Ch9 Advanced Tree(高等樹)

Min-Max Heap、Deap、SMMH => 適合製作Double-ended Priority Tree

其Delete min、Delete Max、Insert X，皆在Time = O(logn)完成

Min-Max Heap(最小-最大堆積)

是一棵Complete B.T，且滿足下列

1. 此tree階層是以min-level、Max-level交替呈現
2. Root位於min-level
3. 若X位於min-level，則代表以X為Root之子樹中，X具有最小值

Deap(Double-ended Heap)(雙堆積)

是一棵Complete B.T，滿足下列

1. Root不存Data
2. Root之左子樹為Min-Heap
3. Root之右子樹為Max-Heap
4. 令i為Min-Heap中某個Node編號，令j為Max-Heap中對應的Node編號

則Deap[i] <= Deap[j]

其中j = i + 2的(log2(i+1)取上限再-2)次方

If (j > n) then j = [j/2]

SMMH(Symmetric Min-Max Heap)  
是一種Complete B.T，Root不存Data

滿足下列

1. Left sibling <= Right sibling
2. Node X 有祖父(grandparent)，則祖父的左子點必<=X
3. 祖父的右子點必 >= X

等同於

i的子孫中(不含i)中最小值為i的左子點

i的子孫中，最大值為i的右子點

Extended Binary Tree(延伸二元樹)

N個Nodes之B.T，若以Link List表示，則會有N+1條null Link，在這些Null Link上，加上特殊Node叫作External(or Failure)Nodes，其它Node叫Internal Node，此種Binary Tree稱之

External Node數 = Internal 數 + 1

定理:E=I+2N (N = Internal Node數)

Weighted External Path Length(WEPL)

加權外部路徑長度

Huffman Algo求min WEPL值

令W = N個外部節點加權值集合

Steps:

1. 從W中取出2個最小的加權值
2. 建立extended B.T
3. 將這2個權值和視為新權值，加入W中
4. Repeat 1~3 Until W中只剩一個權值，即作(n-1)個回合，此tree叫作Huffman Tree，其WEPL值即為min WEPL

AVL Tree

是一棵Height Balanced BST

若不為空，則滿足:

1. |HL-HR| <= 1，HL、HR為左、右子樹的高度
2. 左、右子樹也是AVL Tree

Node’s Balance Factor = Node’s左子樹高 – Node’s右子樹高

=> 只有三種可能 -1、0、1

AVL不平衡之cases:

4種:LL、LR、RL、RR

如何判斷:新點置入後，若有某個離它最近的Node變成不平衡，則看此新點在此不平衡點的甚麼子樹方向，只須看兩層，分出LL、LR、RL、RR

調整原則:

1. 中間鍵值往上拉，小的放左，大的放右，此三個Node是標示LL、LR、RL、RR兩條線所連結的三個Node
2. 孤兒Node or 子樹依BST性質即可置入正確位置

DS版本 => LL、LR、RL、RR 各視為一種Rotation

Alg版本 => LL、RR = 視為各一種Rotation LR、RL = 視為各兩種Rotation

AVL Tree: Insert X 時，頂多發生一種rotation(DS版本)、兩種rotation(Alg版本)

LL、RR => 2個pointers changed

LR、RL => 4個pointers changed

1. B.T若Root之左、右子樹皆為AVL Tree，則整棵B.T也是AVL Tree => False
2. 在AVL Tree中，任何Node之左、右子樹必為AVL Tree => True
3. AVL Tree中，任何Leaf之間的Level值差必<=1 => False
4. 對AVL Tree使用Inorder Traversal可得小->大 排序 => True

定理:形成高度H之AVL Tree，所需之Node樹 = F(H+2)-1 (F=>費氏數)

M-way Search Tree (Tree’s Degree = m, m >= 2)

緣由:因為在External Search方面，資料量太大，無法全部一次置於memory中，進行search，必須借助外部儲存體保存(如Disk) Data，在分批載入搜尋，要提升External Search之效率，必須降低Disk I/O次數，即降低Search Tree’s Height => 最有效的方法:加大Tree’s Degree => m-way search tree

B tree of order m

是一棵Balanced m-way Search Tree，主要用於external search及external sort

滿足下列:

Root至少有>=2個子點，即2<=Root’s Degree<=m

除了Root及Failure Nodes之外，其餘的Node之Degree必須在

m/2取上限 <=Degree<=m

所有的Failure Nodes皆位於同Level(即Balance之意)

B tree of order 3 => 整棵只有2種Nodes => 2-3 Tree

B tree of order 4 => 整棵只有3種Nodes => 2-3-4 Tree

Insert X in B tree of order m

1. 先Search for x，其Failure Node之Parent 為置入Node
2. 置入X於Node內
3. 檢查Node內key數是否有overflow

Case1:沒有(即key數 <= (m-1)

Case2:有(即key數=m個 or > m-1)，則須作Split(分裂)處理，且Split後，必須針對父點 goto3

Delete x in B tree of order m

1. 先Search for x，找出X位於哪個Node
2. 分兩大狀況處理
   1. X位於Leaf

刪X

檢查此Node是否underflow(key數不足)(破產)

Case1:沒有不足，Exit

Case2:不足，是否可作rotation調整(跟兄弟借錢)，若可以，則作rotation，不行則必須作combine處理(跟父點借錢)，且combine完後，必須針對父點goto => 檢查是否underflow循環去執行

* 1. X位於Non-leaf

X位於Non-leaf，則以X的左子樹中最大值或X的右子樹最小值，去取代X(設Y)，而Y必定在Leaf，故需檢查Leaf少一個key，回X位於Leaf的狀況去處理

Red-Black Tree [Algo版]

是一棵Balanced Search Tree，且滿足:

1. Node的顏色非黑即紅
2. Root一律為黑色
3. Null也視為黑色
4. 紅色Node的兩個子點必須是黑色
5. Root到任何不同Leaf之Path上皆具有相同數目的黑色Nodes => Balanced

Insert、Delete、Search X Time = O(logn)

Note: Balanced Search Tree之上述時間皆O(logn) 如: AVL Tree、B Tree

Red-Black Tree[DS版本]

是2-3-4 Tree對應的BST，滿足

1. Link的color非黑即紅
2. 若Link在2-3-4 Tree中存在，則視為黑色Link，否則為紅色Link
3. 任何path上不可出現連續紅色的Link
4. Root到任何不同Leaf之路徑，皆具有相同數目之黑色Links

Note:

1. 若2-3-4 Tree高度 = logn，則R-B Tree最小高度=logn(全部是2-Node)，最大高度=2\*logn(全部是4-Node)
2. R-B Tree中，皆為黑色Links(root到某Leaf) => 最短path

最長path => 黑、紅Links交錯

1. R-B Tree(AVL 也是)之Insert X 頂多發生一種rotation (2次rotations for double-rotation)
2. R-B Tree之Delete X 頂多發生2種rotation(3次rotations for single+double rotation) => 比AVL Tree優秀
3. 若B.T root之左右子樹皆為R-B Tree，則整棵B.T也是R-B Tree => False
4. 在R-B Tree中，任何Node之左右子樹也是R-B Tree => False
5. R-B Tree之Search Insert Delete X 皆為O(logn) => True
6. R-B Tree之Insert X 需 O(1) rotations => True
7. R-B Tree用Inorder Traversal 可得 小->大排序 => True