Ch5 Tree與Binary Tree

Tree是由>0個Nodes構成之有限集合，不可以為空

Root’s level = 1

(有些版本、題目設為0開始)

Forest

>=0顆互斥樹，所構成之集合，可以為空

Tree的表示方法

法一:利用Link List直接表示Tree

如果Tree’s degree = k，故一個Node準備K個Link欄位

缺點:浪費Link Space，只有(n-1)條Link有用(n個節點)

法二:將Tree化成Binary Tree，再用B.T表示方法儲存

法三:child-sibling方法

Node Structure為Data child sibling

Child: Left-most child only

Sibling: Next-right sibling

法四:括號法

利用父(子1子2…子m)表現父與子點之組成關係，且可以巢狀表示

Binary Tree是由>=0個Nodes所構成之有限集合，可以為空

滿足:

1. 有Root及左右子樹
2. 左右子樹也是Binary Tree

Binary Tree三個基本定理(假設 root之Level = 1)

1. B.T中第i Level之最多Node數:
2. 高度k之Binary Tree，最多Node數:

最少Node數:k(一層一個點)

1. 非空的B.T

Leaf個數:N0個

Degree = 2 ，Node數:N2個

N0 = N2 + 1

Binary Tree的種類

1. 斜曲的B.T => 左斜、右斜
2. Full B.T => 高度=k , 最多Node數 =
3. Complete B.T => 高度=k, n個Nodes , 滿足 < n < ，由上而下，由左而右的Node增長順序

Complete B.T之定理

某Node之編號是i

左子點編號:2\*i

右子點編號:2\*i + 1

父點編號:i/2 (取整數)

1. Complete、Full B.T必為最小樹高
2. Complete B.T n1 = 1 or n1 = 0 => true
3. 有一個B.T，若Root的左右子樹皆為Complete B.T，則整棵B.T必為Complete B.T => False，換成Full、Binary Search Tree、HEAP、AVL Tree、Red-Black-Tree 皆False
4. 一個complete B.T任何Node之左右子樹必為Complete B.T => True，換成Full BST、AVL Tree、HEAP、皆True，Red-Black Tree => False
5. 在Complete B.T，若Node無左子點、右子點，必為Leaf => True
6. N個Nodes之Complete B.T高度必為最小化 => True (full 也是)

Strict B.T

在B.T中，Non-leaf必有2個子點，即Degree = 1, Node數 = 0個

1. Full B.T 必為 Strict B.T => True
2. Complete B.T 必為 Strict B.T => False
3. Strict B.T 必為 Full B.T => False
4. Strict B.T 必為 Complete B.T => False

B.T的表示方法

1. 利用Array表示，仿用Full B.T之Node順序，將Data填入Index中

優點:易於存取左、右子點、父點，針對Full、Complete B.T有充分利用空間

缺點:Node增刪不便、對於斜曲的B.T極度浪費空間

1. 利用Link List表示，Node Structure => Lchild、Data、Rchild，Lchild、Rchild分別指向左、右子樹

優點:Node增刪方便、對於斜曲的B.T存放，較省空間

缺點:不易存取父點，Link空間約50%浪費

B.T Traversal

方法:

1. Preorder: DLR
2. Inorder: LDR
3. Postorder: LRD

Stack應用

1. Levelorder

Queue應用

給予前序、中序 or 後序、中序必可決定唯一的Binary Tree，

前序、後序無法決定唯一，可能>=1棵以上

前序:ABC

後序:CBA

互為反向，可決定出N個不同的B.T

1. 左斜
2. 右斜
3. 先左在右
4. 先右在左

無法唯一(4棵不同的B.T)

1. B.T的前序=中序 => Empty、Root一個點、右斜
2. B.T的後序=中序 => Empty、Root一個點、左斜
3. B.T的前序=後序 => Empty、Root一個點

決定唯一的B.T

1. Level-order + 前序 => False
2. Level-order + 後序 => False
3. Level-order + 中序 => True
4. Complete B.T(or Full) + 前/後/中/Level-order =>True

Binary Search Tree

是一棵Binary Tree，若不為空，則滿足

1. 左子樹中所有Data值均<=Root
2. 右子樹中所有Data值均>=Root
3. 左右子樹也是BST

如何用BST排序

1. 將input data建成BST
2. 對於BST做Inorder Traversal即可得小->大

做Input是小->大 input 建出右斜的BST

反之，大->小 建出左斜的BST

成功搜尋X之比較次數=X之Level值

失敗搜尋X之比較次數=Null Level值 – 1 (=Failure Node)

那些可決定unique B.T

1. BST+前序 => True
2. BST+中序 => Fasle
3. BST+後序 => True
4. BST+Level-order => True
5. 若Root之左右子樹皆為BST，則整個BT也是BST => False
6. BST中，任何一個Node之左右子樹也是BST => True
7. 在BST中，左子樹之所有Data必小於右子樹的所有Data => True
8. 在BST中，父點皆大於子點之值 => False
9. BST實施Postorder，可得大->小 => False

Heap

可分為Max-Heap、min-Heap

以Max-Heap而言，定義

1. 是一棵Complete Binary Tree
2. 所有父點均 >= 其子點值
3. Root具有最大值
4. 製作priority queue之最合適的Data Structure
5. 適合用Array保存
6. 若B.T root之左、右子樹為Heap，則整棵B.T也是Heap => False
7. In a Heap，任何一個Node之左、右子樹也是Heap => True
8. In a Heap，父點皆>=其子點值 => True
9. In a Heap，左兄弟必<=右兄弟 => False
10. Heap適合用Link list保存 => False

針對Min-Heap的相關操作之時間複雜度

1. Insert X = O(logn)
2. Delete-Min = O(logn)
3. Find-Min = O(1)
4. Build Heap = O(n)
5. Merge two Heaps = O(n)
6. Delete x = O(logn)
7. Find-Max = O(n)
8. Decrease key of a node = O(logn)

Thread Binary Tree

定義:

一個n個Nodes的Binary Tree，如果以Link List來製作，則會有n+1條null Links，為了充分利用這些Null Links，因此將他們指向其他Node，此種B.T稱之

規則: 以中序順序為指引

1. 若X -> Lchild為Null，則視它為左引線，將它指向中序順序中X的前一個Node
2. 若X -> Rchild為Null，則視它為右引線，將它指向中序順序中X的下一個Node

Data Structure如下:

LeftThread Lchild Data Rchild RightThread

LeftThread => False:表Lchild指向左子點 True:表Lchild為左引線

RightThread => False:表Rchild指向右子點 True:表Rchild為右引線

新增一個Head Node不存任何data

兩種情況:

1. 空樹: LeftThread:T RightThread:F
2. 非空樹: LeftThread:F RightThread:F

簡化的中序追蹤系統 => no stack no recursion

規則:

從Head node出發，不斷地問(找)每個Node的Insuc為何，並列印(但Head Node不用印)，最後又回到Head結束

Insert t node成為S的右子點

Case1: S原本沒有右子點

Case2: S原本有右子點

Note: Insert t 成為S的左子點，將Alg中，left改Right，Right改Left，Insuc(x)改Inpre(x)