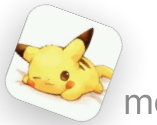
Computer Programming 2 Lab

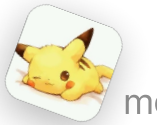
2023-06-07

Mozix Chien



Outline

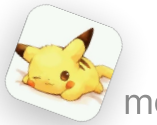
- Time Complexity
- Sorting Algorithm
- Dynamic Programming
- Tree
- Graph
- Others



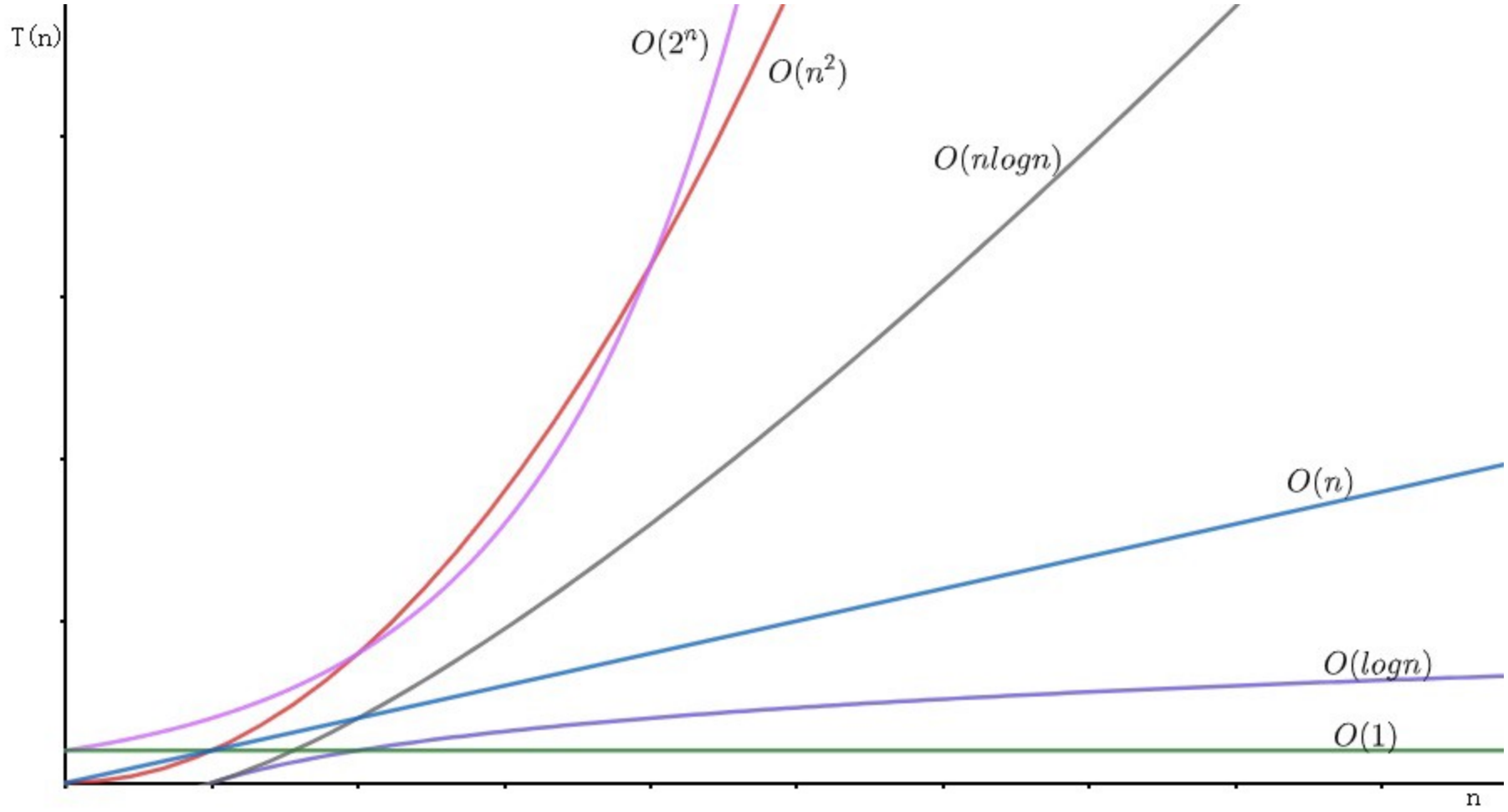
Time Complexity

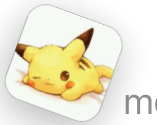
Time Complexity

Time complexity measures algorithm efficiency. It's expressed in big O notation, representing how running time grows with input size.



Time Complexity



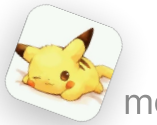


Sorting Algorithm

Sorting Algorithm

Selection Sort

1. 從「未排序好的數字」中找到最小(大)值
2. 把最小(大)值丟到「未排序好的數字」的最左邊，把它標示成已排序好

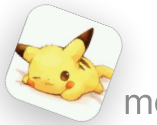


Sorting Algorithm

Selection Sort

```
[45, 3, 27, 80, 41, 51, 7]
```

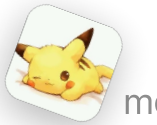
```
[|45, 3, 27, 80, 41, 51, 7]
[3 | 45, 27, 80, 41, 51, 7]
[3, 7 | 45, 27, 80, 41, 51]
[3, 7, 27 | 45, 80, 41, 51]
[3, 7, 27, 41 | 45, 80, 51]
[3, 7, 27, 41, 45 | 80, 51]
[3, 7, 27, 41, 45, 51 | 80]
```



Sorting Algorithm

Insertion Sort

1. 從「未排序過的數字」中選擇一個數
2. 把這個的數往前插入在適當的位置

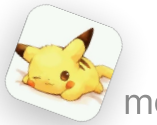


Sort 排序

Insertion Sort

```
[45, 3, 27, 80, 41, 51, 7]
```

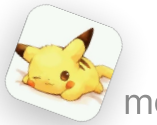
```
[45 | 3, 27, 80, 41, 51, 7]
[3, 45 | 27, 80, 41, 51, 7]
[3, 27, 45 | 80, 41, 51, 7]
[3, 27, 45, 80 | 41, 51, 7]
[3, 27, 41, 45, 80 | 51, 7]
[3, 27, 41, 45, 51, 80 | 7]
[3, 7, 27, 41, 45, 51, 80 ]
```



Sorting Algorithm

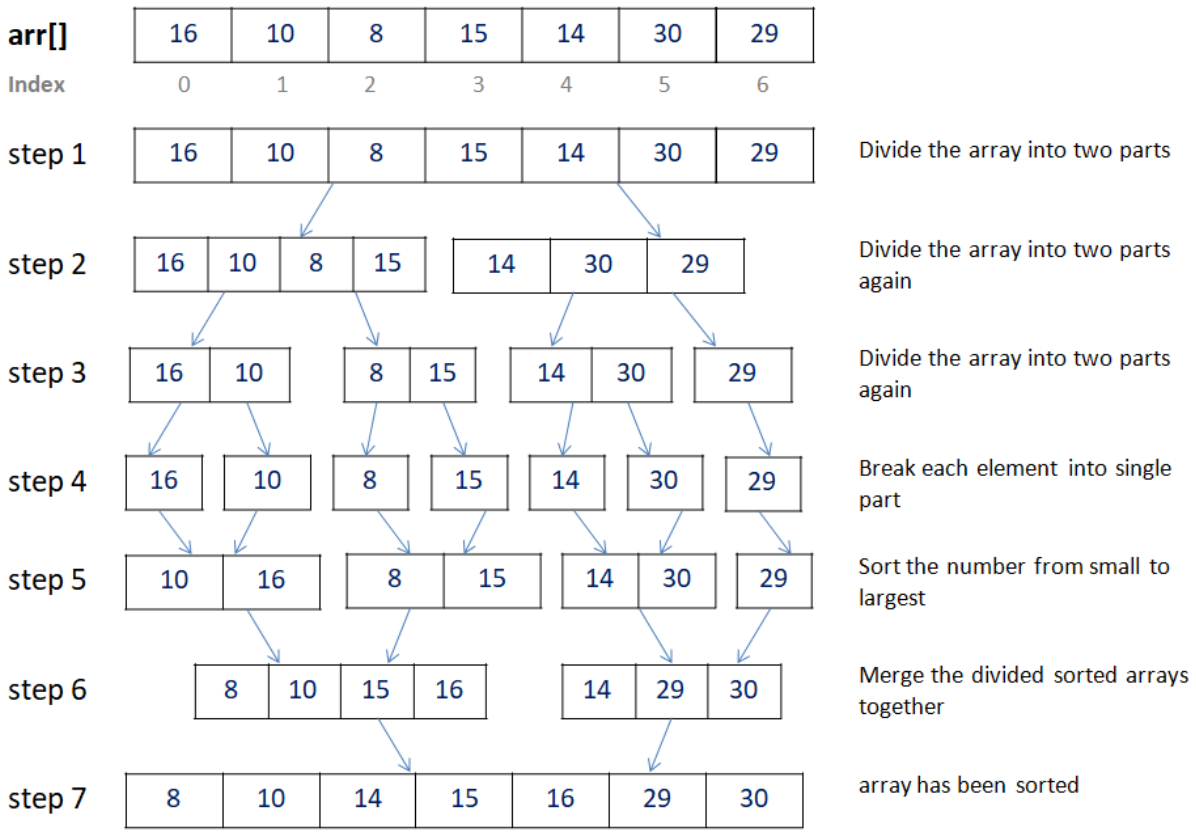
Bubble Sort

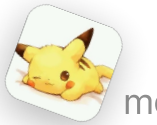
1. 兩兩比較，將較小(大)者往後移動，隨後繼續比較後兩者，持續比較至陣列尾端
2. 總共做 N 組



Sorting Algorithm

Merge Sort





Sorting Algorithm

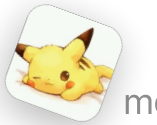
Stability

- 相同的資料，排序前後的順序不變

例：

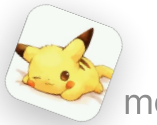
```
1, 5, 8, 3, 5'
```

```
1, 3, 5, 5', 8
```

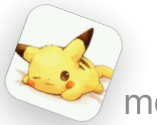


Sorting Algorithm

Algorithm	Best	Worst	Avarage	Space	Stability
Selection Sort	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(1)$	Unstable
Insertion Sort	$O(n)$	$O(n^2)$	$O(n^2)$	$O(1)$	Stable
Bubble Sort	$O(n)$	$O(n^2)$	$O(n^2)$	$O(1)$	Stable
Merge Sort	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$	$O(n)$	stable
Quick Sort	$O(n \log n)$	$O(n^2)$	$O(n \log n)$	$O(n)$	Unstable
Heap Sort	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$	$O(n)$	Unstable

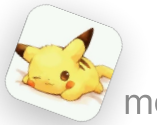


Dynamic Programming



Dynamic Programming

- 透過將問題轉化為數個子問題，遞推來求解
- 可以先試著統整歸納「遞推式」，在將其轉為程式碼



Dynamic Programming

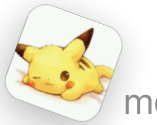
Coin Change

給定數個金幣價值 `coins`，以及目標價值 `amount`，問 `amount` 可由多少最少的金幣組成，如果無法組成輸出 -1

遞推式

```
V[i] = min(V[i-C] + 1, V[i])
```

- V 為目標價值，C 為硬幣價值
- 亦即 **考慮目標價值為 V 時，從價值 V - C 的數量再加 1 會不會比原本來的還要好**



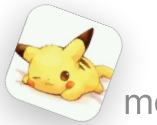
Dynamic Programming

Kanpsack Problem

有 n 個重量與價值分別是 w_i 和 v_i 的物品。想裝入一個最大限重為 W 的背包，想求背包可裝入的最大價值。

Brute Force

- 枚舉所有物品放入背包的狀況，找其最佳解
- 時間複雜度 $O(2^n)$

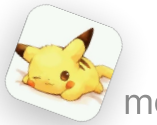


Dynamic Programming

Kanpsack Problem

```
dp[i][j] = dp[i-1][j] // 背包重量裝不下這個物品
         = max(
            dp[i-1][j-w[i]] + v[i], // 裝看看這個物品
            dp[i-1][j]              // 不裝這個物品
         )
```

- 時間複雜度 $O(n * W)$

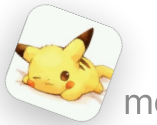


Tree

Tree

簡介

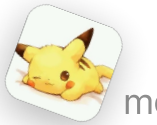
- 樹是一種 **階層式(Hierarchical)** 的 **資料結構**
- 樹的組成包含 **枝(邊)** 與 **節點**
- 樹中不可包含 **環(Ring)**
- 每個 **子節點只能有一個父節點**
- 依照有無根節點可分為 **有根樹** 與 **無根樹**
- 依照子節點個數可分為 **二元樹** 或 **K元樹**



Tree

名詞介紹

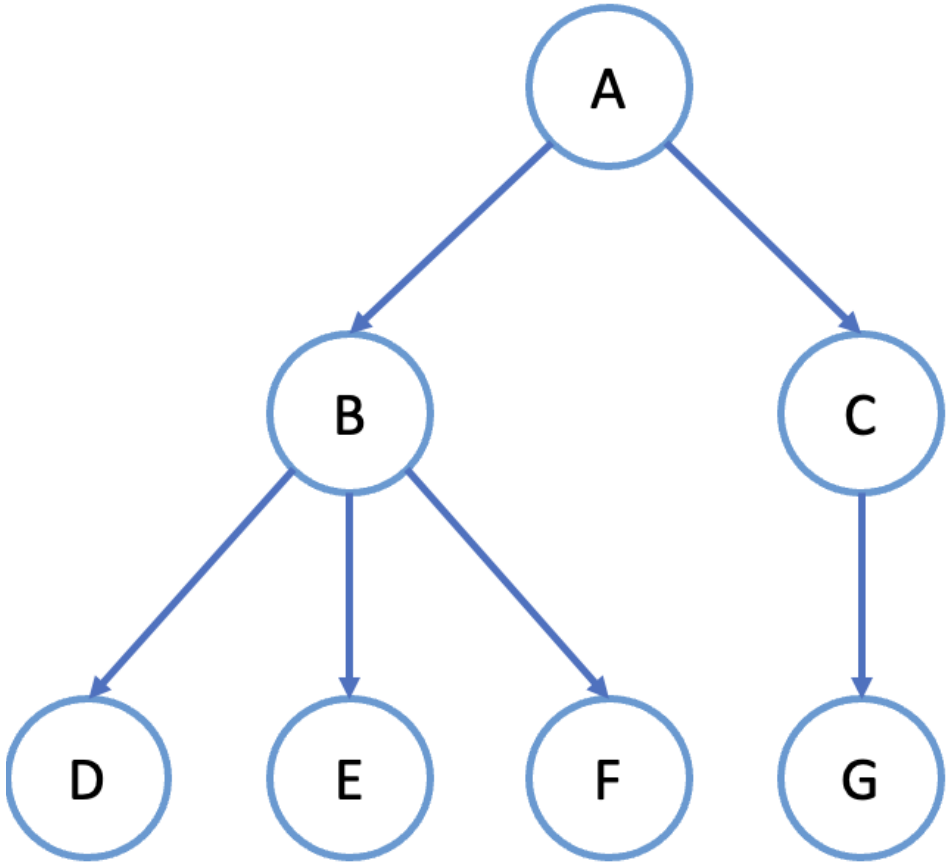
- 節點 (node) : 樹上的每個分支點
- 枝(邊) (branch) : 連接節點與節點間的邊
- 根節點 (root) : 一棵樹可想做是由一個點(根)開始分支
- 葉節點 (leaf) : 無法繼續分支的節點、沒有子節點的節點
- 父節點 (parent) : 相臨的兩個點，較靠近根的為父節點
- 子節點 (child) : 相臨的兩個點，較遠離根的為子節點
- 權重 (weight) : 若邊有權重，則樹權重為所有邊權重的總和
- 樹高 (high) : 樹最深的深度為樹高

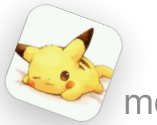


Tree

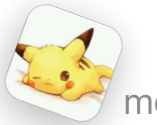
特性

1. 樹沒有環。
2. 樹上所有點之間都相連通。
3. 任意兩點之間只有唯一一條路徑。
4. 在樹上任意添加一條邊，就會產生環。
5. 在樹上任意刪除一條邊，一顆樹就裂成兩棵樹。
6. 邊數等於點數減一。





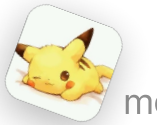
Graph



Graph

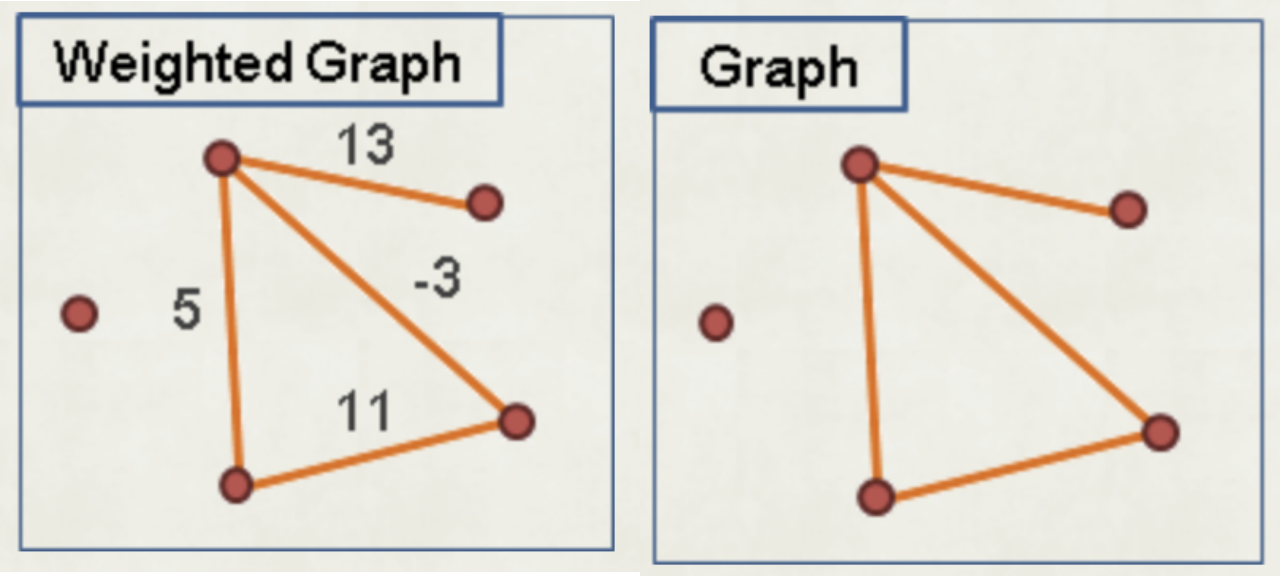
簡介

- 圖是一種 **描述相鄰關係** 的 **資料結構**
- 圖的組成包含 **點(Vertex)** 與 **邊(edge)**
- 圖中的點由邊相連接，**兩點之間可能不只有一條邊**
- 依照邊有無方向可分為 **有向圖** 與 **無向圖**
- 依照邊有無權重可分為 **加權圖** 與 **無權圖**
- 一個完全連通且沒有環的圖稱為 **樹**



Graph

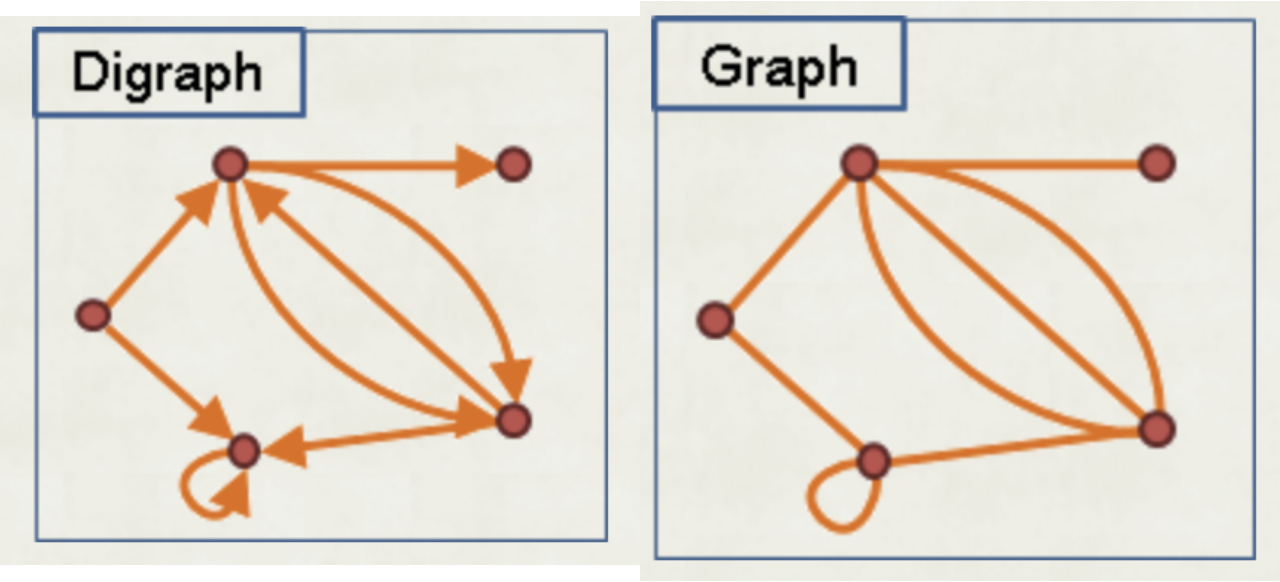
加權圖與無權圖



Graph

有向圖與無向圖

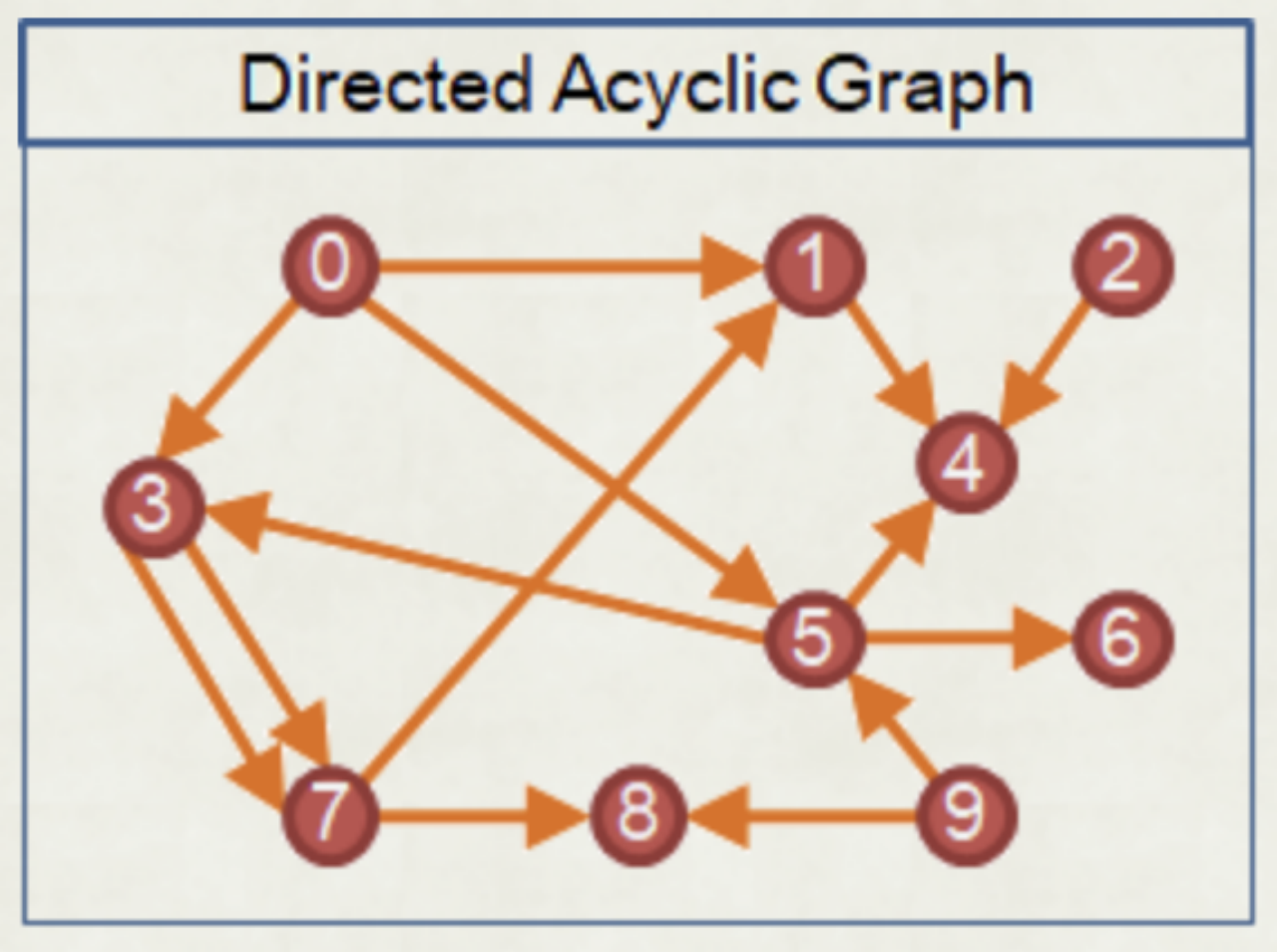
- 當有向圖沒有環時，稱為 **有向無環圖(Directed Acyclic Graph, DAG)**



Graph

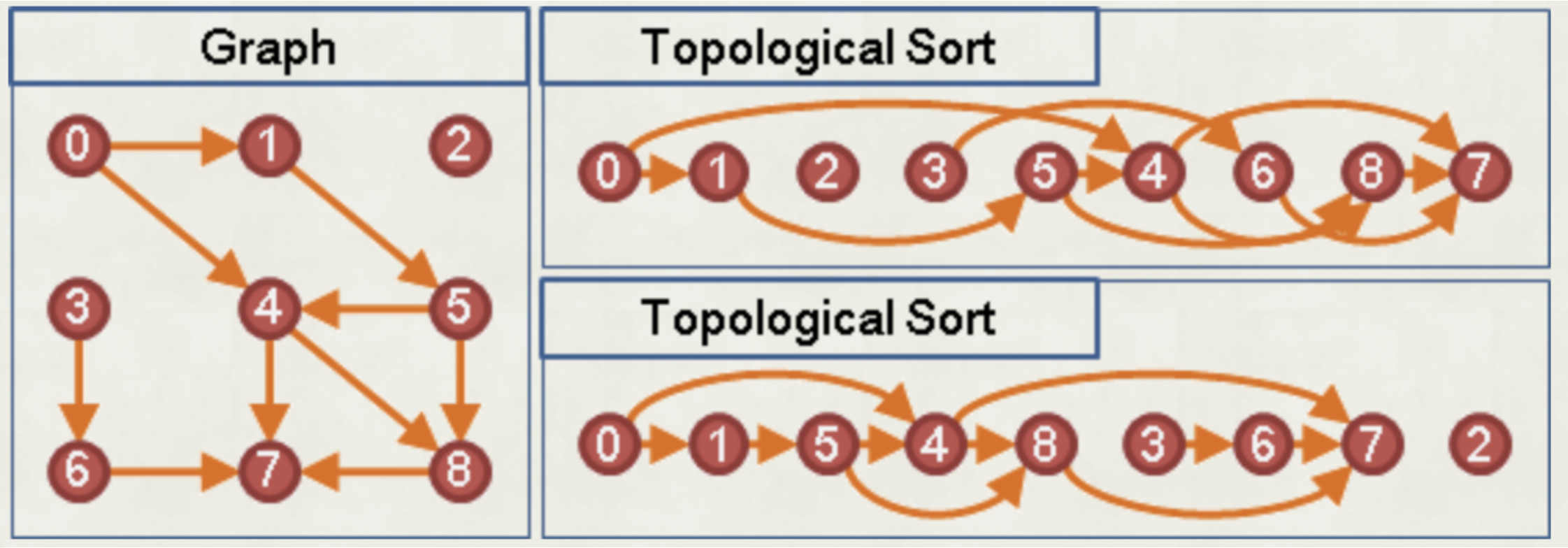
有向無環圖(Directed Acyclic Graph, DAG)

- 我們可以對有向無環圖進行拓樸排序(Topological Sort)



Graph

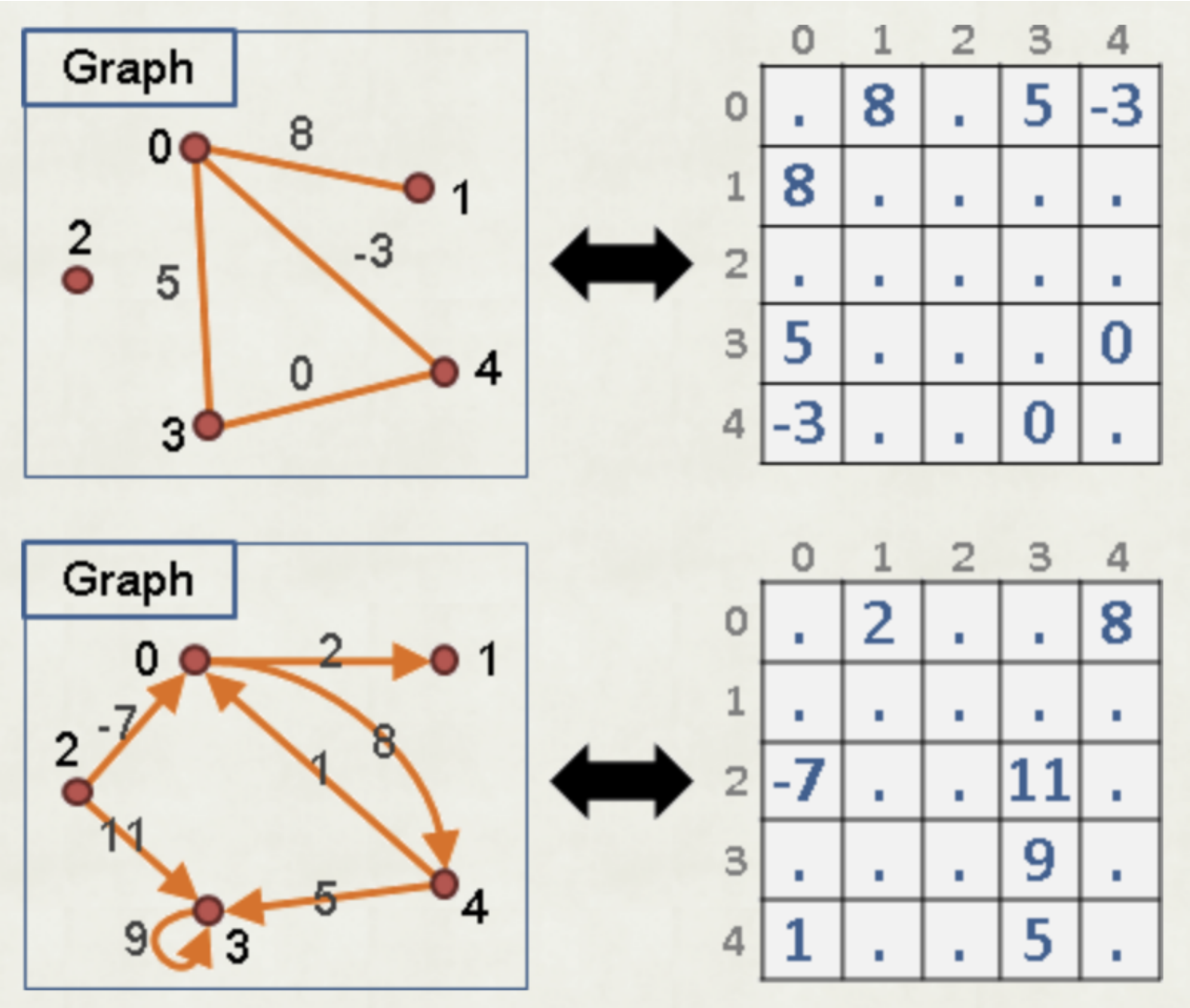
有向無環圖(Directed Acyclic Graph, DAG)



Graph

Adjacency Matrix

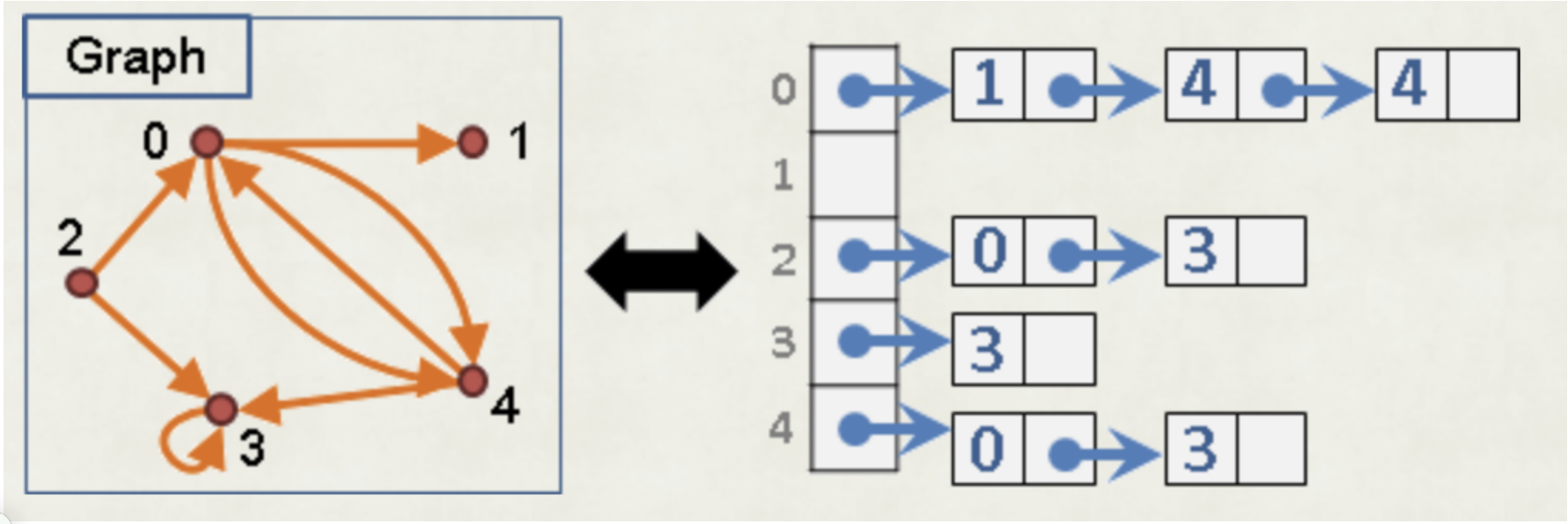
- 利用相鄰矩陣紀錄圖的連接關係
- 可以包含 **有向圖** 與 **無向圖**
- 但無法處理 **多重邊**

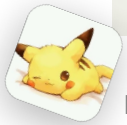


Graph

Adjacency List

- 利用相鄰串列紀錄圖的連接關係
- 可以處理 **多重邊**





Any Questions?