---------------------------- 22 ----------------------------

Elementary Graph Algorithms

Mergeable Heap (Chapter 19)

22-1x

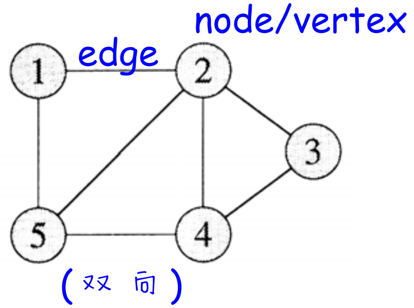
(min-heap) Procedure	Binary heap (worst-case)	Fibonacci heap (amortized)	array
MAKE-HEAP (empty)	$\Theta(1)$	$\Theta(1)$	$O(1)$
INSERT	$\Theta(\lg n)$	$\Theta(1)$	$O(1)$
MINIMUM	$\Theta(1)$	$\Theta(1)$	$O(n)$
EXTRACT-MIN	$\Theta(\lg n)$	$O(\lg n)$	$O(n)$
UNION	$\Theta(n)$!!!	$\Theta(1)$	$O(n)$
DECREASE-KEY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

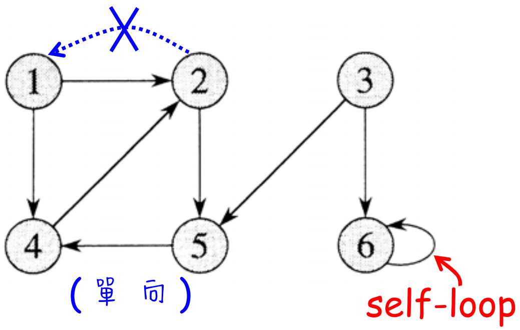
22.1 *Representations of graphs*

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

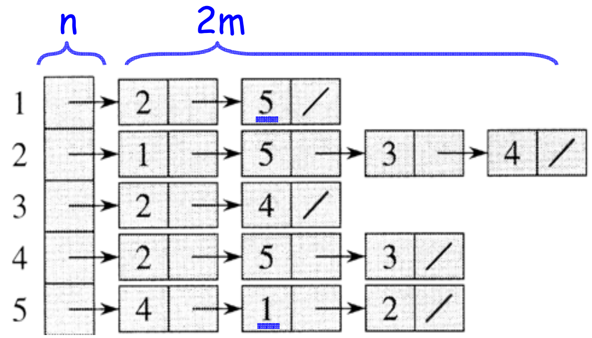
$n = |V| = V$ $m = |E| = E$

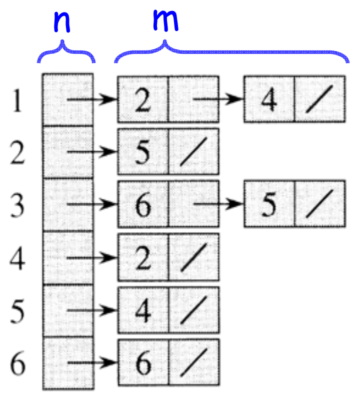
An undirected graph A directed graph





Adjacency-list





$m \le C^n_2 < n^2$

* $O(n+m)$ memory (for sparse G --- m is small)
* It's hard to determine whether $e=(u, v)$ is in E. $e = (3,1)$?
* It can be extended to *weighted graphs*.

Adjacency-matrix

$$\begin{array}{c|ccccc} & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & 0 & 1 & 0 & 0 & 1 \\ 2 & 1 & 0 & 1 & 1 & 1 \\ 3 & 0 & 1 & 0 & 1 & 0 \\ 4 & 0 & 1 & 1 & 0 & 1 \\ 5 & 1 & 1 & 0 & 1 & 0 \end{array}$$

n

$e = (5, 2)$

$$\begin{array}{c|cccccc} & 1 & 2 & 3 & 4 & 5 & 6 \\ \hline 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 2 & 0 & 0 & 0 & 0 & 1 & 0 \\ 3 & 0 & 0 & 0 & 0 & 1 & 1 \\ 4 & 0 & 1 & 0 & 0 & 0 & 0 \\ 5 & 0 & 0 & 0 & 1 & 0 & 0 \\ 6 & 0 & 0 & 0 & 0 & 0 & 1 \end{array}$$

self-loop

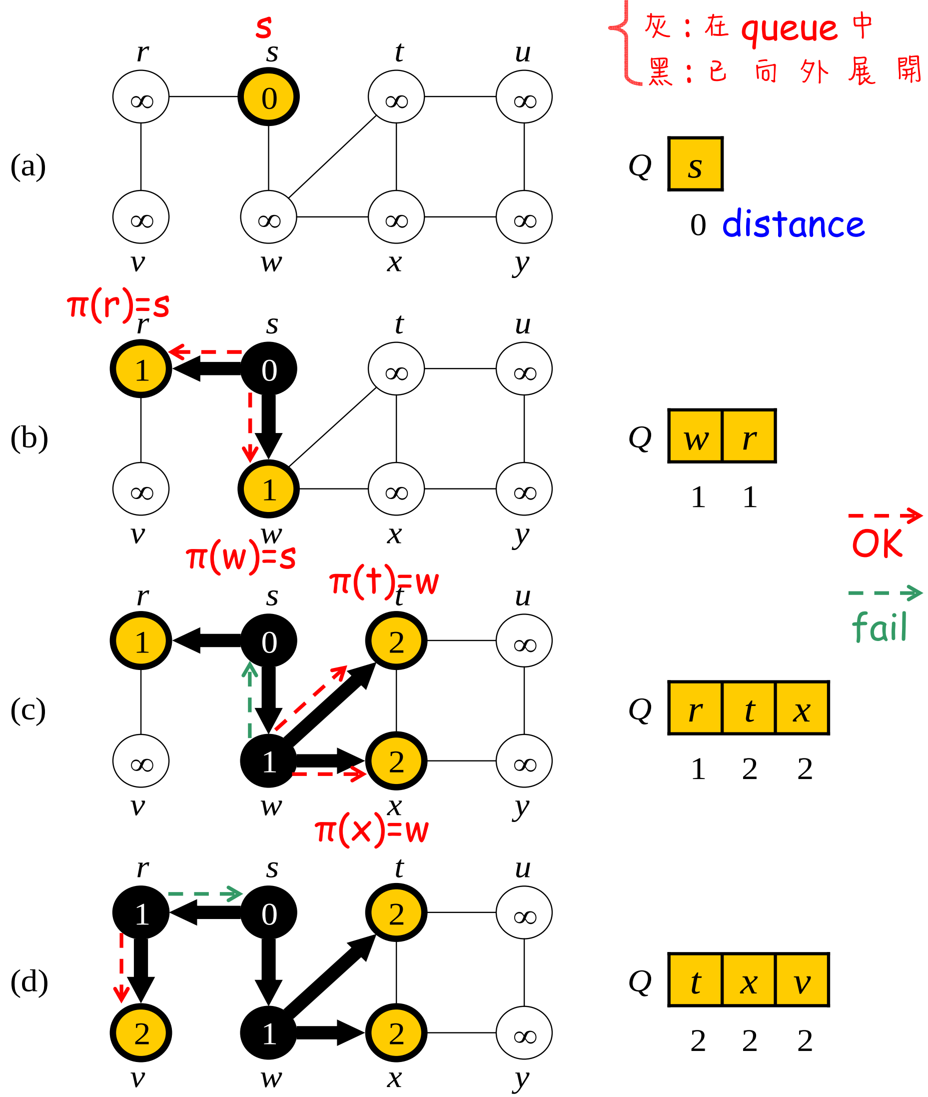
* $O(n^2)$ memory (for dense G --- m is close to n^2)
* It can be extended to *weighted graphs*.
* For unweighted G, 1-bit is enough for an edge. $\longrightarrow O(n^2)$ bits

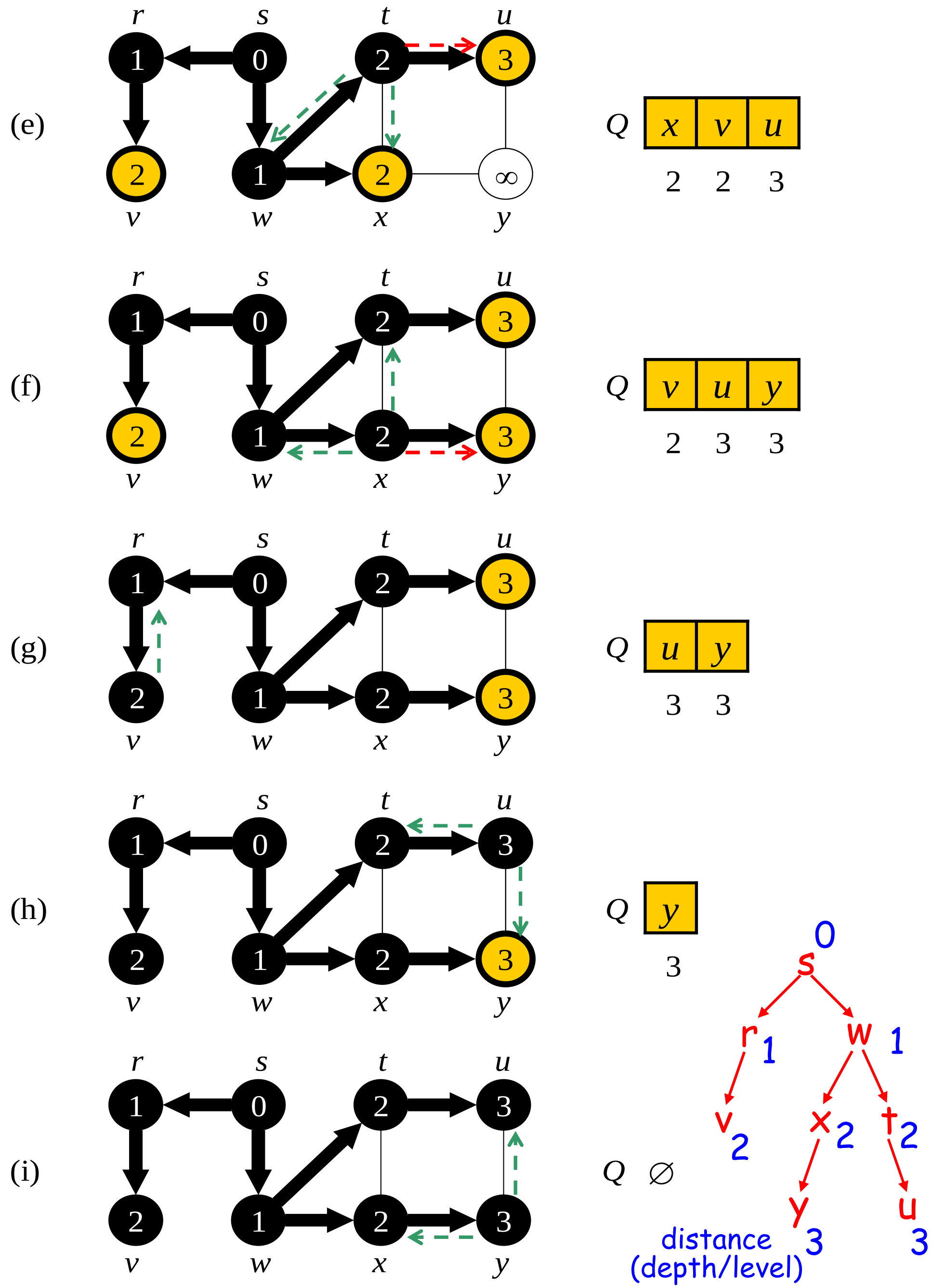
22.2 Breadth-first search

Breadth-first search / Breadth-first tree

Given *G*=(*V*, *E*) and a **source** vertex *s*∈*V*

白：未被見過
灰：在 queue 中
黑：已向外展開





白：未被見過
灰：在 queue 中
黑：已向外展開

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

Amortized
有人多，有人少
但加總相同

$\sum \text{degree}(u) = O(2m) = O(m)$

一根 edge 最多被看兩次

* $\pi(v)$: the predecessor of *v*.
* $O(V+E)$ time: using adjacency list, each edge is scanned at most twice. $V_\pi \subseteq V$ directed, not unique
* 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)

single source shortest path problem

沒指定 source
visit all nodes

22.3 Depth-first search / Depth-first forest

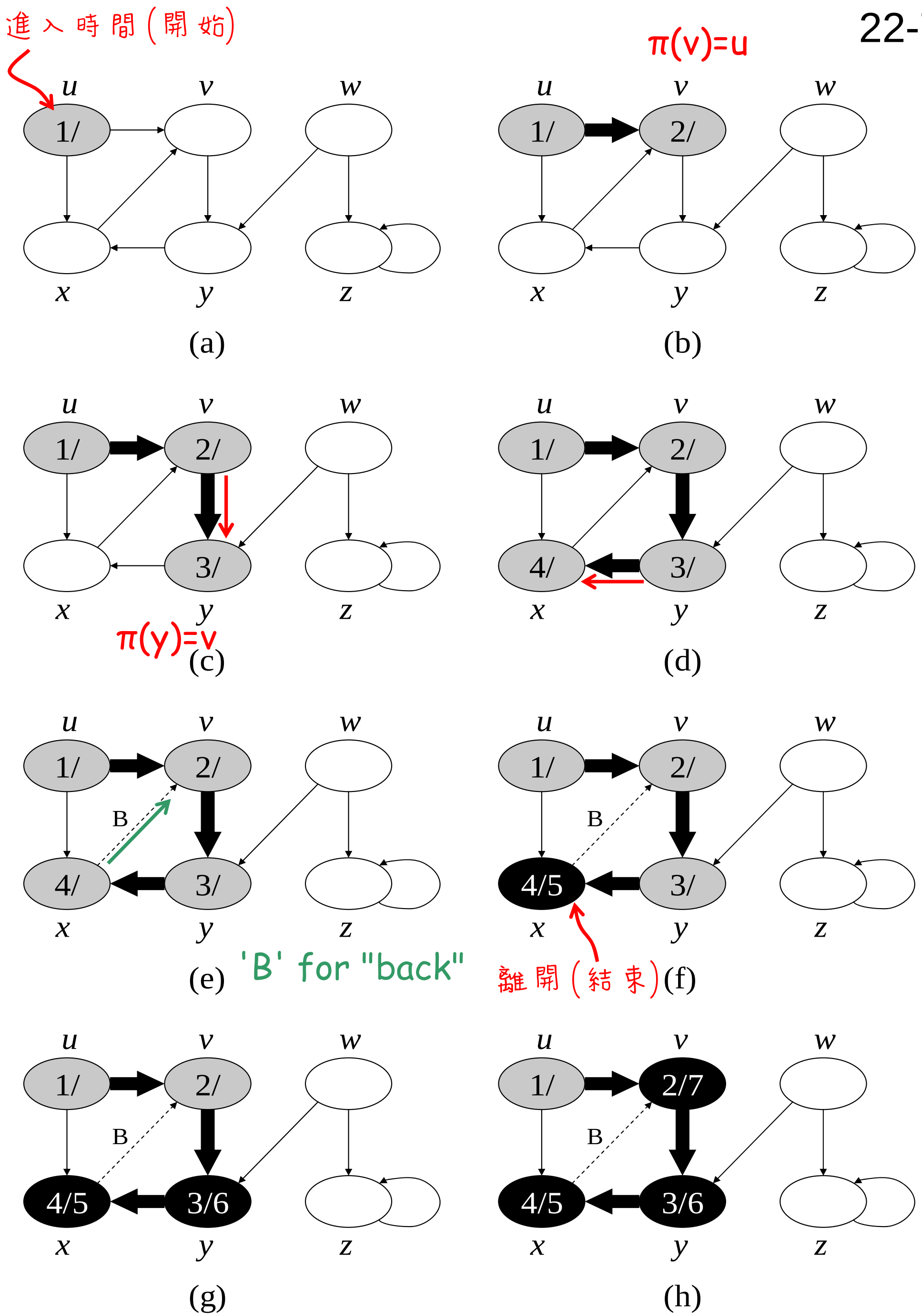
白：未被見過
灰：進入(展開中)
黑：結束(離開)

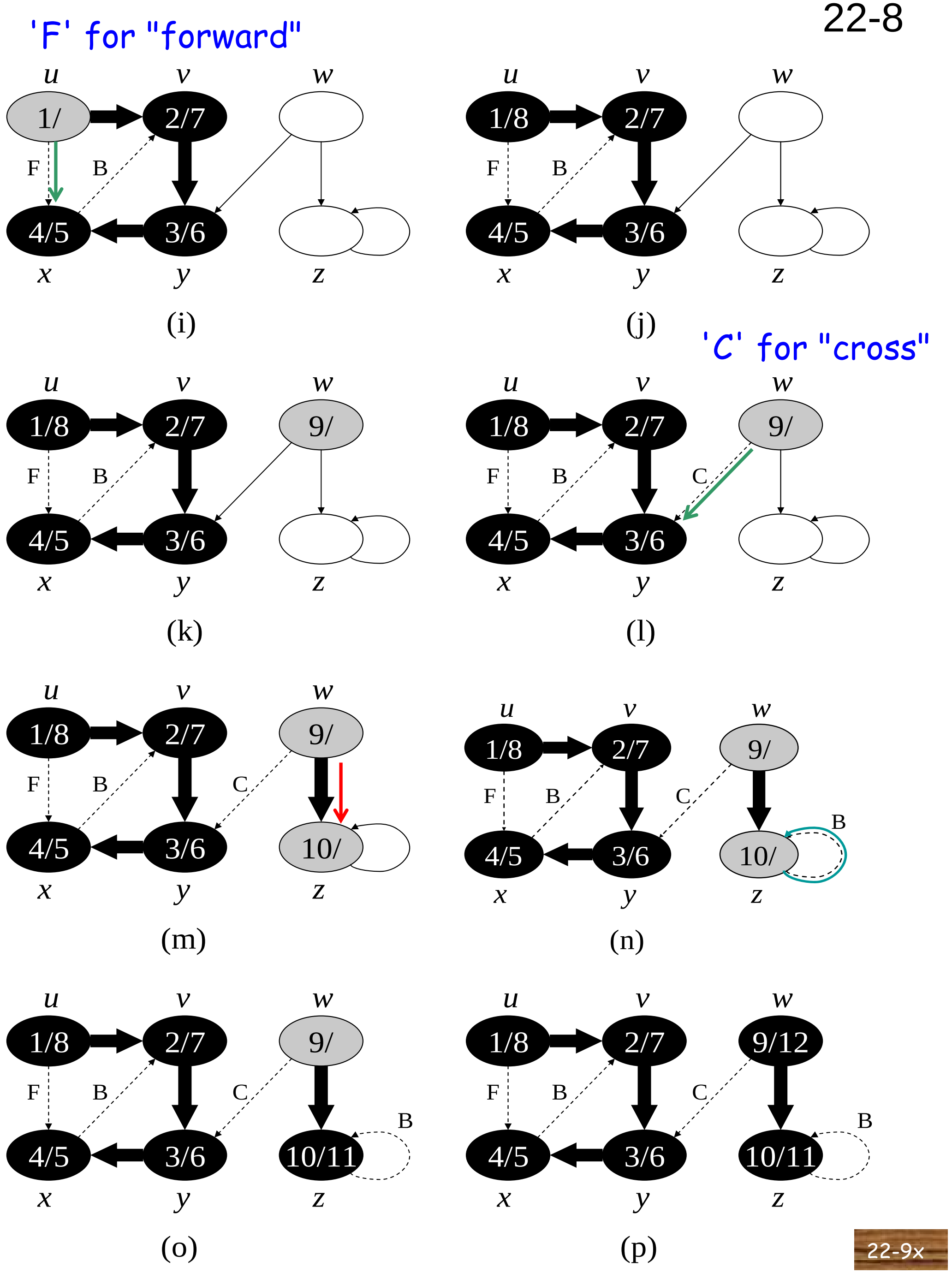
```
DFS(G)
for each vertex u ∈ V[G]
    do color[u] ← WHITE
       π[u] ← NIL
time ← 0
for each vertex u ∈ V[G]
    do if color[u] = WHITE            /*找到一個未見過的*/
          then DFS-VISIT(u)           /*往下展開一棵 tree*/
```

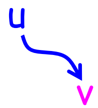
```
DFS-VISIT(u)
color[u] ← GRAY   展開中     ▷ White vertex u has just been discovered.
time ← time + 1
d[u] ← time
for each v ∈ Adj[u]          ▷ Explore edge (u, v).
    do if color[v] = WHITE
          then π[v] ← u
               DFS-VISIT(v)   /* recursive 往下展開
color[u] ← BLACK   結束      ▷ Blacken u; it is finished.
f[u] ← time ← time + 1       /* 無路可走，退回上一個 node
```

* 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: $G_\pi=(V, E_\pi)$

directed, not unique


進入時間(開始)
π(v)=u
π(y)=v
'B' for "back"
離開(結束)
u v w x y z
(a) (b) (c) (d) (e) (f) (g) (h)


'F' for "forward"
'C' for "cross"
u v w x y z
F B C
(i) (j) (k) (l) (m) (n) (o) (p)

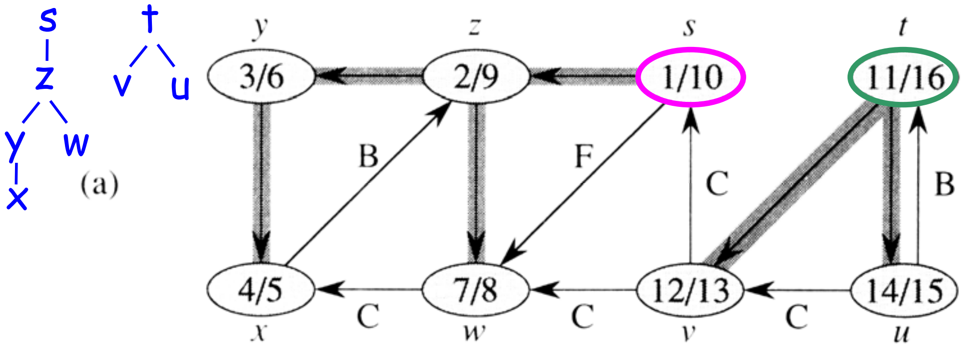


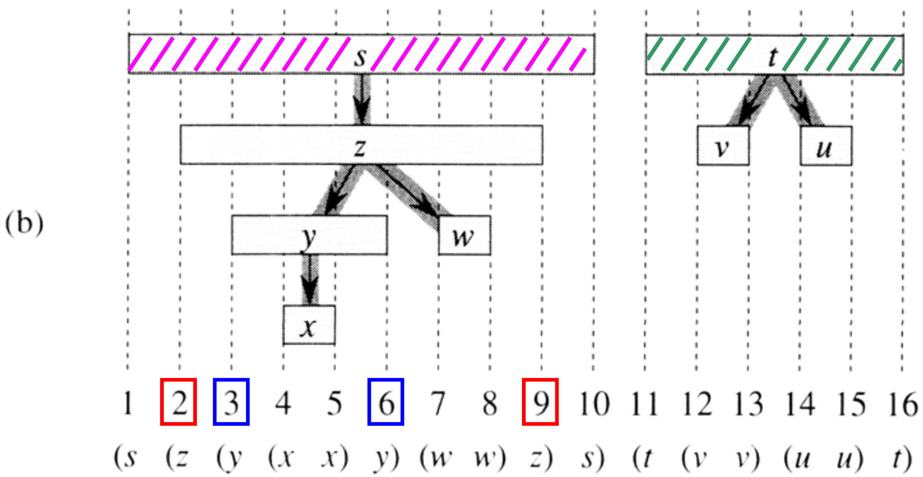
Corollary 22.8(Nesting of descendants' intervals)
Vertex *v* is a proper descendant of vertex *u* in the depth-forest for a (directed or undirected) graph *G* if and only if $d[u] < d[v] < f[v] < f[u]$.

O(n) time checking (stack)

*parenthesis structure: (well-formed)
discover *u* → "(*u*" finish *u* → "*u*)"

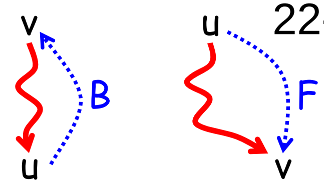
22-9y
22-9z



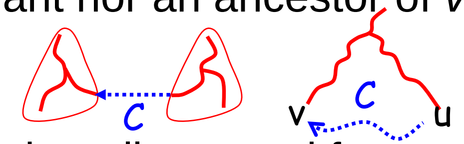


☆ if neither u or v is a descendant of the other
⇨ their intervals are disjoint !!!

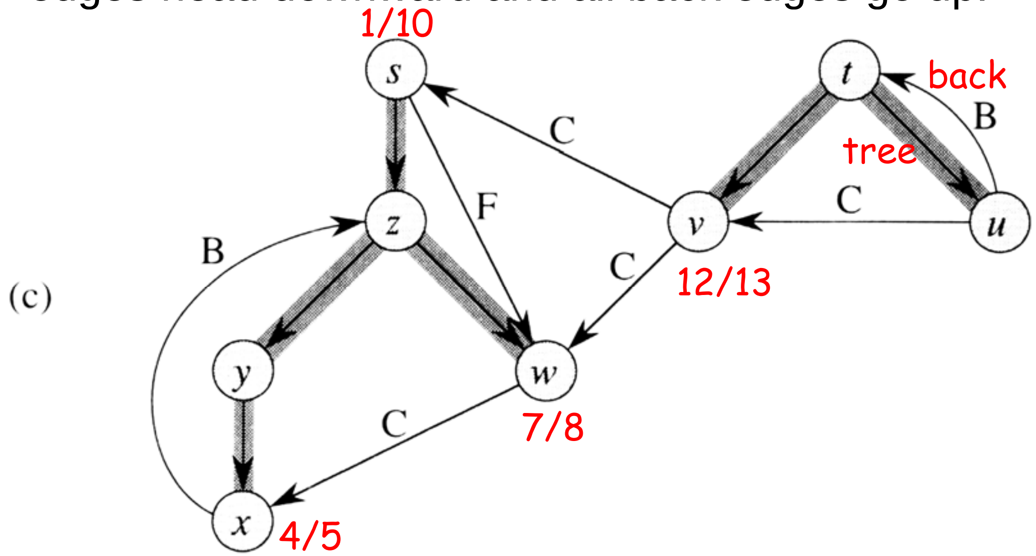
Classification of edges



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.



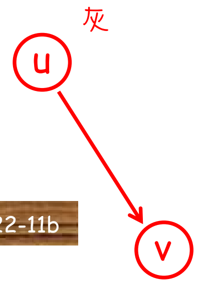
Example: redraw *G* such that all tree and forward edges head downward and all back edges go up.



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 22-11a
3. v is black $\rightarrow$ forward if $d[u]<d[v]$ 22-11b
   cross if $d[u]>d[v]$

22-11x

* If G is an undirected graph, an edge is classified as the first type that applies.
(Equivalently, the first time we see it)

Theorem 22.10 In a depth-first search of an undirected graph G, every edge is either a tree edge or a back edge. 22-11c

No cross edges !

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

22-11d

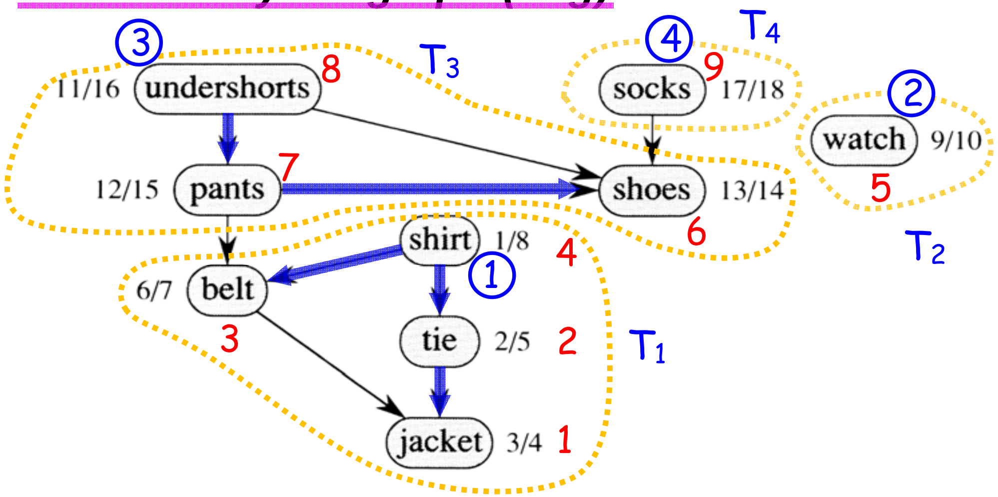
$\boxed{\text{Assume } d[u]<d[v].}$ Since e is in the adjacent list of u, v must be discovered and finished before we finished u.

$d[u] < d[v] < f[v] < f[u]$

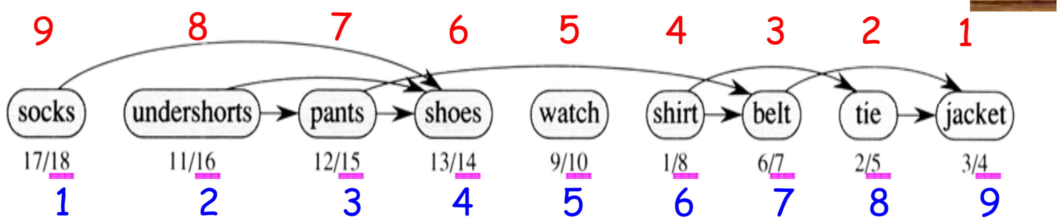
① If e is encountered from u to v, e is a tree edge.
② Otherwise, e is a back edge, since u is still gray at the time e is encountered.

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. 22-12a

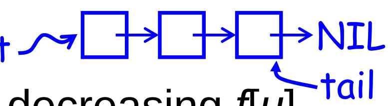


output sequence

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

(stack)



☆ * Output vertices in order of decreasing $f[u]$. 22-12c

* $O(V+E)$ (using a stack, instead of sorting)

22.5 Strongly connected components

22-13x

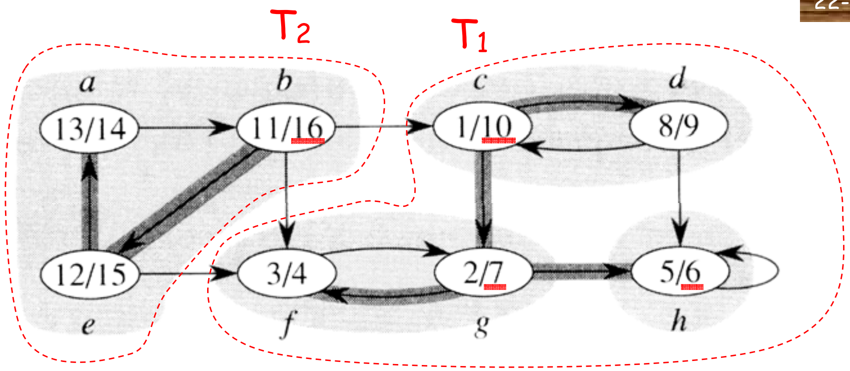
* For an undirected G, performing DFS once can obtain all "connected components"

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 $v \to u$.

22-13y

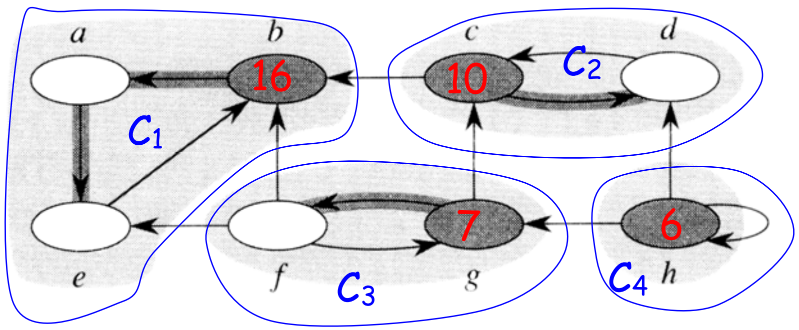
G:

(a)



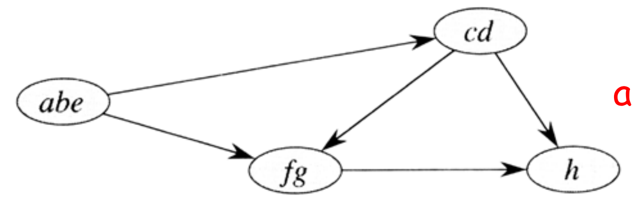
G^T:

(b)



Components

(c)



a DAG

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

* $O(V+E)$

u v u v

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

* Correctness: ??? (Refer to textbook)

22-14x

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.)