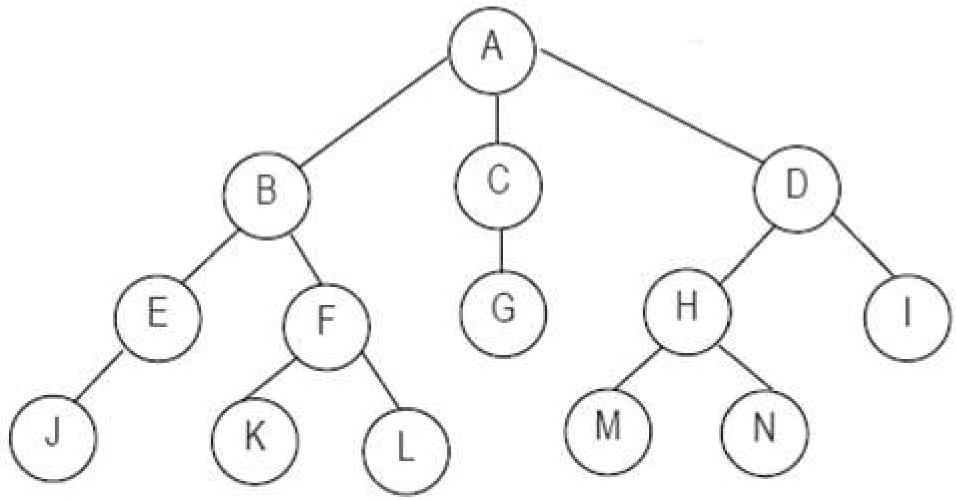
Chapter 5 樹狀結構(1)

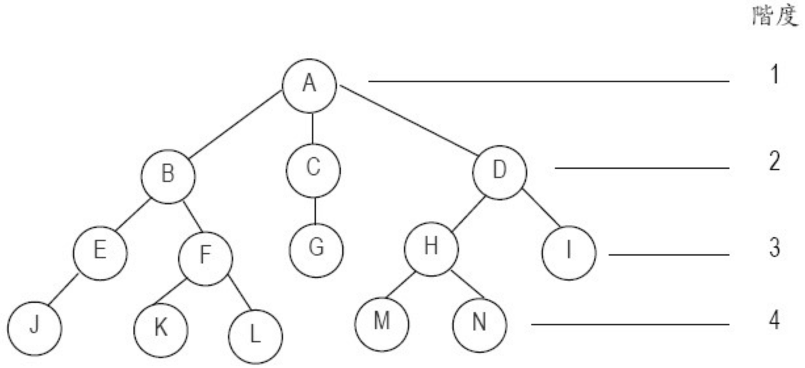
樹(tree)

- 定義：樹是由一個或多個節點所組成的有限集合，並使
  1. 存在一個特定節點稱為樹根(root)
  2. 剩下的節點分割成 $n \geq 0$ 個無交集的集合 $T_1, \ldots, T_n$， 其中每一個集合都是一棵樹， $T_1, \ldots, T_n$ 稱為此樹根的子樹(subtree)
- Example:
  - 樹根是A
  - 有三個子樹B, C, D



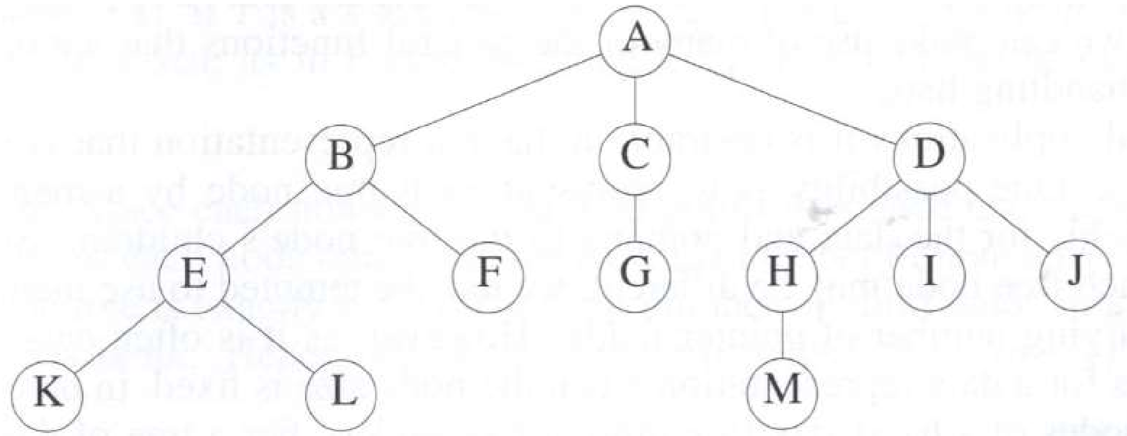
與樹相關的專有名詞

- 節點(node)：14個節點(A, B, … N) 。
- 節點的分支度(degree)：子樹的數目 ，如A, B, C的分支度為3, 2, 1。
- 終點節點(terminal node)或樹葉節點(leaf node)：分支度為0的節點(如J, K, L, G, M, N, I)。
- 非終點節點(nonterminal)：分支度不為0的節點(如A, B, E, F, C, D, H, I)
- 子節點(children node)與父節點(parent node)：某節點X的子樹的root稱為X的子節點(如B的子節點為E, F)， 而X為子節點的父節點(如E,F的復節點為B)。
- 兄弟節點(sibling node)：有相同的父節點的節點，如B的兄弟節點為C, D
- 祖先(ancestor)節點：為此節點到root所經的節點，如F的祖先為B, A
- 樹的分支度：為樹終結點最大的分支度，如此樹的分支度為3
- 階度(level)：root是level one, 若一個節點在level n, 則其子節點在level n+1, 如B在level 2、E在level 3、J在level 4
- 高度(height)或深度(depth)：為此樹最大的level，如此樹的高度為4



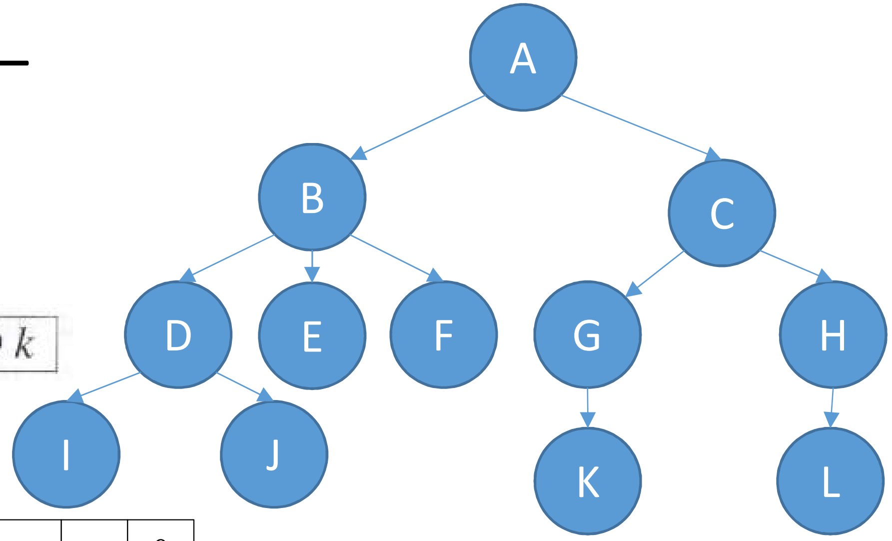
作業05-01

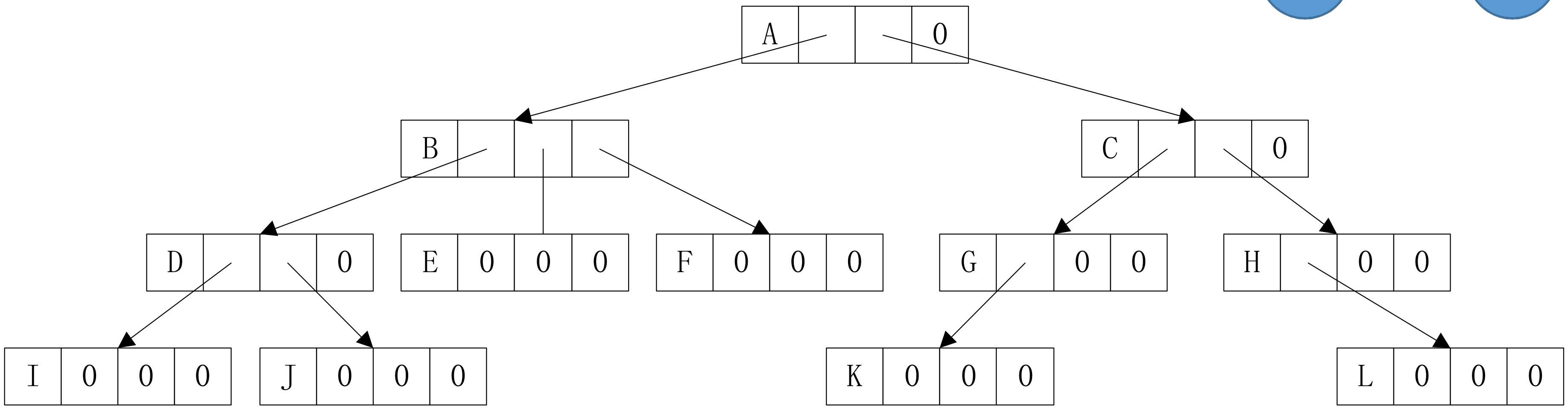
- B節點的分支度?
- 樹葉節點?
- 非終點節點?
- M的父節點?
- E的子節點?
- E的兄弟節點?
- H的祖先節點?
- 此樹的分支度?
- E在階度?
- 此樹的深度(高度)?



Representation of Tree – Linked List

DATA	CHILD 1	CHILD 2	…	CHILD k



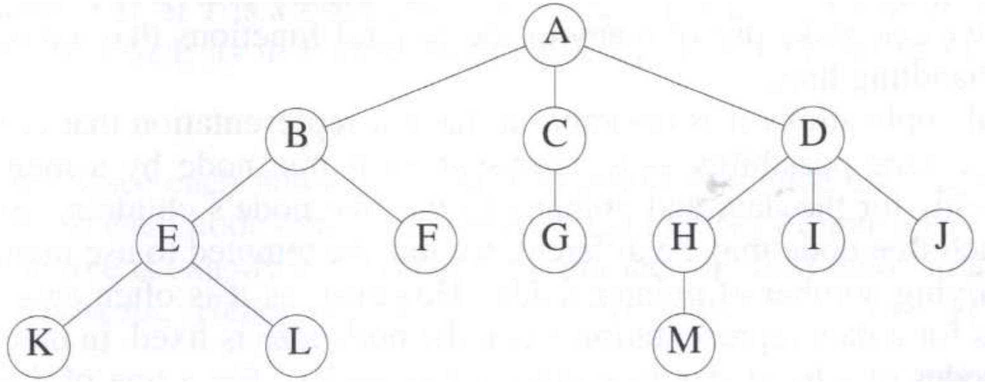


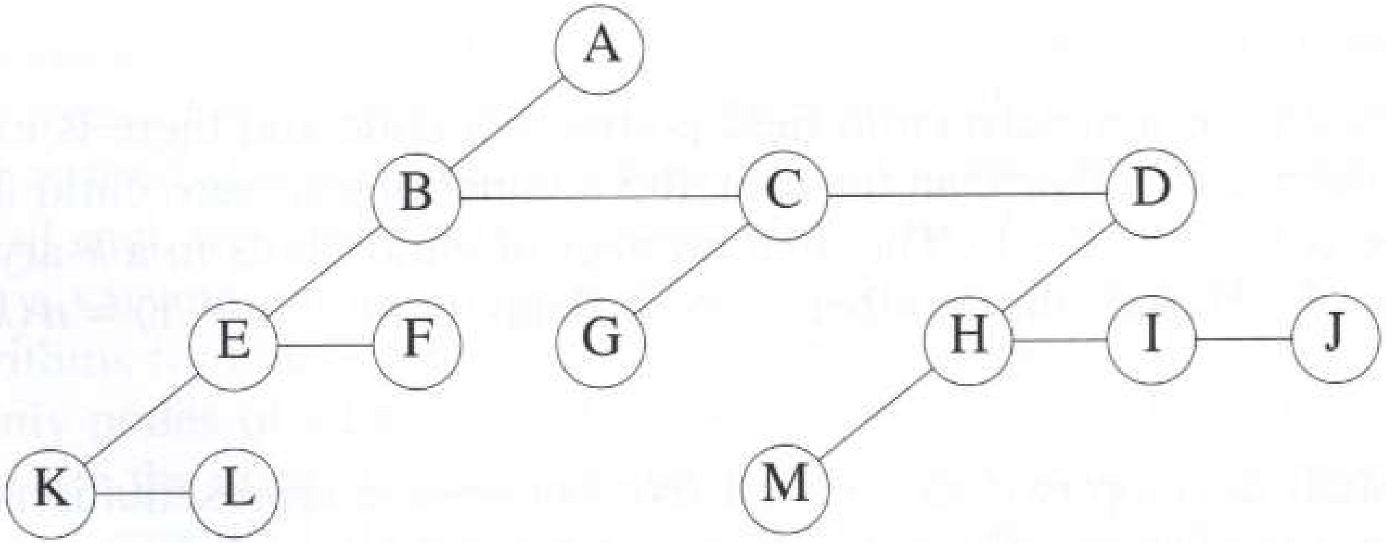
- If T is a k-ary tree (i.e., a tree of degree k) with n nodes, each having a fixed size as in above Figure, then $n(k - 1) + 1$ of the nk child fields are 0, $n \geq 1$.

$$nk - (n-1) = n(k - 1) + 1$$

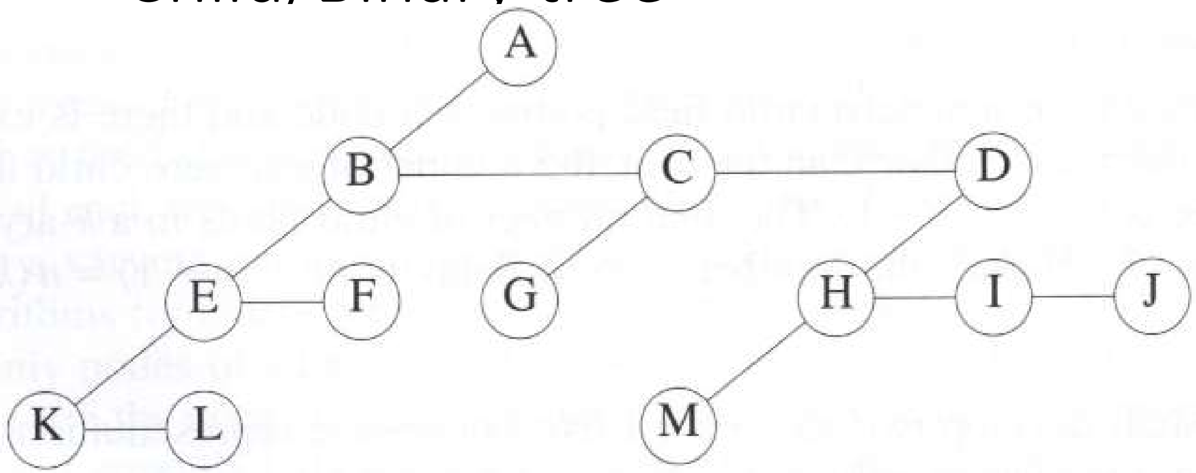
Representation of Tree –
Left Child-Right Sibling Representation

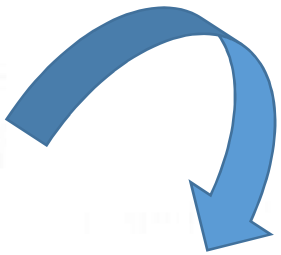
data	
left child	right sibling



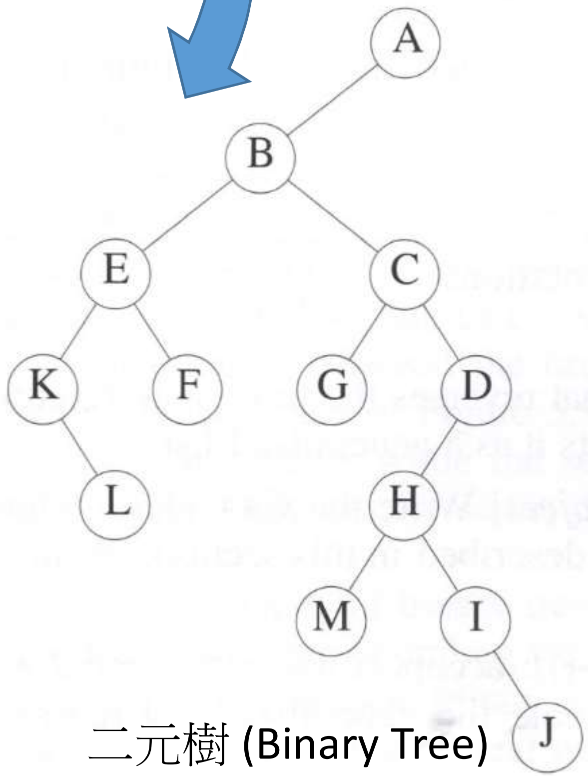


Representation of Tree –
Degree Two Tree Representation/ Left Child-Right Child/Binary tree





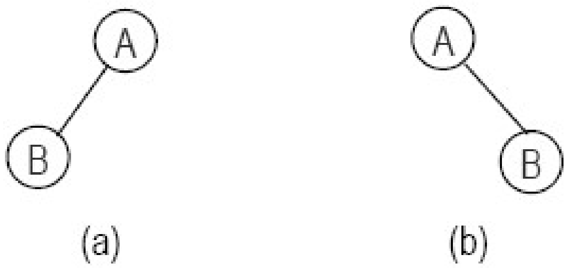
順時鐘轉45度



二元樹 (Binary Tree)

二元樹 (Binary Tree)

- 二元樹是由節點所組成的有限集合，這個集合若不是空集合，就是由樹根、右子樹（right subtree）及左子樹（left subtree）所組合而成。其中右子樹和左子樹可以為空集合。
- 二元樹與一般樹不同的地方是：
  1. 二元樹的節點個數可以是零，而一般樹至少由一個節點所組成。
  2. 二元樹有排列順序的關係，而一般樹則沒有。
     - 下圖的兩個二元樹是不同的
  3. 二元樹中每一節點的分支度至多為2，而一般樹則無此限制。



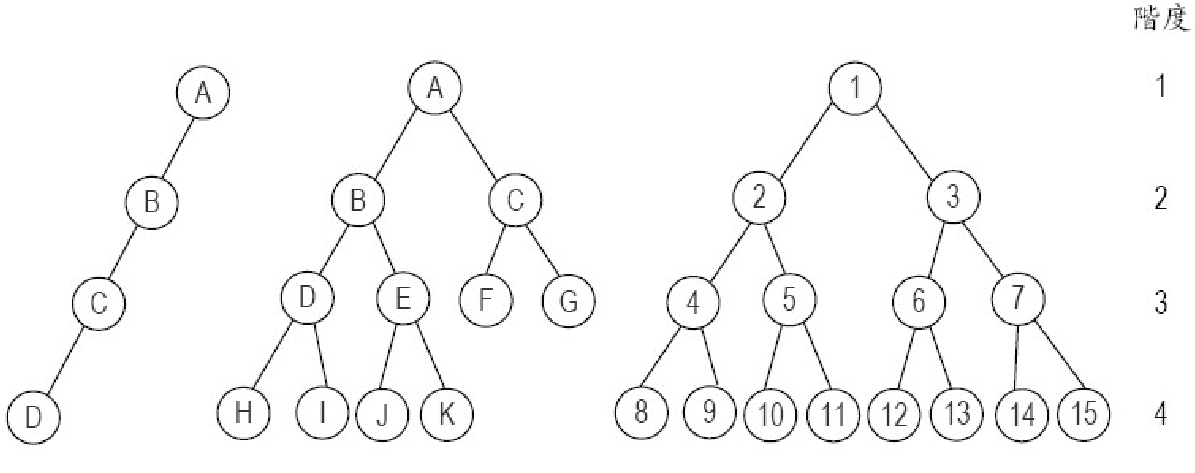
二元樹 ADT

```
class TreeNode{
    char data;
    TreeNode leftChild;
    TreeNode rightChild;
}
```

```
class BinaryTree {
    BinaryTree(); // creates an empty binary tree
    BinaryTree(int data, BinaryTree bt1, BinaryTree bt2);
    // creates a binary tree whose left subtree is bt1, whose
    // right subtree is bt2, and whose root node contains item
    boolean isEmpty();// return true if the binary tree is empty
    BinaryTree leftSubtree(); // return the left subtree of this
    BinaryTree rightSubtree(); // return the right subtree of this
    int RootData();// return the data in the root node of this
}
```

二元樹的種類

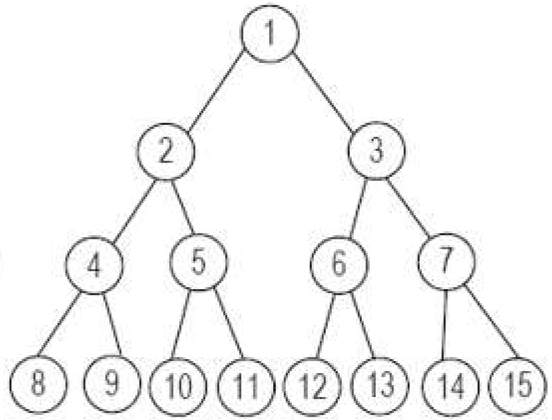
- 左/右斜樹(left/right skewed tree)：圖(a)。
- 完滿二元樹(fully binary tree) ：圖(c)。
  - 非終端節點都有兩個分支度的二元樹。
  - 完滿二元樹共有2^k-1個節點,k為二元樹的深度。
- 完整二元樹(complete binary tree) ：圖(b)。
  - 與完滿二元樹非常相似,但節點個數少於2^k-1。
  - 第i<k層有2^i-1個節點,第i=k層的節點由左至右順序編排。



(a) (b) (c)

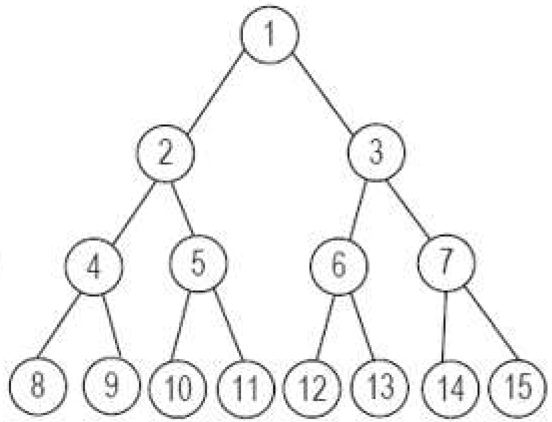
二元樹的特性

- 二元樹在第 i 階度的最多節點數為 2^{i-1}，i≥1。(證明使用數學歸納法)
  - i = 1， $2^{1-1} = 1$，成立
  - 假設i = i -1成立， 所以第 i -1階度的最多節點數為 $2^{i-1-1}=2^{i-2}$
  - i=i時，第i-1層有2^{i-2}個節點，而每個節點會產生兩個分支，所以第i皆會有 $2*2^{i-2}= 2^{i-1}$個節點。



二元樹的特性

- 高度(或深度)為k的二元樹，最多的節點數為2^k-1，k≥1。
  - 將1~k階中的節點加起來就是階度為k的二元樹，最多的節點數
  - 2^{1-1}，$+2^{2-1}$，$+ \ \ldots + 2^{k-1} = \sum_{i=1}^{k} 2^{i-1} = 2^k$-1
- 對任一非空二元樹T，若n_0為樹葉節點數，n_2為分支度為 2 的節點數，則$n_0=n_2+1$。
  - 令n為所有節點數，n_1為分支度為 1 的節點數，可得$n=n_0+ n_1 +n_2$ (a)
  - 除了root外，所有節點都有一個分支連向它，設此二元樹有B個分支，則n=B+1，$B = n_1 +2n_2$
  - $n = B+1 = n_1 +2n_2+1$, 代入(a)式，則$n_1 +2n_2+1 = n_0 + n_1 +n_2 \Rightarrow n_0 = n_2+1$
- 一個n個節點的完滿二元樹其高度為$log_2(n+1)$
  - $2^k-1 = n \Rightarrow 2^k = n + 1 \Rightarrow k = log_2(n+1)$
- 一個n個節點的完整二元樹其高度為$\lceil log_2(n+1) \rceil$



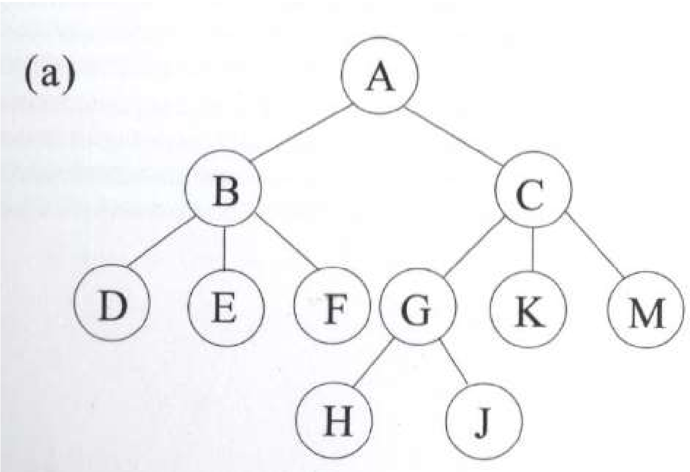
$S_n = a_1 \frac{1-q^n}{1-q}$

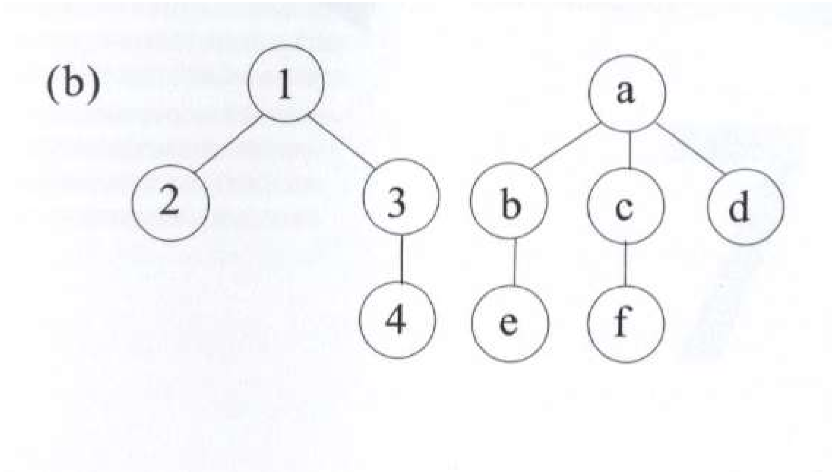
作業05-02

1. 若一個二元樹有200個節點，則其最小階度為何?
2. 若一個二元樹有200個葉節點，則其最小階度為何?
3. 若一個二元樹有100個樹葉節點，則分支度為 2 的節點數=?
4. 若一個二元樹有55個節點，其中分支度為 1 的節點數有22個，求樹葉節點數?

作業05-03

請將下列的一般樹化為二元樹

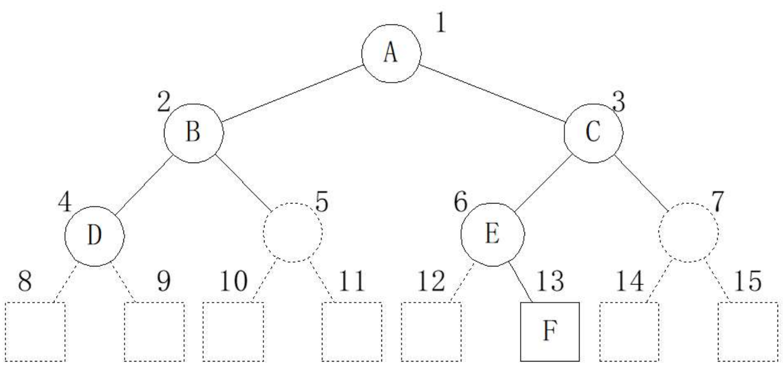


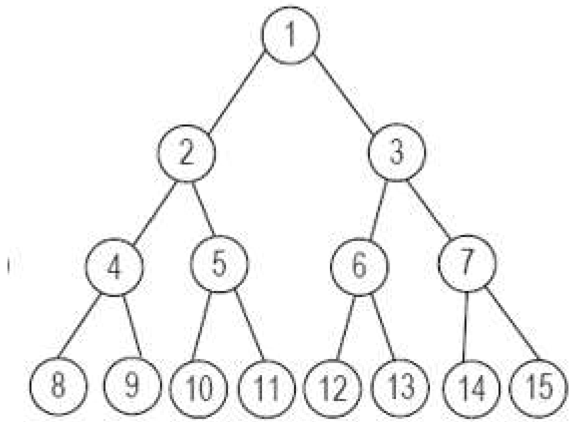


二元樹的表示方法－陣列 (1)

- 將二元樹的節點儲存在一維陣列
  - 下圖為儲存在一維陣列的表示法。
  - 若運用在非完整二元樹或滿枝二元樹時，可能會造成許多空間的浪費 → 使用鏈結串列。

(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)	(11)	(12)	(13)	(14)	(15)
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1st level	2nd level		3rd level				4th level							

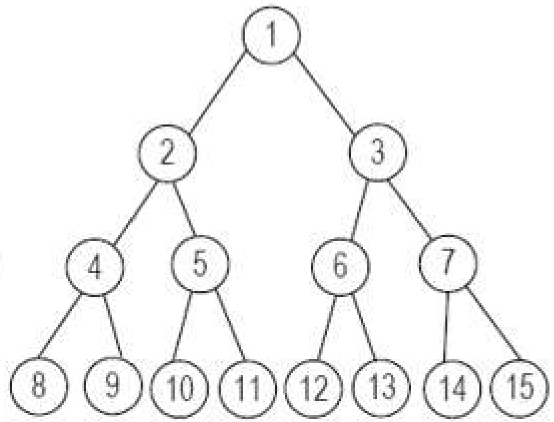




二元樹的表示方法－陣列 (2)

含有n個節點的完整二元樹，其深度為 $\lceil \mathrm{Log}_2(n+1) \rceil$，如果從樹根開始，由上而下，由左而右將每一個節點編號，編號為1, 2, ..., n，那麼對於任何一個節點i, 1≦i≦n 均滿足以下三個特性：(假設陣列註標是從1到n)

1. 若i≠1，其父節點位於 $\lfloor i/2 \rfloor$ 位置，若i=1時無父節點。
2. 若2i≦n，節點i其左子樹根位於2i位置。若2i>n，則節點i沒有左子節點
3. 若2i+1≦n，節點i其右子樹根位於2i+1位置。若2i+1>n，則節點i沒有右子節點



Proof：

若2i≦n，節點i其左子樹根位於2i位置。若2i>n，則節點i沒有左子節點

Proof:

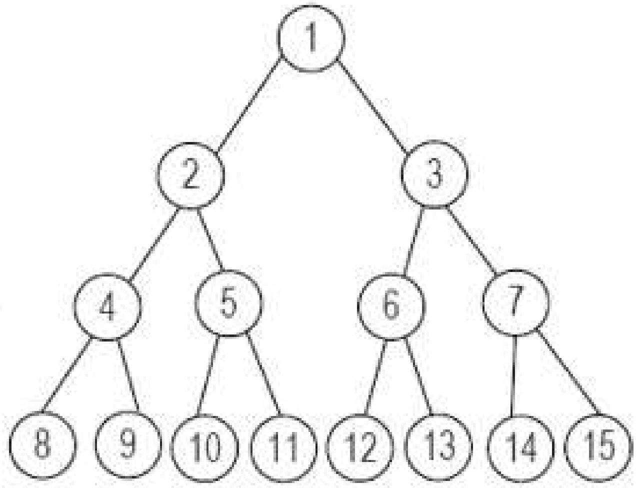
i=1，節點1其左子樹根位於2i=2位置，除非2>n，則節點1沒有左子節點

i=j成立，節點j其左子樹根位於2j位置，除非2j>n，則節點j沒有左子節點

i=j+1時，節點j+1其左子樹根位於2(j+1)，

因節點的左節點為2j右節點為2j+1，所以

下一個節點2j+2為j+1節點的左節點。



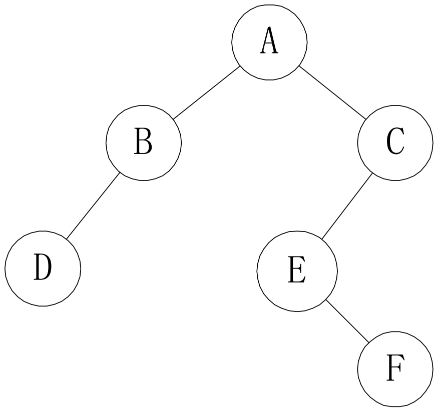
二元樹的表示方法－鏈結串列(1)

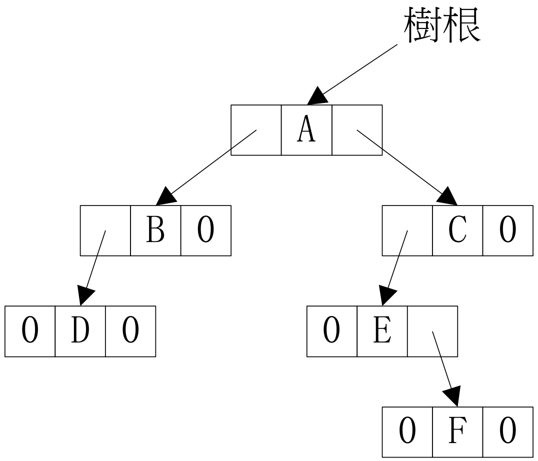
- 二元樹的節點結構

左子樹鏈結 leftChild	節點資料欄 data	右子樹鏈結 rightChild

```
class TreeNode{
    char data;
    TreeNode leftChild;
    TreeNode rightChild;
}
```

- 二元樹的鏈結表示法

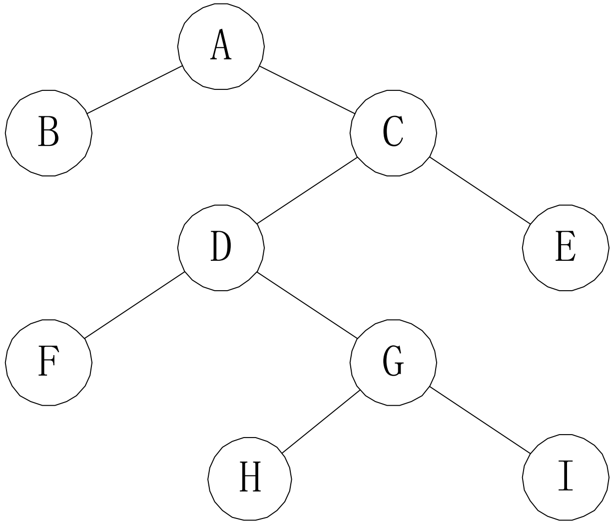




二元樹的追蹤(1)

- 拜訪樹中每個節點各一次稱為追蹤(Traversal)。
  - 中序追蹤(Inorder Traversal)
    - 中序追蹤法是依左子樹, 樹根節點, 右子樹之順序來拜訪每一個節點, 亦即以LDR之順序, 因樹根放在中間而得名。
  - 前序追蹤 (Preorder Traversal)
    - 拜訪節點之次序是樹根節點, 左子樹, 最後才是右子樹，亦即以DLR之順序。
  - 後序追蹤(Postorder Traversal)
    - 由於我們規定先左子樹再右子樹, 因此後序追蹤之次序為左子樹, 右子樹, 最後才是樹根節點，亦即以LRD之順序。

二元樹的追蹤(2)



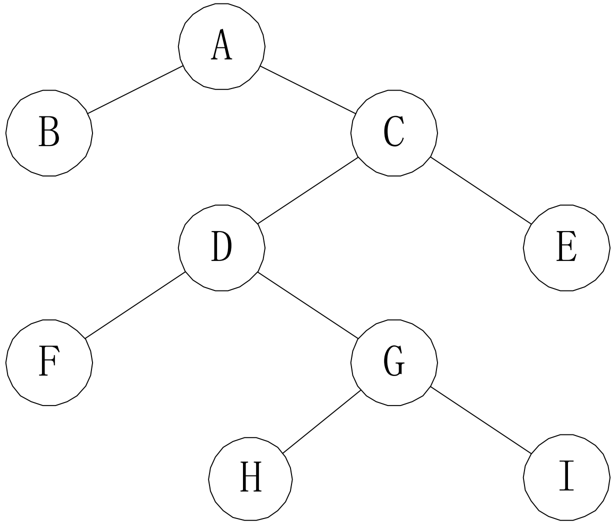
- 中序追蹤結果為：B、A、F、D、H、G、I、C、E
- 前序追蹤結果為：A、B、C、D、F、G、H、I、E
- 後序追蹤結果為：B、F、H、I、G、D、E、C、A

二元樹的追蹤(3)

```
public void inorder(TreeNode currentNode)
{
        if(currentNode!=null){
                inorder(currentNode.LeftChild);
                println(currentNode.data);
                inorder(currentNode->rightChild);
        }
}
```

```
public void preorder(TreeNode currentNode)
{
        if(currentNode!=null){
                println(currentNode.data);
                preorder(currentNode.leftChild);
                preorder(currentNode.rightChild);
        }
}
```

```
public void postorder(TreeNode currentNode)
{
        if(currentNode!=null){
                postorder(currentNode.leftChild);
                postorder(currentNode.rightChild);
                println(currentNode.data);
        }
}
```



- 中序追蹤結果為：

B、A、F、D、H、G、I、C、E

- 前序追蹤結果為：

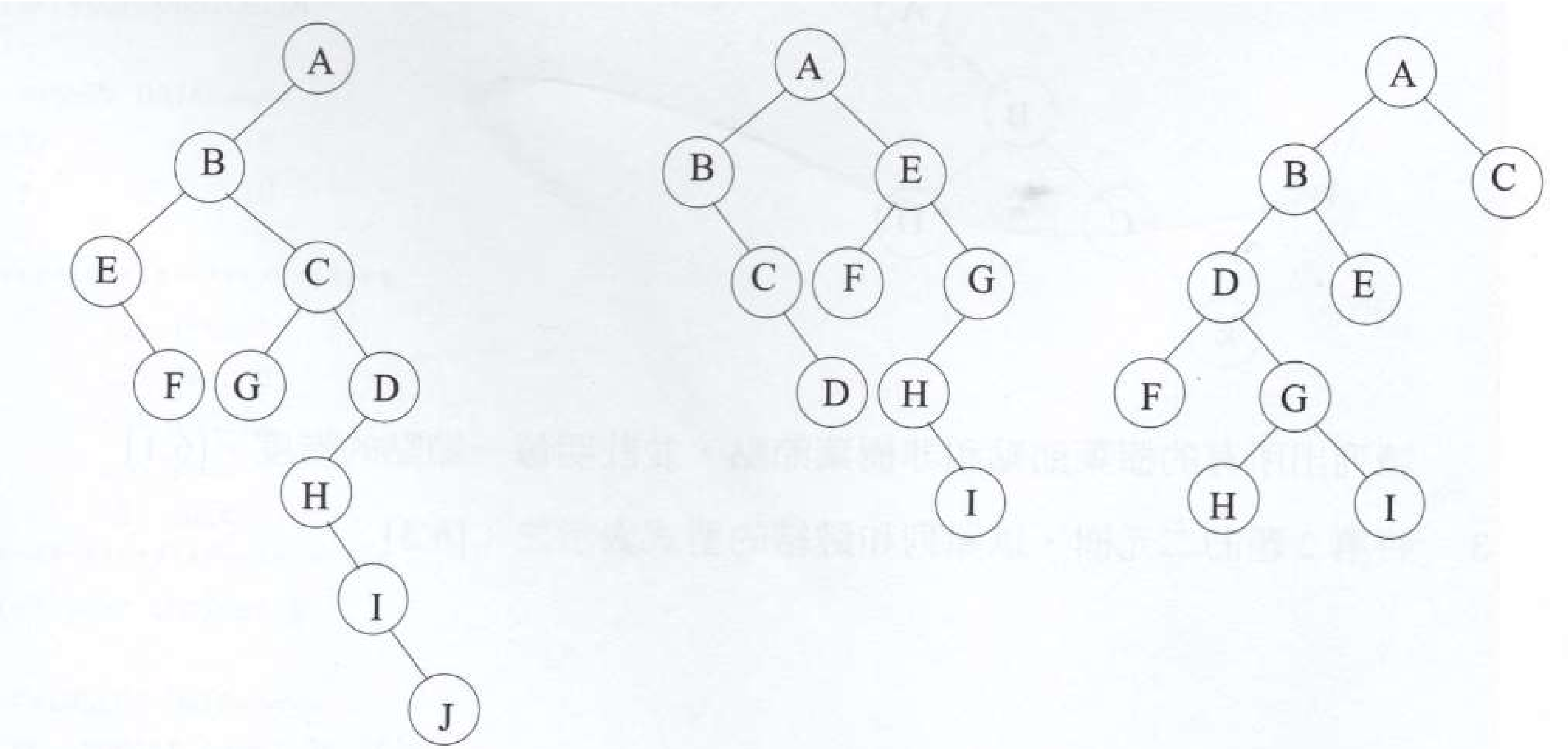
A、B、C、D、F、G、H、I、E

- 後序追蹤結果為：

B、F、H、I、G、D、E、C、A

作業05-04

- 請寫出下列三顆二元樹的中序、前序、後序追蹤的結果。



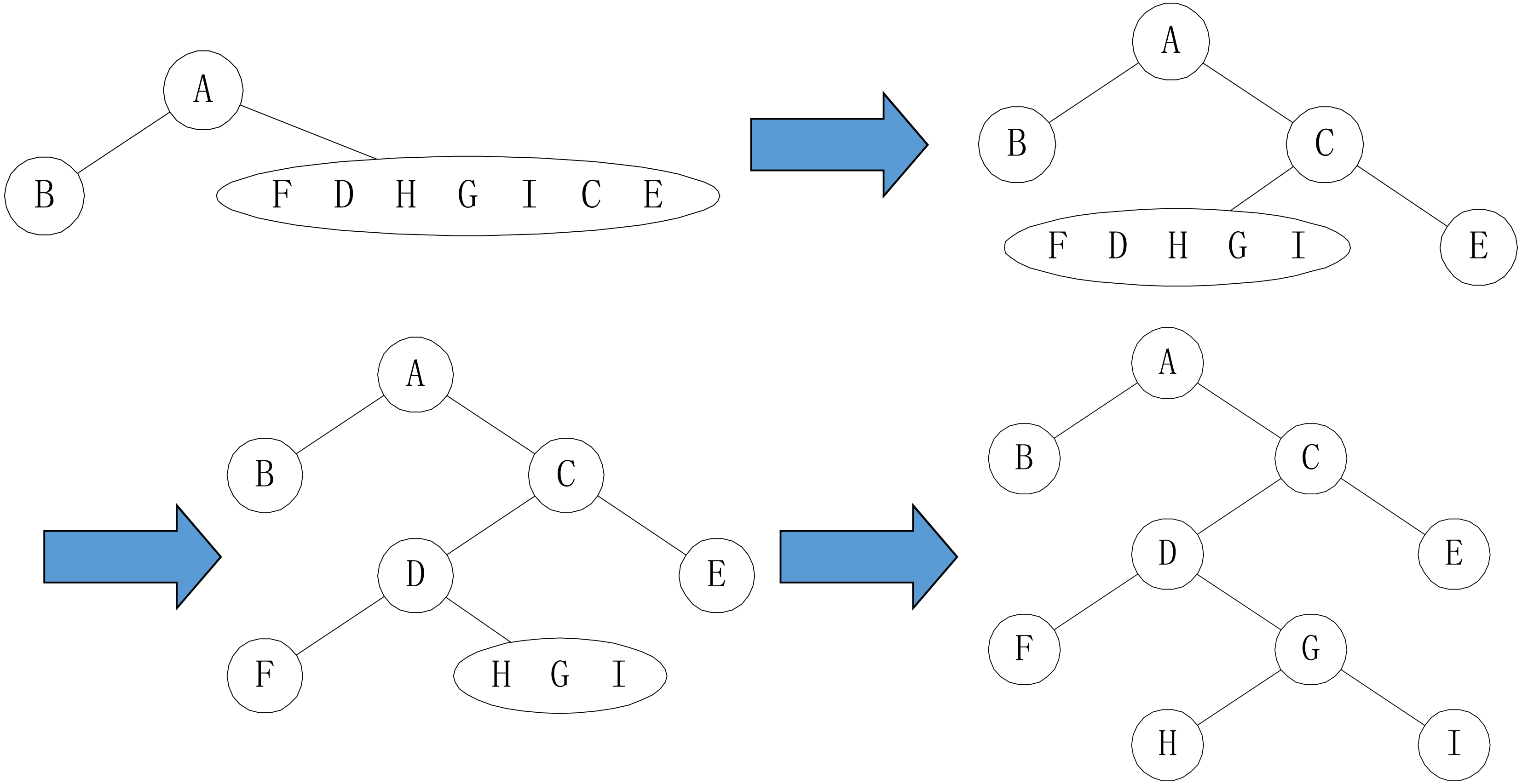
(a) (b) (c)

二元樹的追蹤(3)

- 上述三種追蹤有以下幾點特性:
  1. 樹根為前序追蹤的第一個節點,也是後序追蹤的最後一個節點。
  2. 已知前序追蹤與中序追蹤結果,便能決定出唯一的二元樹。
  3. 已知後序追蹤與中序追蹤結果,便能決定出唯一的二元樹。

二元樹的追蹤(4)

- 已知二元樹之前序追蹤結果為：A、B、C、D、F、G、H、I、E,且中序追蹤結果為：B、A、F、D、H、G、I、C、E，則該二元樹為何？

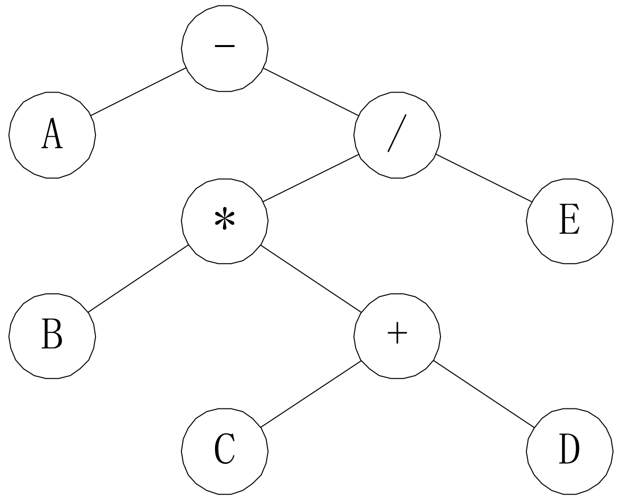


作業05-05

- 已知二元樹之前序追蹤結果為：A、B、C、D、E、F、G、H、I,且中序追蹤結果為：B、C、A、E、D、G、H、F、I，則該二元樹為何？

將中置運算式轉換成二元樹

1. 依運算子之優先順序將中置式加適當括號。
2. 由最內層開始, 將運算子放在樹根位置, 而左運算元當作左子樹根, 右運算元當作右子樹根, 一層一層地脫去括號即可建造出相對的二元數。



中序運算式A-B*(C+D)/E

1. (A-((B*(C+D))/E))

- 後序追蹤可獲後置式: ABCD+*E/-
- 前序追蹤可獲前置式: -A/*B+CDE