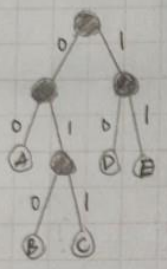
My Notes
Important Concepts worth keeping

Today: / /

11 Heap (Compelet binary tree)

Huffman coding 霍夫曼編碼



@ Encoder

A:00	D:10
B:010	E:11
C:011	

Text: EAEBAECDEA

1100 11 010 00 11 011 10 11 00

優點:

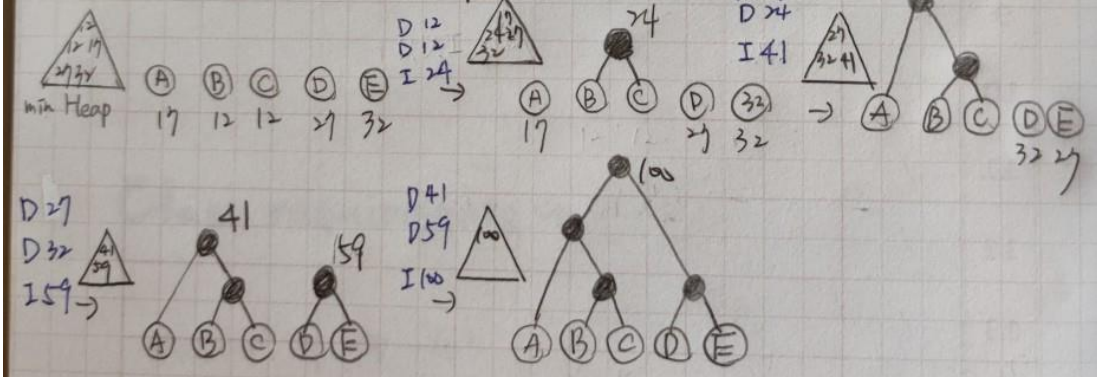
1. Avarage code length is minimized.
2. 出現Frequency高，代碼長度短，反之較長
3. a priority queue can be used to build the binary tree → heap

@ When use heap?

字出現次數

Character	A	B	C	D	E
Frequency	17	12	12	27	32

特別：此 binary tree 從樹葉開始做！



※ Heap是為了做出↑ binary tree所需要使用的資料結構[資料結構]

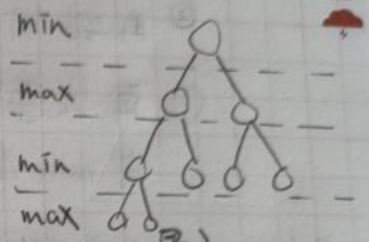
"Punctuality:Showing esteem for others by doing the right things at the right time."

My Questions
Problems & Difficulties needing exploration

My learning weather report

Double-ended Priority Queues [同時可知 Max、Min]

- Min-Max heap
- Double-end heap (Deap)



Min-Max heap

Insert : ① 檢查此節點所在 (Min or Max 層)

level = (int($\log_2(i+1)$) % 2) → 0 Min層 / 1 Max層

② 與其父節點比較大小

若此節點在 Max 層，而父節點大於此節點，swap

My Opinions
Thoughts, inspirations, and su~~ggestions~~

father = (i-1)/2

③ 與其祖父節點比較大小

if((i-1)/2 > 2) grandfather = ((i-3))/4

Delete : 1. Delete Smallest (刪root) / Largest (看刪誰

① 與其兩child比較，是否要換

② 在往下跟子祖比較

grandchildren = (i*4)+j

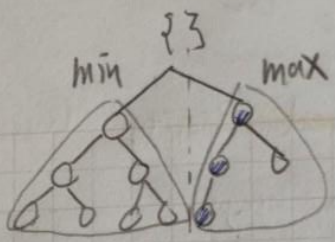
j = 3, 4, 5, 6

在對的時間，做對的事，
以表明對人的重視。
《培基文教》

My Notes

Important Concepts worth keeping

Today: / /



Heap.

Insert: ① 先找此node在 Min or Max 上

middle = int(log$_2$(i+1))/2

② 再與相对應節點比較

③ 再依序往上檢查至 smallest or Largest,

若 node 有與其父節點 大 or 小 就 swap.

Delete: ① 走至樹葉

② 與对應節點比較

③ 若交換,再往上依序檢查要不要換

"Do not worry about

My Questions
Problems & Difficulties needing exploration

My learning weather report

Forest (union) of Heaps

- Binomial heap = 項式堆積
- Fibonacci heap

Binomial tree of order k

The root has k children

Merge by two binomial tree of order k-1

Number of nods : 2^k

Tree height = k+1 $\Rightarrow O(\log n)$

$C_i^k = \frac{k!}{i!(k-i)!}$ node at level i, for $i = 0, \cdots, k$

↳ $13 = (1101)_2 = 2^3+2^2+0+2^0$

root

Example :

Binomial tree of order 4 (B_4)?

level 0: $C_0^4 = 1$
level 1: $C_1^4 = 4$
level 2: $C_2^4 = 6$
level 3: $C_3^4 = 4$
level 4: $C_4^4 = 1$

① 5 levels
② level 2 = 4 nodes
③ total $2^4 = 16$ nodes

密碼 cipher key

不要為明天憂慮，
因為明天自有明天的憂慮，
一天的難處一天當就夠了。
《新約聖經》

Today:

My Notes

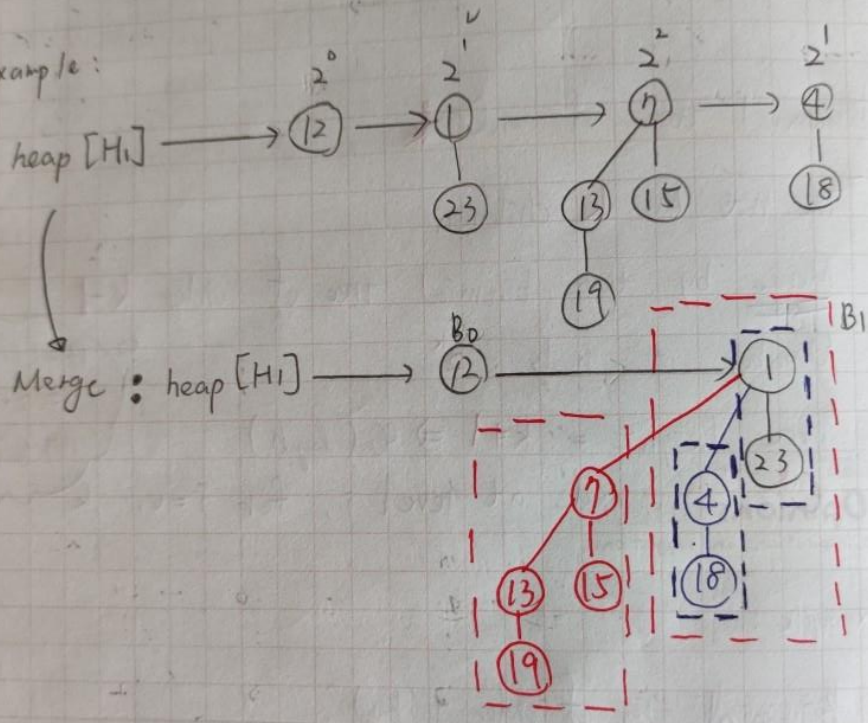
Important Concepts worth keeping

Merge

① 每個 root 都是 1 個 min heap

② Two binomial tree of the same order can be Merged

Example:



Example:

What does a binomial heap of "26" node looks like?

$26 = (11010)_2 = 2^4 + 2^3 + 0 + 2^1 + 0$ (3 root)

$208 = (11001000)_2 = 2^7 + 2^6 + 2^3$ (3 root)

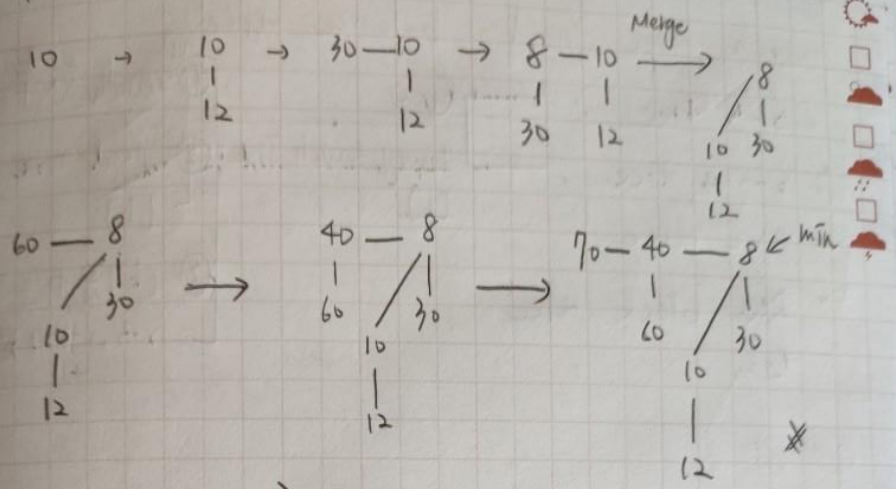
208 — 0
104 — 0
52 — 0
26 — 1
13 — 0
6 — 0
3 — 1

So in everything, do to others what you would have them do to you...
《New Testament》

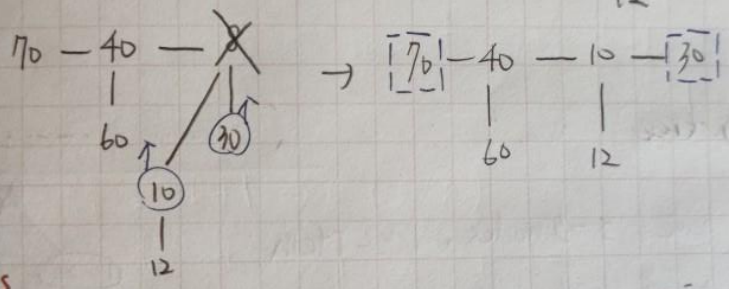
My Questions
Problems & Difficulties needing exploration

My learning weather report

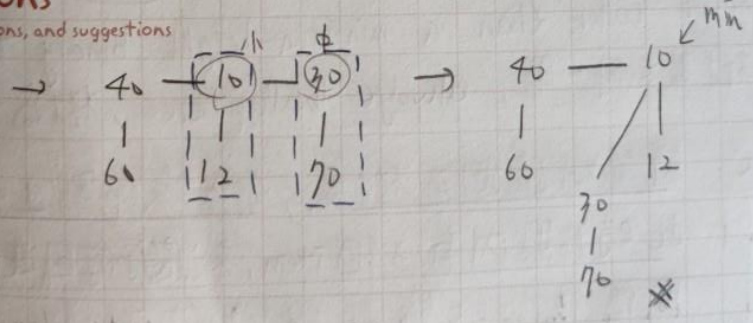
Insert: 10, 12, 30, 8, 60, 40, 70



Delete:



My Opinions
Thoughts, inspirations, and suggestions



密碼 cipher key

所以無論何事，
你們想要人怎樣待你們，
你們也要怎樣待人。
《新約聖經》

My Notes L3 Search

Important Concepts worth keeping

Today: / /

search Tree and balance tree

- 2-3 tree
- 2-3-4 tree
- AVL tree
- Red-black tree

V.S

binary search Tree [b]

① search 效率高於[b]

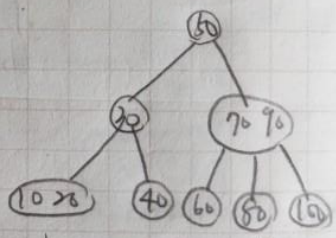
② need more storage than [b] 儲存空間

③ [b]是由上往下長

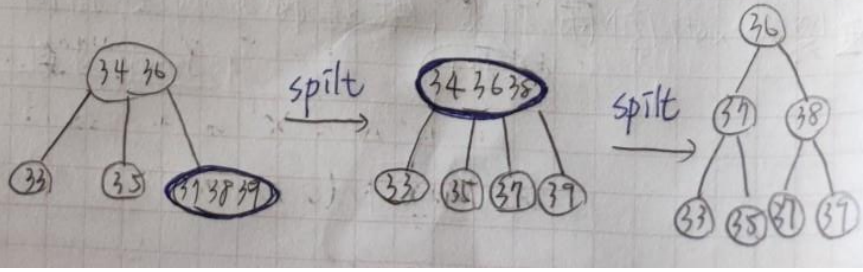
2-3, 2-3-4是由下往上長

2-3 tree

1. have 2~3 nodes, 1~2 item
2. never taller than a minimun-height binary tree $[\log_2(n+1)]$
3. 樹葉在同一層, always balance, 樹高一樣



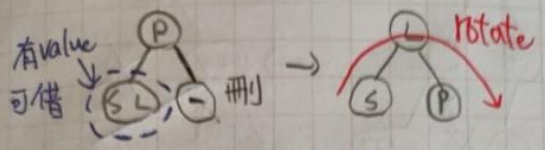
Insert: 新資料遞迴至樹葉, 加入該節點並sorted, 若此時節點內有3個item, 就依序往回進行spilt()



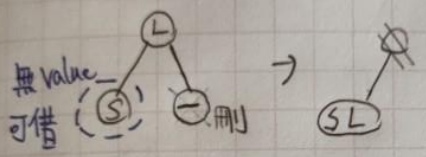
My Questions
Problems & Difficulties needing exploration

My learning weather report

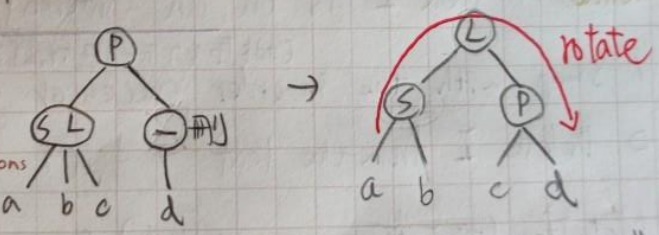
Delete : (a) Redistribute values → 當 Delete item 位於"樹葉"
重新分配



(b) Merge into a leaf → " " " "



(C) Redistribute values and child → " 位於"中間"
重新分配



My Opinions
Thoughts, inspirations, and suggestions

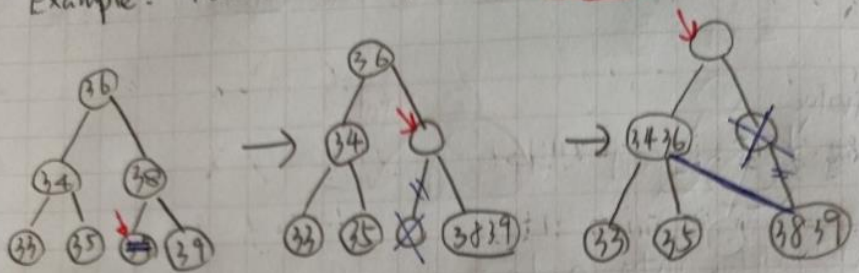
(d) Merge into internal node → 位於"root"

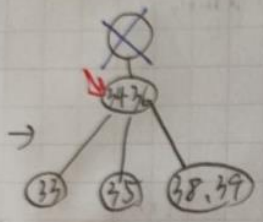
步驟：1. Delete I from the leaf
2. If the leaf now contain one item, you are done
3. else, choose one of the following operation to fix
(a) Redistribute values and (c) child
(b) Merge into a leaf
(d) Merge into Interal node

密碼 cipher key

終始功：以終為始
－史蒂芬．柯維

Example: Delete item 37 [若此節點 item 為 empty 就呼叫 fix()]





② What if item I is on an internal (內部) node?
[排序的下一個 item]
1. swap with the in-order successor on leaf
2. Delete I from the leaf → fix()

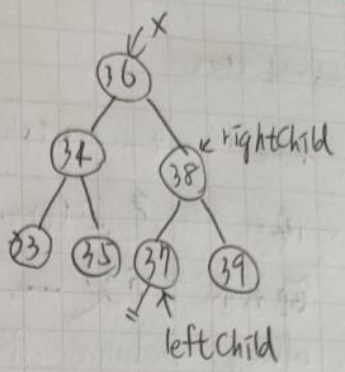
deleteItem(in ttTree, in theKey) {
X = the tree node whose search key equals theKey;
if(X is not leaf)
Y = successor(X);
swapKey(X, Y);
Delete key from X;
if(X now has no item) fix(X);
}

"We recognize individual differences with respect to talents, character, capability, and background. We believe that full development of one's potential signifies success"
(CYCU's Education Philosophy)

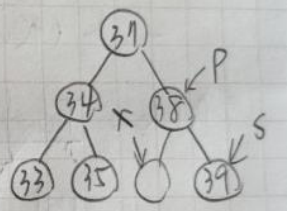
My Questions
Problems & Difficulties needing exploration

My learning weather report

```
Ptr Successor ( Ptr x ) {
  rightChild = x -> right;
  leftChild = rightChild -> left;
  while ( leftChild -> left != NULL)
    leftChild = leftChild -> left;
  return leftChild;
```



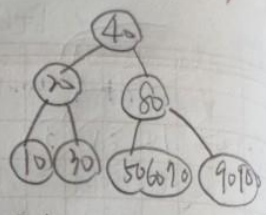

```
void fix ( Ptr x)
  if( x == root ) remove the root;
  else
    p = parant of x;
    if( the nearst silbing of x has two items)
      Redistribute items among the sibling p and x;
      if(x is not leaf)
          Move appropriate child from sibling to x;
    else // merge
      s = the nearst sibling of x;
      Move appropriate item done from p to s
      if ( x is not leaf) Move x's child to s;
      remove x;
      if ( p now has no item) fix(p);
```



密碼 cipher key

我們了解人人各承不同之稟賦，
其性格、能力與環境各異，
故充分發揮個人潛力就是成功。
《中原大學教育理念》

Today: / /



2-3-4 tree

1、have 2~4 nodes (child nodes), 1~3 item

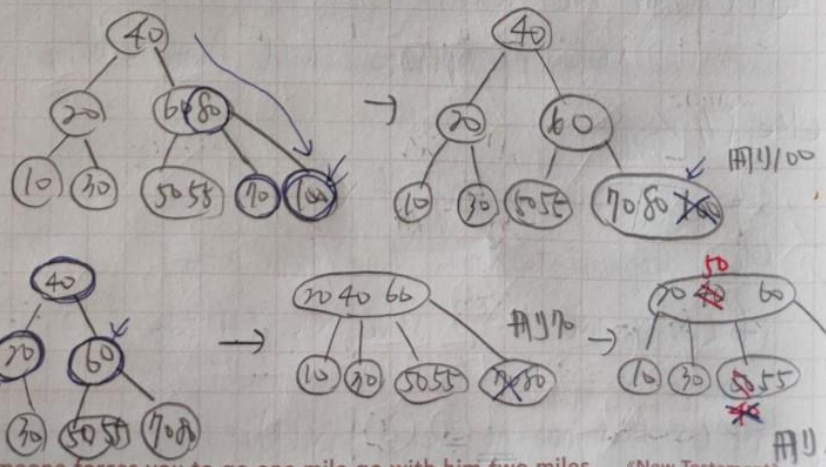
2、優化不用遞迴(比2-3 tree好), 往下只要遇到3個item的節點就先分裂 — 節省新增時間; 但樹高變高, search會變慢

3. always balance, 樹高一樣

Insert: Splits occur only at the path from the root to leaf, finally at leaf add item (downward)
(No upward recursion is needed!)

Delete: Transformation (Merge) occur only at the path from root to leaf (downward) → 此節點只有1個item就Merge
(No upward recursion is needed!)

Ex: Delete 100、70、40



If someone forces you to go one mile, go with him two miles. 《New Testament》

Problems & Difficulties needing explanation

weather report

AVL tree

1. 樹高可能差1, balance

Insert : ① +2, 0、+1 rotateLL( )

② +2, -1 rotateLR( )

③ -2, 0, -1 rotateRR( )

④ -2, +1 rotateRL( )

Delete : 此節點, if 只有一子節點, 將其往上提

else 有二子節點, 找 in-order successor 往上提

再往回依序檢查是否平衡, 不平衡則"旋轉"!

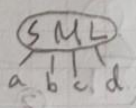
My Opinions

Thoughts, inspirations, and suggestions

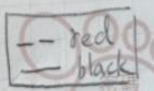
Red-black tree (一種 binary search tree)

1. has the advantage of a 2-3-4 tree, without the storage overhead
2. rotation like AVL tree [為了 balance]
3. Red-black tree 可对應至少一個 2-3-4 tree

↓

对應: 4-node 

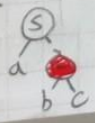
→ red



1 red 1 black

3-node 

⇒ 

or 

cipher key

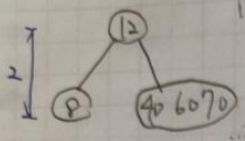
《馬太福音》 13

My Notes

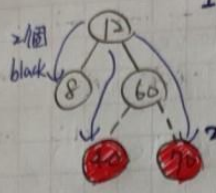
Important Concepts worth keeping

Today / /

对应性质



2-3-4 tree

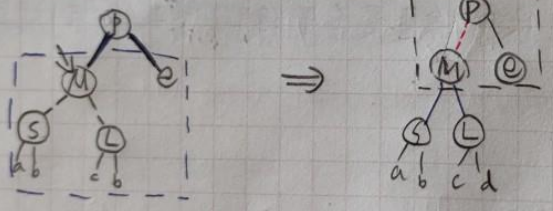


red-black tree

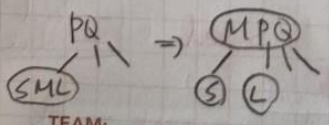
1. Every external path has an equal number of black pointers
2. black painter 数量 = 2-3-4 tree 的 height

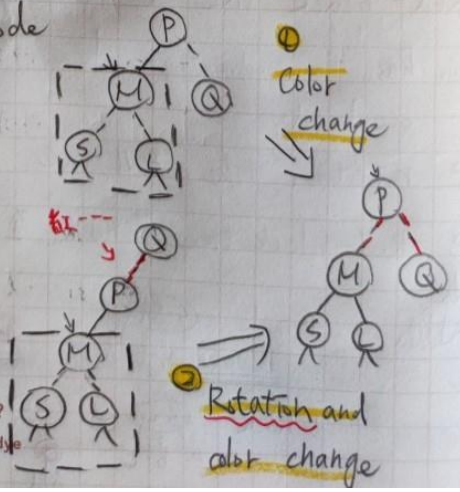
☆ Split on red black tree

1. Parant is a 2-node
   - only change the colors



2. Parant is a 3-node
   - Color change
   - rotation





② Rotation and color change

TEAM:
Together Everyone Argue and Mad ?
Together Each Achieve More ! —Sundyle

My Questions
Problems & Difficulties needing exploration

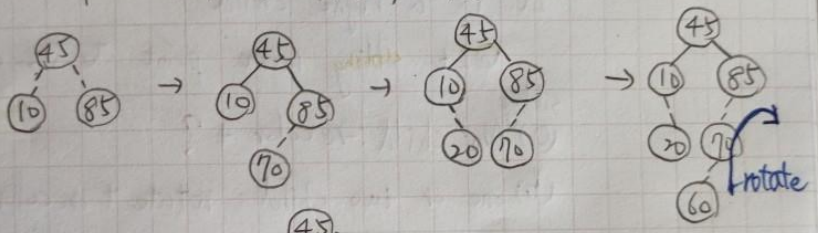
My learning weather report

Insert : 1. Splits of a node with two red pointers (like the 4-node in 2-3-4 tree) occur only on path from the root to leaf (downward!)

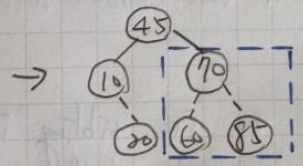
2. Set the pointer to a new-add node as red

3. Rotate if there are two consecutive(連續) red pointer

Example : 45, 85, 10, 70, 20, 60



My Opinions
Thoughts, inspirations, and suggestions



Delete : ① two childern → swap with the in-order successor

② one childern → pointed to by a black pointer (child指向)

③ leaf → ① pointed to by a red pointer
delete the leaf

② pointed to by a black pointer
★ 一定 have sibling

三種

1. Pointer to its parant is red
   It is a right child
   @ It sibling must point to its child by red
   sibling
   (a) No child : recolor
   (b) one or two child : rotate + recolor

2. Pointer to its parant is black
   It is a right child
   @ It sibling must point to its child by red
   sibling
   (a) No child : recolor + ?
   (b) one or two child : rotate + recolor

3. Pointer to its sibling is red
   It is a right child
   @ It sibling must have to black pointer
   @ LL rotation + recolor
   sibling
   (a) No child : ok
   (b) one child : rotate
   (d) Two child : rotate + recolor