

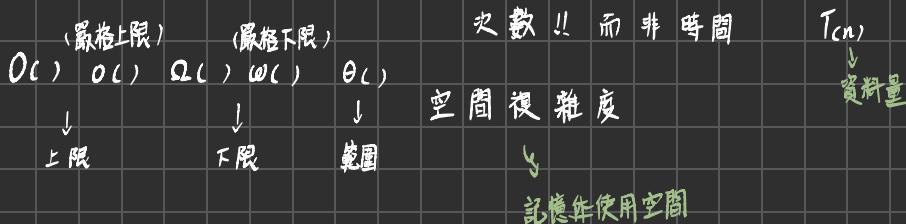
各式

資料 \rightarrow 資料結構 + 演算法 \rightarrow 資訊

演算法的效率

演算法：
解法問題的步驟

時間複雜度 Time Complexity



$T(n) = \text{常數} \rightarrow \text{不受資料量影響}$

$O(1)$ ex. 兩資料交換

$$T(n) = \text{對數 } \log_2 n \quad * \log n = \log_2 n \quad \text{高斯} \\ O(\log n) \text{ ex. 二分法搜尋 次數} \leq \lceil \log_2 n \rceil + 1$$

多對數 $\log_2 n \log_2 n$

指數 $=^n$ 最差要放幾次 $O(2^n)$

ex. 暴力枚舉和子集合 (陣列是否存在子集合合等於零)

多項式 一次 $O(n)$ ex. 單次完整掃描 (找出最大值)

二次 $O(n^2)$ ex. 雙層迴圈比較 (陣列是否有重複值)

$$\sum_{i=0}^{n-1} (n-1-i) = n(n-1)/2 = \Theta(n^2)$$

階乘 $n!$ $O(n!)$ ex. 位置排列 (有條件座位排列)

多對數
 $O(\log^2 n)$

複雜度: $O(n!) > O(2^n) > O(n^2) > O(n \log n) > O(n) > O(\log n) > O(1)$

階乘 指數 平方 線性對數 線性 對數 常數

漸進表示法

簡化時間函數

選取函數中最具代表性的函數，忽略小的

用於表示資料量 n 無限大的成長趨勢

(沒有上 or 下界 \Rightarrow 無法辨認)

* 加減用公式去做

O - "big-O" notation 上限

取大於最高項係數

$O(gcn)) = \{ fcn: \text{存在常數 } c \text{ 和 } n_0, fcn \underset{\sim}{\leq} c gcn, \text{ for } n \geq n_0 \}$

\hookrightarrow 嚴格上限沒有等號 \Rightarrow 記為小寫

Ω - "Omega" notation 下限

取小於最高項係數

$\Omega(gcn)) = \{ fcn: \text{存在常數 } c \text{ 和 } n_0, fcn \underset{\sim}{\geq} c gcn, \text{ for } n \geq n_0 \}$

Θ - "Theta" notation 輪圍

小於最高項係數 \hookrightarrow 大於最高項係數

$\Theta(gcn)) = \{ fcn: \text{存在常數 } c_1, c_2, n_0, c_1 gcn \leq fcn \leq c_2 gcn, \text{ for } n \geq n_0 \}$

* n_0 只是用於確認不等式在這之後都符合

* c 是為了覆蓋 fcn)

靜態資料 vs. 動態資料

編譯時宣告 空間和大小可隨時改變
大小固定

ex. Array

ex. Linked List

Python !!

串列 List []

△ 可容納不同資料型態、不同長度

List [a:b] 讀取連續

△ 有順序性 (或線性)

$a \rightarrow b-1$

△ 可更動

[List[i] for i in (a,b,c)]

△ 相當於動態的陣列 Array

取非連續位

串列名稱 = [元素1, 元素2, ...]

拿出來於新址再讀取

* List [:] 讀取全部

終點 -1 !!

必須小 \rightarrow 大 !!

語法

若元素為 List，則視為一元素

- `append(新元素)` 加入一新元素
- `extend([元素])` 加入多筆新元素

拆開為多元素

- `pop(索引值)` 移除索引值位置元素
- `remove(元素)` 移除元素

預設為-1 (最後)

`List[index] = 新元素`

- `insert(index, 元素)` 在第 index 插入元素

`.count(元素)` 計算次數

`.index(元素)` 找出元素位置

`.reverse()` 反轉

`.sort()` 排序

二維串列 List 2D [[]]

跳過！

列索引 行索引

迴圈

跳過！

元組 Tuple ()

△ 不可修改

△ 與餘相似於串列

元組名稱 = (元素, 元素, ...)

* 只有單一元素宣告 `tuple = ()`

集合 Set {}

△ 元素不重複

△ 不具有順序性

△ 可修改

交集 : &

聯集 : |

差集 : -

- `pop(元素)` 移除元素
- `add(元素)` 新增元素

`.issubset(set)` 是否為子集合 (T/F)

`.issuperset(set)` 是否為父集合 (T/F)

`.intersection(set)` 交集

`.union(set)` 聯集

字典 Dictionary {}

字典名稱 = {鍵 (Key) : 值 (Value)}

dict['Key'] = New Value	修改	• pop (Key)	移除	• values	回傳所有 Value
• update ({Key: Value})	新增	• keys()	回傳所有 Key		

陣列 Array

- 一維 → 向量
- 二維 or 多維 → 矩陣

- △ “靜態”的資料結構
- △ 包含相同的資料型態
- △ 線性序列 (有序性)
- △ 在記憶體空間中具有連續性

記憶體位置？

△ 以列 (row) 為主 ex. C, C++, Python * Numpy

△ 以行 (column) 為主 .ctypes, data 第一筆資料位置 .shape 結構形狀

宣告後無法更動結構

⇒ 修改會重新指派位置

一維陣列 Index ↙ 每個元素佔用位元

$loc(A[i]) = I_0 + i \times d$

↳ 陣列起點

	1	2	...	j	...	n
1	0	0	...	0	...	0
2	0	0	...	0	...	0
⋮	⋮	⋮	⋮	⋮	⋮	⋮
m	0	0	...	0	...	0

二維陣列

row ↗ column

⇒ 每列 (row) 有 n 元素

逐列存放 : $loc(A[i][j]) = I_0 + i \times n \times d + j \times d$

逐行存放 : $loc(A[i][j]) = I_0 + j \times m \times d + i \times d$

⇒ 每行 (column) 有 n 元素

陣列加法 & 乘法

一維陣列

加法

△ 大小相同

$$[1, 2, 3] + [4, 5, 6] = [5, 7, 9]$$

* Python $A+B \Rightarrow$ 合併 AB



運用 Numpy 可直接實現

乘法

$$3 * [1, 2, 3] = [3, 6, 9]$$



* Python $c * A \Rightarrow$ 重複 c 次 A

二維陣列 \Rightarrow 矩陣

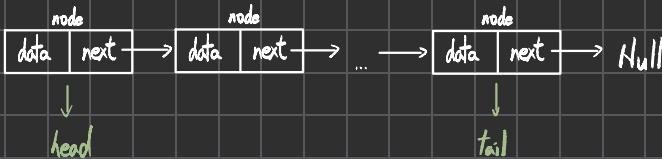
直接運算問題同一維

numpy \Rightarrow @ 乘法 .T 轉置

Linked List

△ 動態資料

△ 記憶體位置不連續



Singly Linked List (單向)



```
class Node :
```

```
    def __init__(self, data = None, next = None):  
        self.data = data  
        self.next = next
```

```
class SingleLinkedList :
```

```
    def __init__(self, head = None):  
        self.head = head
```

: (新增、删除、修改...)

概念

(while current.next)

新增 (表尾) : current = head \rightarrow 找到 current = 尾 \rightarrow 新增

新增 (表头) : 新增 \rightarrow 修改 head

Doubly Linked List



```
class Node :
```

```
    def __init__(self, data = None, next = None, prev = None):  
        self.data = data  
        self.next = next  
        self.prev = prev
```

Circular Linked List

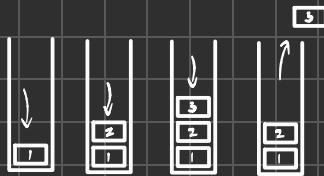


△ 只有一個節點 Next/Prev 指向自己

△ 找尾節點

```
while current.next != self.head:  
    current = current.next
```

Stack 索量



Last-In-First-Out
(LIFO)

重要 function .

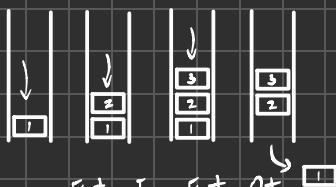
- I. Create
- II. Push → 新增
- III. Pop
- IV. IsEmpty → 空集合 or Not
- V. Full → 最大上限

變數

- I. Capacity → 容量
- II. items = []

* 課堂用內建 list 運算呈現

Queue 佇列



First-In-First-Out
(FIFO)

重要 function .

- I. Create
- II. Enqueue → 新增
- III. Dequeue
- IV. IsEmpty → 空集合 or Not
- V. Full → 最大上限

變數

- I. Capacity → 容量
- II. items = []

運算式表示法

運算元

$(A + B) \times C + D / E$

運算子

⇒ 中序法 (infix)

(運算子在運算元中間)

中序法轉後序法：

step I. 幾個運算子，就有

幾個括號

$((A + B) \times C) + (D / E))$

(前序取代左邊)

step II. 把運算子取代最近

右手邊括號

$((A + B) \times C) + (D / E))$

↑ ↑ ↓ ↑

$((A B + C \times D E) / +$

前序法 (prefix)

(運算子在運算元前面)

後序法 (postfix)

(運算子在運算元後面)

ex. 1 3 + 4 * 10 2 / +

(前序從後往前面疊) * 上對下

⇒ 3 ⇒ 3 ⇒ 1+3

1 1 1 4

* 過運算子取最上面兩個
做運算 (下對上)！

step III. 刪除左手邊括號

$A B + C \times D E / +$

⇒ 4 4 * 4 10 16 16 10

⇒ 2 2 / 2 10 10 16 16 +

⇒ 10 10 10 10 5 5

⇒ 16 16 16 16 16 16

⇒

21

樹 TREE

階層式數據結構

不可形成迴圈

一個以上節點就是樹

分支度 (Degree)

節點子樹個數

階層 / 階度 (Level)

高度 (Height)

邊 (Edge)

父與子連接線

根節點 (Root)

父節點 (Parent)

子節點 (Child)

葉節點 (Leaf)

沒有子節點的節點

兄弟節點 (Siblings)

同父節點

非終端節點 (Nonterminal Nodes)

同代 (Generation)

同階層

根節點

第 0 階層

節點

葉節點

第 1 階層

葉節點

節點

葉節點

葉節點

：

二元樹 Binary Tree

最多只有 2 個子節點：左子節點 / 右子節點
(Left Child) (Right Child)

vs. 一般的 Tree

Tree 不可為 \emptyset ，但 Binary Tree 可以
樹的分支度 $d \geq 0$ ，但 Binary Tree $0 \leq d \leq 2$
樹的子樹沒有次序，但 Binary Tree 有

$$\max(\text{深度 } k \text{ 的 Binary Tree 的 node 數}) = 2^k - 1$$

△ 完滿二元樹 Fully Binary Tree

$$\text{高度} = h \text{ s.t. } \text{node} = 2^h - 1$$

△ 完整二元樹 Complete Binary Tree

- I. 除最後一層，其餘層的節點數達最大
- II. 由左而右，不可跳過



△ 歪斜樹 Skewed Binary Tree

完全沒有 Left / Right node
(左歪斜樹 / 右歪斜樹)

左歪斜樹



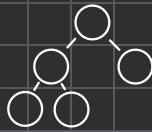
右歪斜樹



or

△ 嚴格二元樹 Strictly Binary Tree

非終端節點有非空左右子樹



二元樹之走訪 Binary Tree Traversal

△ 前序



根 → 左 → 右

△ 中序



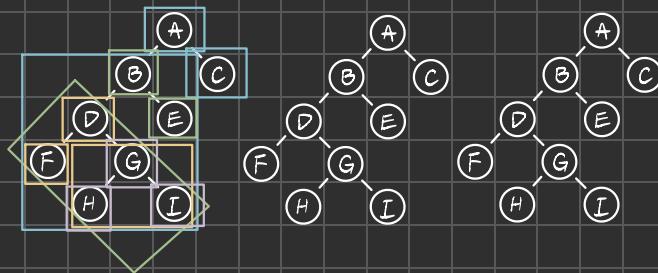
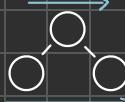
左 → 根 → 右

△ 後序



左 → 右 → 根

△ 層序



二元搜尋樹 Binary Search Tree (BST)

每個 Node 的數值必大於 Left Child, 且小於 Right Child

中序走訪 ⇒ 小 → 大 排列

I. 搜尋

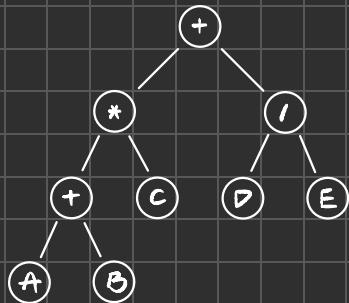
最多做 Tree's Height + 1 次
大往右，小往左

II. 插入

III. 刪除

運算式樹

Ex. $(A+B)*C + D / E$



內部節點 \Rightarrow 運算子

葉節點 \Rightarrow 運算元

平衡搜尋樹 (Balanced Search Tree)

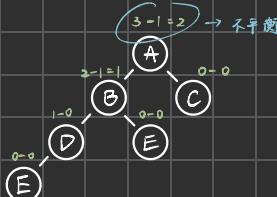
左 / 右子樹高度不相差 1

高度盡可能接近 $O(\log n)$

How to know?

(左 - 右子樹高度)

計算平衡因子 (係數)
 $\leq 1 \Rightarrow$ 平衡



② AVL 樹

△ 漢轉

I. LL 型 (LEFT-LEFT)



II. RR 型 (RIGHT-RIGHT)



III. LR 型

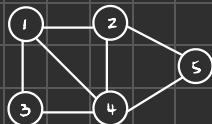


IV. RL 型



圖形結構

由頂點 (vertex) 和邊 (edge) 所組成的集合，以 $G(V, E)$



$$V = \{1, 2, 3, 4, 5\}$$

$$E = \{(1,2), (1,3), (1,4), (2,3), (2,4), (2,5), (3,4), (4,5)\}$$

邊的方向：有向 / 無向

邊的權重：無 / 有

尤拉環 (Eulerian Cycle)

頂點分枝為偶數，可從某頂點出發
經每邊一次再回起點

尤拉鏈 (Eulerian Chain)

允許其中兩個頂點的分歧有奇數，
從奇數頂點出發，但不回到起點

霍夫曼編碼 (Huffman Coding)

數據壓縮

頻率高 \Rightarrow 短編碼
頻率低 \Rightarrow 長編碼

v.s. 傳統

節省位元數
(傳統固定長度)

流程

I. 統計頻率
II. 為符號建立節點

權重 = 頻率

III. 排序所有節點

重複合併兩圈最小節點

Δ 父節點 Z : $Z.\text{weight} = x.\text{weight} + y.\text{weight}$

Δ 標記邊：左 0 右 1

