------------------------------ 22 ------------------------------
Elementary Graph Algorithms

Mergeable Heap (Chapter 19)

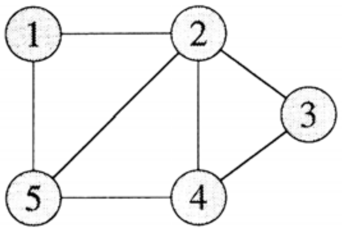
Procedure	Binary heap (worst-case)	Fibonacci heap (amortized)
MAKE-HEAP	$\Theta(1)$	$\Theta(1)$
INSERT	$\Theta(\lg n)$	$\Theta(1)$
MINIMUM	$\Theta(1)$	$\Theta(1)$
EXTRACT-MIN	$\Theta(\lg n)$	$O(\lg n)$
UNION	$\Theta(n)$	$\Theta(1)$
DECREASE-KEY	$\Theta(\lg n)$	$\Theta(1)$
DELETE	$\Theta(\lg n)$	$O(\lg n)$

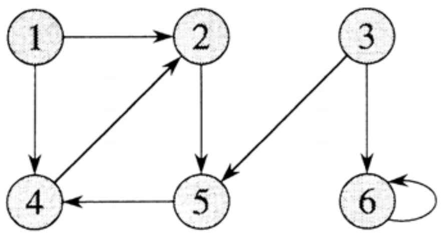
22.1 *Representations of graphs*

G = (*V*, *E*) *V*: vertex set *E*: edge set

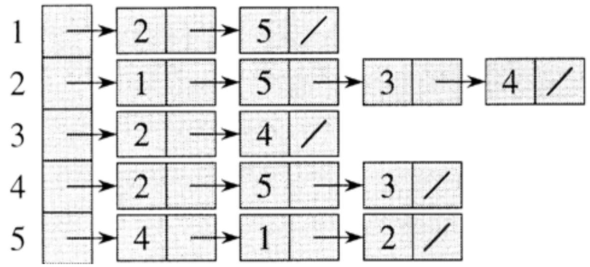
$n = |V|$ $m = |E|$

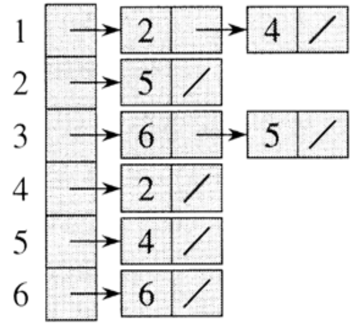
An undirected graph A directed graph





Adjacency-list





* $O(n+m)$ memory (for sparse *G* --- *m* is small)
* It's hard to determine whether $e=(u, v)$ is in *E*.
* It can be extended to *weighted graphs*.

Adjacency-matrix

	1	2	3	4	5
1	0	1	0	0	1
2	1	0	1	1	1
3	0	1	0	1	0
4	0	1	1	0	1
5	1	1	0	1	0

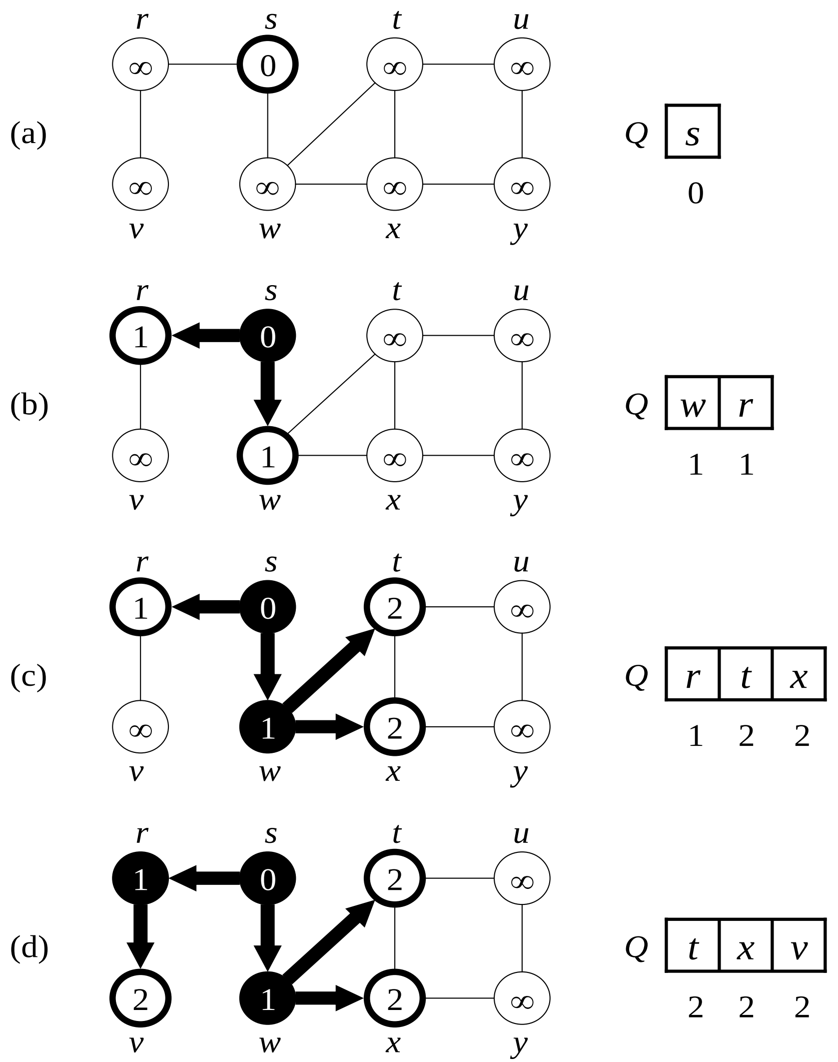
	1	2	3	4	5	6
1	0	1	0	1	0	0
2	0	0	0	0	1	0
3	0	0	0	0	1	1
4	0	1	0	0	0	0
5	0	0	0	1	0	0
6	0	0	0	0	0	1

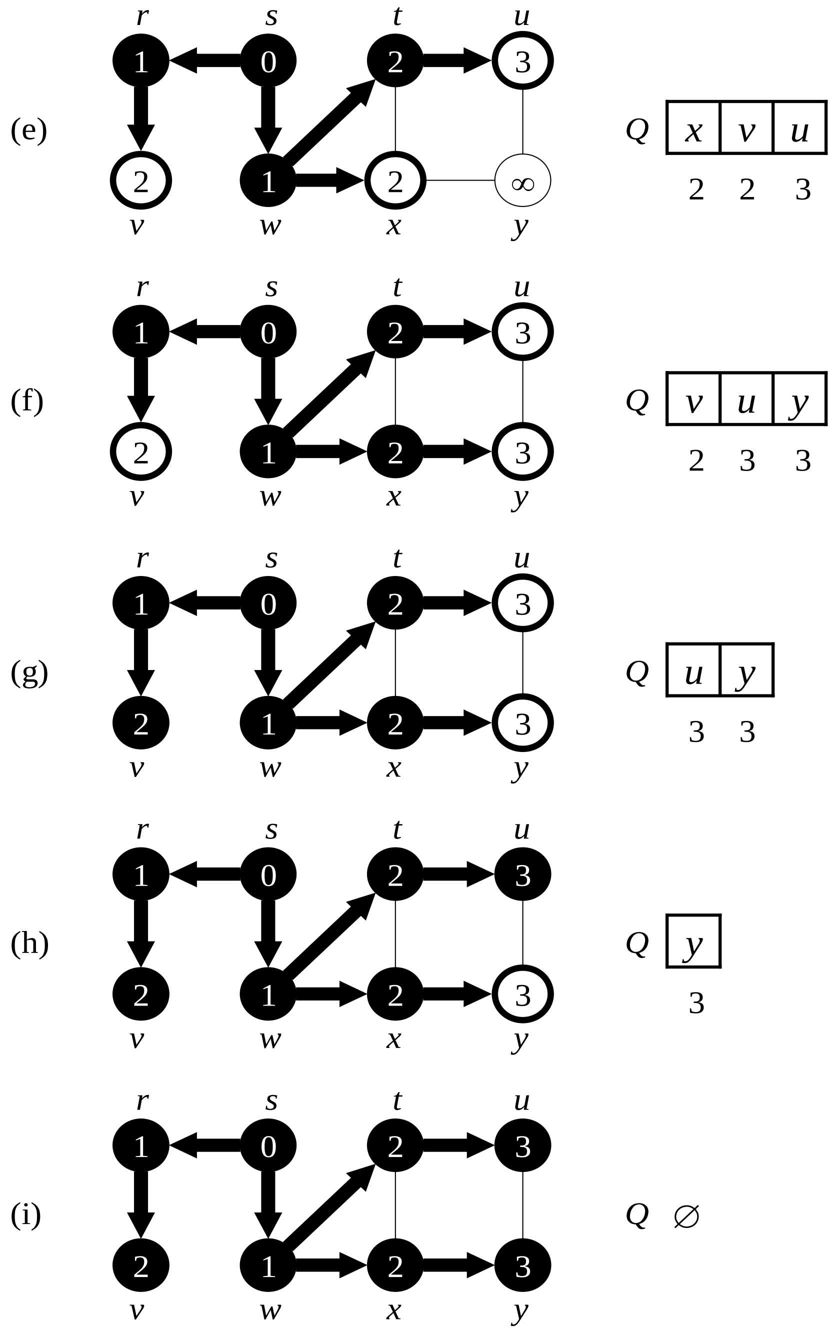
* $O(n^2)$ memory (for dense *G* --- *m* is close to n^2)
* It can be extended to *weighted graphs*.
* For unweighted *G*, 1-bit is enough for an edge.

22.2 Breadth-first search

Breadth-first search / Breadth-first tree

Given G=(V, E) and a **source** vertex $s \in V$





```
BFS(G, s)
for each vertex u ∈ V[G] − {s}
      do color[u] ← WHITE
         d[u] ← ∞
         π[u] ← NIL
color[s] ← GRAY
d[s] ← 0
π[s] ← NIL
Q ← Ø
ENQUEUE(Q, s)
while Q ≠ Ø
      do u ← DEQUEUE(Q)
         for each v ∈ Adj[u]
             do if color[v] = WHITE
                   then color[v] ← GRAY
                        d[v] ← d[u] + 1
                        π[v] ← u
                        ENQUEUE(Q, v)
         color[u] ← BLACK
```

* $\pi(v)$: the predecessor of *v*.
* $O(V+E)$ time: using adjacency list, each edge is scanned at most twice.
* Breadth-first tree $G_\pi=(V_\pi, E_\pi)$ (rooted tree)
* The path in breadth-first tree from *s* to *v* is a shortest path (containing the fewest number of edges) from *s* to *v*. (unweighted)

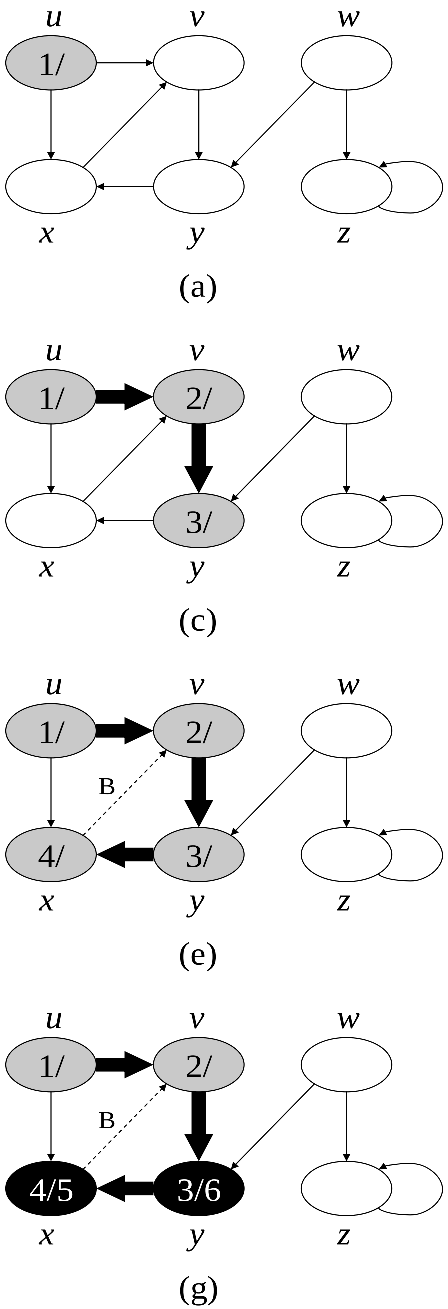
22.3 Depth-first search / Depth-first forest

```
DFS(G)
for each vertex u ∈ V[G]
      do color[u] ← WHITE
         π[u] ← NIL
time ← 0
for each vertex u ∈ V[G]
      do if color[u] = WHITE
            then DFS-VISIT(u)
```

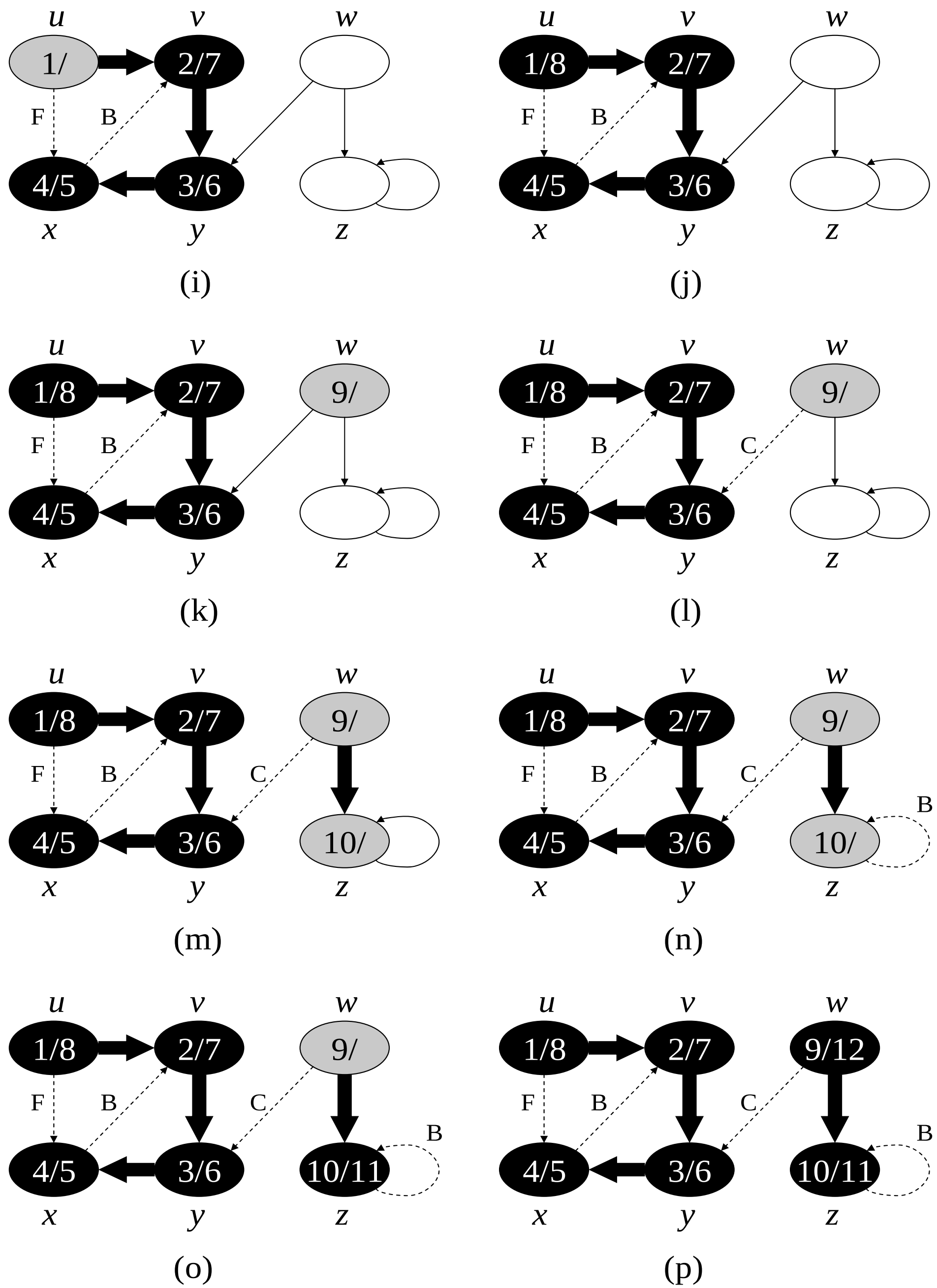
```
DFS-VISIT(u)
color[u] ← GRAY            ▷ White vertex u has just been
time ← time + 1                                  discovered.
d[u] ← time
for each v ∈ Adj[u]        ▷ Explore edge (u, v).
      do if color[v] = WHITE
            then π[v] ← u
                 DFS-VISIT(v)
color[u] ← BLACK           ▷ Blacken u; it is finished.
f[u] ← time ← time + 1
```

* No specified source.
* $d[v]/f[v]$ — $d[v]$: time when *v* is discovered; $f[v]$: time when *v* is finished
* $\pi(v)$: the predecessor of *v*.
* |--- white ---| $d[v]$ |--- gray ---| $f[v]$ |--- black ---|
* $O(V+E)$ *Depth-first forest: $G_\pi=(V, E_\pi)$

22-7


(a) u v w x y z 1/
(b) u v w x y z 1/ 2/
(c) u v w x y z 1/ 2/ 3/
(d) u v w x y z 1/ 2/ 4/ 3/
(e) u v w x y z 1/ 2/ B 4/ 3/
(f) u v w x y z 1/ 2/ B 4/5 3/
(g) u v w x y z 1/ 2/ B 4/5 3/6
(h) u v w x y z 1/ 2/7 B 4/5 3/6

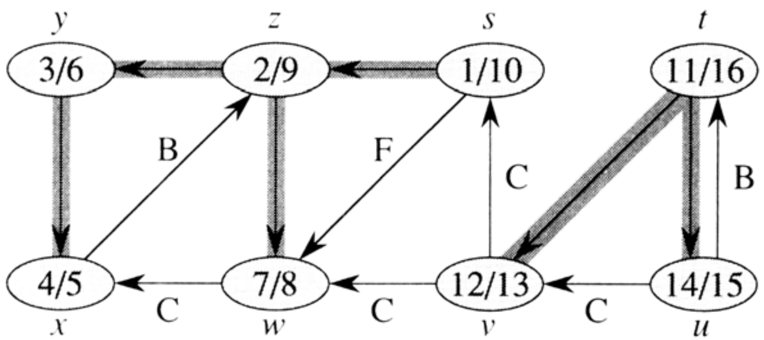
22-8


(i) u v w x y z 1/ 2/7 F B 4/5 3/6
(j) u v w x y z 1/8 2/7 F B 4/5 3/6
(k) u v w x y z 1/8 2/7 9/ F B 4/5 3/6
(l) u v w x y z 1/8 2/7 9/ F B C 4/5 3/6
(m) u v w x y z 1/8 2/7 9/ F B C 4/5 3/6 10/
(n) u v w x y z 1/8 2/7 9/ F B C B 4/5 3/6 10/
(o) u v w x y z 1/8 2/7 9/ F B C B 4/5 3/6 10/11
(p) u v w x y z 1/8 2/7 9/12 F B C B 4/5 3/6 10/11

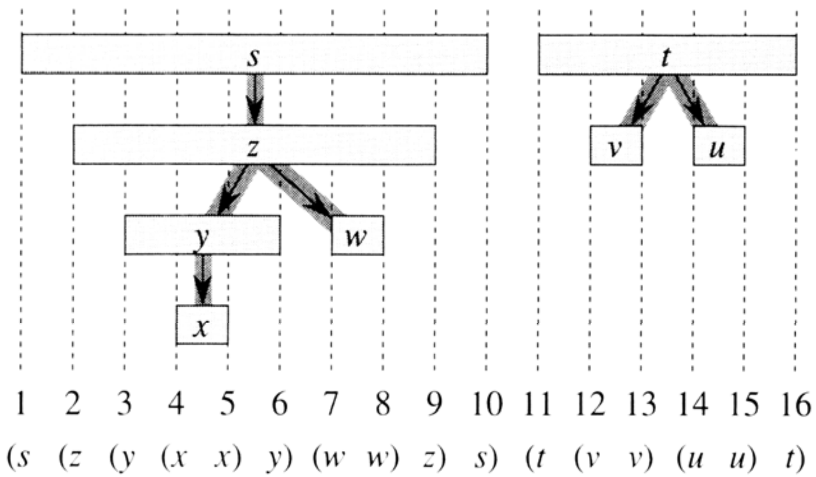
Corollary 22.8(Nesting of descendants' intervals)
Vertex *v* is a proper descendant of vertex *u* in the depth-forest for a (directed or undirected) graph *G* if and only if $d[u] < d[v] < f[v] < f[u]$.

*parenthesis structure: (well-formed)
discover $u \rightarrow$ "(*u*" finish $u \rightarrow$ "*u*)"

(a)



(b)

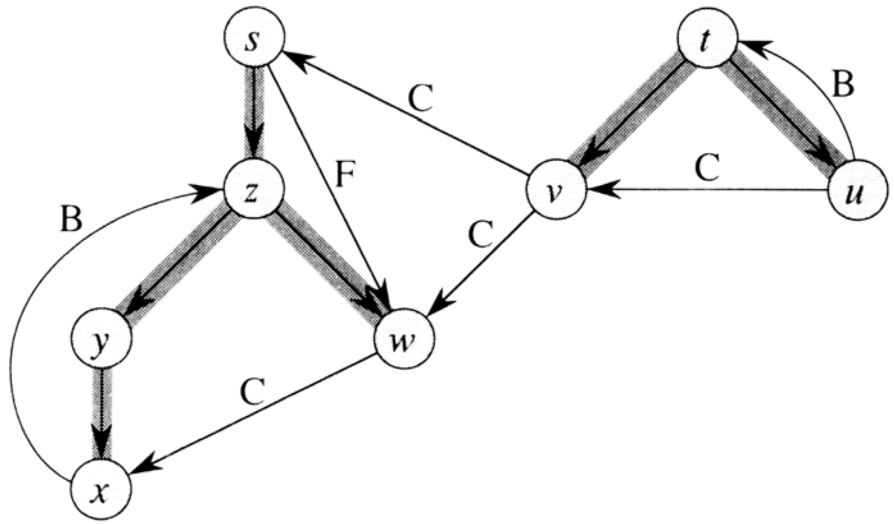


Classification of edges

1. ***Tree edges:*** edges in G_π
2. ***Back edges:*** non-tree edges (*u*, *v*) such that *u* is a descendant of *v* in G_π. (including self-loop)
3. ***Forward edges:*** non-tree edges (*u*, *v*) such that *u* is an ancestor of *v* in G_π.
4. ***Cross edges:*** non-tree edges (*u*, *v*) such that *u* is neither a descendant nor an ancestor of *v* in G_π.

Example: redraw *G* such that all tree and forward edges head downward and all back edges go up.

(c)



In this drawing, all cross edges are from right to left.

Modify DFS algorithm to classify edges

When an edge (u, v) is encountered:

1. v is white $\rightarrow$ tree
2. v is gray $\rightarrow$ back
3. v is black $\rightarrow$ forward if $d[u]<d[v]$
   cross if $d[u]>d[v]$

* If G is an undirected graph, an edge is classified as the first type that applies.

Theorem 22.10 In a depth-first search of an undirected graph G, every edge is either a tree edge or a back edge.

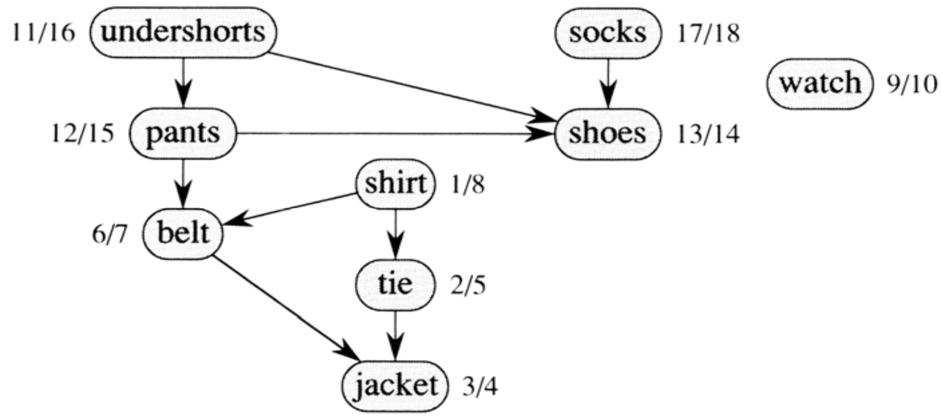
Proof. Let $e=(u, v)$ be an edge in G.

Assume $d[u]<d[v]$. Since e is in the adjacent list of u, v must be discovered and finished before we finished u.

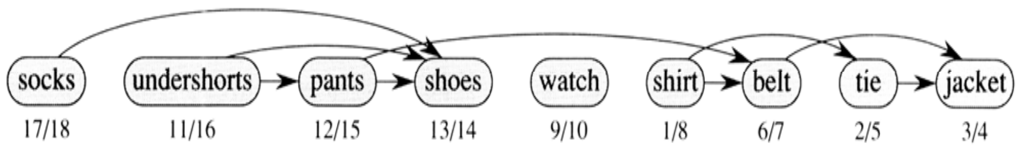
If e is encountered from u to v, e is a tree edge. Otherwise, e is a back edge, since u is still gray at the time e is encountered.

22.4 Topological sort

directed acyclic graph (dag)



Topological sort: order the vertices into a sequence such that if <u, v> is in G, u is before v.




```
TOPOLOGICAL-SORT(G)
call DFS(G) to compute finishing times f[v] for each vertex v
as each vertex is finished, insert it onto the front of a linked list
return the linked list of vertices
```

* Output vertices in order of decreasing $f[u]$.
* $O(V+E)$

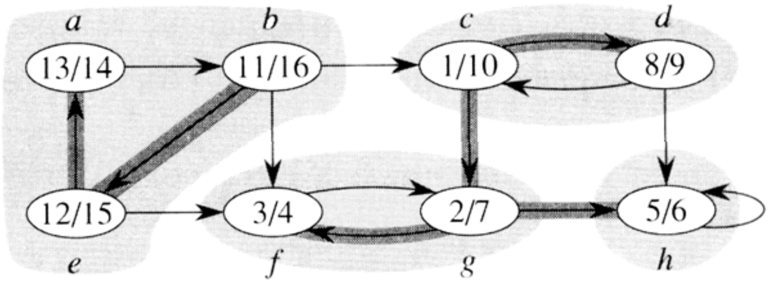
22.5 Strongly connected components

* For an undirected G, performing DFS once can obtain all "connected components"

Strongly connected components (directed):
a maximal set of vertices $U \subseteq V$ such that for every pair of u, $v \in U$, we have both $u \to v$ and $v \to u$.

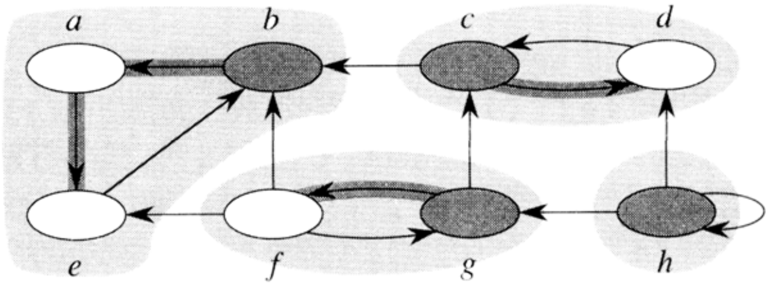
G:

(a)



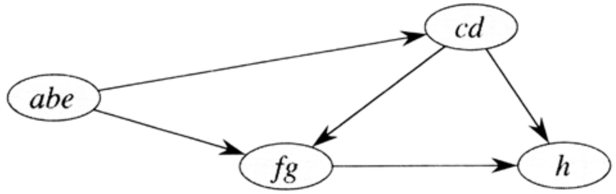
G^T:

(b)



Components

(c)



STRONGLY-CONNECTED-COMPONENTS(G)

```
1  call DFS(G) to compute finishing times f[u] for each vertex u
2  compute G^T
3  call DFS(G^T), but in the main loop of DFS, consider the vertices
          in order of decreasing f[u] (as computed in line 1)
4  output the vertices of each tree in the depth-first forest formed in line 3
       as a separate strongly connected component
```

* $O(V+E)$

* Note that G and G^T have the same components.

* Correctness: ??? (Refer to textbook)

Homework: Ex. 22.1-6, 22.2-4, Prob. 22-2 (d)(f). (While doing Prob. 22-2(d)(f), you can use the properties in (a)(b)(c)(e) without proving.)