Chap 22 – Elementary Graph Algorithms

*22.1 Representations of graphs

22.2 Breadth-first search

22.3 Depth-first search

22.4 Topological sort

22.5 Strongly connected components

22.2 Breadth-first search

- Breadth-first search
  - $G = (V, E)$ directed or undirected; $s \in V$ source vertex
  - BFS explores the graph G level-by-level and computes

    $v.d =$ distance (smallest # of edges) from s to v, $\forall v \in V$

    $\quad\quad = $ length of shortest path $s \rightsquigarrow v$

    $v.\pi =$ predecessor of v on shortest path $s \rightsquigarrow v$
  - $v.\pi$ induces a breadth-first tree: $\{(v, v.\pi : v \in V - \{s\})\}$
  - As BFS progresses, every vertex has a color
    - WHITE = undiscovered
    - GRAY = discovered, but not finished
    - BLACK = finished

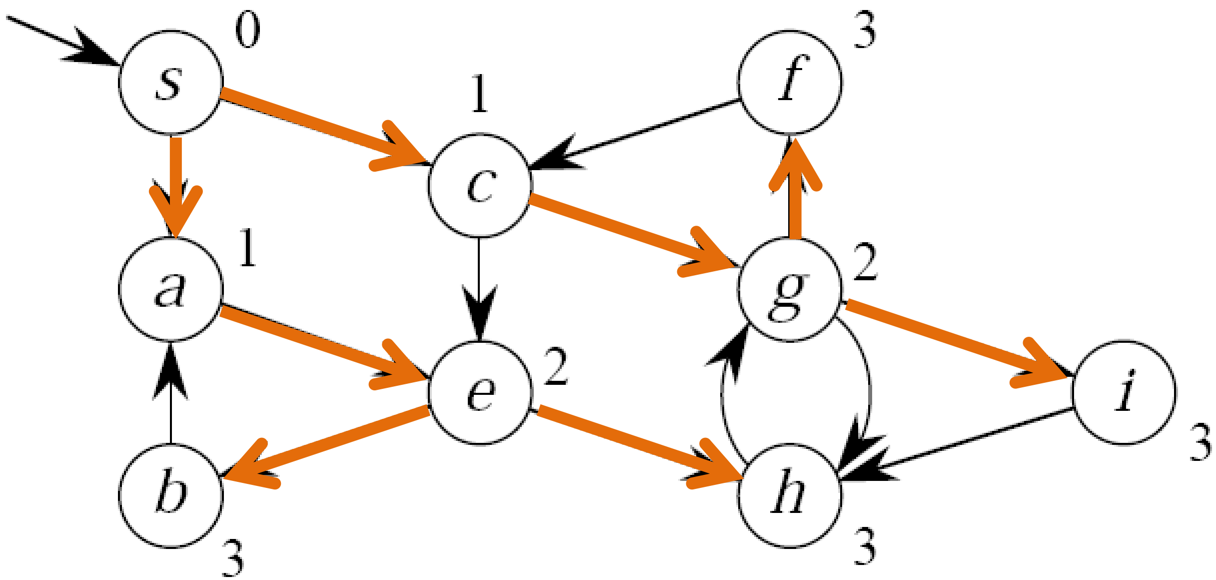
22.2 Breadth-first search

- Breadth-first search
  - BFS(V, E, s)

for each $u \in V - \{s\}$
    $u.d = \infty$
    $u.\pi = \text{NIL}$
    $u.color = \text{WHITE}$
$s.d = 0$
$s.\pi = \text{NIL}$
$s.color = \text{GRAY}$
$Q = \emptyset$
$\textsc{Enqueue}(Q, s)$
while $Q \neq \emptyset$
    $u = \textsc{Dequeue}(Q)$
    for each $v \in G.Adj[u]$
        if $v.color == \text{WHITE}$
            $v.d = u.d + 1$
            $v.\pi = u$
            $v.color = \text{GRAY}$
            $\textsc{Enqueue}(Q, v)$
    $u.color = \text{BLACK}$

22.2 Breadth-first search

- Breadth-first search
  - Example



$$Q = \{s^0\} \to \{a^1, c^1\} \to \{c^1, e^2\} \to \{e^2, g^2\} \to \{g^2, b^3, h^3\}$$
$$\to \{b^3, h^3, i^3, f^3\} \to \{h^3, i^3, f^3\} \to \{i^3, f^3\} \to \{f^3\} \to \emptyset$$

22.2 Breadth-first search

- Breadth-first search
  - BFS may not discover all vertices.
  - Time: $O(V+E)$
    - $O(V)$ $\because$ each vertex is enqueued at most once
    - $O(V)$

      $\because$ for directed graph, each edge (u, v) is examined at most once when u is dequeued.

      For undirected graph, each edge $\{u, v\}$ is examined at most twice when u and v are dequeued.

22.3 Depth-first search

- Depth-first search
  - Depth-first search explores the graph path-by-path.
  - No source vertex is given – if any undiscovered vertices remain, DFS selects one of them as a new source and searches from that source.
  - Comment

    In the book, BFS is limited to one source, but DFS may search from multiple sources.

    Why?

    It is because BFS and DFS are typically used this way.

22.3 Depth-first search

- Depth-first search
  - DFS computes two timestamps on each vertex:
    $v.d$ = discovery time (i.e. when v is grayed)
    $v.f$ = finishing time (i.e. when v is blacken)
  - It also computes
    $v.\pi$ = predecessor of v
  - Since DFS may repeat from multiple sources , $v.\pi$ induces a depth-first forest comprising several depth-first trees, one for each source vertex.

22.3 Depth-first search

- Depth-first search
  - DFS(G)

    for each $u \in G.V$

    $\quad u.color = \text{WHITE}$

    $\quad u.\pi = \text{NIL}$

    $time = 0$

    for each $u \in G.V$

    $\quad$ **if** $u.color == \text{WHITE}$

    $\quad\quad$ DFS_VISIT(G, u)

  - $time$ is a global variable.

DFS_VISIT(G, u)

$time = time + 1$

$u.d = time$

$u.color = \text{GRAY}$

for each $v \in G.Adj[u]$

$\quad$ **if** $v.color == \text{WHITE}$

$\quad\quad v.\pi = u$

$\quad\quad$ DFS_VISIT(G, v)

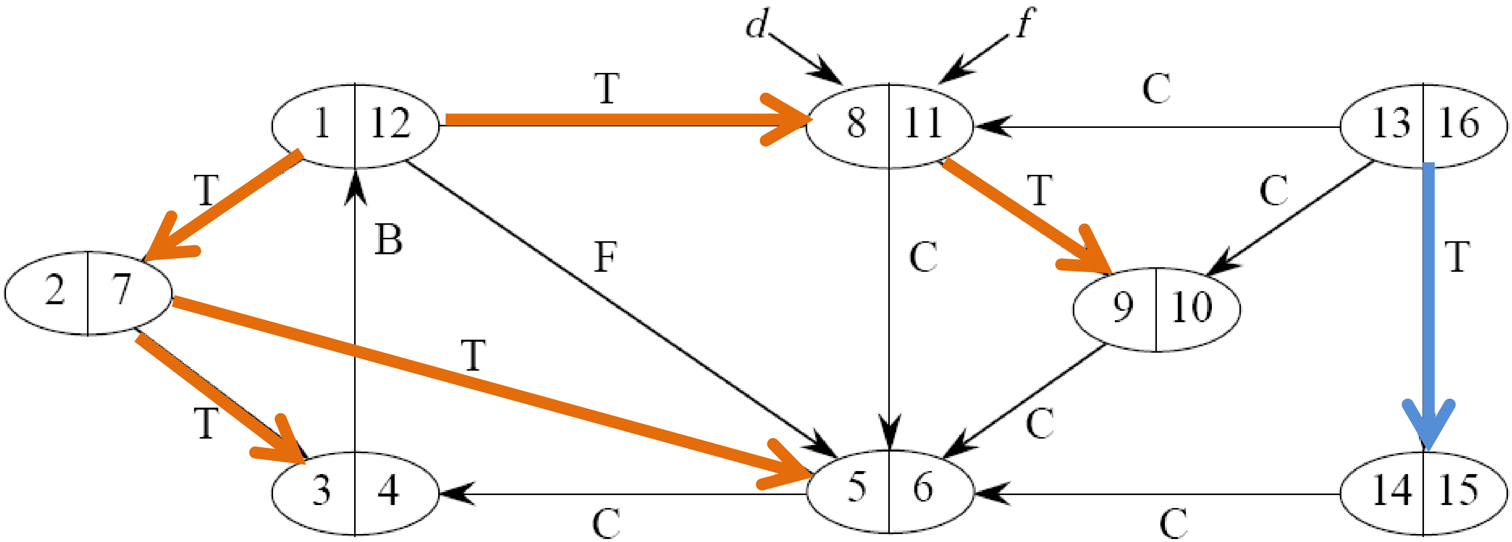
$u.color = \text{BLACK}$

$time = time + 1$

$u.f = time$

22.3 Depth-first search

- Depth-first search
  - Time: $\Theta(V+E)$
    - Similar to BFS analysis.
    - Θ not O $\because$ guaranteed to examine every vertex and edge
  - Example



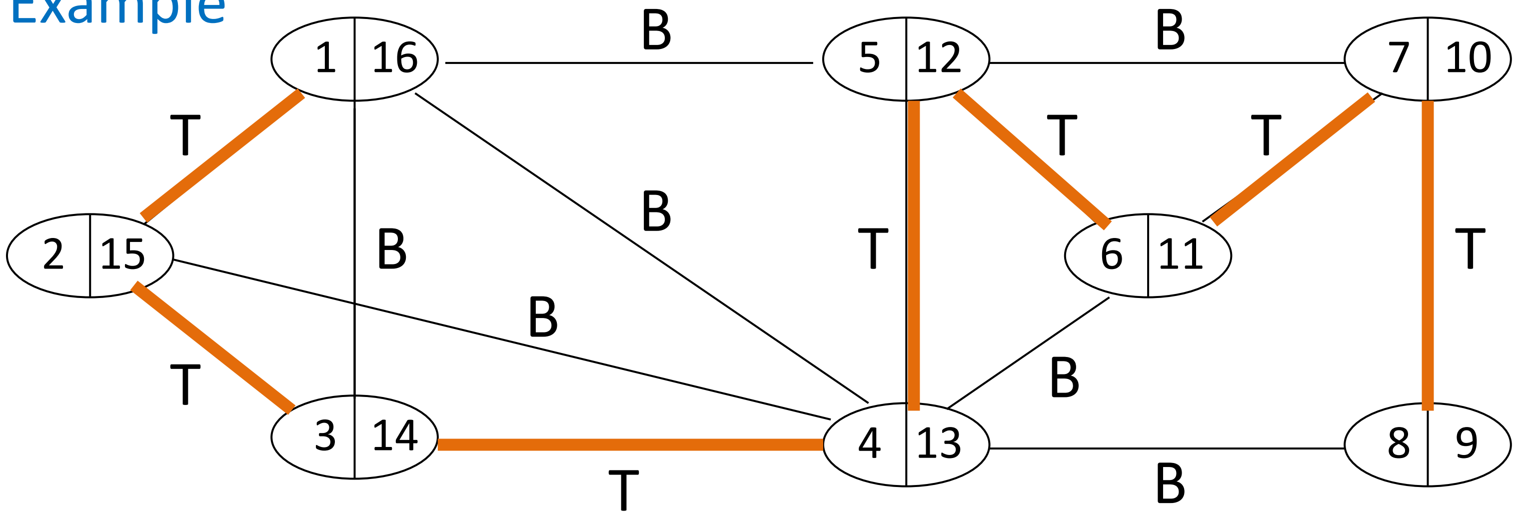
22.3 Depth-first search

- Properties of depth-first search
  - Classification of edges
    - *Tree edge:* edges in the depth-first forest
    - *Back edge:* (u, v), where u is a descendant of v.
    - *Forward edge:* (u, v), where v is a descendant of u, but not a tree edge.
    - *Cross edge:* Any other edge between vertices in the same depth-first tree or in different depth-first trees.
  - $\because$ (u, v) and (v, u) are the same edge in an undirected graph, an edge is classified by the first type above that matches.

22.3 Depth-first search

- Depth-first search
  - **THEOREM**

    In a DFS of an undirected graph, every edge is either a tree edge or a back edge.
  - Example



Every edge not in the tree is a back (not forward) edge.

22.3 Depth-first search

- Depth-first search
  - **THEOREM (PARENTHESIS THEOREM)**

    For any u, v, exactly one of the following holds:
    - $[u.d, u.f] \cap [v.d, v.f] = \emptyset$ and neither of u and v is a descendant of the other.
    - $[u.d, u.f] \subsetneq [v.d, v.f]$ and u is a descendant of v.
    - $[v.d, v.f] \subsetneq [u.d, u.f]$ and v is a descendant of u.
  - **THEOREM (WHITE-PATH THEOREM)**

    v is a descendant of u iff at the time $u.d$ that the search discovers u, there is a path $u \leadsto v$ consisting of only white vertices (except for u, which was just colored gray).

22.4 Topological sort

- Topological sort
  - $G = (V, E)$ directed acyclic graph (dag)

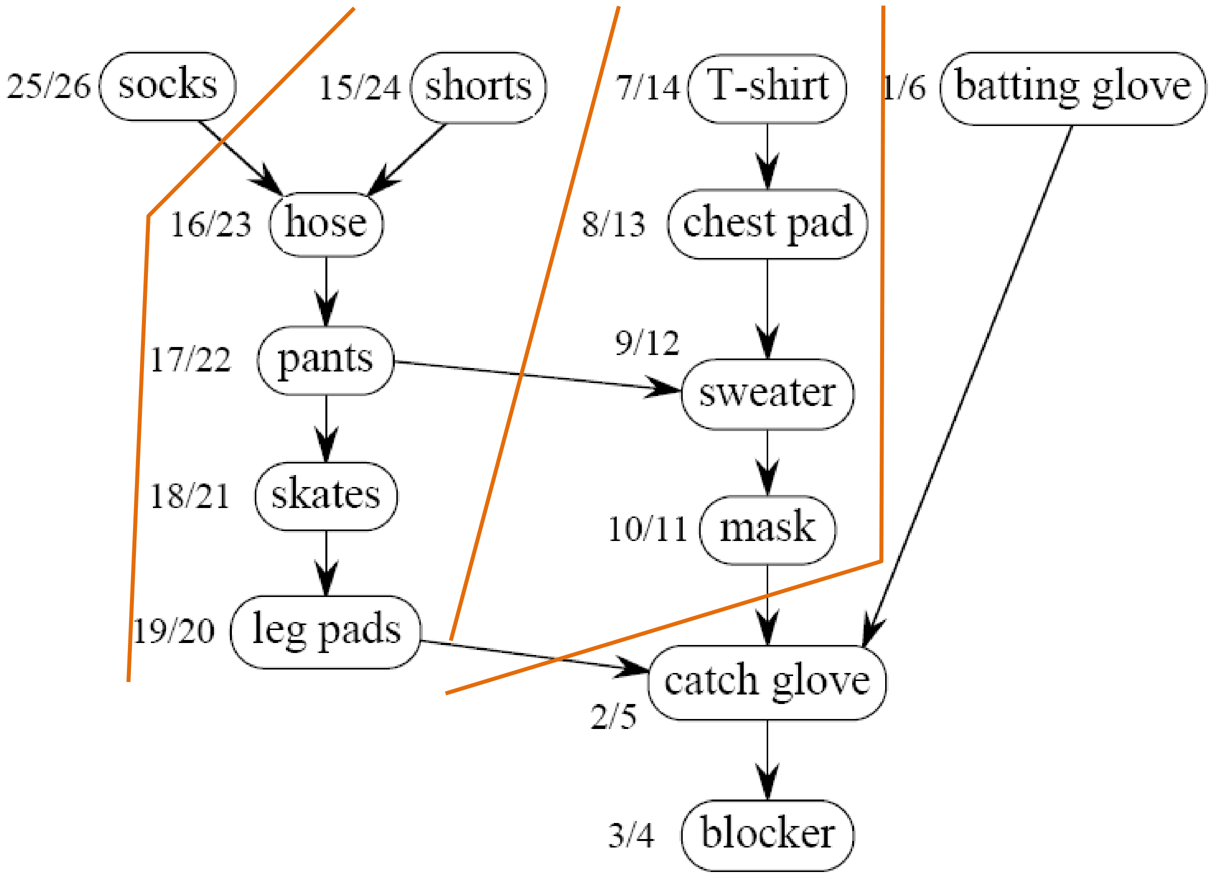
    A topological sort of G is a linear ordering on V such that if $(u, v) \in E$ the u appears somewhere before v.
  - TOPOLOGICAL-SORT(G)

    1 Call DFS(G) to compute finishing times $v.f$ for all v

    2 Output vertices in order of *decreasing* finishing times
  - Time: $\Theta(V + E)$
    - Don't need to sort by finishing times
    - Just insert the vertices onto the *front* of a linked list as they're finished.

22.4 Topological sort

- Topological sort
  - Example



Order:

26	socks
24	shorts
23	hose
22	pants
21	skates
20	leg pads
14	t-shirt
13	chest pad
12	sweater
11	mask
6	batting glove
5	catch glove
4	blocker

22.4 Topological sort

- Depth-first search
  - **THEOREM**

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

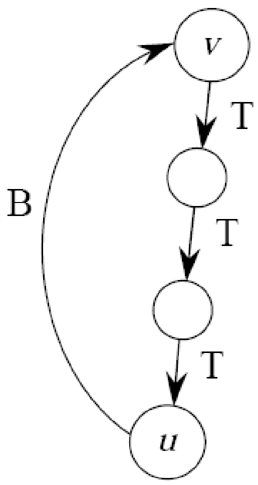
    Proof

    $\Rightarrow$ Suppose (u, v) is a back edge.

    Then, u is a descendant of v in a depth-first tree:

    Therefore, $v \rightsquigarrow u \rightarrow v$ is a cycle.

    A contradiction.



22.4 Topological sort

- Depth-first search
  - **THEOREM** (Cont'd)

    $\Leftarrow$ Suppose G contains a cycle c.

    Let v be the first vertex discovered in c

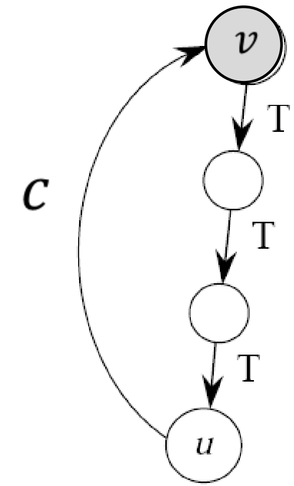
    Let (u, v) be the preceding edge in c

    At time $v.d$, vertices of c form a white path $v \rightsquigarrow u$ $\because$ v be the first vertex discovered in c.

    By the white-pace theorem, u is a descendant of v.

    Therefore, (u, v) is a back page.

    A contradiction.



22.4 Topological sort

- Depth-first search
  - **THEOREM** TOPOLOGICAL-SORT(G) is correct.

    Proof

    Need to show that $(u, v) \in E \Rightarrow u.f > v.f$

    When we explore (u, v), $u.color ==$ GRAY.

    Case 1: $v.color ==$ GRAY

    Then, u is a descendant of v

    $\Rightarrow (u, v)$ is a back edge $\Rightarrow G$ is cyclic. A contradiction.

    Case 2: $v.color ==$ WHITE

    Then, v is a descendant of u

    $\Rightarrow [v.d, v.f] \subsetneq [u.d, u.f] \Rightarrow u.f > v.f$, as desired.

22.4 Topological sort

- Depth-first search
  - **Theorem** (Cont'd)

    Case 3: $v.color == \text{BLACK}$

    Then, v is already finished.

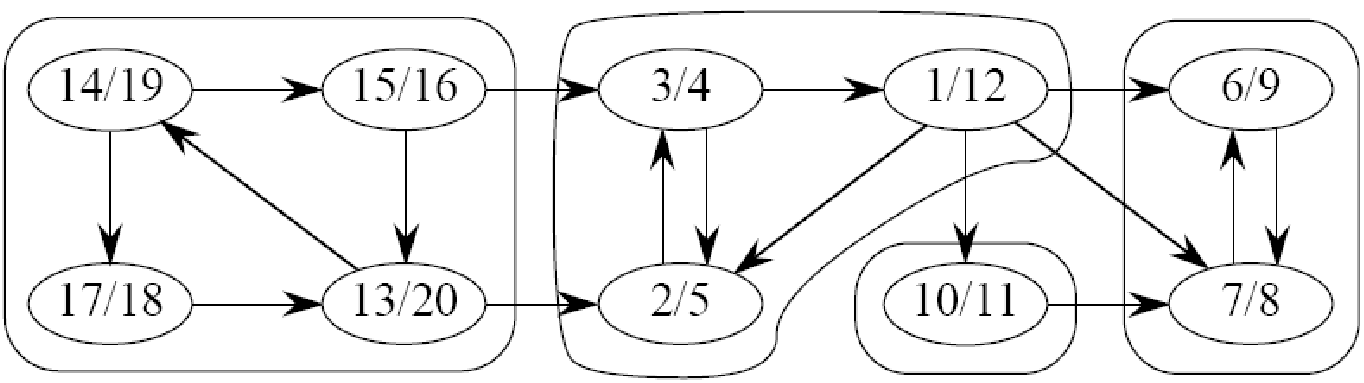
    Since we're exploring (u, v), we have not yet finished u.

    $\Rightarrow u.f > v.f$, as desired.

22.5 Strongly connected components

- Strongly connected components
  - $G = (V, E)$ directed graph

    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 $v \rightsquigarrow u$.
  - Example

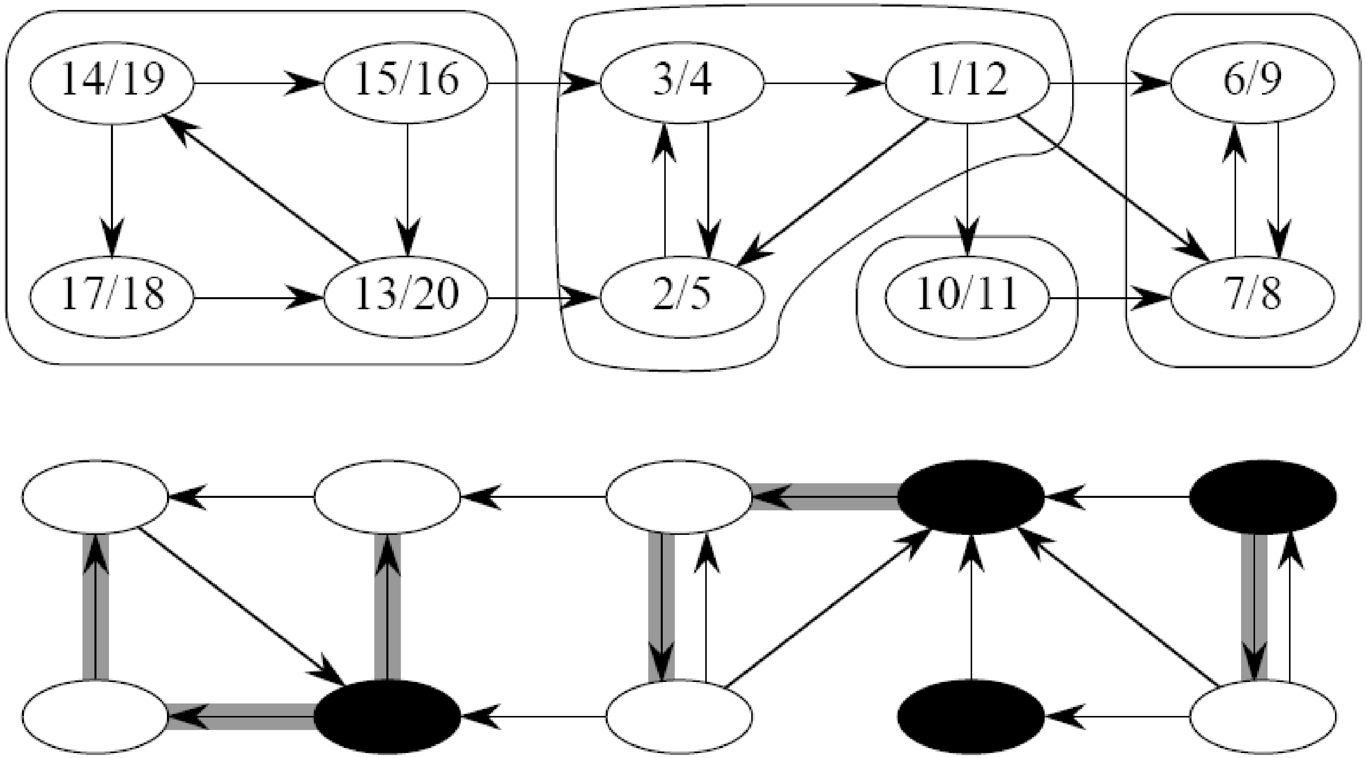


22.5 Strongly connected components

- Strongly connected components
  - SCC(G)
    1. Call DFS(G) to compute finishing times $v.f$ for all v
    2. Compute G^T, i.e. the transpose of G
    3. Call DFS(G^T), but consider vertices in topological sort order, i.e. in order of decreasing $v.f$ found in Step 1
    4. Output the vertices of each depth-first tree as a SCC
  - Time: $\Theta(V + E)$
    - $G^T = (V, E^T)$, where $E^T = \{(u, v) : (v, u) \in E\}$, can be created in $\Theta(V + E)$ time using adjacency list
    - G and G^T have the same SCC's, $\because u \leadsto^G v$ iff $v \leadsto^{G^T} u$

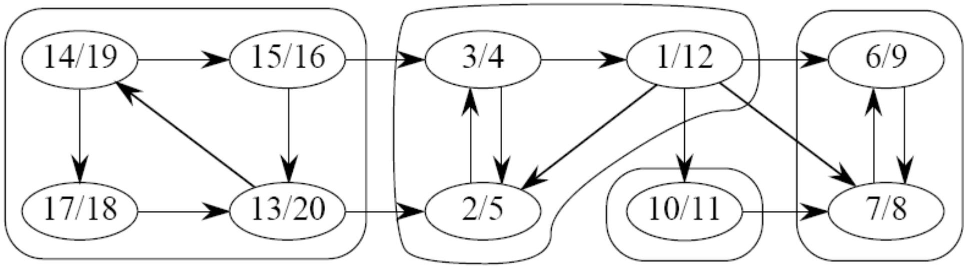
22.5 Strongly connected components

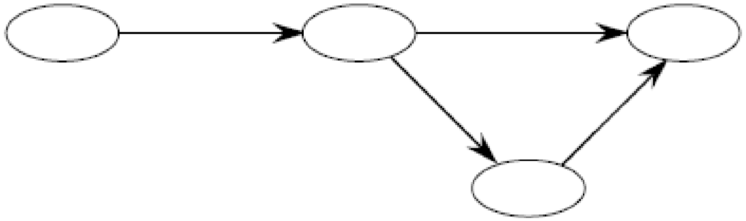
- Strongly connected components
  - Example (Cont'd)



22.5 Strongly connected components

- Strongly connected components
  - Component graph
    - $G^{SCC} = (V^{SCC}, E^{SCC})$
    - V^{SCC} has one vertex for each SCC in G.
    - E^{SCC} as an edge if there's an edge between the corresponding SCC's in G.
  - Example





22.5 Strongly connected components

- Strongly connected components
  - **Lemma** G^{SCC} is a dag.

    More formally, 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 can't also be a path $v' \rightsquigarrow v$ in G.

    Proof

    If $v' \rightsquigarrow v$ exists, then $u \rightsquigarrow u' \rightsquigarrow v'$ and $v' \rightsquigarrow v \rightsquigarrow u$.

    Therefore, u and v' are reachable from each other, so they aren't in separated SCC's

22.5 Strongly connected components

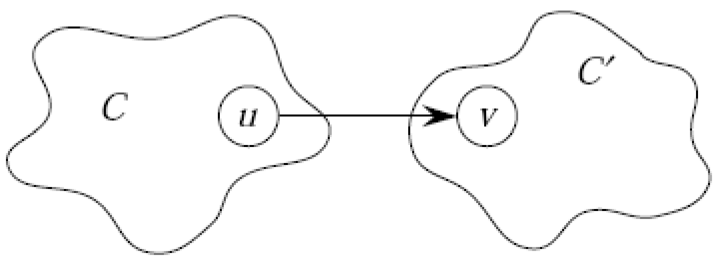
- Strongly connected components
  - Let $U \subseteq V$, define

    $d(U) = \min_{u \in U}\{u.d\}$, i.e. the earliest discovery time in U

    $f(U) = \max_{u \in U}\{u.f\}$, i.e. the latest finishing time in U

    where $u.d$ and $u.f$ refer to the 1st DFS.
  - **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, $f(C) > f(C')$

22.5 Strongly connected components

- Strongly connected components
  - *Proof* of **LEMMA**

    Case 1: $d(C) < d(C')$

    Let $x \in C$ be the 1st discovered vertex

    $\Rightarrow$ At time $x.d = d(C)$, all vertices in C and C' are white.

    $\Rightarrow$ $\exists$ a path $x \leadsto y$ of white vertices in C and C', where $y \in C \cup C' - \{x\}$, since C and C' are SCC's and there is an edge (u, v) from C to C'

    $\Rightarrow$ By the white-space theorem, y is a descendant of x in depth-first tree

    $\Rightarrow$ By the parenthesis theorem, $[y.d, y.f] \subsetneq [x.d, x.f]$

    $\Rightarrow$ $x.f = f(C) > f(C')$

22.5 Strongly connected components

- Strongly connected components
  - *Proof* of **LEMMA**

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

    Let $y \in C'$ be the 1st discovered vertex

    By similar argument, all vertices in C' are descendants of y

    $\Rightarrow y.f = f(C')$

    On the other hand, G^{scc} is a dag and there is an edge

    (u, v) from C to C'

    $\Rightarrow \nexists$ a path from C' to C

    $\Rightarrow$ all vertices in C aren't descendants of y

    $\Rightarrow$ at time $y.f = f(C')$, all vertices in C are still white

    $\Rightarrow f(C) > f(C')$

22.5 Strongly connected components

- Strongly connected components
  - **COROLLARY**

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

    Then, $f(C) < f(C')$

    Proof

    $(u, v) \in E^T \Rightarrow (v, u) \in E \Rightarrow f(C) < f(C')$
  - **COROLLARY**

    Let C and C' be distinct SCC's in $G = (V, E)$. Suppose that $f(C) > f(C')$, then there is no edge from C to C' in G^T.

22.5 Strongly connected components

- Strongly connected components
  - **THEOREM** SCC(G) is correct.

    Proof

    Let $C_1, C_2, C_3, \ldots$ be SCC's with $f(C_1) > f(C_2) > f(C_3) > \cdots$

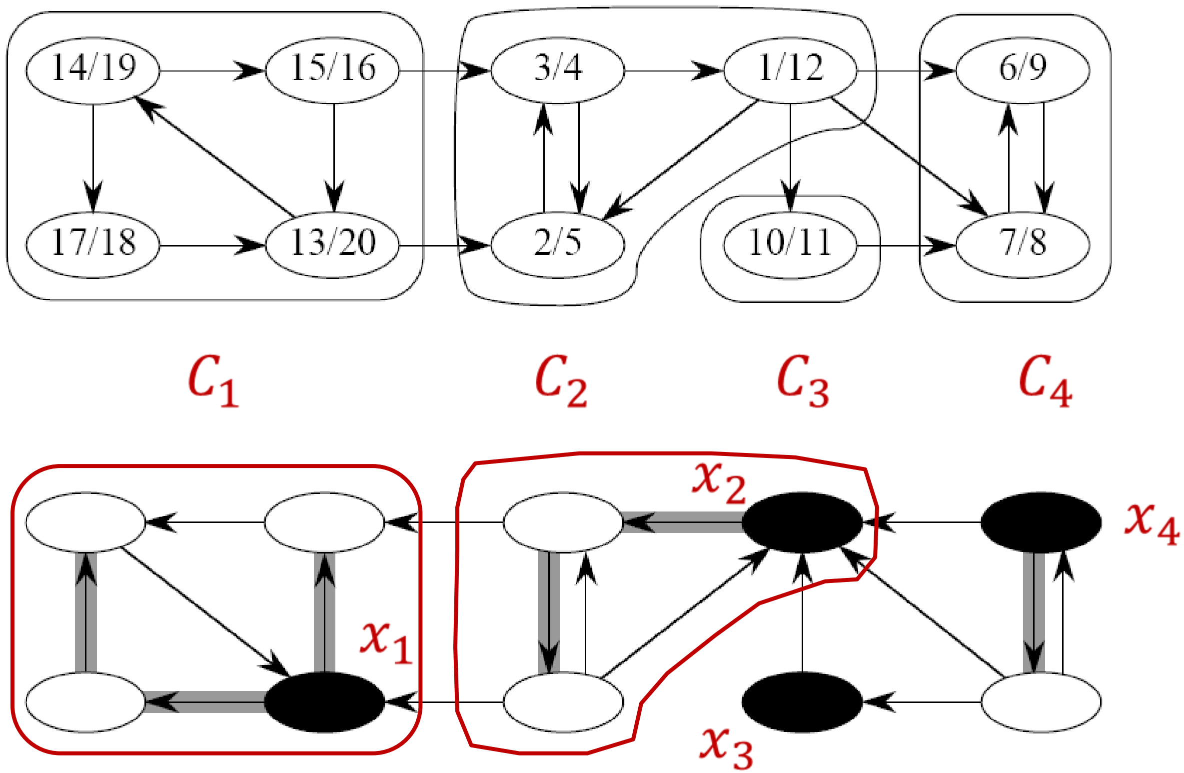
    Let $x_i \in C_i$ be such that $x_i.f = f(C_i)$

    The last corollary says that there is no edge from C_i to C_j, $i > j$, in G^T.

    Therefore, DFS(G^T) starts with x_1 and visits **only** vertices in C_1, which means that the depth-first tree rooted at x_1 contains **exactly** the vertices of C_1.

22.5 Strongly connected components

- Strongly connected components
  - **THEOREM** (Cont'd)



22.5 Strongly connected components

- Strongly connected components
  - **THEOREM** (Cont'd)

    Next, DFS(G^T) selects x_2 as a new root and visits
    - vertices in C_2 $-$ gets tree edges to these
    - vertices in already-visited SCC C_1 $-$ gets no tree edges to these

    Therefore, the depth-first tree rooted at x_2 contains exactly the vertices of C_2.

    The process continues until all the SCC's are found.