Average Running Time

```
Rabin-Karp-Matcher(T,P,d,q
n=T.length
m=P.length
h=d^(m-1) mod q
p=0
t=0
for i=1 to m
    p=(dp+P[i]) mod q
    t=(dt+T[i]) mod q
for s=0 to n-m
    if p==t
        if P[1..m]==T[s+1..s+m]
            print "Pattern occurs with shift" s
    if s<n-m
        t=(d(t-T[s+1]h)+T[s+m+1]) mod q
```

平常的時候, valid shift很少
(假設有c個)
不會每次都有modulo的hit.
假設字串各種排列組合出現的機率相等
則spurious hit的機率可當成$1/q$.

Pre-processing: $O(m)$

則比對花的時間:
Spurious hit共花O((n-m+1)/q)=O(n/q)次
總共比對花的時間為
O((n-m+1)+(m(c+n/q)))

迴圈跑n-m+1次

If c=O(1) and q≥m, → O(n+m)=O(n)

Hit的時候比對: $O(m)$

算Prefix function花多少時間?

```
Compute-Prefix-Function(P)
m=P.length
let π[1..m] be a new array
π[1] = 0
k=0
for q=2 to m
    while k>0 and P[k+1]!=P[q]
        k=π[k]
    if P[k+1]==P[q]
        k=k+1
    π[q] = k
return π
```

進入迴圈的時候k<q, 且q每次增加, k有時候不增加
所以k<q永遠成立

所以$\pi[q] = k < q$.

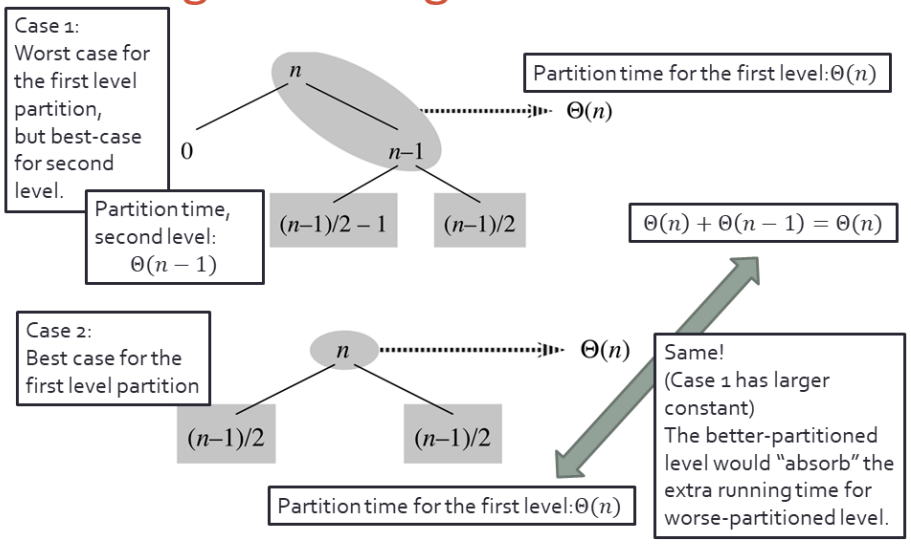
所以每執行一次迴圈
就減少k一次
且k永遠不是負的

k只會在這邊增加,
因此最多總共增加m-1次(迴圈執行次數)

Total: O(m)

最後: 既然有增加才有得減少, while loop總共執行的次數不會超過O(m)

Average running time



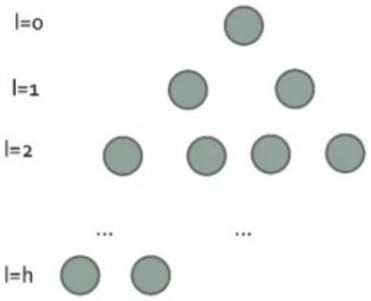
Some terms related to sorting

- **Stability:**
- If $a_i = a_j$ (equal key value) , then they maintain the same order before and after sorting.

- **In-place:**
- Directly sort the keys at **their current memory locations**. Therefore, only O(1) additional space is needed for sorting.

- **Adaptability:**
- If **part of the sequence is sorted**, then the time complexity of the sorting algorithm reduces.

Time Complexity: Trial 2



- 所花的時間為: $O(n)$

$$2^0 h + 2^1(h-1) + 2^2(h-2) + \cdots + 2^{h-1} \cdot 1$$

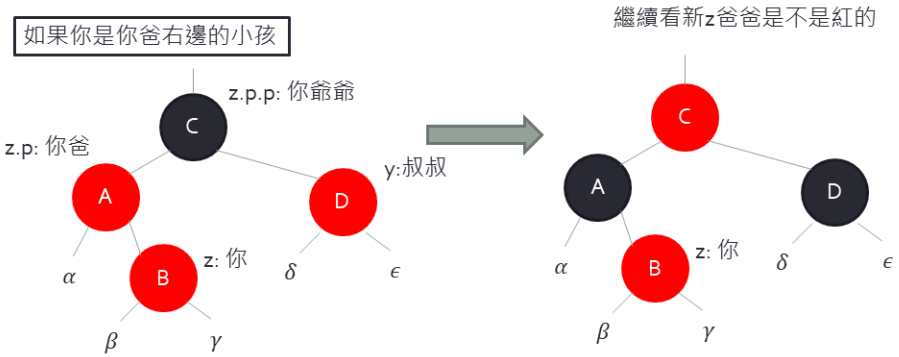
- $= \sum_{i=0}^{h} 2^i (h-i)$
- Let $S = \sum_{i=0}^{h} 2^i (h-i)$.
- $2S = 2h + 4(h-1) + \cdots + 2^h$
- $2S - S = -h + 2 + 4 + \cdots + 2^h$
- $S = 2^{h+1} - h - 2$

$= 2 \cdot 2^{O(\log n)} - O(\log n) - 2$

$= O(n)$ ~~$- O(\log n) - 2$~~

You can also see a different proof on p.159 of Cormen.

情形一：你的叔叔是紅的



注意看每一個步驟是否有保持紅黑樹第五個條件:
從C開始走到α~ ϵ任一個分支所經過的黑色node數目在修改前後必須相同!

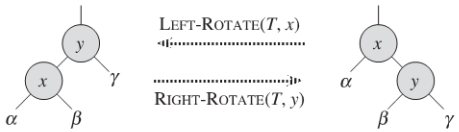
紅黑樹

- 每個node都分配一個顏色: 紅或黑
- 沒有children的地方都補上external node, 又叫NIL

- 規則們:
- 1. 每個node不是黑就是紅
- 2. root是黑色的
- 3. 每個leaf (external node, or NIL)都是黑色
- 4. 如果一個node是紅的, 則它的兩個小孩都是黑的
- 5. 對每個node來說, 從它到它的所有子孫葉子node的路徑上含有**一樣數目的黑色node** (不包含他自己)

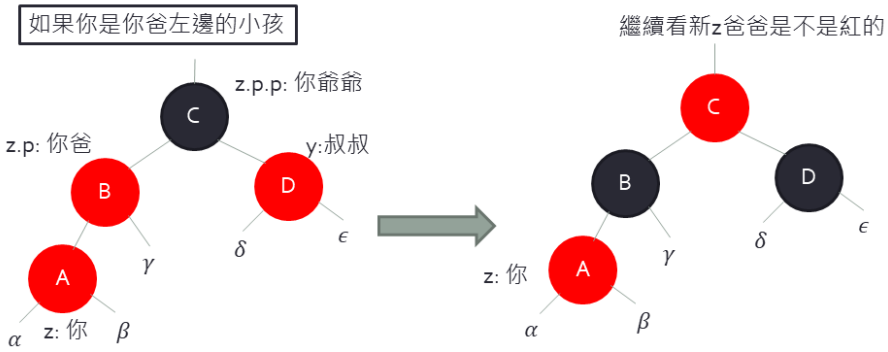
Pseudo-code: Left-Rotate(T,x)

```
LEFT-ROTATE(T, x)
1   y = x.right
2   x.right = y.left
3   if y.left ≠ T.nil
4       y.left.p = x
5   y.p = x.p
6   if x.p == T.nil
7       T.root = y
8   elseif x == x.p.left
9       x.p.left = y
10  else x.p.right = y
11  y.left = x
12  x.p = y
```



Can you write the pseudo-code for Right-Rotate(T,y)?

情形一：你的叔叔是紅的



注意看每一個步驟是否有保持紅黑樹第五個條件:
從C開始走到α~ϵ任一個分支所經過的黑色node數目在修改前後必須相同!

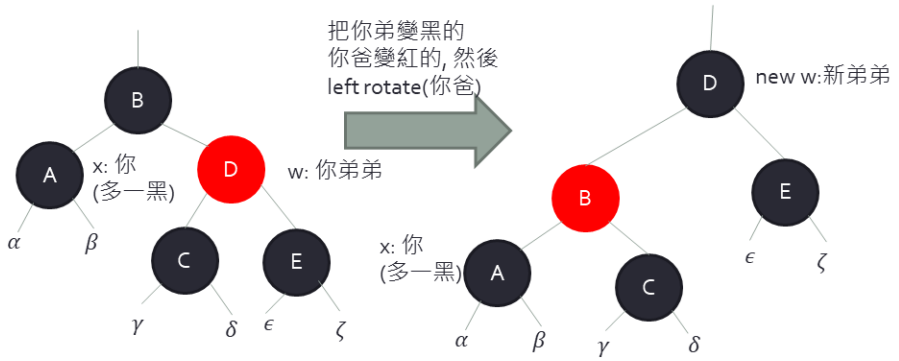
Pseudo-code：插入後修理R-B tree

```
RB-Insert-Fixup(T,z)
While z.p.color==RED  (如果你爸是紅色的)
      if z.p==z.p.p.left (如果你爸是你爺爺左邊的小孩)
            y=z.p.p.right (那麼叔叔就是你爺爺右邊的小孩)
            if y.color==RED (如果叔叔是紅色的)
                  z.p.color=BLACK (就把爸爸設成黑色的)
                  y.color=BLACK (叔叔也設成黑色的)
                  z.p.p.color=RED (爺爺設成紅色的)
                  z=z.p.p (把自己變成爺爺)
            else if z==z.p.right (如果你是你爸右邊的小孩)
                        z=z.p (把自己變成你爸)
                        LEFT-ROTATE(T,z)
                  z.p.color=BLACK (把你爸變成黑色)
                  z.p.p.color=RED (把你爺爺變成紅色)
                  RIGHT-ROTATE(T,z.p.p)
      else if z.p==z.p.p.right (如果你爸是你爺爺右邊的小孩,
                                    跟前面一樣, 只是left, right都反
過來) ……

T.root.color=BLACK (最後把root改成黑色)
```

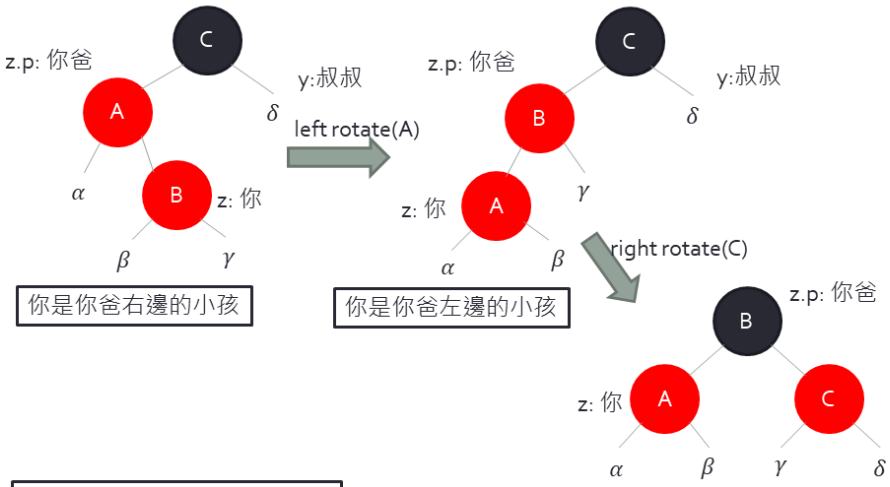
情形1　情形2　情形3　情形4-6: 鏡射的狀況

情形一：你的弟弟是紅的



轉換成情形二、三、或四

情形二與三：你的叔叔是黑的



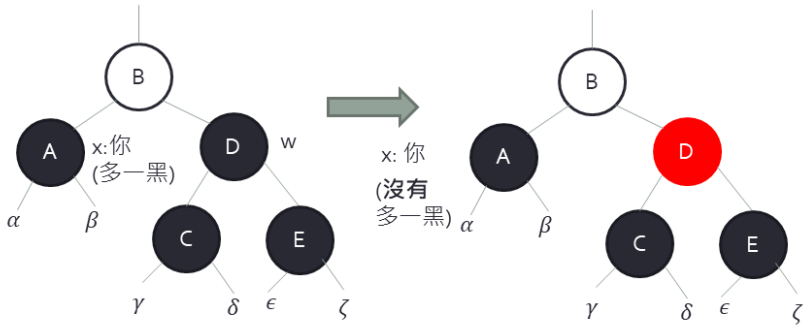
你是你爸右邊的小孩

你是你爸左邊的小孩

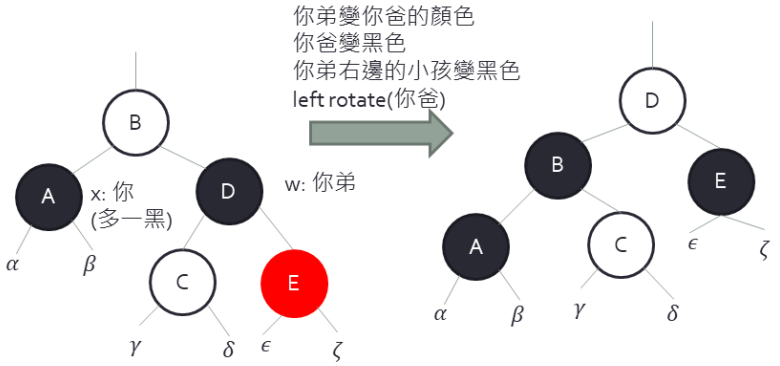
注意看每一個步驟是否有保持紅黑樹第五個條件

需要繼續往上層看嗎? 不用! 因為B還是黑的!

情形二：你的弟弟是黑的&
你的姪子們都是黑的



情形四:你的弟弟是黑的&
你的右邊姪子(弟弟的右邊小孩)是紅的

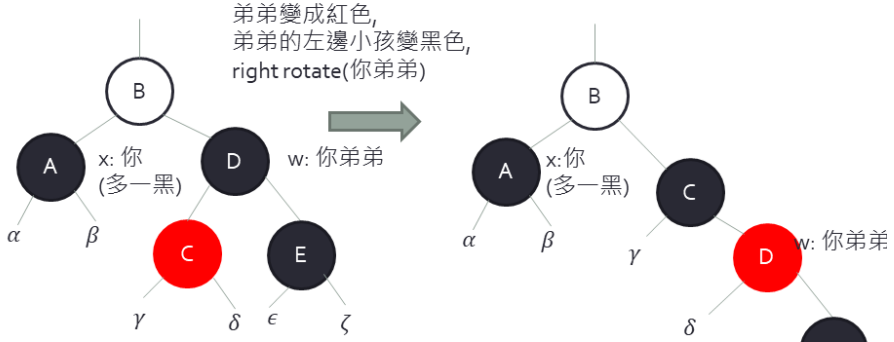


Pseudo-code: 刪除後修理R-B tree

```
RB-Delete-Fixup(T,x)
while x!=T.root  && x.color==BLACK  (你不是root而且是黑的)
        if (x==x.p.left)  (你是你爸的左邊小孩)
                w=x.p.right  (你弟就是你爸右邊小孩)
                if w.color==RED  (如果你弟是紅色的)
                        w.color=BLACK  (把你弟設成黑色)
                        x.p.color=RED   (你爸設成紅色)
                        LEFT-ROTATE(T,x.p)
                        w=x.p.right      (你新弟弟是現在你爸右邊小孩)
                if w.left.color==BLACK  && w.right.color==BLACK  (兩個姪子都黑)
                        w.color=RED  (把弟弟設成紅的)
                        x=x.p  (你變成你爸)
                else if w.right.color==BLACK  (你姪子右黑左紅)
                                w.left.color=BLACK   (左邊姪子設成黑的)
                                w.color=RED  (你弟設成紅的)
                                RIGHT-ROTATE(T,w)
                                w=x.p.right  (新弟弟是現在你爸右邊小孩)
                        w.color=x.p.color  (弟弟設射程爸爸的顏色)
                        x.p.color=BLACK  (爸爸設成黑的)
                        w.right.color=BLACK  (右邊姪子設成黑的)
                        LEFT-ROTATE(T,x.p)
                        x=T.root  (你直跳root,  準備出去)
        else (x==x.p.right)  (如果你是你爸右邊的小孩,跟前面一樣,  只是left,  right都反過來)
x.color=BLACK  (如果跳出迴圈了,  也就是x是紅加黑,  或者是root是黑加黑,  那麼就把它設成一個黑色即可)
```

情形1
情形2
情形3
情形4
情形5-8

情形三：你的弟弟是黑的&
你的姪子們(弟弟的小孩)是左紅右黑



變成情形 4