L5 stack 堆

Last In First out (LIFO "後進先出")

簡易使用檢查成對括弧、辨識語言、找路徑、代數運算式

打字的應用

```
while (!eOL) 讀                 while (ADT不是空的)
  if (不是倒退鍵)               { 取出最後一筆資料
    加入adT                       去ch
  else if (adT不是空的           然後remove
    Remove the last                最後一筆資料
  else //沒東西可刪             } //while
    忽略倒退鍵
```

找括號

寫match fuction

迴圈 { 左就push / 右pop 是空的用bool balance = 0 回報錯誤

三括號的話 → 除外的說法就是 左就count++ 右就count-- count會小於0或結尾>0就不對

最後stack會是empty且balance=1才正確

辨識語言字串，分隔

迴文 L={w$w'}, w'=反(w)

方法前半部push 後半部pop

```
while(!$) {                     while(!eos && iL)
  stack.push(ch)                  if(stack不是空的) {
  ch=next char                      stackpop(top);
} // while                          ch=next char
bool iL = true                      if(top != ch)
  (in language)                       iL=false
                                  } //
                                  else
                                    iL=false
                                if(iL && stack是空的)
                                  return true;
                                else
                                  return false;
```

實現ADT

用array實現堆疊ADT

linked list

ADT list

建構 && 解構

isempty()

pop()

push

getTop

pop(傳value)

push - List.insert(1, value)

gettop - List.retrieve(1, top)

pop [List.remove(1)

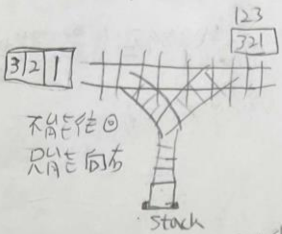
---

array, pointer 固定，非固定大小

pointer 有效率

ADTList 再利用 [illegible]

鐵軌排列



push,pop(1) 2右 3張
1往右 1下. (1,2,3) 3,2,1
2往下 (2,3,1) 132 (1,3,2) 2,1,3 (1,2,3) 3,2,1
鐵 (3,2,1) 123 (3,1,2) 2,3,1 (X) (3,1,2)

231 ✓
213 ✓
123 ✓
132 ✓
321 ✓ 鐵
312 X

((a+b)(c*d))
ab+cd*+

push = 2(n-1), pop = 4(n+1)
push轉pop
pop轉push

pemutation 用push, pop
(1,2,3)(),(),()

這種
有 C^6_3 種，但是要去除錯誤排列，(push,pop,(pop), (push,push,pop)
去除 4pop 和 2push = $C^6_4 = C\binom{2n}{n+1}$ (pop (push push pop push pop))

★ Catalan number: 1, 1, 2, 5, 14, 42 (0 1 2 3 4 5)

$CN_n = C\binom{2n}{n} - C\binom{2n}{n+1} = \frac{1}{n+1}C\binom{2n}{n} \Rightarrow C_n = \frac{(2n)!}{(n+1)!n!}$

$C\binom{6}{3} - C\binom{6}{4} = \frac{6!}{3!3!} - \frac{6!}{4!2!} = \frac{6!}{3!3!}(1-\frac{3}{4}) = \frac{1}{4}C\binom{6}{3}$

後序式求解、中序式轉成後序式

假設沒一元(-1)、指數運算子, only letter and number, 且文法正確的後序

先pop兩運算元，計算後再push結果

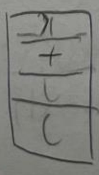
c-d
a+b
(a+b)*(c-d)

ab+cd-*

中序轉後序

運算元、運算子、括弧、優先權

把前改後倒過來讀



右括號就往下推運算元
有優先如果下個大於
等於就要往下找

序 序 方法

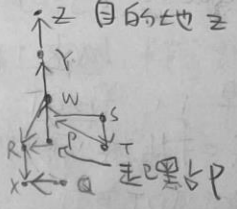
*+ab-cd

堆跟遞迴相近 → 程式結構完全不一樣

遞迴用隱含堆疊

city
origin destination

```
bool Map::isPath(int ori, int des){
  Stack aStack
  int top, next; // city
  bool success
  unvisitAll();  // clear marks
  aStack.push(ori); // P
  markVisited(ori); // 起點
  aStack.getTop(top); // 回起點 or 到終點
  While(!aStack.isEmpty() && top != des){
    success = getNextCity(top, next);
    if(!success) aStack.pop(); // backtrack
    else {aStack.push(next), markVisited(next)}
    if(!aStack.isEmpty())
      aStack.getTop(top);
  } // while
  if(aStack.isEmpty())
    return false; // 沒路
  else
    return true; // 有路
} // map
```



找最短路線(用堆疊)

遞迴 迷宮也是 像上面一樣 用回朔用掉死路

路線圖

L6 Queue 佇列 (First in, first-out)(FIFO)

最後資料 back, rear

〃早 〃 front

例子：排隊、飛機起飛、買票、模擬、print (simulation)

模擬

行為模型、統計、預測
(modeling the behavior)(statistics)(predict)

例子：時間驅動 (time driven)
- 事件驅動 (event driven)
  - Arrival events (輸入決定時間)
  - Depature events (模擬決定時間)

Teller / queue

加本構
- Single teller / single queue
- Multiple teller / single queue
- Multiple teller / Multiple queue

Arrival	duration	Departure	waiting time

基本例 ADT實作

不同於 stack

dequeue 先 vs pop 後

getFront 先 vs getTop 後

dequeue(getFront) vs pop(getTop)

---

stack vs queue in 比較

stack	queue
反向	正向
top	front
[1] ← top	[1 2 1] front back

array-based implementation (later)

pointer-base enqueue (linked list)

front 前端、back 後端

circular linked list

only back (後端) 環狀

環狀 array 問題會滿空相同 (相對位置)

解法1：計數器 count

解法2：旗標空白(最大)比似 dummy head

解法3：旗標 isfull();

---

pointer-base implement 有效率的 (執行效率)

ADT list implement class 好撰寫省 program time (撰寫效率)

位置導向 ADT: stack, queue (only end position), list (All)

L7 Algorithm Efficiency 演算法 (Big O)

分析演算法 (比較不同解法效率的工具)

- Time efficiency, space efficiency
  時間效應 空間效應
- 比較兩演算法
  significant differences 顯著差異
  不被 clever coding tricks 影響

三個差異來看程式好壞 (非演算法)

- specific implementation 實作
- computer 電腦
- data 資料, dataset

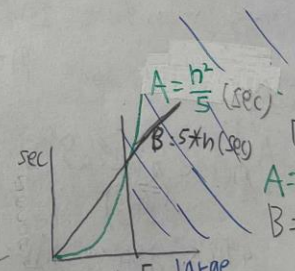
演算法A、演算法B
程式A、程式B
比較希望相同 (難較少)
dataset

example 1
Traverse Linked list of n node 走訪
n+1 comparisons, n+1 assignment, n write

example 2
Hanoi Tower 河內塔 運算次數
2^n-1 moves

$M(n) = M(n-1)+1+M(n-1)$ for $n>1$

Algorithm growth rate
problem size 成長速率



隨著n增加
A=指數成長
B=線性成長

通常care這裡

演算法 | A需 n^2 部分 time | $A \propto n^2$
B需 n | $B \propto n$
B faster than A
growth-rate function

- Algorithm A is $O(n^2)$ 或 order n^2
- Algorithm B is $O(n)$ 或 order n
- Big O notation
  通常會忽略低位階和常數
  $O(3n^2)$ $O(\frac{1}{2}n^2)$

位階比

$$O(1) < O(\log_2 n) < O(n) < O(n*\log_2 n) < O(n^2) < O(n^3) < O(2^n)$$

objective

find the worst case ✓
average case ✓
Best case ×

	stable?	Worst	best	Average	
bubble sort	✓	$O(n^2)$	$O(n)$	$O(n^2)$	
selection	×	$O(n^2)$	$O(n^2)$	$O(n^2)$	
insertion	✓	$O(n^2)$	$O(n)$	$O(n^2)$	
merge	✓	$O(n*\log_2 n)$	$O(n*\log_2 n)$	$O(n*\log_2 n)$	
quick	×	$O(n^2)$	$O(n*\log_2 n)$	$O(n*\log_2 n)$	
heap	×	$O(n*\log_2 n)$	$O(n\log_2 n)$	$O(n*\log_2 n)$	
radix	✓	$O(n)$	$O(n)$	$O(n)$	or $O(nd)$
shell	×	$O(n*\log_2 n)^2$	$O(n*\log_2 n)$	介於$O(n*\log_2 n)^2$	

— bubble sort 每一回合把最大移去最右去

(大隊接力) Selection sort 適合長的或小n的資料 (n大swap太多次)

(撲克牌) Insertion sort 用一個local wall向右推且排序

merge 分組後合併
(遞迴)

quick 分組後遞迴
(pivot 軸)

radix 分解key分配至buckets

objective

find the worst case ✓
average case ✓
Best case ×

	stable?	Worst	best	Average	
bubble sort	✓	$O(n^2)$	$O(n)$	$O(n^2)$	
selection	×	$O(n^2)$	$O(n^2)$	$O(n^2)$	
insertion	✓	$O(n^2)$	$O(n)$	$O(n^2)$	
merge	✓	$O(n*\log_2 n)$	$O(n*\log_2 n)$	$O(n*\log_2 n)$	
quick	×	$O(n^2)$	$O(n*\log_2 n)$	$O(n*\log_2 n)$	
heap	×	$O(n*\log_2 n)$	$O(n\log_2 n)$	$O(n*\log_2 n)$	
radix	✓	$O(n)$	$O(n)$	$O(n)$	or $O(nd)$
shell	×	$O(n*\log_2 n)^2$	$O(n*\log_2 n)$	介於$O(n*\log_2 n)^2$	

— bubble sort 每一回合把最大移去最右去

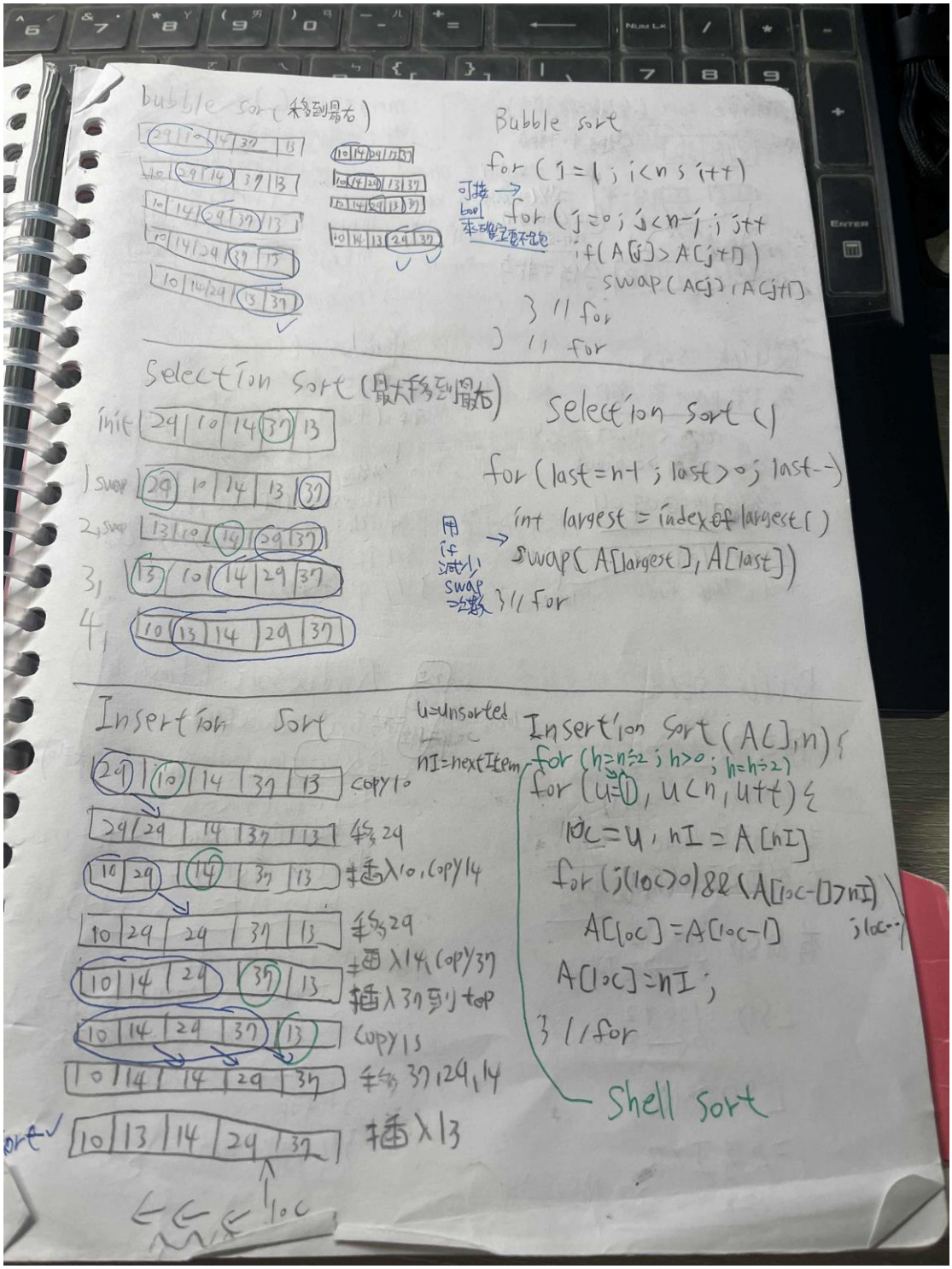
(大隊接力) Selection sort 適合長的或小n的資料 (n大swap太多次)

(撲克牌) Insertion sort 用一個local wall向右推且排序

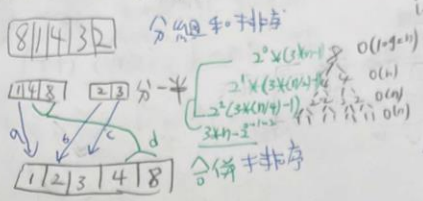
merge 分組後合併
(遞迴)

quick 分組後遞迴
(pivot 軸)

radix 分解key分配至buckets


bubble sort (移到最右)
Bubble sort
for (i=1; i<n; i++)
for (j=0; j<n-i; j++
if(A[j]>A[j+1])
swap(A[j], A[j+1])
} // for
} // for
可接 bool 來確定要不要跑
Selection Sort (最大移到最右)
Selection sort ()
for (last=n-1; last>0; last--)
int largest = index of largest()
swap(A[largest], A[last])
} // for
用 if 減少 swap 次數
Insertion Sort
u=unsorted
nI=nextItem
Insertion Sort(A[], n){
for (h=n/2; h>0; h=h/2)
for (u=0, u<n, u++){
loc=u, nI=A[nI]
for (; (loc>0) && (A[loc-1]>nI)
A[loc]=A[loc-1] ; loc--)
A[loc]=nI;
} //for
Shell sort
copy 10
移 29
插入 10, copy 14
移 29
插入 14, copy 37
插入 37 到 top
copy 13
移 37, 29, 14
插入 13
loc

merge sort (先分組後排序)




```
mergesort ( ) {
if (first < last) {
  int mid = (first+last) / 2
  mergesort (first, mid)
  mergesort (mid+1, last)
  merge (first, mid, last)
}// if
}// mergesort
```

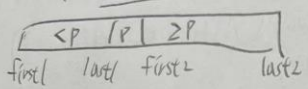
Quick sort

先找 pivot 來分組

item < pivot

item >= pivot

後遞迴呼叫




```
quicksort ( ) {
int pivotindex
if (first < last){
  partition (first, last, pivotindex)
  quicksort (first, pivotindex-1)
  quicksort (pivotindex+1, last)
}// if
}// quicksort
```

Radix sort

用基數與容器分組

(0~9) (bucket)

先分解取 value 分組

在分配到 bucket 串接

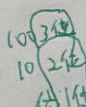
有 MSD 10012 左→右排序

LSD 10012 左←右排序

LSD > MSD

二次高正確

// MSD 會錯

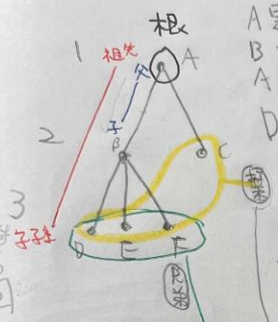



```
Radixsort (first, last)
int bucket[10]
for(base=1; (maxdata/base)>0; base*=10)
  for(i=first; i<=last; i++)
    bucket[(A[i]/base)%10+1]++ //count
  for(i=1; i<10; i++)
    bucket[i] += bucket[i-1] //串接 分組
  for(i=first; i<=last; i++)
    temp[bucket[(A[i]/base)%10]++] = A[i]
  for(i=first; i<=last; i++)
    A[i] = temp[i]
}// for
}// Radixsort
```

L8 木討

binary tree 位置導向 (stack / list / queue)

binary search tree 內容導向 (sorted list)



A是B的父Node (parent)

B ——— 子Node (child)

A是D的祖先Node (Ancestor)

D是A的子孫Node (descendant)

根 (root)：沒有父節點的唯一Node

子樹 (subtree)：某個Node與其子與子孫，exp：(B)

葉 (Leaf)：沒有子節點的Node

兄弟 (siblings)：有共同的父節點之Node

樹高 (height)：從根到葉最長路徑的節點數量

Binary tree

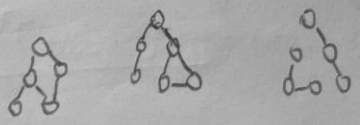
有零節點(empty)

或一樹有小於等於2的子節點 (左子樹 left subtree, 右子樹 right subtree)

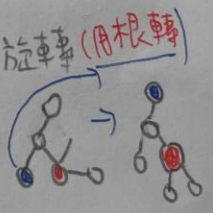
和二個樹根

不可相斷，三角形，短路各

height(T) = 1 + max{h(L), h(R)}



可旋轉 (用根轉)



Full Binary Trees 完全樹

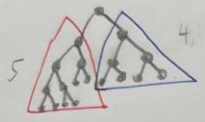
每個都已經滿，像這時間不用，限制較多(數量固定

Complete Binary Trees 完整樹

] full ⇒ level = h-1

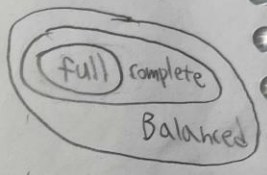
在 levels ≤ h-2 都各有兩個子節點且
level = h-1 有二子那個節點右方都有二子
而 level = h 只有一子時就只有左子點。

Balanced Binary Trees 平衡樹



樹高差不超過 1 就行
完整樹 = 平衡樹
完全樹

包含關係



用 ADT 來實現 by array or linked list
(多個free)
(開數列)

左子 = 2*父+1
右子 = 2*父+2
父 = (子-1)/2

完整樹樹高 (Full h)

level	該level Node	該樹Node
1	$1=2^0$	$1=2^1-1$
2	$2=2^1$	$3=2^2-1$
3	$4=2^2$	$7=2^3-1$
⋮	⋮	⋮
h	2^{h-1}	2^h-1

最小/最大樹高

完整樹高 (complete) 最小

$n \le 2^h-1 \Rightarrow \log_2(n+1) \le \log_2(2^h)$

$\Rightarrow h \ge \log_2(n+1) \Rightarrow h = \lceil \log_2(n+1) \rceil$

最大

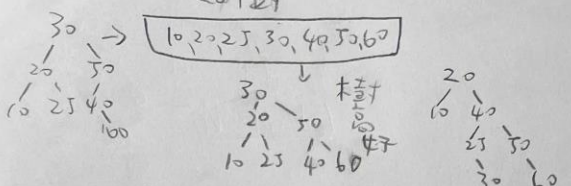
skewed 二元樹：n
傾斜

最大

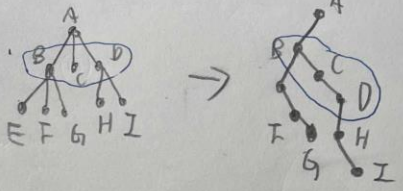
$(2^{h-1}-1)+1 \le n \Rightarrow \log_2(2^{h-1}) \le \log_2(n)$

$\Rightarrow h \le \log_2(n)+1 \Rightarrow h = \lfloor \log_2(n) \rfloor + 1$

中序走訪平衡樹



左小孩：最左側小孩
右小孩：右邊的兄弟



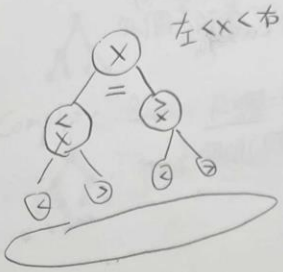
降低分支數
且可確定分支數量

二元搜尋樹 Binary Search Tree

左<X<右

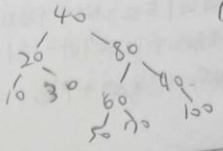
[Average case: O(log n)
Worst case: O(n)

struct BST
item
*left
*right
};



40, 20, 10, 80, 90, 100, 30, 60, 50, 70

(insert)



delete

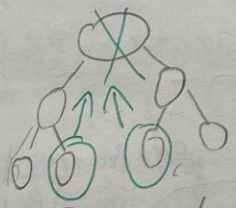
刪掉要設NULL

一個Node 用父Node接下面的Node

二個Node ???

用 Inorder 去尋找替代

successor

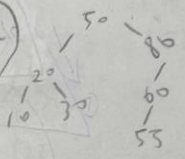


合手右小孩最左子孫

or 右小孩

copy上去最上然後delete

+55 -40 -70 -100 -40 -80



同高什麼交換數沒有指

加入移除次序會影響樹高

隨機次序加入可逼近最小樹高

operation	Average case	Worst case
Retrieval	O(log n)	O(n)
Insertion	O(log n)	O(n)
Deletion	O(log n)	O(n)
Traversal	O(n)	O(n)

Tree Sort 1. 建樹 and by n 插入

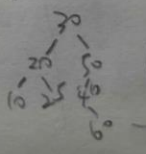
Average case O(n×log n)

Worst case O(n*n)

2. 中序走訪 Inorder traverse

O(n)

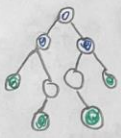
應用前序走訪成還原破壞樹



30 20 10 25 50 40 60

原本

二元樹滿足性質



N_2 = 3節點有兩子 (recursive的非base case)

N_0 = 4葉 (recursive的base case)

$-N_0 = N_2 + 1$ (葉-內部點=1)

B = 8 Branches (edges) 邊=點-1 ⇒(根)的關係

$- B = |E| = 2 * N_2 + 1 * N_1$

$N_0 + N_2 + N_1 = |V| = |E| + 1$

$= 2 * N_2 + 1 * N_1 + 1$

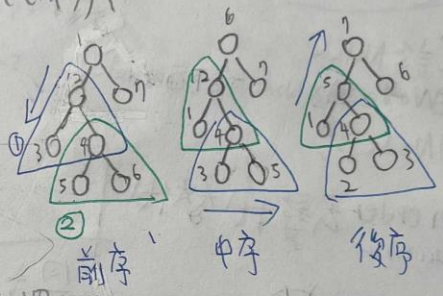
traverse (樹)

if (二元樹 is not empty) {

前序 preorder → traverse (左)

中序 inorder → traverse (右)

後序 postorder → } // if



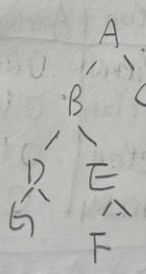
① ② ③

前序 中序 後序 前後對稱

用 走訪與兩序還原樹 (前後序不可還原)

exp = post = GDFEBCA

In = GDBFEAC

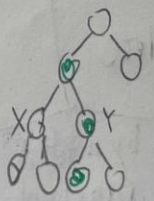


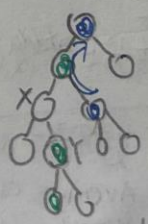
exp = Pre = ABCdefG

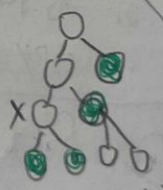
in = ABCdefG

successor

後序後者 中序後者 前序後者







1 右小孩

2 父親

3 祖先

1 右小孩左孩

2 右小孩

3 父親 右上角-祖先

1 小孩

2 兄弟

3 祖先的孩