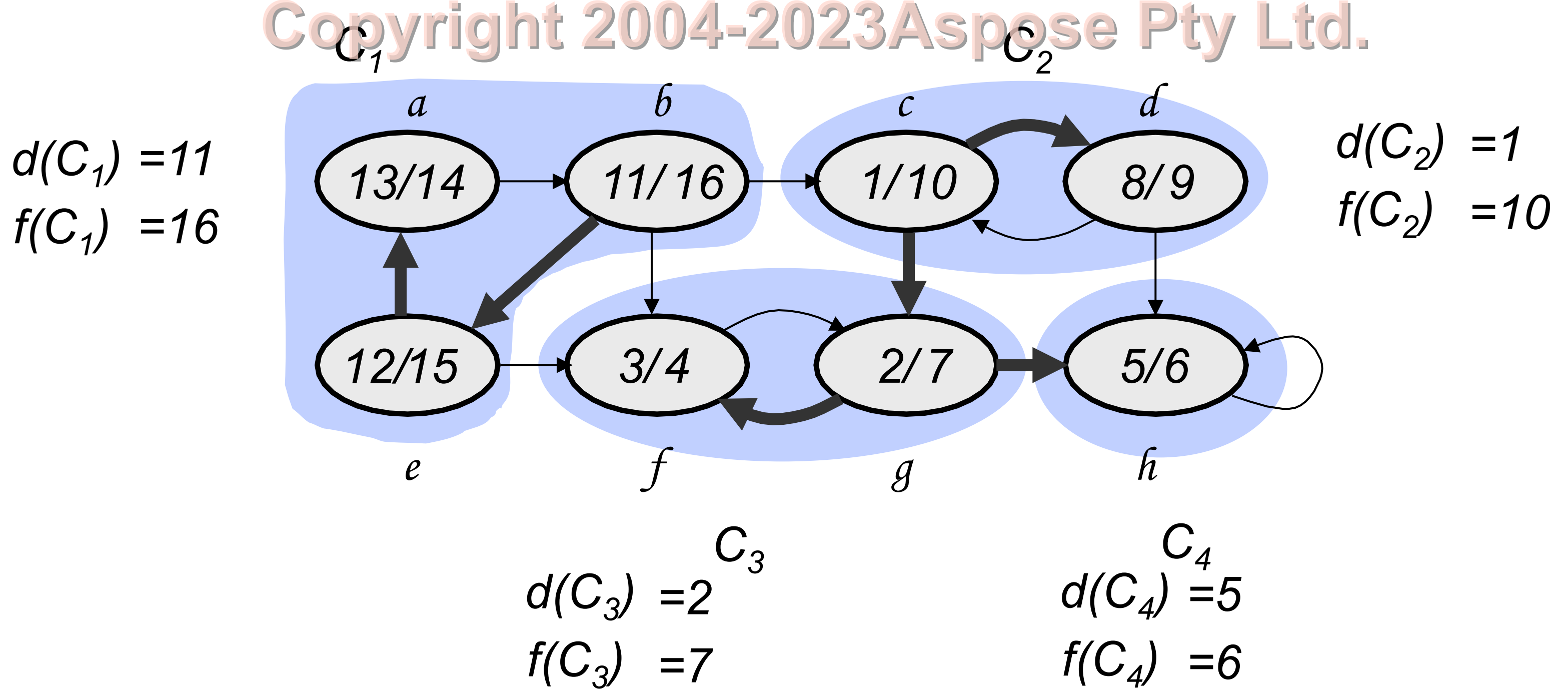
- Extend notation for `d` (starting time) and `f` (finishing time) to sets of vertices $U \subseteq V$:
  - $d(U) = \min_{u \in U} \{ d[u] \}$ (earliest discovery time)
  - $f(U) = \max_{u \in U} \{ f[u] \}$ (latest finishing time)



Lemma 2

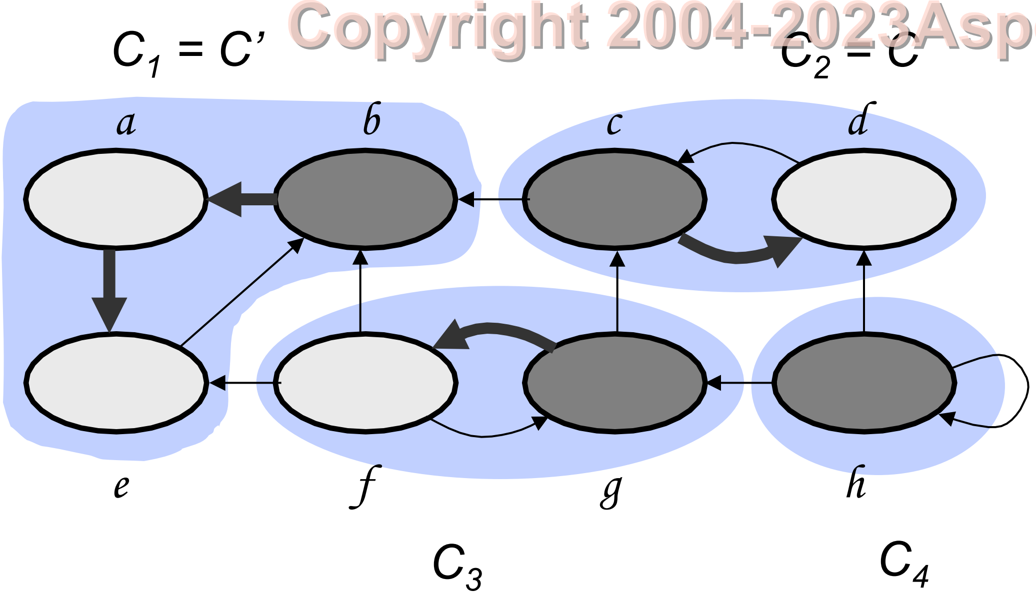
- Let C and C' be distinct SCCs in a directed graph G = (V, E). If there is an edge (u, v) ∈ E, where u ∈ C and v ∈ C' then f(C) > f(C').

- Consider C_1 and C_2, connected by edge (b, c)



Corollary

- Let C and C' be distinct SCCs in a directed graph G = (V, E). If there is an edge (u, v) ∈ E^T, where u ∈ C and v ∈ C' then f(C) < f(C').
- Consider C_2 and C_1, connected by edge (c, b)



- Since (c, b) ∈ E^T ⇒ (b, c) ∈ E
- From previous lemma:

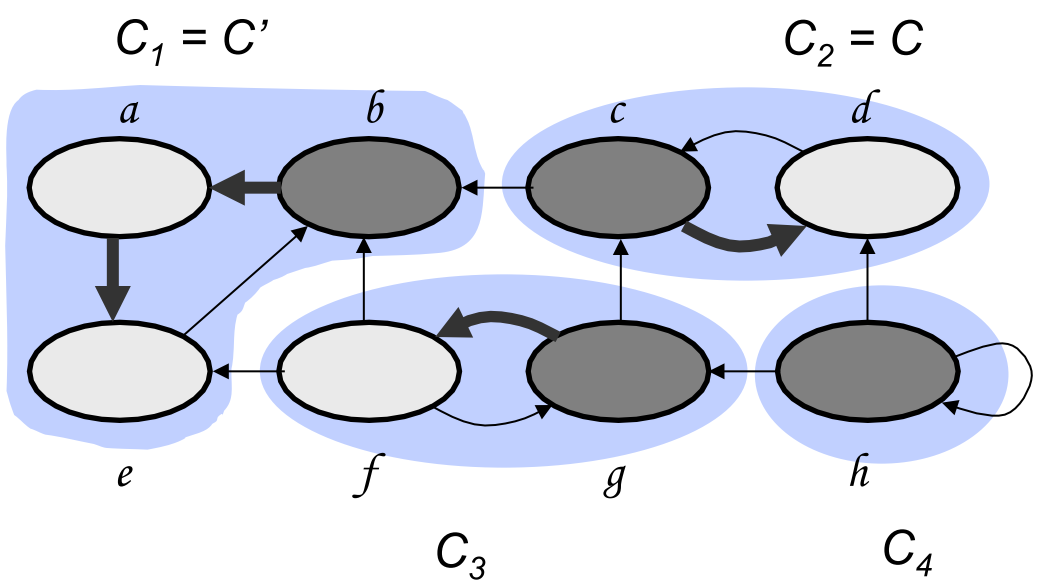
$f(C_1) > f(C_2)$

$f(C') > f(C)$

$f(C) < f(C')$

Corollary

- Each edge in G^T that goes between different components goes from a component with an earlier finish time (in the DFS) to one with a later finish time



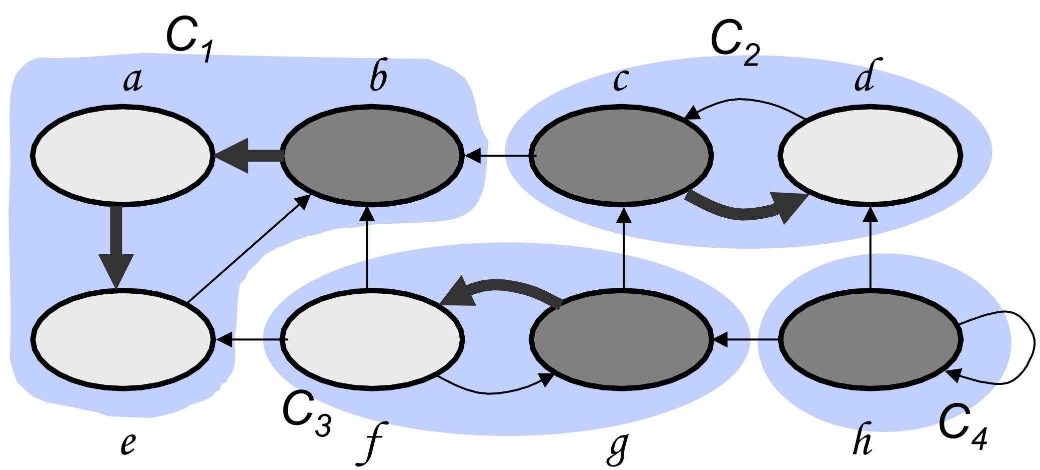
Why does SCC Work?

- When we do the second DFS, on G^T, we start with a component C such that f(C) is maximum (b, in our case)
- We start from b and visit all vertices in C_1
- From corollary: f(C) > f(C') in G for all C ≠ C' ⇒ there are no edges from C to any other SCCs in G^T

⇒ DFS will visit only vertices in C_1

⇒ The depth-first tree rooted at b contains exactly the vertices of C_1

b	e	a	c	d	g	h	f
16	15	14	10	9	7	6	4



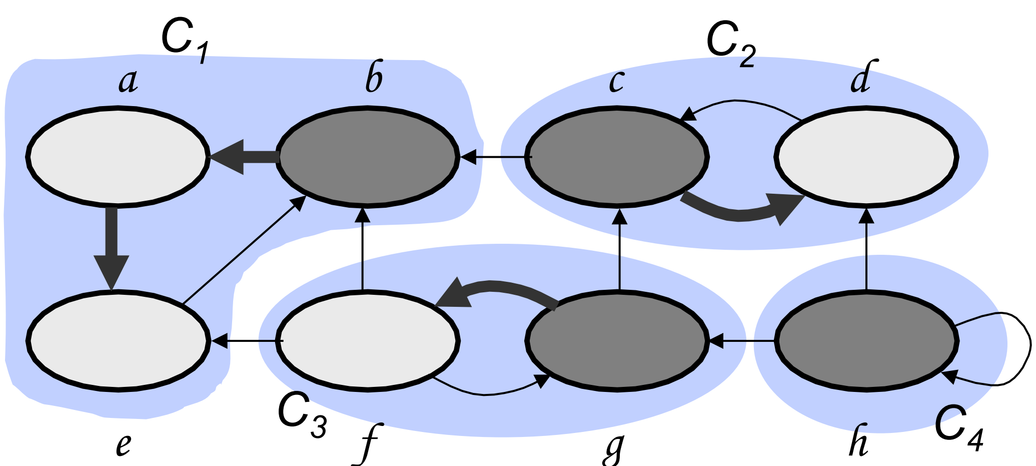
Why does SCC Work? (cont.)

- The next root chosen in the second DFS is in SCC C_2 such that f(C) is maximum over all SCC's other than C_1
- DFS visits all vertices in C_2
  - the only edges out of C_2 go to C_1, which we've already visited

$\Rightarrow$ The only tree edges will be to vertices in C_2

- Each time we choose a new root it can reach only:
  - vertices in its own component
  - vertices in components *already visited*

b	e	a	c	d	g	h	f
16	15	14	10	9	7	6	4



Reference

- Book: Cormen – Chapter 22 – Section 22.5
- Exercise:
  - 22.5-1: Number of componets change?
  - 22.5-6: Minimize edge list
  - 22.5-7: Semiconnected graph