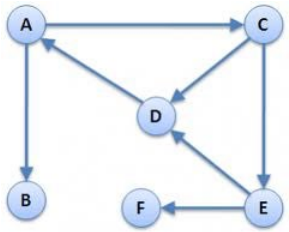
20.Elementary Graph Algorithms 基礎圖學演算法

中國文化大學
資訊工程學系 張耀鴻
112學年度第2學期

Review (of basic graph definitions)

- **Directed graph (digraph)**: G=(V, E) (有向圖)
  - V: **Vertex** set (點)
  - E: **Edge** set (邊)
  - The **edge (u,v)** is an ordered set (有序集)
  - If (u,v) is an edge, then (u,v) is **incident *from*** u and is **incident *to*** v. 以某節點為出發點的邊之個數
  - **Out degree** of a vertex = # edges incident from it
  - **In degree** of a vertex = # edges incident to it 進入某節點的邊之個數



- **Undirected graph** G=(V, E) (無向圖)
  - The edge {u,v} is an unordered pair 邊 $\overline{uv}$ 為無序集(無方向性)
  - The edge (u,v) is **incident *on*** u and v 邊 $\overline{uv}$ 連接 u 和 v
  - **Degree** of a vertex = # edges incident on it

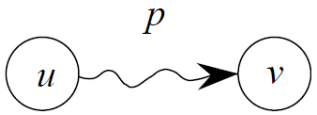
    與某節點連接的邊之個數

Review (cont.)

長度為k的路徑：即連接k個邊的路徑

- A **path** of **length** k from u to u′ is a sequence $\langle v_0, v_1, v_2, \ldots, v_k \rangle$ s.t. $u = v_0$, $u' = v_k$, and $(v_{i-1}, v_i) \in E$

可到達：u, v之間有條路徑將它們相連

- 

v is **reachable** from u via p

簡單路徑：路徑上每個邊不重覆

- A path is **simple** if $v_0, v_1, v_2, \ldots, v_k$ are distinct
- G=(V,E) is **connected** if every vertex is **reachable** from all other vertices.

連通：每個節點皆可從任意節點到達

Review (cont.)

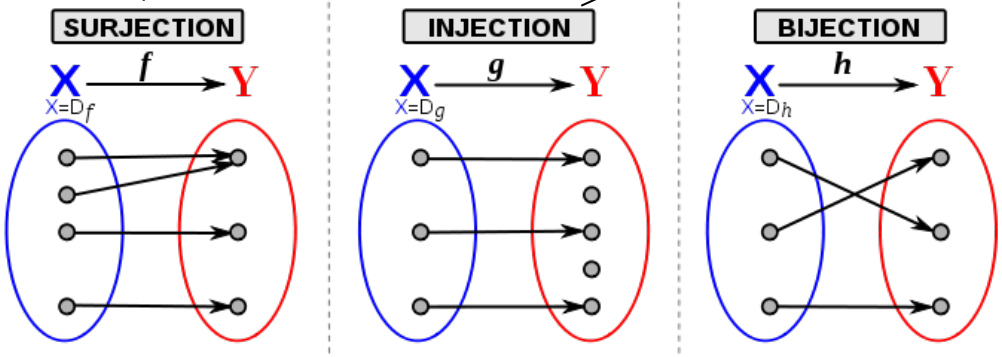
- G=(V,E) and G′=(V′,E′) are **isomorphic**
  - ✓ if there exists a bijection f: V→V′ s.t.
    (u, v) ∈ E ⇔ (f(u), f(v)) ∈ E′
- **(Read CLRS, Appendix B.4)**

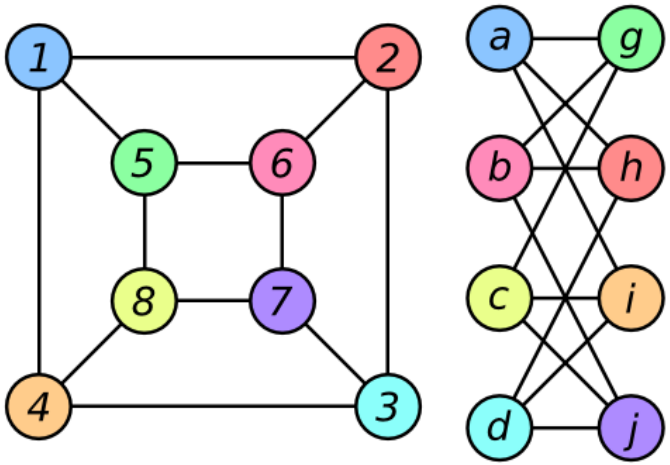
同構: A圖形的每個邊與B圖形的每個邊存在一種映射的關係

映射 = Surjection + Injection (1-1)

滿射: 每個y值皆會有對應的x (多對一)

內射: 每個y值頂多只有一個對應的x





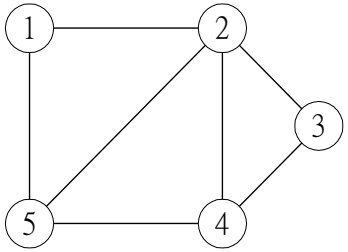
20.1 Representations of graphs

鄰接矩陣

- adjacency-matrix representation (dense)
- adjacency-list representation (sparse)

鄰接串列

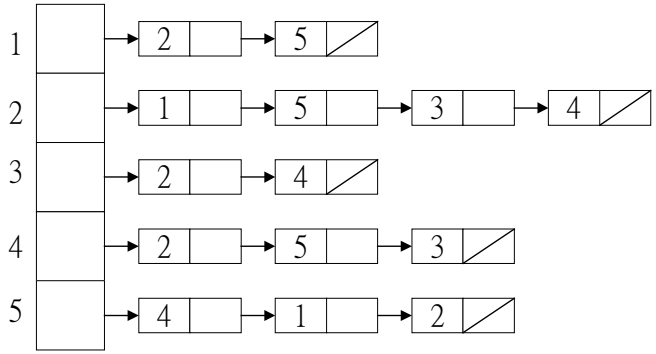
Undirected graph



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

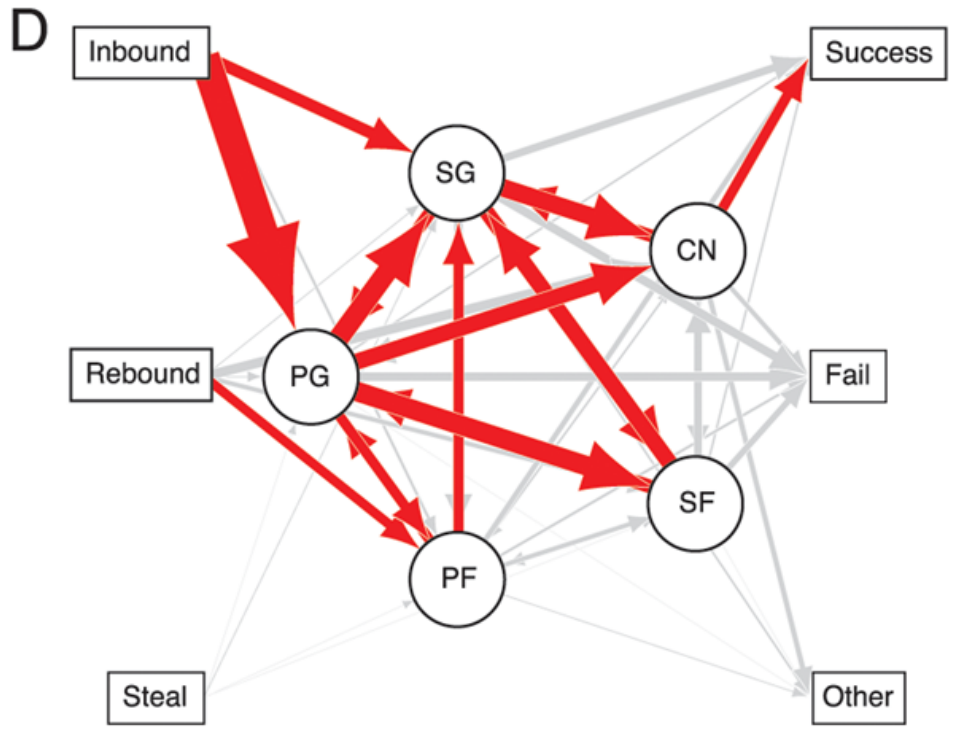
$|V| \times |V|$ matrix $A = (a_{ij})$

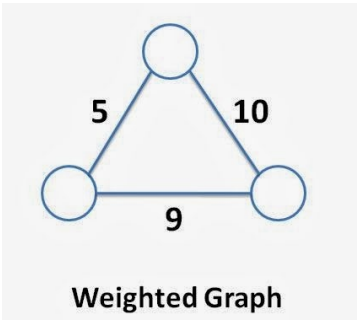
$$a_{ij} = \begin{cases} 1 & \text{if } (i, j) \in E, \\ 0 & \text{otherwise.} \end{cases}$$



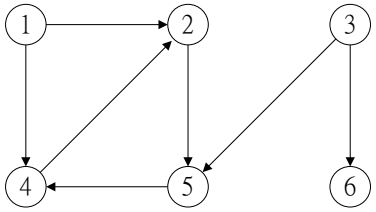
權重圖

Weighted graphs: $w: E \to R$

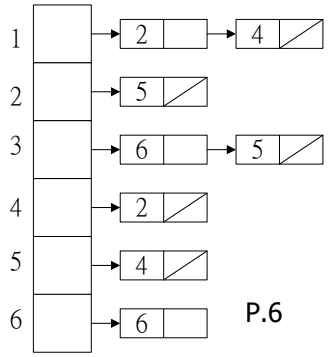




Directed graph



	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



Running time of graph algorithm

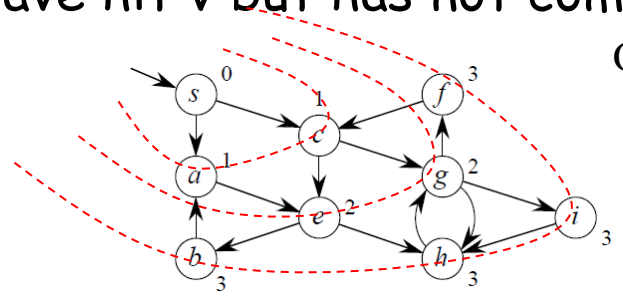
- The running time of an algorithm on graph $G=(V, E)$, it's often in terms of both $|V|$ and $|E|$.
  圖形演算法的執行時間, 常以節點數($|V|$)和邊的個數($|E|$)表示.
- In asymptotic notation—and *only in asymptotic notation—we'll drop the* cardinality. Example: $O(V+E)$.
  在漸近表示法中, 可省略基數記號(| |), 直接以 V 和 E 表示.
- e.g. Given a graph G=(V, E) using adjacent matrix representation. 以鄰接矩陣表示
  - ✓ Space 空間: $\Theta(V^2)$
  - ✓ Time: to list all vertices adjacent to u: $\Theta(V)$ 列出所有與 u 相鄰的節點
  - ✓ Time: to determine whether (u,v) $\in$ E: $\Theta(1)$ 決定u 與 v 是否相鄰

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

20.2 Breadth-first search 廣度搜尋

- The archetype for many graph algorithms 許多圖學演算法基於BFS
  - ✓ Prim's minimum-spanning-tree algorithm Prim's 最小生成樹
  - ✓ Dijkstra's single-source shortest-path algorithm Dijkstra's 最短路徑
- **Input**: Graph $G=(V,E)$, and a source vertex $s \in V$
- **Output**: v.d = distance (smallest # of edges) from s to v, for all $v \in V$ 從起始點到 v 的矩離
- **Idea**: Send a *wave* out from s. 想像從s出發的水波
  - ✓ First hits all vertices 1 edge from s. 最先遇到所有與 s 距離為1 的節點
  - ✓ From there, hits all vertices 2 edges from s, ...etc.
  - ✓ Use FIFO queue to maintain wavefront. 接著遭遇所有與 s 距離為2 的節點 以先進先出佇列Q記錄水波前線
  - ✓ $v \in Q$ iff wave hit v but has not come out yet

Q中存放被水波遇到但未離開的頂點



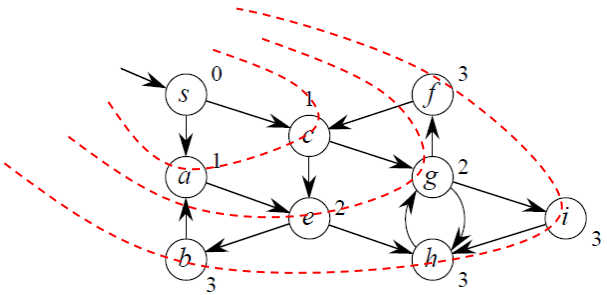
Breadth-first search Algorithm

BFS(V, E, s)

1. **for each** $u \in V - \{s\}$ 先把從起點到其他所有頂點的距離設為無窮大
   $u.d = \infty$
2. $s.d = 0$

v.d: discover time 發現(到達)時間
v.f: finishing time 結束(離開)時間

3. $Q = \phi$ 剛開始佇列Q內是空的
4. **Enqueue**(Q, s) 把起點s加到Q內
5. **while** $Q \neq \phi$ 當佇列內還有頂點沒處理完
6. $u =$ **Dequeue**(Q) 從佇列抓出第一個頂點u
7. **for each** $v \in G.Adj[u]$ 檢視所有與u相連的頂點,
8. **if** $v.d == \infty$ 若是首次檢視
9. $v.d = u.d + 1$ 則該點發現時間是u的發現時間加1
10. **Enqueue**(Q, v) 將該點加入佇列

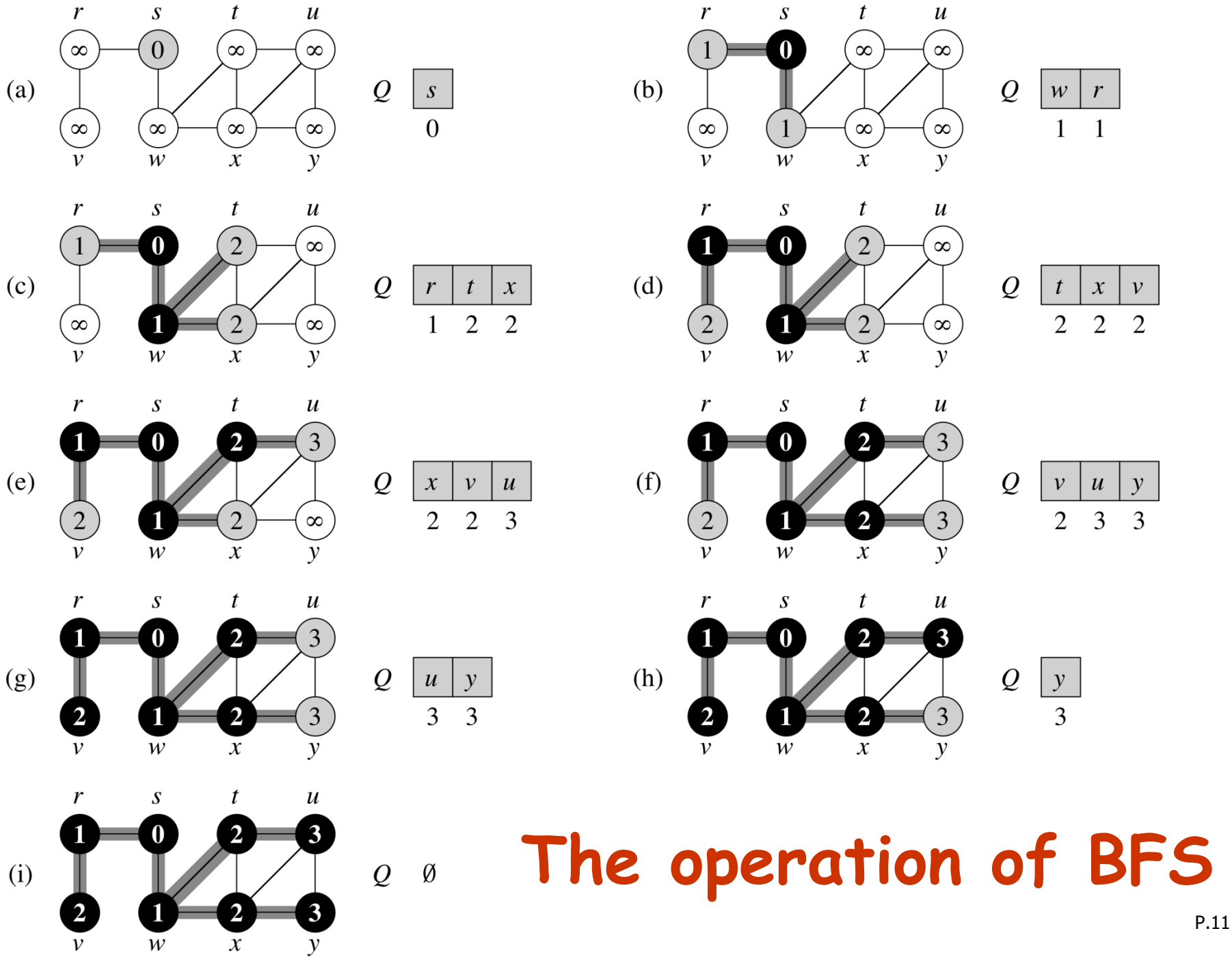


Breadth-first search Algorithm

```
BFS(V, E, s)
1.  for each u ∈ V − {s}
        u.d = ∞
2.  s.d = 0
3.  Q = ϕ
4.  Enqueue(Q, s)
5.  while Q ≠ ϕ
6.      u = Dequeue(Q)
7.      for each v ∈ G.Adj[u]
8.          if v.d == ∞
9.              v.d = u.d + 1
10.             Enqueue(Q, v)
```

v.d: discover time
v.f: finishing time

- **Aggregate analysis (合計分析, Ch17)**
- 第 8 行確保: 所有節點最多只會放進 Queue中一次.
- 故知 dequeue 最多也是一次(Line 6).
- 因此花在 Queue 運算上的時間=O(V).
- 每個邊 (u, v) 只會在 u 被 dequeue 時才會被檢視 (Line 7).
- i.e., scan each adjacent list at most once. 換言之, 每個鄰接串列頂點最多被掃描一次
- 於是, 花在掃描相鄰串列的時間= O(E).
- 綜上所述, 總共時間加起來 = O(V+E)

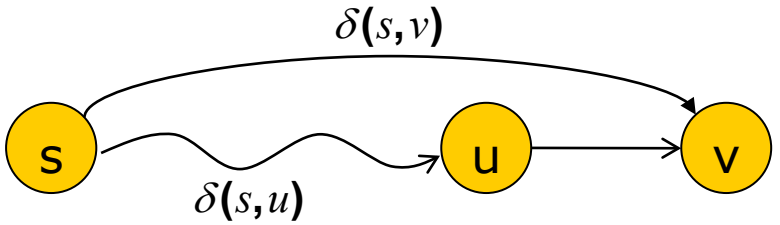


The operation of BFS

Shortest Path 最短路徑

$\delta(s,v)$: **shortest path distance** from s to v

- **Lemma 20.1**.對任意邊(u,v), s到v的最短路徑絕對不會大於s到u的最短路徑加1
  Let G = (V, E) be a directed or undirected graph, and let s ∈ V be an arbitrary vertex. Then for any edge (u,v) ∈ E, $\delta(s,v) \le \delta(s,u) + 1$



- **Proof**:
  1. If u is reachable from s, then so is v.
  2. Shortest path from s to v cannot be longer than the shortest path from s to u followed by edge (u,v), and thus the inequality holds.
  從 s到v 的最短路徑一定會小於或等於 s到u 的最短路徑再加上邊(u,v),故得證.

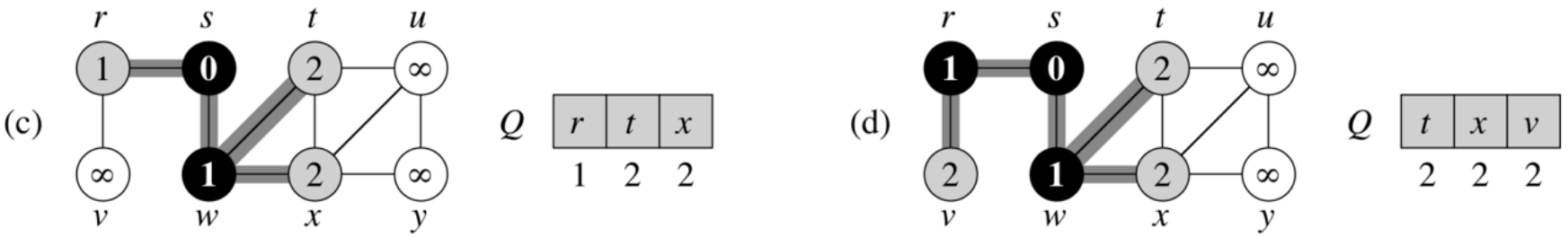
當BFS結束時, 從 s到任意節點 v 的距離永遠大於或等於 s到 v 的最短路徑.

- **Lemma 20.2.** Let $G = (V, E)$ be a directed or undirected graph, and suppose that BFS is run on G from a given source $s \in V$.
  Then upon termination, for each vertex $v \in V$, the value $v.d$ computed by BFS satisfies
  $$v.d \geq \delta(s, v).$$
- **Proof**. (Induction on #times a vertex is placed in the queue)(以歸納法對一個節點放進Queue的次數證明.) 當BFS結束時, 以下式子成立:

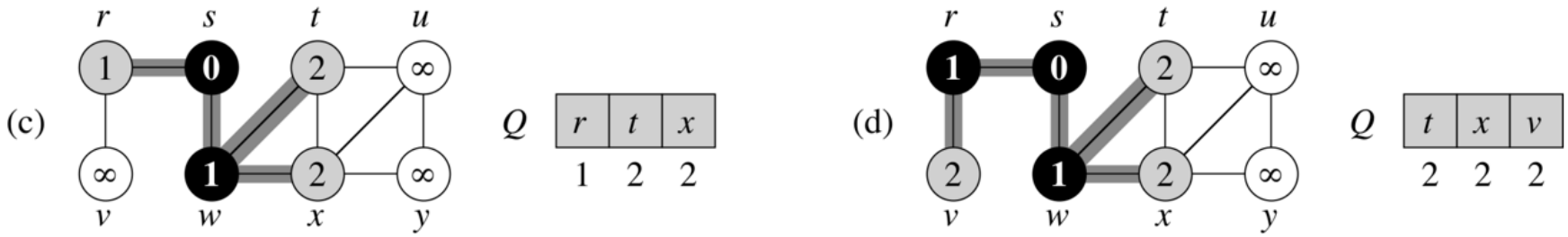
$$\begin{aligned} v.d &= u.d + 1 && \text{(Line 9)} \\ &\geq \delta(s, u) + 1 && \text{(Inductive hypothesis)} \\ &\geq \delta(s, v). && \text{(Lemma 22.1)} \end{aligned}$$

- **Lemma 20.3.** Suppose that during the execution of BFS on a graph $G = (V, E)$, the queue Q contains the vertices $\langle v_1, v_2, \ldots, v_r \rangle$, where v_1 is the head of Q and v_r is the tail. Then $v_r.d \le v_1.d + 1$ and $v_i.d \le v_{i+1}.d + 1$ for $i = 1, 2, \ldots, r-1$.

若在BFS執行過程中, Queue的內容為$\langle v_1, v_2, \ldots, v_r \rangle$.
則從起始點到v_i的距離不會大於下個節點v_{i+1}再加1,
而且從起始點到Queue中最後一個節點的距離不會大於第一個節點再加1.



例: r移出Queue, v加入Queue, 此時 v.d <= r.d+1, 而r.d <=t.d+1, 因此, v.d <=t.d+1



- Proof of Lemma 20.3.
  (induction on #queue operations)

(仔細檢視虛擬碼, 並以歸納法對Queue運算的次數證明)

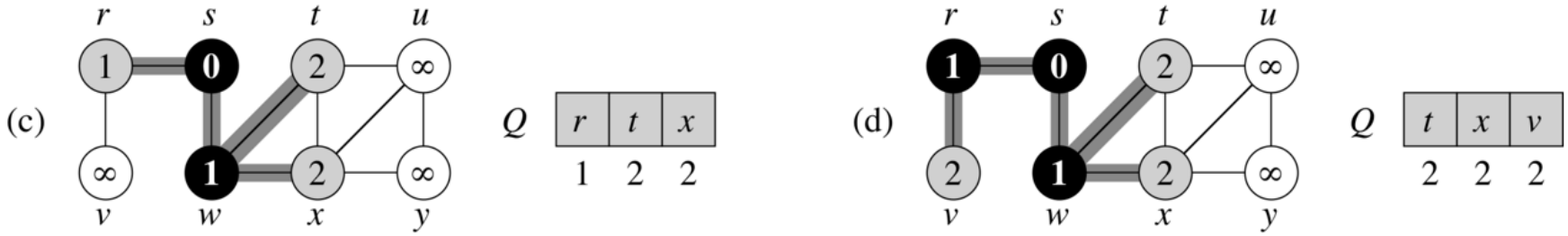
當 v_1 dequeue 時, v_2 成為新的頭, 由歸納假設知, $v_1.d \le v_2.d + 1$
同時 $v_r.d \le v_1.d + 1 \le v_2.d + 1$ 亦成立

當 Line 10 **Enqueue**(Q, v) 時, 此新節點成為v_{r+1}.
假設 u 是上一個被移出 Queue 的節點, 由歸納假設知,
對於新的頭而言, $v_1.d \ge u.d$ (Inductive hypothesis)

$$\text{Thus, } v_{r+1}.d = v.d = u.d + 1 \le v_1.d + 1$$

$$\text{Also have } v_r.d \le u.d + 1 = v.d = v_{r+1}.d$$

因此, 當 v engueue 時, Lemma 20.3 亦成立.



- **Corollary 20.4**. Suppose vertices v_i and v_j are enqueued during the execution of BFS, and that v_i is enqueued before v_j is enqueued.

  Then $v_i.d \le v_j.d$ at the time that v_j is enqueued.

  若在BFS執行過程中, v_i 比 v_j 更早進入 Queue.
  則 $v_i.d \le v_j.d$ (起始點到v_i 的距離不會超過 v_j.)

- ***proof*** Immediate form Lemma 20.3 and the property that each vertex receives a finite d value at most once during the course of BFS.

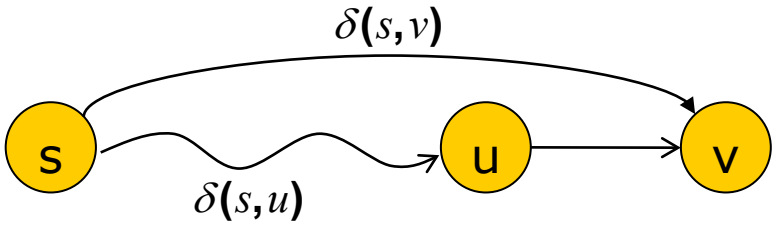
  直接由引理 20.3 以及BFS 執行時每個節點只會被附予一個d值即可得證.

- We can now prove that breadth-first search correctly finds shortest-path distances.

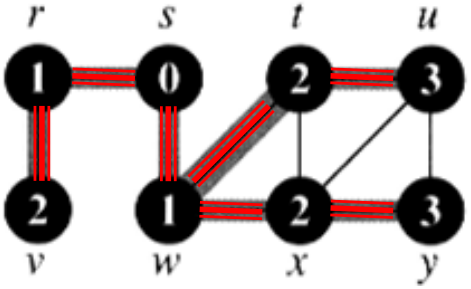
Theorem 20.5 (Correctness of BFS)

- Let $G = (V, E)$ be a directed or undirected graph, and suppose that BFS is run on G from a given source $s \in V$.
  Then, during its execution, BFS discovers every vertex $v \in V$ that is reachable from the source s, and upon termination $v.d \geq \delta(s, v)$ for all $v \in V$.
  Moreover, for any vertex $v \neq s$ that is reachable from s, one of the shortest paths from s to v is the shortest path from s to $v.\pi$ followed by the edge $(v.\pi, v)$.
- ***Proof***. By induction.

當BFS結束時, $v.d \geq \delta(s, v)$, 而且 $\delta(s, v) = \delta(s, v.\pi) +$ 邊 $(v.\pi, v)$
換言之, s 到 v 的最短路徑 = s 到 v 前一點的最短路徑 +邊 $(v.\pi, v)$



Breadth-first trees 廣度優先樹



- For a graph $G = (V, E)$ with source s, we define the ***predecessor subgraph*** of G as
  $G_\pi = (V_\pi, E_\pi)$ where 前身子圖G_π：由圖G中所有節點和邊的前身所組成
  $V_\pi = \{v \in V : v.\pi \neq \text{NIL}\} \cup \{s\}$, and ($V_\pi$：節點V的前一個節點)
  $E_\pi = \{(v.\pi, v): v \in V_\pi - \{s\}\}$. ($E_\pi$：節點V與前一個節點所成的邊)
  The edges in E_π are called ***tree edges***. (E_π 又稱為樹邊)

- **Lemma 20.6.** When applied to a directed or undirected graph $G = (V, E)$, procedure BFS constructs π so that the predecessor subgraph $G_\pi = (V_\pi, E_\pi)$ is a ***breadth-first tree***.

  由BFS 所形成的樹為廣度優先樹

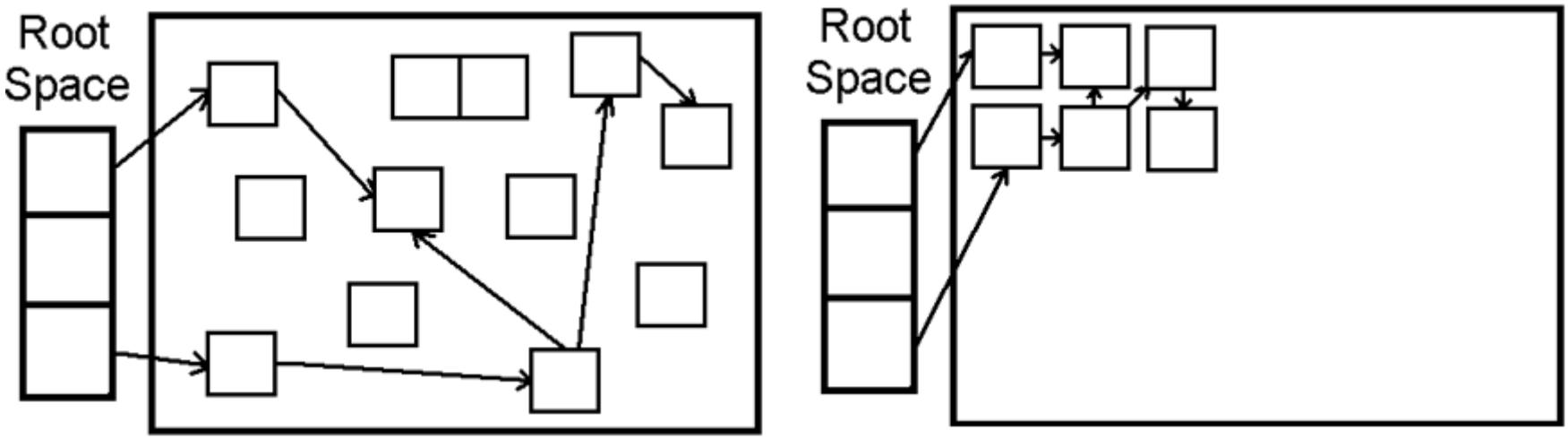
PRINT_PATH(G, s, v)

```
PRINT_PATH(G, s, v )
1  if  v == s
2  then print s
3  else if  v.π == NIL
4         then print "no path from" s "to" v  "exist"
5         else PRINT_PATH (G, s, v.π)
6                print v
```

Applications of BFS

- **Garbage Collection:**
  - ✓ The garbage collection technique, "Cheney's algorithm" uses breadth-first traversal for copying garbage collection.

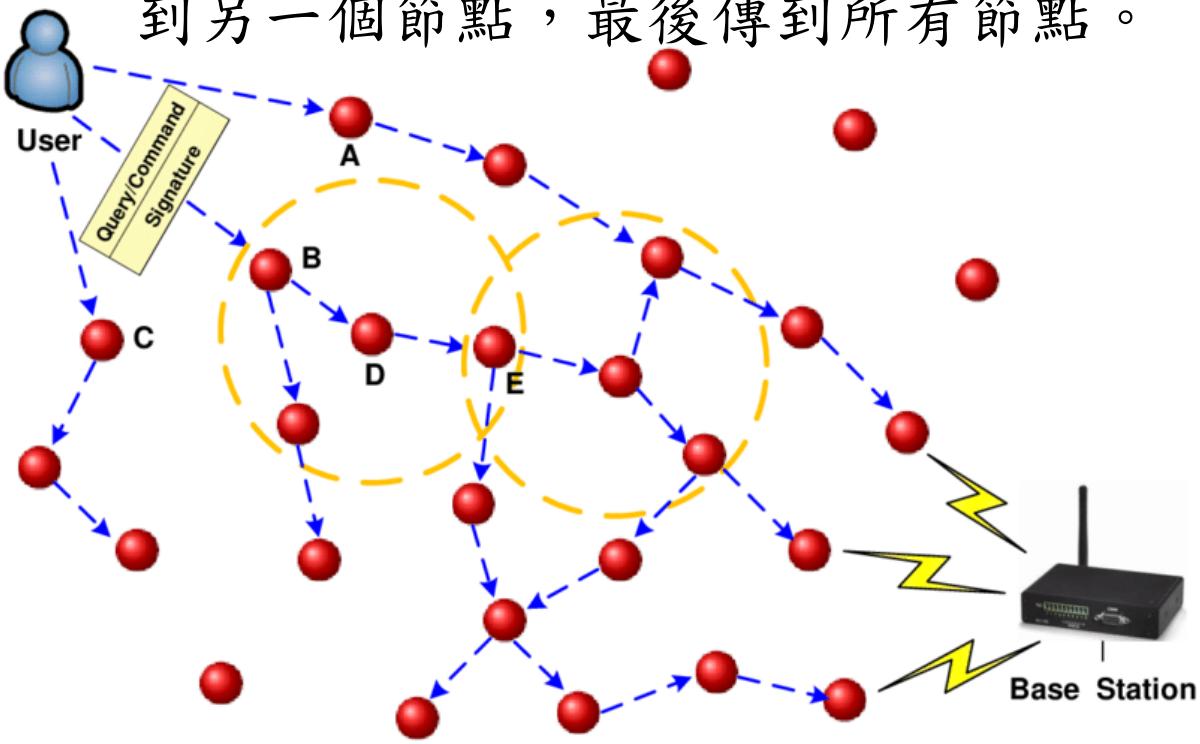
    Cheney's演算法以BFS演算法走訪記憶體節點來進行垃圾收集。



Applications of BFS (Cont'd)

- **Broadcasting In Networks:**
  - ✓ A packet travels from one node to another using the BFS technique in the broadcasting network to reach all nodes.

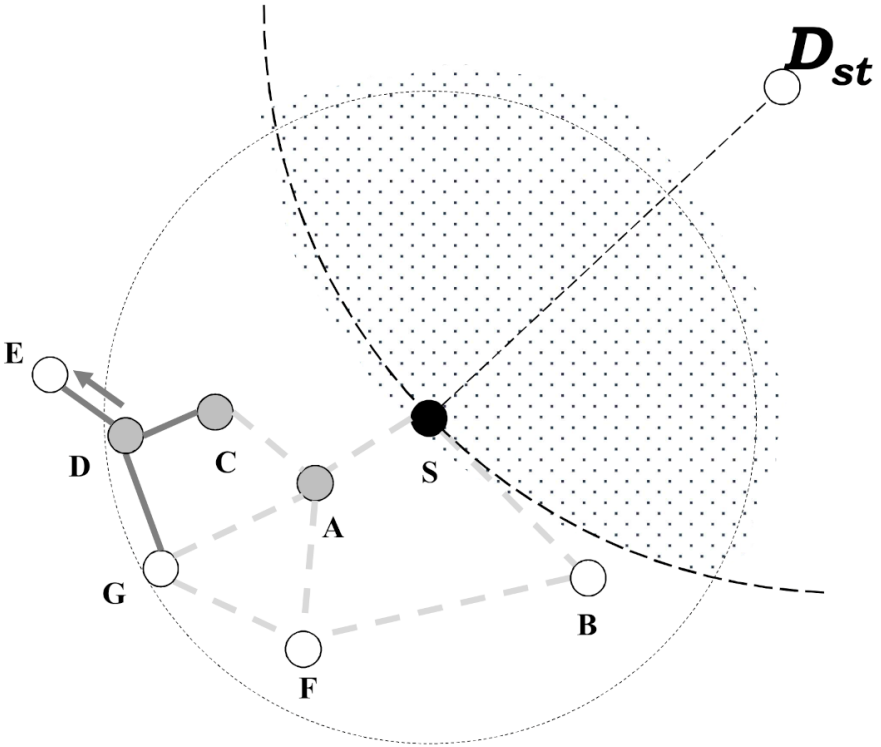
網路廣播時，封包用BFS技術從一個節點傳輸到另一個節點，最後傳到所有節點。



Applications of BFS (Cont'd)

- **GPS Navigation:**
  - ✓ We can use BFS in GPS navigation to find all the adjacent or neighboring location nodes.

GPS導航時，以BFS來尋找所有相鄰或鄰近位置的節點。



Applications of BFS (Cont'd)

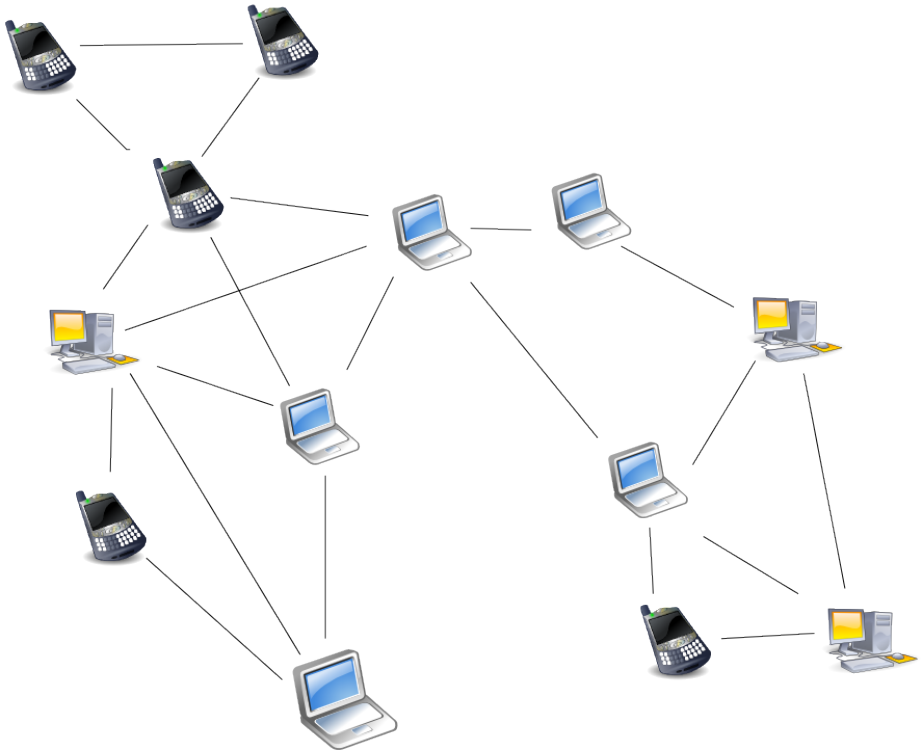
- **Social Networking Websites:**
  - ✓ Given a person 'P', we can find all the people within a distance, 'd' from p using BFS till the d levels.

    給定某個人P，可用BFS找到與P距離d的所有人。

Applications of BFS (Cont'd)

- **Peer To Peer Networks:**
  - Again BFS can be used in peer to peer networks to find all the adjacent nodes.

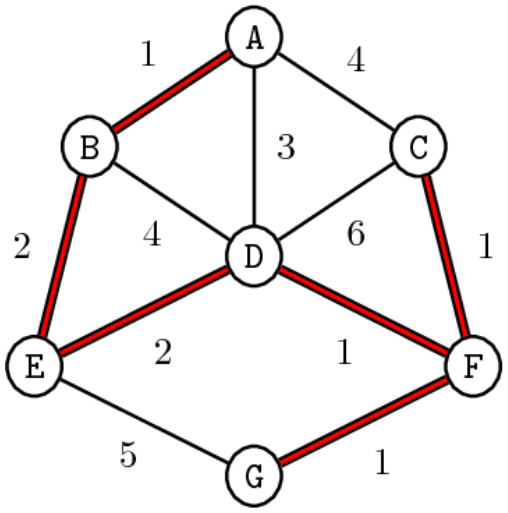
BFS可以在P2P網路中找到所有相鄰的節點



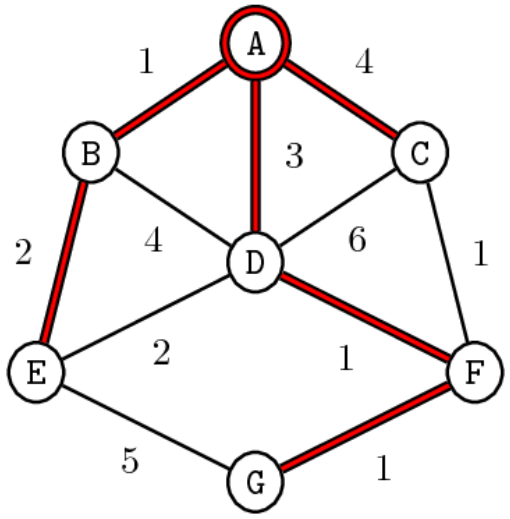
Applications of BFS (Cont'd)

- **Shortest Path And Minimum Spanning Tree In The Un-weighted Graph:**
  - ✓ BFS technique is used to find the shortest path i.e. the path with the least number of edges. Similarly, we can also find a minimum spanning tree using BFS.
    利用BFS技術可找出最短路徑，即圖中邊數最少的路徑。同樣也可以用BFS來找出圖中最小生成樹。

Minimum Spanning Tree



Single-Source Shortest Path



20.3 Depth-First Search 深度優先搜尋

- **Input**: $G = (V, E)$, **no source vertex**. 無起始點
- **Output**: 2 ***timestamps*** on each vertex: 時間戳記
  - ✓ $v.d$ = discovery time 發現時間
  - ✓ $v.f$ = finishing time 結束時間
- Will methodically explore *every* edge. 系統化探測每個邊
  - ✓ Start over from different vertices as necessary.
- As soon as we discover a vertex, explore from it.
  - ✓ Unlike BFS, which puts a vertex on a queue.

一但發現一個節點, 便由此為出發點探測下一個 (無需Queue)

- As DFS progresses, every vertex has a ***color***: 節點顏色
  - ✓ WHITE = undiscovered 未發現
  - ✓ GRAY = discovered, but not done exploring from it 還沒完
  - ✓ BLACK = finished (have found everything reachable from it)已結束
- Discovery and finishing times: 發現時間與結束時間
  - ✓ Unique integers from 1 to 2|*V* |. 介於 1 到 2|V| 之間
  - ✓ For all *v*, *v.d* < *v.f* . 發現時間比結束時間早
- In other words, $1 \le v.d < v.f \le 2|V|$

1 ≤發現時間 ≤結束時間 ≤節點數 x 2

Depth-First Search

DFS(G)

```
1  for each u ∈ G.V
2      u.color = WHITE
3  time = 0
4  for each u ∈ G.V
5      if u.color == WHITE
6          DFS-VISIT(G, u)
```

$\Theta(V)$

一開始每個頂點都是白色(未發現)

時間一開始為0

$\Theta(V)$

一一處理每一個頂點

若該頂點狀態是未發現,則從該頂點出發進行深度搜尋拜訪所有與該點相連的其他頂點.

Depth-First Search

記錄初次拜訪時間

檢查每個相連鄰居

若是新鄰居則去拜訪一下

所有鄰居都拜訪完則記錄完成時間

```
DFS-VISIT(G, u)
1  time = time + 1
2  u.d = time    // discover u
3  u.color = GRAY
4  for each v ∈ G.Adj[u]
5      if v.color == WHITE
6          DFS-VISIT (G, v)
               // explore
7  u.color = BLACK
8  time = time + 1
9  u.f = time    // finish u
```

$|Adj[v]|$

Depth-First Search

DFS(G)

```
1  for each u ∈ G.V                    } Θ(V)
2      u.color = WHITE
3  time = 0
4  for each u ∈ G.V                    } Θ(V)
5      if u.color == WHITE
6          DFS-VISIT(G, u)
```

DFS-VISIT(G, u)

```
1  time = time + 1
2  u.d = time    // discover u
3  u.color = GRAY
4  for each v ∈ G.Adj[u]               } |Adj[v]|
5      if v.color == WHITE
6          DFS-VISIT (v) // explore
7  u.color = BLACK
8  time = time + 1
9  u.f = time    // finish u
```

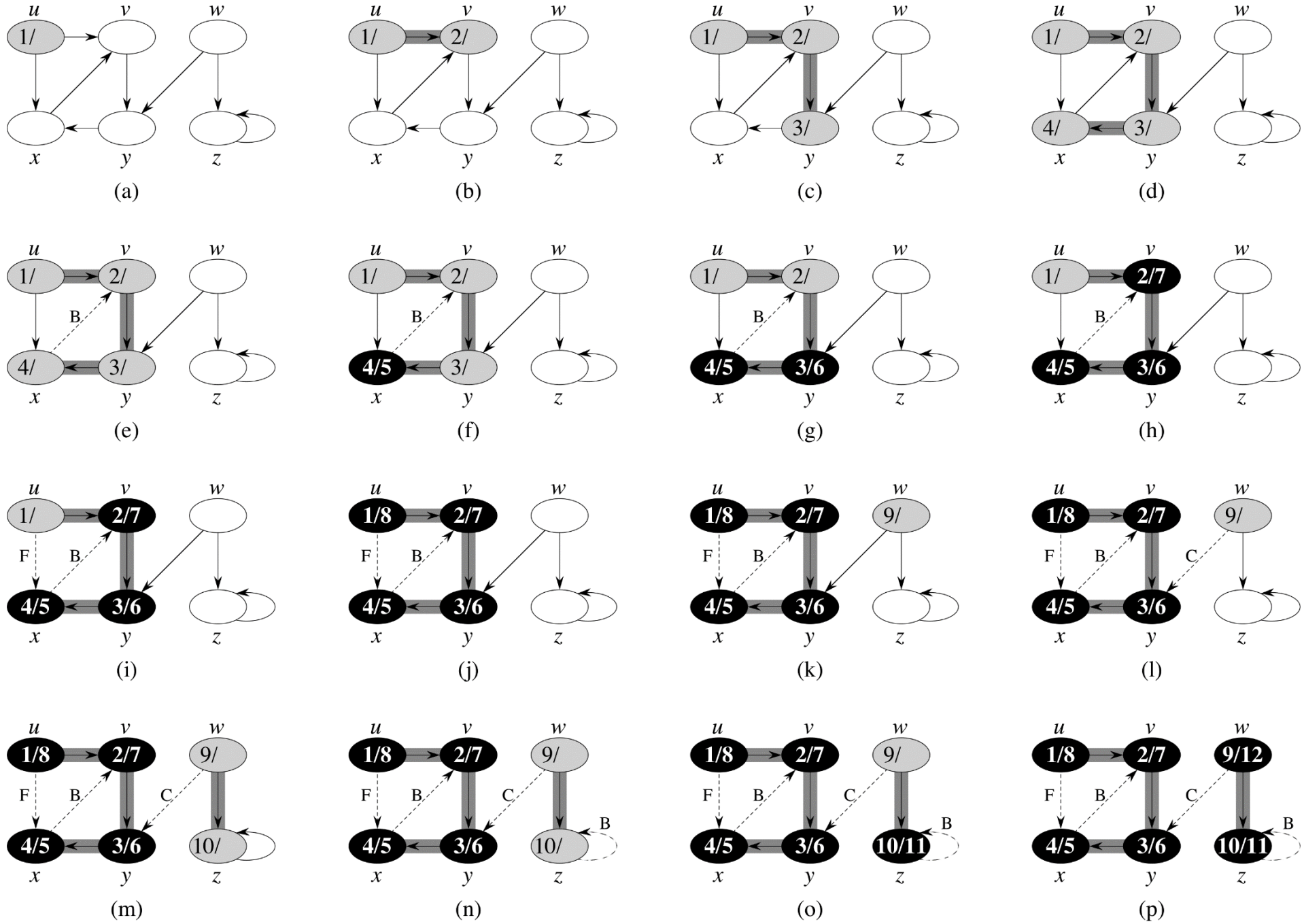
Aggregate analysis:

每一個節點呼叫一次 DFS-VISIT ➔ **Θ(*V*)**

執行 DFS-VISIT 時, Line 4-6 for-迴圈會做 $|Adj[v]|$ 次 ➔ 總共 $2|E|$

therefore, complexity = **Θ (*V*+*E*)**

$$\sum_{v \in V} |Adj[v]| = \Theta(E)$$


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

Properties of depth-first search

Theorem 20.6. (Parenthesis theorem) 括弧定理

In any depth-first search of a graph $G = (V, E)$ for any two vertices u and v, exactly one of the following conditions holds:

在DFS執行過程中，考慮任意兩節點 u, v，以下三種情況恰有一種成立:

1. the intervals $[u.d, u.f]$ and $[v.d, v.f]$ are entirely disjoint.
   u, v的發現時間和結束時間不重疊
2. the interval $[v.d, v.f]$ is contained entirely within the interval $[u.d, u.f]$, and v is a descendant of u, or (v 為 u 的後裔)
3. the interval $[u.d, u.f]$ is contained entirely within the interval $[v.d, v.f]$, and u is a descendant of v. (v 為 u 的祖先)

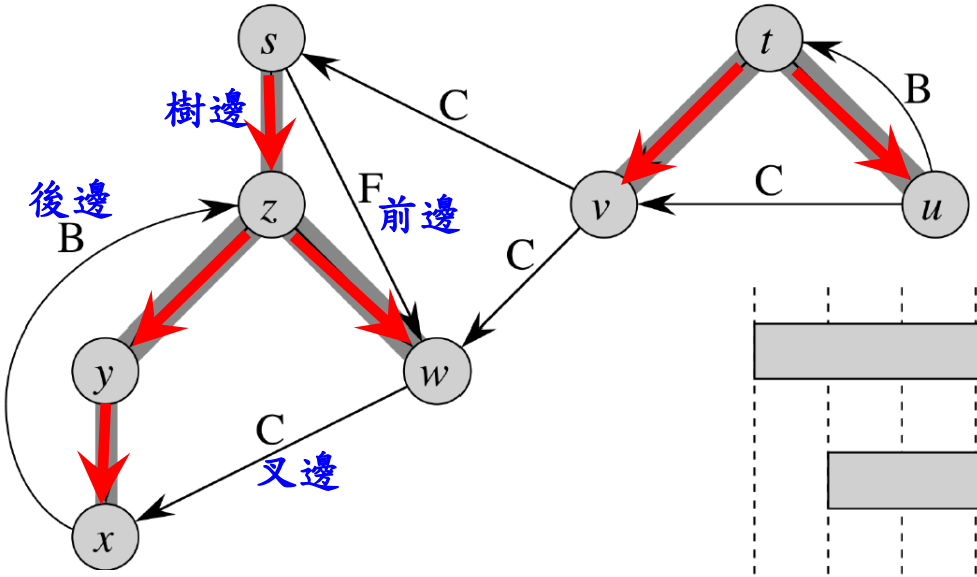
- Like parentheses:
  - ✓ ( ) [ ]　　([ ])　　[( )]　　➔ OK
  - ✓ ([ )]　　[( ])　　➔ Not OK

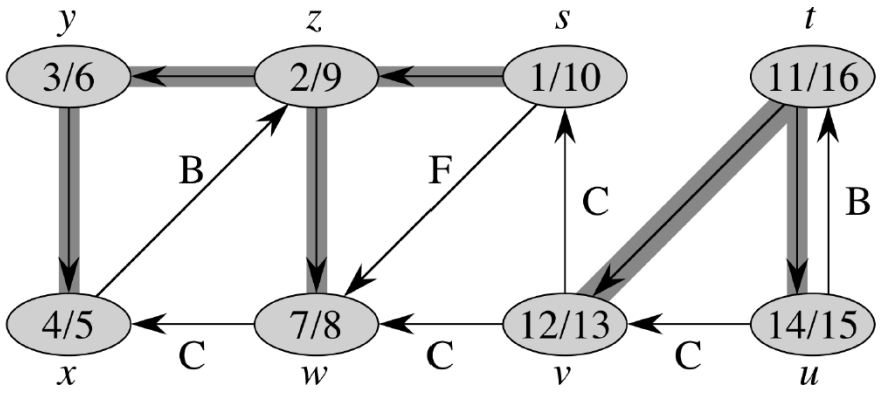
Corollary 22.8. (Nesting of descendants' interval) 巢狀後裔區間

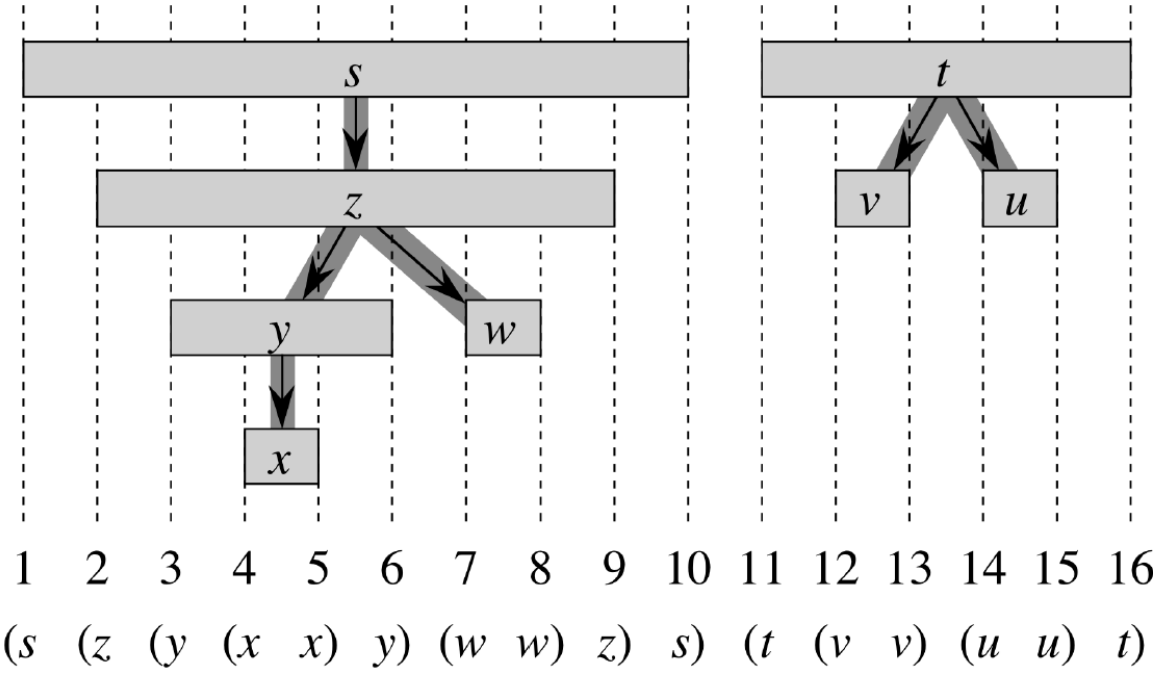
Vertex v is a ***proper descendant*** of a vertex u in the depth-first forest for a (directed or undirected) graph G if and only if $u.d < v.d < v.f < u.f$.

節點 v 為 u 的真後裔 ⇔ v 比 u 晚發現, 而且早結束

Property of DFS







在圖G的深度優先森林中, 節點 v 是 u 的後裔
⇔ 在發現 u 時, 可從 u 出發經由白點抵達 v

Theorem 20.9 (**white path theorem**) 白路徑定理

In a **depth-first forest** of a (directed or undirected) graph $G = (V, E)$, vertex v is a descendant of vertex u if and only if at time $u.d$ that the search discover u, vertex v can be reached from u along a path consisting entirely of white vertices.

Classification of edges: 邊的分類

樹邊 Tree edges: 在深度優先森林中, 由探索 (u, v) 邊所找到
後邊 Back edges: 邊 (u, v), 其中 u 是 v 的後代
前邊 Forward edges: 邊(u,v),其中 v 是 u 的後代
叉邊 Cross edges: any other edges

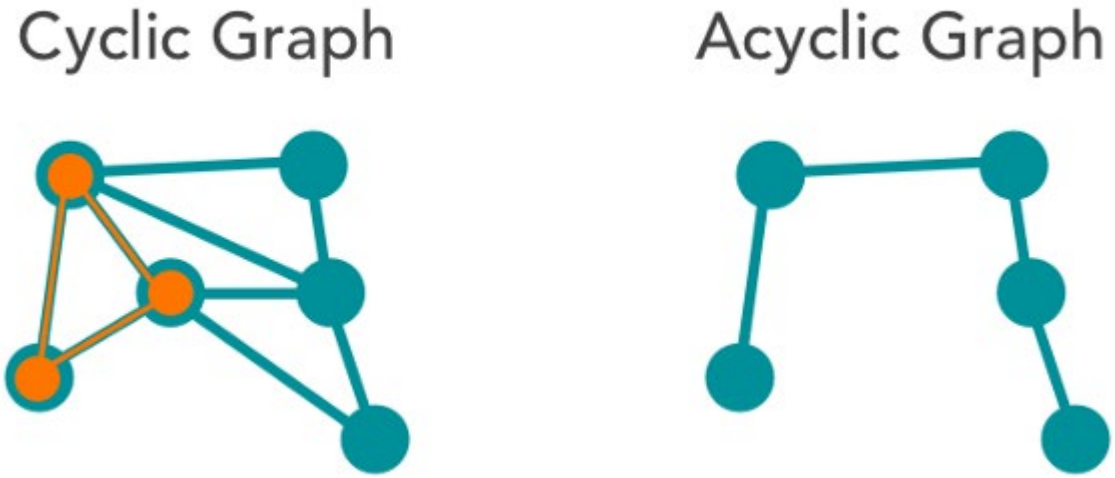
Theorem 20.10. In a depth-first search of an *un*directed graph G, every edge of G is either a **tree edge** or a **back edge**.

在無向圖G進行DFS, 所有的邊不是樹邊就是後邊

Applications of DFS

- **Detecting Cycles In The Graph:** 檢測圖中的迴圈
  - ✓ If we find a back edge while performing DFS in a graph then we can conclude that the graph has a cycle. Hence DFS is used to detect the cycles in a graph.

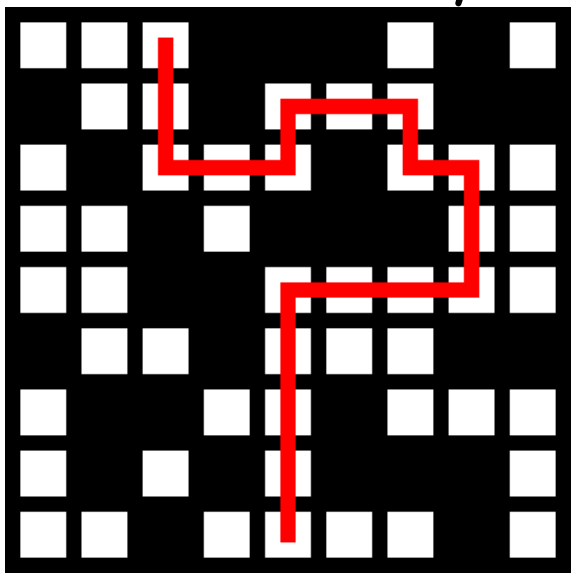
  如果在圖中執行DFS時發現了一個back edge，那麼我們可以得出結論：圖有一個迴圈。因此，DFS被用來檢測圖中的迴圈



Applications of DFS (Cont'd)

- **Pathfinding:** 路徑搜尋
  - ✓ Given two vertices x and y, we can find the path between x and y using DFS. We start with vertex x and then push all the vertices on the way to the stack till we encounter y. The contents of the stack give the path between x and y.

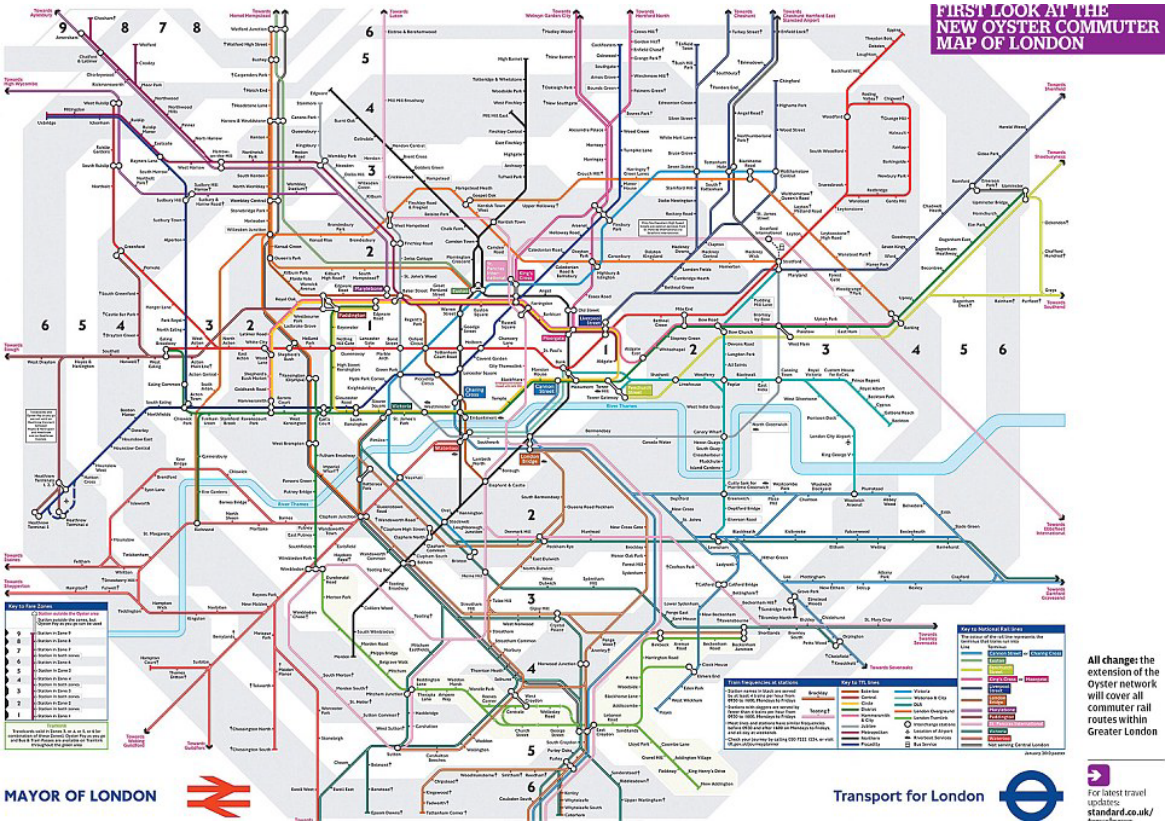
  給定兩個頂點x和y，可以用DFS找到x和y之間的路徑。從頂點x開始，然後將所有頂點推到stack中，直到遇見y為止。stack的內容即是x和y之間的路徑。



Applications of DFS (Cont'd)

- **Minimum Spanning Tree And Shortest Path:**
  - DFS traversal of the un-weighted graph gives us a minimum spanning tree and shortest path between nodes.

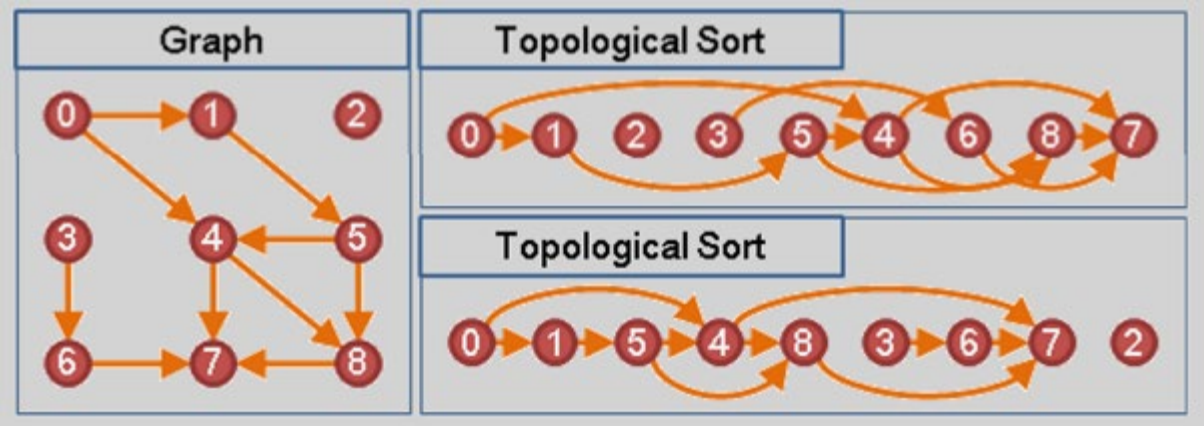
    走訪DFS可找出了最小生成樹和節點間的最短路徑。



Applications of DFS (Cont'd)

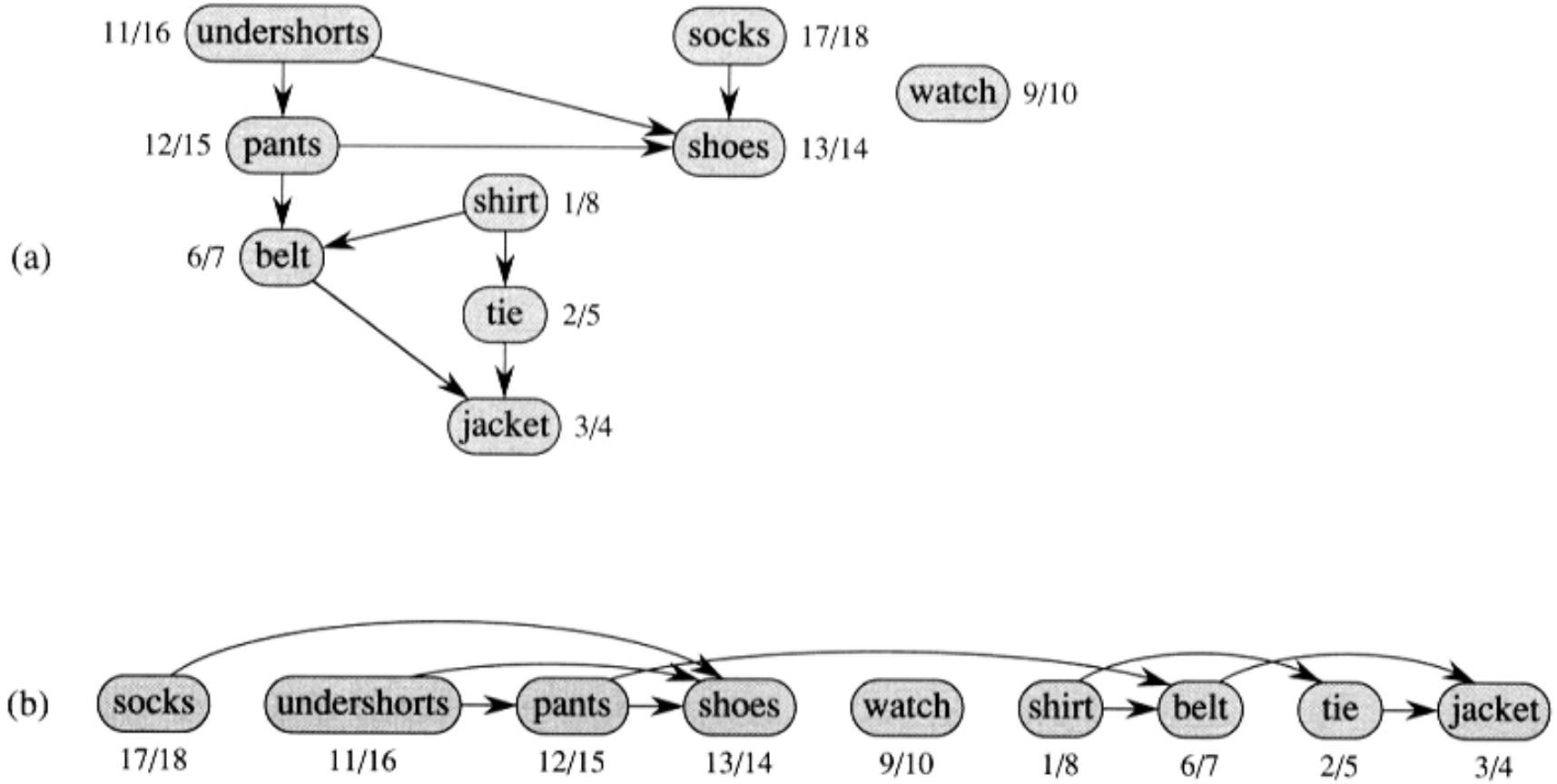
- **Topological Sorting:**
  - We use topological sorting when we need to schedule the jobs from the given dependencies among jobs. In the computer science field, we use it mostly for resolving symbol dependencies in linkers, data serialization, instruction scheduling, etc. DFS is widely used in Topological sorting.

    當我們需要根據工作之間給定的依賴關係來調度工作時，可使用拓撲排序。在資訊科學領域中，DFS主要用於解決連接器中的符號依賴、資料序列化、指令調度等問題。DFS廣泛應用於拓撲排序。



20.4 Topological sort

A topological sort of a dag $G = (V, E)$ is a linear ordering of all its vertices such that if G contains an edge (u, v), then u appears before v in the ordering.


(a)
11/16
undershorts
socks
17/18
watch
9/10
12/15
pants
shoes
13/14
shirt
1/8
6/7
belt
tie
2/5
jacket
3/4
(b)
socks
undershorts
pants
shoes
watch
shirt
belt
tie
jacket
17/18
11/16
12/15
13/14
9/10
1/8
6/7
2/5
3/4

TOPOLOGICAL_SORT(*G*)

1 call DFS(*G*) to compute finishing time f(*v*) for each vertex *v*.

2 as each vertex is finished, insert it onto the front of a link list.

3 return the link list of vertices

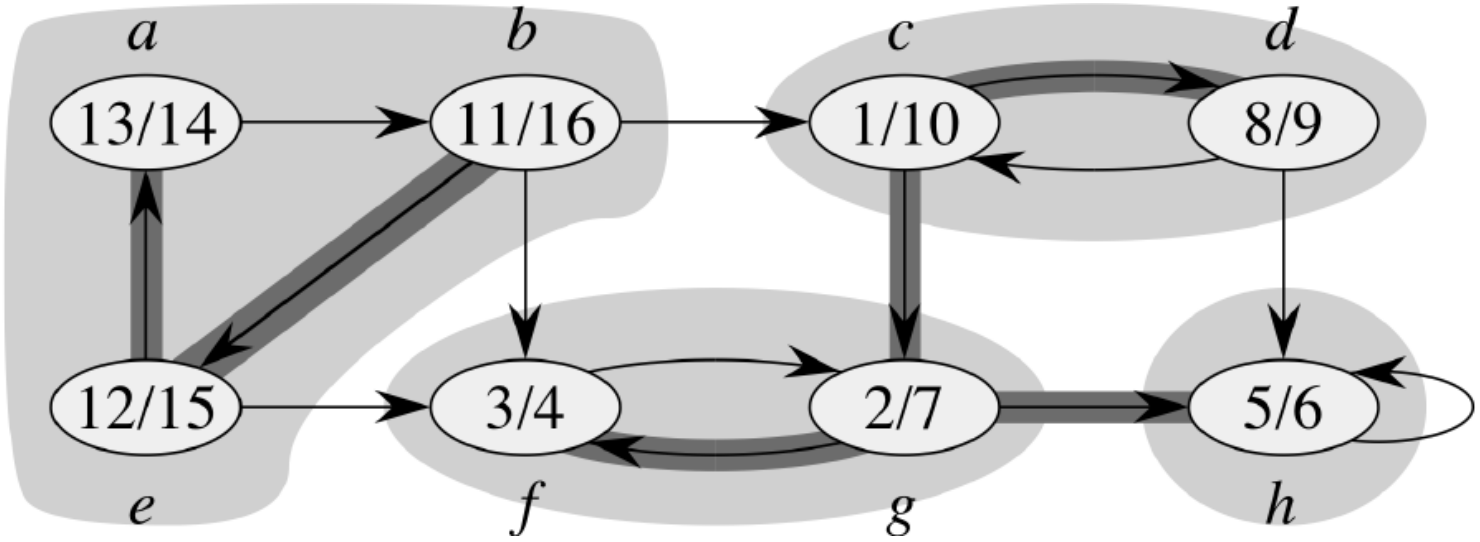
Lemma 22.11. A directed graph G is acyclic if and only if a depth first search of G yields no back edge.

Theorem 22.12. TOPOLOGICAL_SORT(G) produces a topological sort of a directed acyclic graph G.

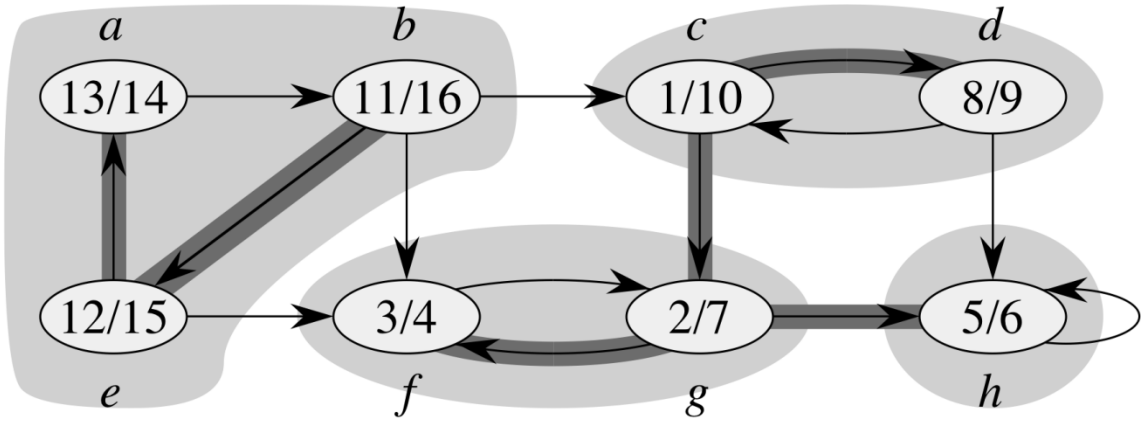
強連通元件

20.5 Strongly connected components

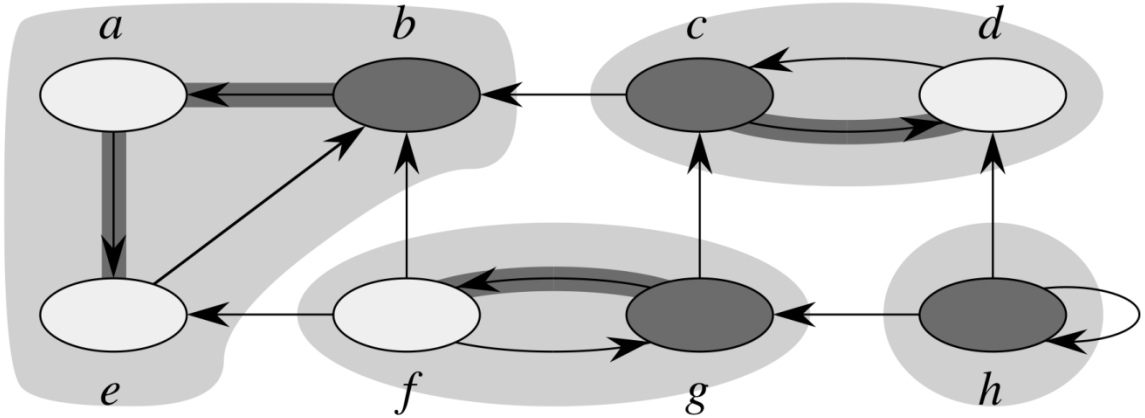
- Strongly connected graph 強連通圖
  - ✓ 若有向圖為 strongly connected
    則任意兩個頂點均有path可到達。
- Strongly connected component 強連通元件為最大強連通子圖
  - ✓ 一個圖的Strongly connected component
    為圖上的 maximal strongly connected subgraph



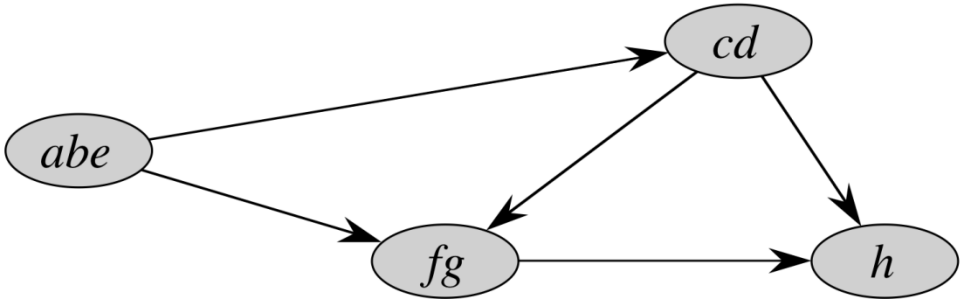
(a)

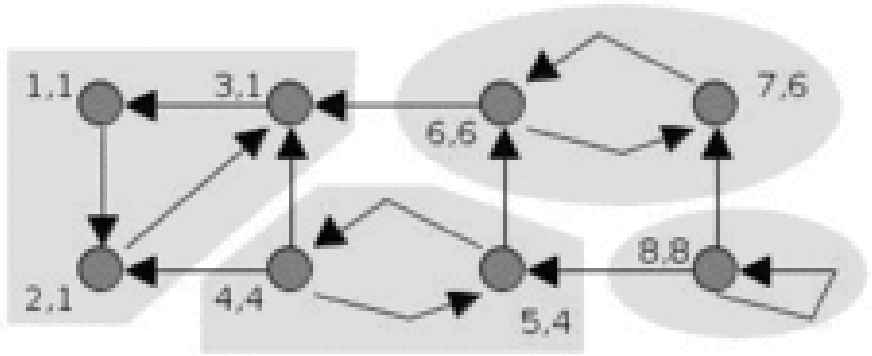


(b)



(c)





STRONGLY_CONNECTED_COMPONENT(*G*)

1 call DFS(G) to compute finishing time $u.f$ for each vertex u $O(V+E)$

呼叫 DFS(G) 算出每個節點的結束時間

2 compute G^T 計算轉置圖 G^T (所有vertex維持不變，edge的方向顛倒)

3 call DFS(G^T), but in the main loop of DFS, consider the vertices in the order of decreasing $u.f$. 以G的轉置圖呼叫 DFS(G^T), 並以結束時間較慢的節點為優先考量 $O(V+E)$

4 Output the vertices of each tree in the depth-first forest of step 3 as a separate strongly connected components.
步驟3所得的深度優先森林即為強連通元件

Complexity: $O(V+E)$

Lemma 22.13. Let C and C' be distinct strongly connected components in directed graph $G = (V, E)$, let u, $v \in C$, let u', $v' \in C'$, and suppose that there is a path $u \rightsquigarrow u'$ in G. Then there cannot also be a path $v' \rightsquigarrow v$ in G.

Lemma 22.14. Let C and C' be distinct strongly connected components in directed graph $G = (V, E)$. Suppose that there is an edge$(u, v) \in E$, where $u \in C$ and $v \in C'$. Then $f(C) > f(C')$.

Corollary 22.15. Let C and C' be distinct strongly connected components in directed graph $G = (V, E)$. Suppose that there is an edge $(u, v) \in E^{\mathrm{T}}$, where $u \in C$ and $v \in C'$. Then $f(C) < f(C')$.

Theorem 22.16. Strongly-connected-component(G) correctly computes the strongly connected components of a directed graph G.

Summary

- Basic definitions
- Representations
  - ✓ Adjacency matrix
  - ✓ Adjacency list
- BFS
  - ✓ Shortest path
- DFS
  - ✓ Parenthesis theorem
  - ✓ White path theorem
  - ✓ Tree edge, back edge, forward edge, cross edge
- Strongly connected components