Algorithms
Chapter 22
Elementary Graph Algorithms

Associate Professor: Ching-Chi Lin
林清池 副教授
chingchi.lin@gmail.com

Department of Computer Science and Engineering
National Taiwan Ocean University

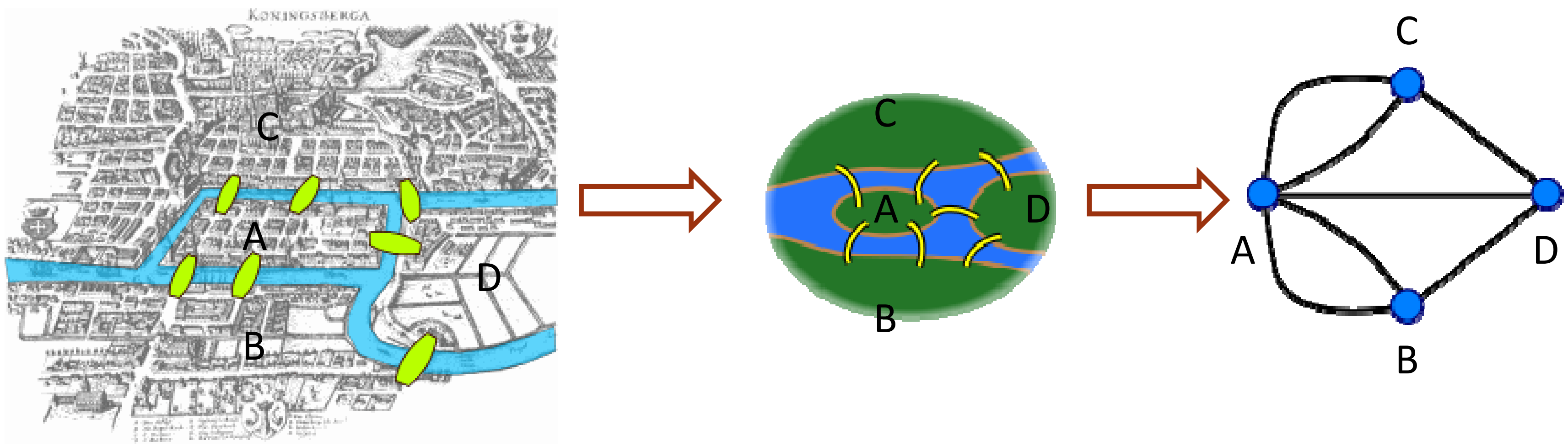
Outline

- **Representations of graphs** 圖的表示法
- Breadth-first search 廣度优先
- Depth-first search 兩种拜訪圖的方法：深度优先
- Topological sort 拓樸排序：算先後順序
- Strongly connected components

Konigsberg Bridge Problem 生活問題 $\xrightarrow{\text{graph}}$ 数学問題

- Can we walk across all the bridges **exactly once** in returning back to the starting land area ?

  在不重覆情況下走過每一座橋一次,最後回到原点



- Transferring to Graph model
  - Land → vertex
  - Bridge → edge

Graph representation

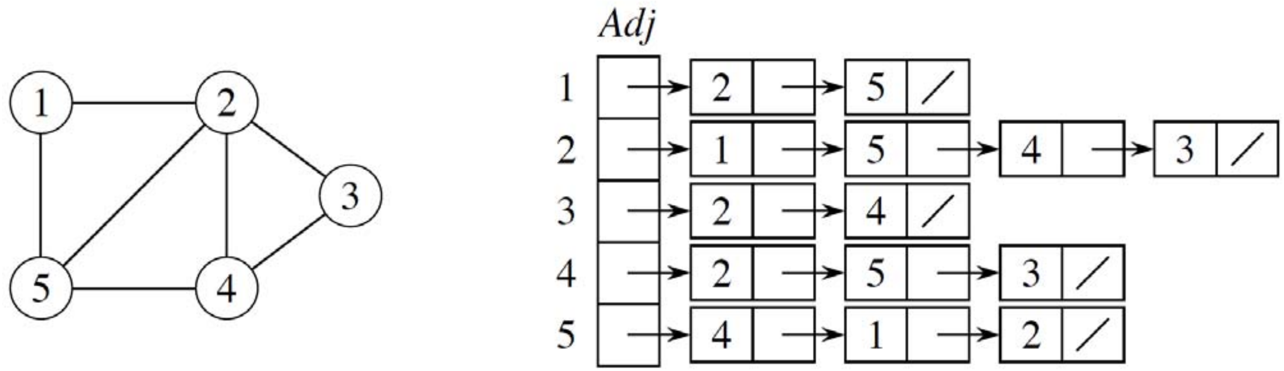
- Given a graph $G = (V, E)$. 圖：有向圖、無向圖
  - May be either directed or undirected.
  - Two standard ways to represent a graph:
    - Adjacency lists, when the graph is **sparse**.
    - Adjacency matrix, when the graph is **dense**.

表示法：list → 邊較少
matrix → 邊較多

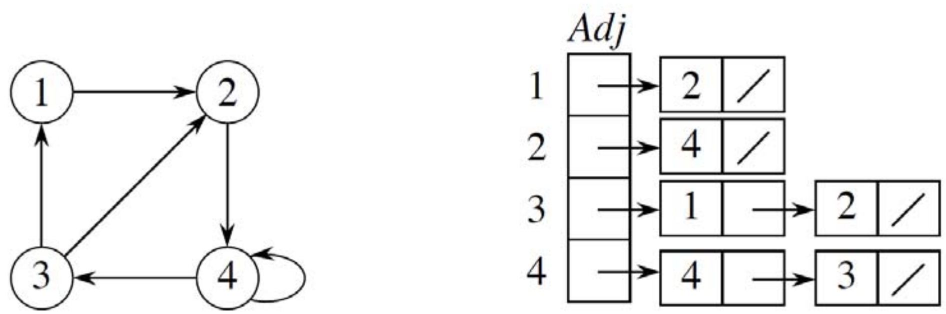
- When expressing the running time of an algorithm, it's often in terms of both $|V|$ and $|E|$, where $|V| = n$ and $|E| = m$.
  - Example: $O(n+m)$. 通常用 $|V|=n$ 和 $|E|=m$ 表示時間複雜度

Adjacency lists$_{1/2}$

- Example: For an undirected graph: 無向圖



- Example: For a directed graph: 有向圖

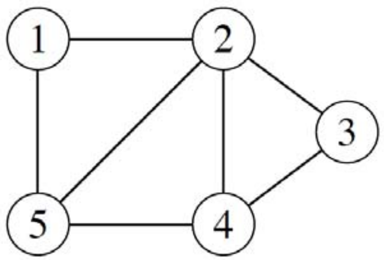


Adjacency lists$_{2/2}$

- Array *Adj* of ***n*** lists, one per vertex.
- Vertex *u*'s list has all vertices *v* such that $(u, v) \in E$.
- If edges have **weights**, can put the weights in the lists.
  - Weight: $w : E \rightarrow R$. 可以将 weight 放在 list 中
- Space: $\Theta(n + m)$.
- Time:
  - list all vertices adjacent to *u*: $\Theta(\deg(u))$. 列出所有的邻居 $\theta(\deg(u))$
  - determine if $(u, v) \in E$: $\Theta(\deg(u))$. 知道是否相邻 $\theta(\deg(u))$

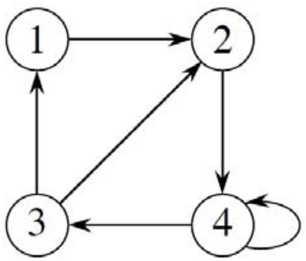
Adjacency matrix$_{1/2}$

- Example: For an undirected graph: 無向圖



	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

- Example: For a directed graph: 有向圖



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

Adjacency matrix$_{2/2}$

- *A* is an ***n*** x ***n*** matrix such that

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

可以將"1"改成 weight

- Can store weights instead of bits for weighted graph.
- Space: $\Theta(n^2)$.
- Time:
  - list all vertices adjacent to u: $\Theta(n)$. 列出所有的鄰居 $\theta(n)$
  - determine if $(u, v) \in E$: $\Theta(1)$. 知道是否相鄰 $\theta(1)$

⇒资料的儲存方式会影響查詢時間

Outline

- Representations of graphs
- **Breadth-first search**
- Depth-first search
- Topological sort
- Strongly connected components

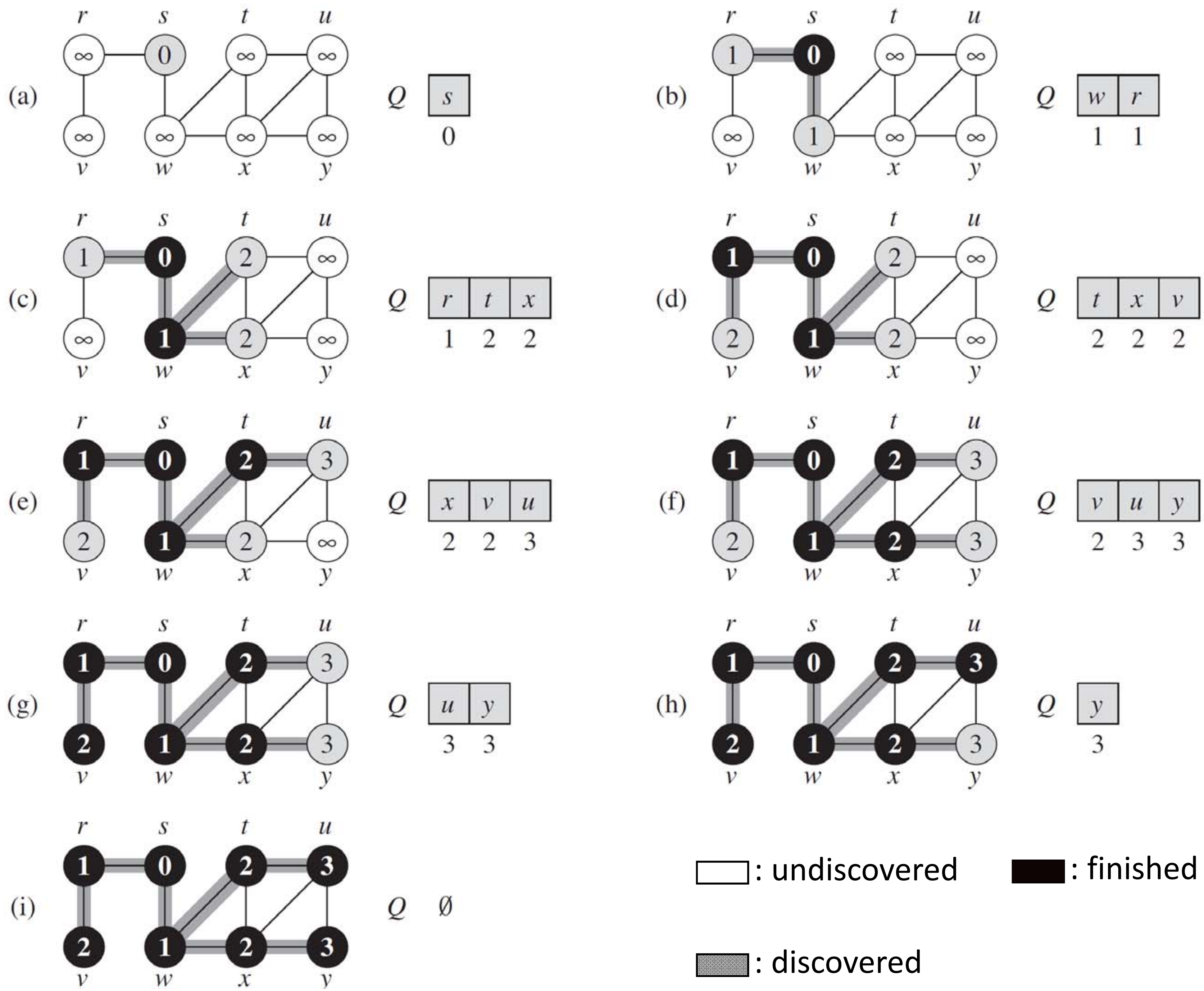
Breadth-first search

有一起始點

- **Input:** A graph $G = (V, E)$ and a distinguished ***source*** vertex s.
  - G can either be directed or undirected. 有向無向圖皆可
- **Output:** The distance (smallest number of edges) from s to each reachable vertex. 可算出與s的距離和"breadth-first tree"
  - As a by-product, it computes a "**breadth-first tree**" with root s that contains all reachable vertices.

先拜訪鄰居,再拜訪鄰居的鄰居⇒拜訪距離"1",再拜訪距離"2",依此類推

- **Idea:** Discover all vertices at distance k from s before discovering any vertices at distance $k + 1$.
  - First hits all vertices 1 edge from s.
  - From there, hits all vertices 2 edges from s.
  - And so on.


: undiscovered
: finished
: discovered

Pseudocode

```
BFS(G, s)
    for each vertex u ∈ V [G] – {s}
        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
        u ← DEQUEUE(Q)
        for each v ∈ Adj[u]
            if color[v] = WHITE
                color[v] ← GRAY
                d[v] ← d[u] + 1
                π[v] ← u
                ENQUEUE (Q, v)
        color[u] ← BLACK
```

初始化s以外的点

初始化s

① 從Q中取出一点u

② 拜訪u尚未被拜訪过的鄰居

(i) 顏色改成灰色

(ii) 距離+1，設定u是父親

(iii) 放入Q中

③ 將u改成黑色

Complexity

- The algorithm uses a first-in, first-out ***queue*** Q to manage the set of gray vertices. 使用Q=先進先出
- $\pi[v]$: the predecessor of v. {元素：元素的條件}
- Breadth-first tree : $G_\pi = (V_\pi, E_\pi)$
  - $V_\pi = \{v \in V: \pi[v] \neq \text{NIL}\} \cup \{s\}$ 點集合
  - $E_\pi = \{(\pi[v], v) : v \in V_\pi - \{s\}\}$ 邊集合
- The path in breadth-first tree from s to v is a shortest path (containing the fewest number of edges) from s to v.
  "breadth-first tree" 上的路徑是其他點到起始點 s 的最短路徑
- Time: $O(n+m)$.
  - $O(n)$: every vertex enqueued at most once. 每一個點最多被放入 queue 中一次
  - $O(m)$: using adjacency list, each edge is scanned at most twice. 使用 adjacency list, 每一個邊最多被看 2 次

Outline

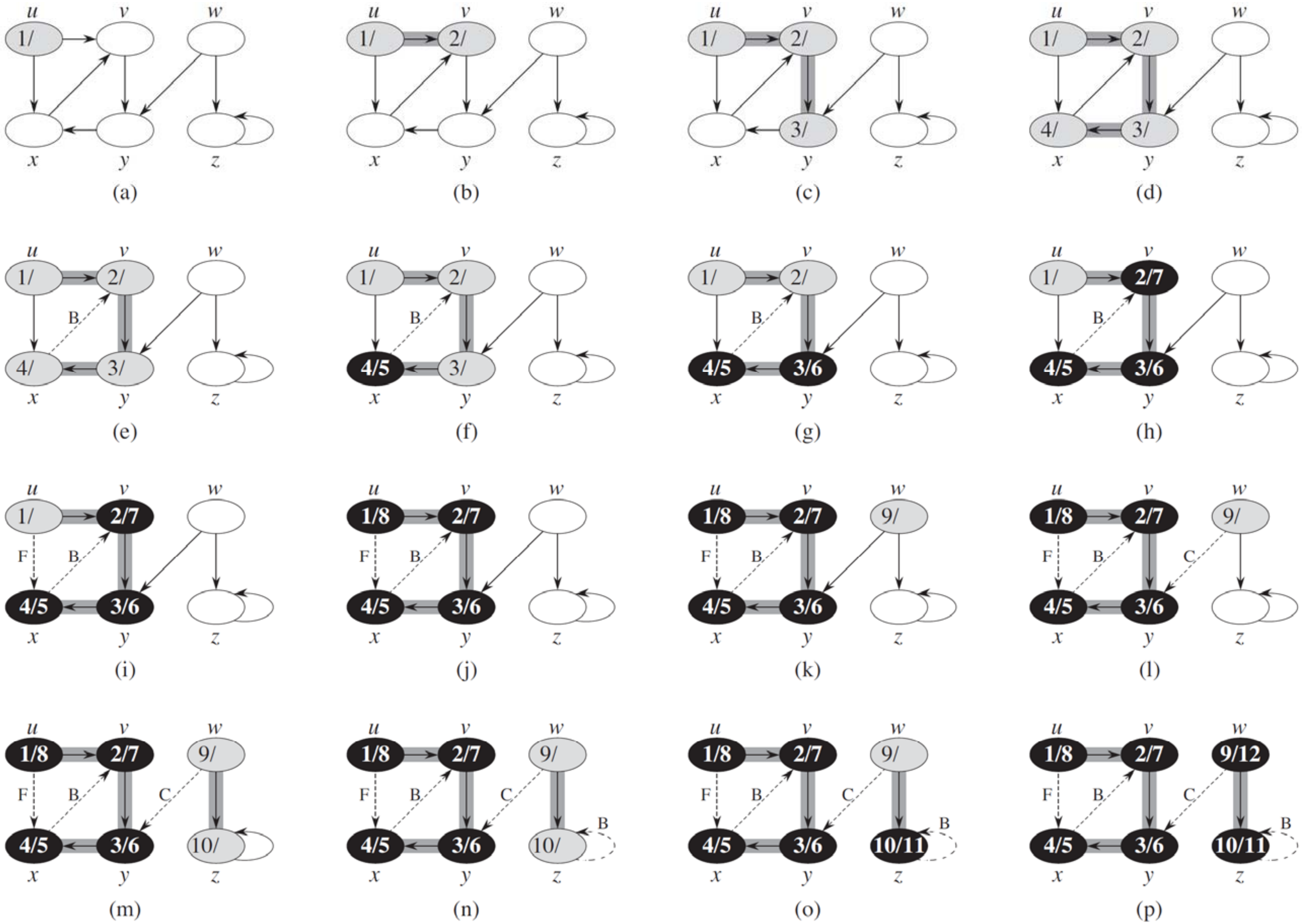
- Representations of graphs
- Breadth-first search
- **Depth-first search**
- Topological sort
- Strongly connected components

先拜訪鄰居的鄰居，再拜訪下一個鄰居

Depth-first search

沒有起始點

- **Input:** A graph $G = (V, E)$. No source vertex is given!
  - G can either be directed or undirected.
- **Output:** Two timestamps: $d[v]$ = discovery time and 第一次拜訪時間
  $f[v]$ = finishing time.
  完成時間，完成所有鄰居的拜訪
  - It also computes a **depth-first forest** $G_\pi = (V, E_\pi)$, where $E_\pi = \{(\pi[v], v) : v \in V \text{ and } \pi[v] \neq \text{NIL}\}$. 樹林
- Will methodically explore **every** edge. 会走过所有的 edge
  - 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 so that we explore from it later. 先拜訪鄰居的鄰居，再拜訪下一個鄰居



: undiscovered : discovered : finished

DEPTH-FIRST SEARCH pseudocode$_{1/2}$

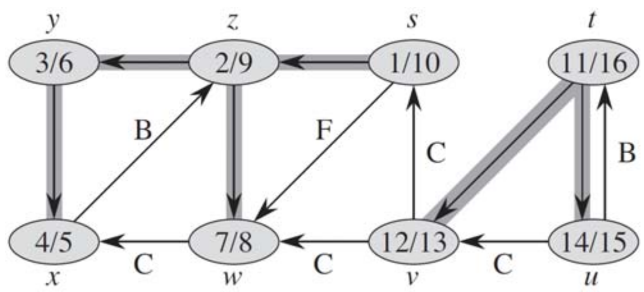
```
DFS(G)
    for each vertex u ∈ V [G]
        color[u] ← WHITE            初始化，設定颜色、父親、時間
        π[v] ← NIL
    time ← 0
    for each vertex u ∈ V [G]
        if color[u] = WHITE         对每一個点檢查，如果沒有拜訪过就 run DFS
            DFS-VISIT(u)

DFS-VISIT(u)
    color[u] ← GRAY       % White vertex u has just been discovered.
    time ← time + 1       u被发現，設定颜色、時間
    d[u] ← time
    for each v ∈ Adj[u]   % Explore edge(u, v).
        if color[v] = WHITE
            π[v] ← u      拜訪u所沒有被拜訪过的鄰居
            DFS-VISIT(v)
    color[u] ← BLACK      % Blacken u; it is finished.
    f [u] ← time ← time +1
```

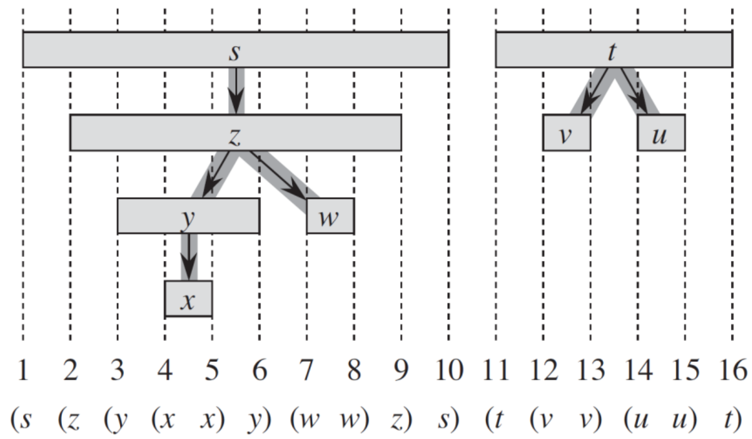
DEPTH-FIRST SEARCH pseudocode$_{2/2}$

- $\pi[v]$: the predecessor of v.
- Discovery and finish times:
  - Unique integers from 1 to $2n$.
  - For all v, $d[v] < f[v]$.
  - In other words, $1 \le d[v] < f[v] \le 2n$.
- Time: $\Theta(n+m)$.
  - $\Theta(n)$: The procedure DFS-VISIT is called exactly once for each vertex $v \in V[G]$. 每一個点剛好被DFS-VISIT呼叫一次
  - $\Theta(m)$: Using adjacency list, each edge is scanned at most twice. 使用adjacency list,每一個边最多被看2次

Properties of depth-first search$_{1/3}$

- Another important property of depth-first search is that discovery and finishing times have **parenthesis structure.** 括号結構
  - When vertex *u* is discovered ➔ represent *u* with "(*u*". 祖先
  - When vertex *u* is finished ➔ represent *u* with "*u*)". 子孫





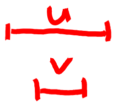
Properties of depth-first search$_{2/3}$

- **Theorem 22.7** (Parenthesis theorem)
  For any two vertices *u* and *v*, exactly one of the following three conditions holds:
  - the intervals [*d*[*u*], *f*[*u*]] and [*d*[*v*], *f*[*v*]] are entirely disjoint, and neither *u* nor *v* is a descendant of the other in the depth-first forest,
  - the interval [*d*[*u*], *f*[*u*]] is contained entirely within the interval [*d*[*v*], *f*[*v*]], and *u* is a descendant of *v* in a depth-first tree, or
  - the interval [*d*[*v*], *f*[*v*]] is contained entirely within the interval [*d*[*u*], *f*[*u*]], and *v* is a descendant of *u* in a depth-first tree.

① u, v 沒有血緣： u v ③ v 是 u 的子孫： u v
② u 是 v 的子孫： v u

Properties of depth-first search$_{3/3}$

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

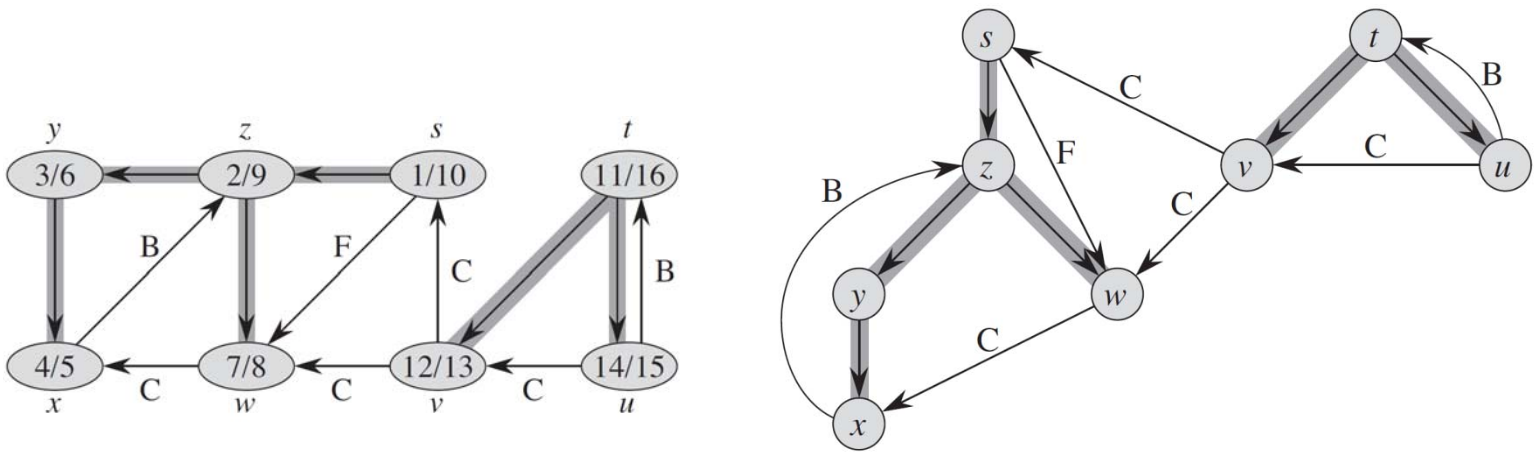
  v是u的子孫⇔ 

- **Theorem 22.9** (White-path theorem)
  Vertex *v* is a descendant of vertex *u* if and only if at the time $d[u]$, there is a *u*-*v* path consisting of only white vertices.

  v是u的子孫⇔在u第一次被拜訪時，有一條u-v path，
  path上的點都是白的

Classification of edges

- Four edge types:
  - **Tree edge:** in the depth-first forest G_π.
  - **Back edge:** non-tree edge (*u, v*) such that *u* is a descendant of *v*. (including self-loop)
  - **Forward edge:** non-tree edge (*u, v*) such that *u* is an ancestor of *v*.
  - **Cross edge:** non-tree edge (*u, v*) such that *u* is neither a descendant nor an ancestor of *v*.



藉由修改DFS來分類edge

Modify DFS algorithm to classify edges

- **Idea :** Each edge (u, v) can be classified by the color of the vertex v that is reached when the edge is first explored.
  - **White:** tree edge.
  - **Gray:** back edge.
  - **Black:** forward edge if $d[u] < d[v]$ and cross edge if $d[u] > d[v]$.

    forward：u 比 v 早被拜訪，cross：u 比 v 晚被拜訪
- If G is an undirected graph, an edge is classified as the **first** type that applies. 如果G是無向圖，edge的屬性由第一次分類決定

  (u,v)：u 拜訪 v 一次，v 也拜訪 u 一次

Theorem 22.10 無向圖只有 tree edge 和 back edge

- **Theorem 22.10 :** In a depth-first search of an undirected graph *G*, every edge of *G* is either a tree edge or a back edge.
- **Proof:**
  - Suppose (*u*, *v*) is an edge in *G* with *d*[*u*] < *d*[*v*]. u較v早被拜訪
  - Since *v* is on *u*'s adjacency list, *v* must be discovered and finished before we finish *u*. 在u完成拜訪前，v一定会被拜訪以及完成拜訪
  - If (*u*, *v*) is explored first from *u* to *v*, then (*u*, *v*) is a tree edge.
  - Otherwise, (*u*, *v*) is a back edge, since *u* is still gray at the time the edge is first explored.

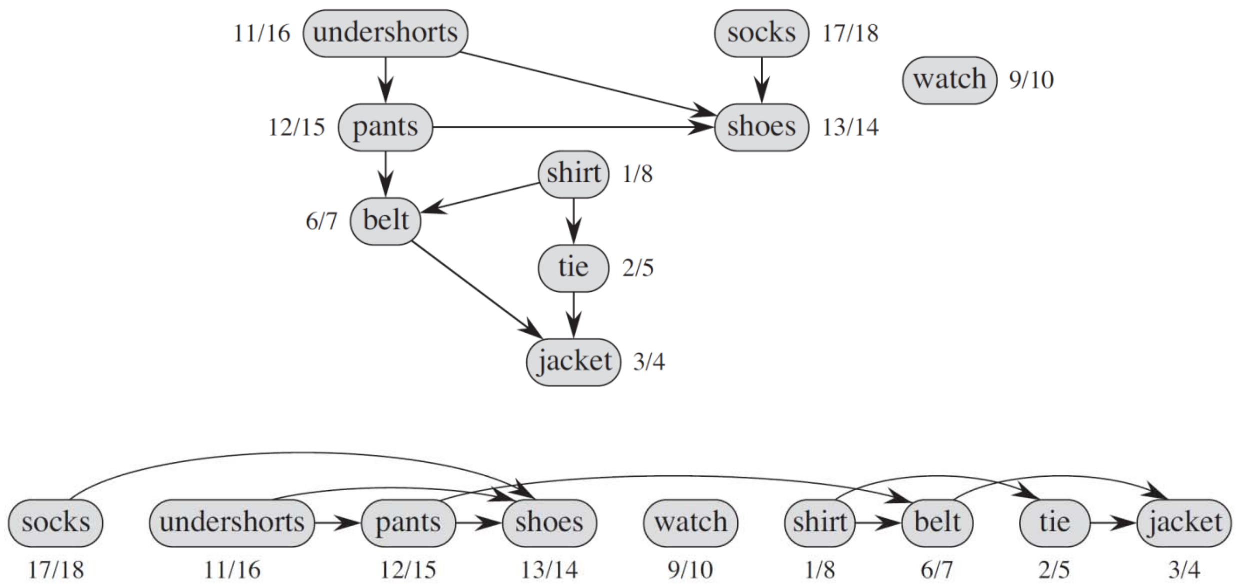
① u 拜訪 v ⇒ tree edge
② v 拜訪 u ⇒ back edge

Outline

- Representations of graphs
- Breadth-first search
- Depth-first search
- **Topological sort**
- Strongly connected components

Topological sort 拓樸排序：应用在沒有 cycle 的有向圖

- Use depth-first search to perform a topological sort of a directed acyclic graph (dag).
- A **topological sort** of a dag *G* 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. $u \to v \Rightarrow$ u在v之前



Pseudocode

TOPOLOGICAL-SORT(G)

1. call DFS(G) to compute finishing times $f[v]$ for each vertex v
2. as each vertex is finished, insert it onto the front of a linked list
3. **return** the linked list of vertices 使用DFS，將完成拜訪的点放入list中

- Time: $\Theta(n+m)$.
    - Depth-first search takes $\Theta(n+m)$ time.
    - It takes $O(1)$ time to insert each of the n vertices.
- Correctness: Refer to textbook.

Outline

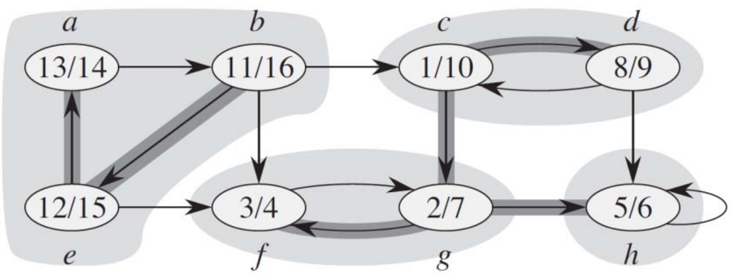
- Representations of graphs
- Breadth-first search
- Depth-first search
- Topological sort
- **Strongly connected components**

強連通：u可以到v，v可以到u 的最大子圖

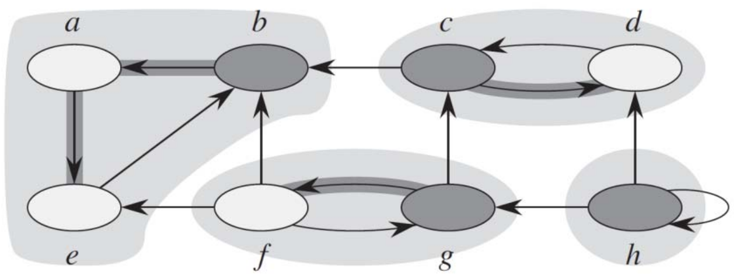
Strongly connected components

- A **strongly connected component** of a directed graph $G = (V, E)$ is a maximal set of vertices $C \in V$ such that for every pair of vertices u and v in C, we have both u-v path and v-u path.
- The **transpose** of a directed graph $G = (V, E)$ is the graph $G^{\mathrm{T}} = (V, E^{\mathrm{T}})$, where $E^{\mathrm{T}} = \{(u, v) : (v, u) \in E\}$. G^{T}：將G中的edge反向
  - E^{T} consists of the edges of G with their directions reversed.
- Observe that G and G^{T} have exactly the same strongly connected components. G和G^{T}中的強連通單元相同

G



G^T



Pseudocode

STRONGLY-CONNECTED-COMPONENTS(*G*)

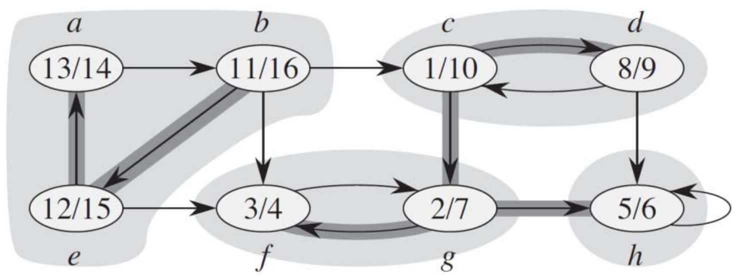
1. call DFS(*G*) to compute finishing times *f*[*u*] for each vertex *u*
2. compute G^T
3. call DFS(G^T), but in the main loop of DFS, consider the vertices in order of decreasing *f*[*u*] (as computed in line 1)
4. output the vertices of each tree in the depth-first forest formed in line 3 as a separate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

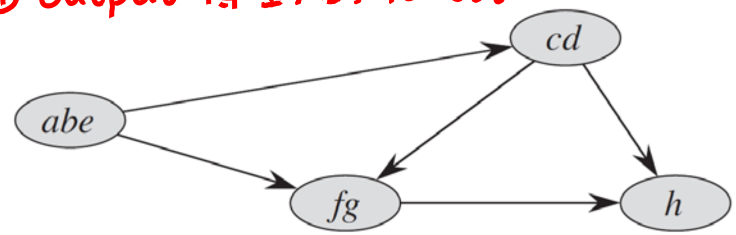
① 用DFS算完成時間 ② 算G^T

③ 後完成的點在G^T上先跑DFS

④ output 得到的 forest

G





Complexity

- Time: $\Theta(n+m)$.
  - Two depth-first searches take $\Theta(n+m)$ time.
- Correctness: Refer to textbook.
- For an undirected graph G, performing DFS once can obtain all "connected components".
  - See data structures chapter 6 for more information.