13.1-5

Show that the longest simple path from a node x in a red-black tree to a descendant leaf has length at most twice that of the shortest simple path from node x to a descendant leaf.

C24106082
陳宏彰

let longest simple path $(a_1, a_2, a_3, \ldots, a_l)$, length $= l$

shortest simple path $(b_1, b_2, b_3, \ldots, b_s)$, length $= S$

by property 5, they have the same numbers of black nodes and there are no repeated red nodes.

$\Rightarrow$ the are at most $\lfloor \frac{l-1}{2} \rfloor$ red node in $a_1 \sim a_l$

and at least $\lceil \frac{l+1}{2} \rceil$ black nodes

$\overset{\text{prop. 5}}{\Longrightarrow} S \geq \lceil \frac{l+1}{2} \rceil \Rightarrow \underline{2S \geq l}$ #

13.1-6

What is the largest possible number of internal nodes in a red-black tree with black-height k? What is the smallest possible number?

At most, there are one red node between two black nodes,

$\Rightarrow h_{max} = 2k+1$

$\Rightarrow$ number of internal nodes $= 2^{2k+1} - 1$

At smallest: all nodes are black

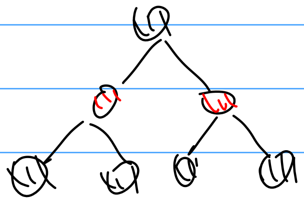
$\Rightarrow h_{min} = k+1$

$\Rightarrow$ number of internal nodes $= 2^{k+1} - 1$

13.1-7

Describe a red-black tree on n keys that realizes the largest possible ratio of red internal nodes to black internal nodes. What is this ratio? What tree has the smallest possible ratio, and what is the ratio?

Most : red and black nodes are alternative



$\Rightarrow \frac{2}{3}$ #

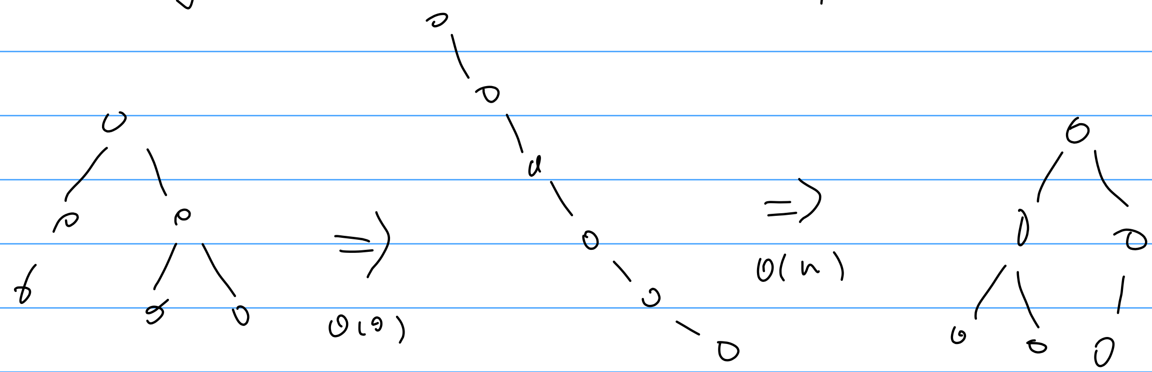
Smallest : All nodes are black $\Rightarrow 0$ #

13.2-4

Show that any arbitrary n-node binary search tree can be transformed into any other arbitrary n-node binary search tree using $O(n)$ rotations. (*Hint:* First show that at most $n-1$ right rotations suffice to transform the tree into a right-going chain.)

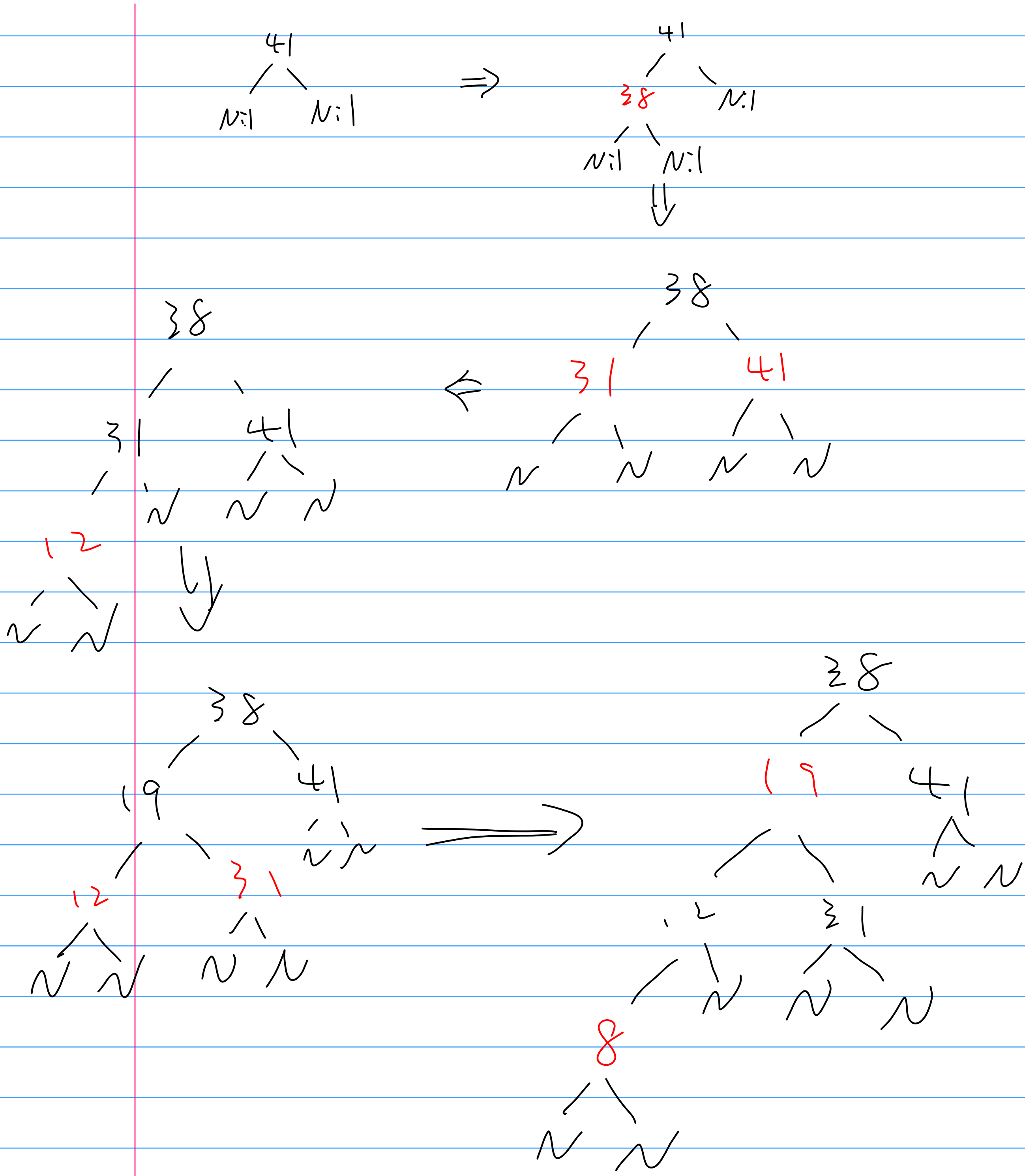
First convert the tree into a right-going chain, which takes at most n times of right rotate $\Rightarrow O(n)$

Then use at most n times left rotate to convert the right-going chain into anothor binary search trees $\Rightarrow O(n)$



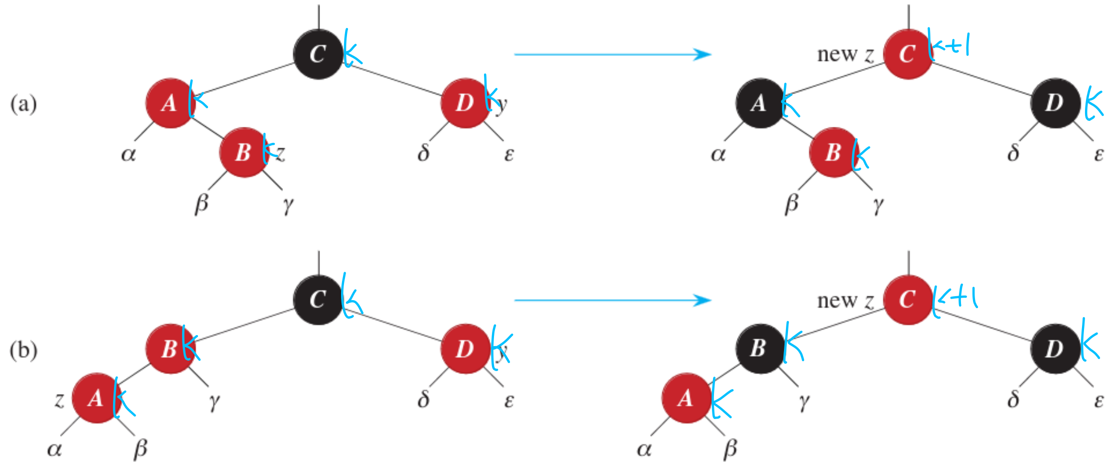
13.3-2

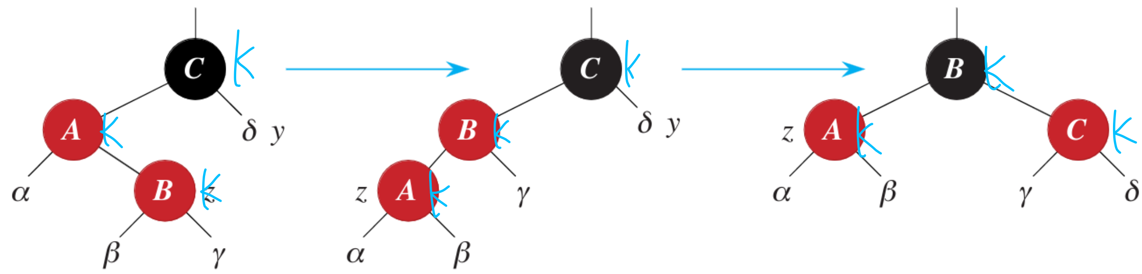
Show the red-black trees that result after successively inserting the keys 41, 38, 31, 12, 19, 8 into an initially empty red-black tree.



13.3-3

Suppose that the black-height of each of the subtrees $\alpha, \beta, \gamma, \delta, \varepsilon$ in Figures 13.5 and 13.6 is k. Label each node in each figure with its black-height to verify that the indicated transformation preserves property 5.





Case 2 Case 3

14.1-1

Show that equation (14.4) follows from equation (14.3) and the initial condition $T(0) = 1$.

$T(n) = 2^n$

when $n=0$ $T(0) = 2^0 = 1$

Assume $T(n-1) = 2^{n-1}$

$$T(n) = 1 + \sum_{j=0}^{n-1} T(j)$$

$$\overset{T(n)=2^n}{=} 1 + \sum_{j=0}^{n-1} 2^j$$

$$= 1 + (2^n - 1)$$

$$= 2^n$$

by induction, $T(n) = 2^n$ #

14.1-6

The Fibonacci numbers are defined by recurrence (3.31) on page 69. Give an $O(n)$-time dynamic-programming algorithm to compute the nth Fibonacci number. Draw the subproblem graph. How many vertices and edges does the graph contain?

$$F_i = \begin{cases} 0 & , i = 0 \\ 1 & , i = 1 \\ F_{i-1} + F_{i-2} & , i \geq 2 \end{cases}$$

```
F(n) {
    f[n+1] = {0, 1, }
    for (i=2; i ≤ n ; i++) {
        f[i] = f[i-1] + f[i-2]
    }
    return f[n]
}
```

14.2-1

Find an optimal parenthesization of a matrix-chain product whose sequence of dimensions is $\langle 5, 10, 3, 12, 5, 50, 6 \rangle$.

$5, 3$ $\quad\quad$ $3, 6$

$$((A_1 \; A_2)((A_3 \; A_4)(A_5 \; A_6)))$$

$$5 \times 10 \times 3 + 3 \times 12 \times 5 + 5 \times 50 \times 6$$

$$+ 3 \times 5 \times 6$$

$$+ 5 \times 3 \times 6$$

$$= 2010 \; \#$$

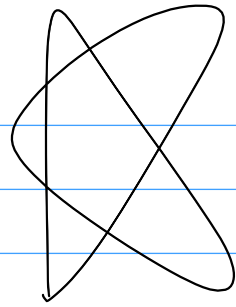
14.2-5

Let $R(i,j)$ be the number of times that table entry $m[i,j]$ is referenced while computing other table entries in a call of MATRIX-CHAIN-ORDER. Show that the total number of references for the entire table is

$$\sum_{i=1}^{n}\sum_{j=i}^{n} R(i,j) = \frac{n^3 - n}{3} .$$

(*Hint:* You may find equation (A.4) on page 1141 useful.)

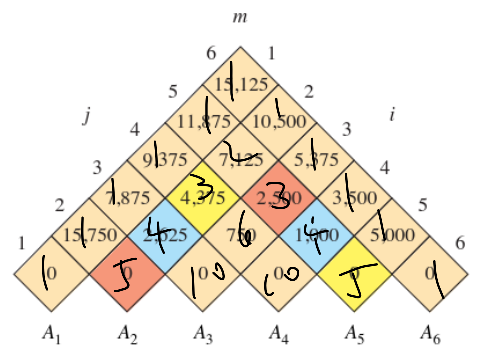
$$\sum_{k=0}^{n} k^2 = \frac{n(n+1)(2n+1)}{6} ,$$



$R(i,j) : \{$

$$\frac{n(n^2-1)}{3} = \frac{n(n+1)(n-1)}{3}$$

$$= \frac{n(n+1)(2n-2)}{6}$$



$$\left(\frac{i}{j}\right)$$

14.3-3

Consider the antithetical variant of the matrix-chain multiplication problem where the goal is to parenthesize the sequence of matrices so as to maximize, rather than minimize, the number of scalar multiplications. Does this problem exhibit optimal substructure?

Assume there are n matrix $A_1, A_2, A_3, , A_n$ and split it between A_k and A_{k-1} and the number of scalar multiplication reach maximum, We can must find maximum in $A_1 \sim A_k$ and $A_{k+1} \sim A_n$.
So this problem exhibit optimal substructure.

14.3-4

As stated, in dynamic programming, you first solve the subproblems and then choose which of them to use in an optimal solution to the problem. Professor Capulet claims that she does not always need to solve all the subproblems in order to find an optimal solution. She suggests that she can find an optimal solution to the matrix-chain multiplication problem by always choosing the matrix A_k at which to split the subproduct $A_i A_{i+1} \cdots A_j$ (by selecting k to minimize the quantity $p_{i-1} p_k p_j$) before solving the subproblems. Find an instance of the matrix-chain multiplication problem for which this greedy approach yields a suboptimal solution.

$5 \quad 3 \quad 2 \quad 1 \quad 4$

greedy $A_1((A_2 A_3) A_4)$ $3 \times 2 \times 1 + 3 \times 1 \times 4 + 5 \times 3 \times 4 = 78$

$((A_1 A_2) A_3) A_4$ $5 \times 3 \times 2 + 5 \times 2 \times 1 + 5 \times 1 \times 4 = 60$

this takes fewer steps than Pf. Capulet's method

14.4-1

Determine an LCS of $\langle 1, \cancel{0, 0}, 1, 0, 1, 0, 1 \rangle$ and $\langle 0, 1, 0, 1, \cancel{1}, 0, \cancel{1, 1}, 0 \rangle$.

0

$\Rightarrow$ 101010 or 010101 #

14.4-4

Show how to compute the length of an LCS using only $2 \cdot \min\{m, n\}$ entries in the c table plus $O(1)$ additional space. Then show how to do the same thing, but using $\min\{m, n\}$ entries plus $O(1)$ additional space.

Assume $m < n \Rightarrow \min(m, n) = m$

if not, exchange X and Y

Because we only need to compute the length but no entire path and we only need the previous row to compute the current row.

When computing k row, tree k-2 row and allocate k+1 row,

In this way, we only use two rows at any time

$\Rightarrow$ space $= 2 \times \min(m, n) + O(1)$

```
LCS-LENGTH(X, Y, m, n)
let b[1 : m, 1 : n] and c[0 : m, 0 : n] be new tables
for i = 1 to m
    c[i, 0] = 0
for j = 0 to n
    c[0, j] = 0
for i = 1 to m          // compute table entries in row-major order
    for j = 1 to n
        if x_i == y_j
            c[i, j] = c[i - 1, j - 1] + 1
            b[i, j] = "↖"
        elseif c[i - 1, j] ≥ c[i, j - 1]
            c[i, j] = c[i - 1, j]
            b[i, j] = "↑"
        else c[i, j] = c[i, j - 1]
            b[i, j] = "←"
return c and b
```

allocate and init the first row of c

if i > 1 { tree c[i-2]; allocate c[i+1] }

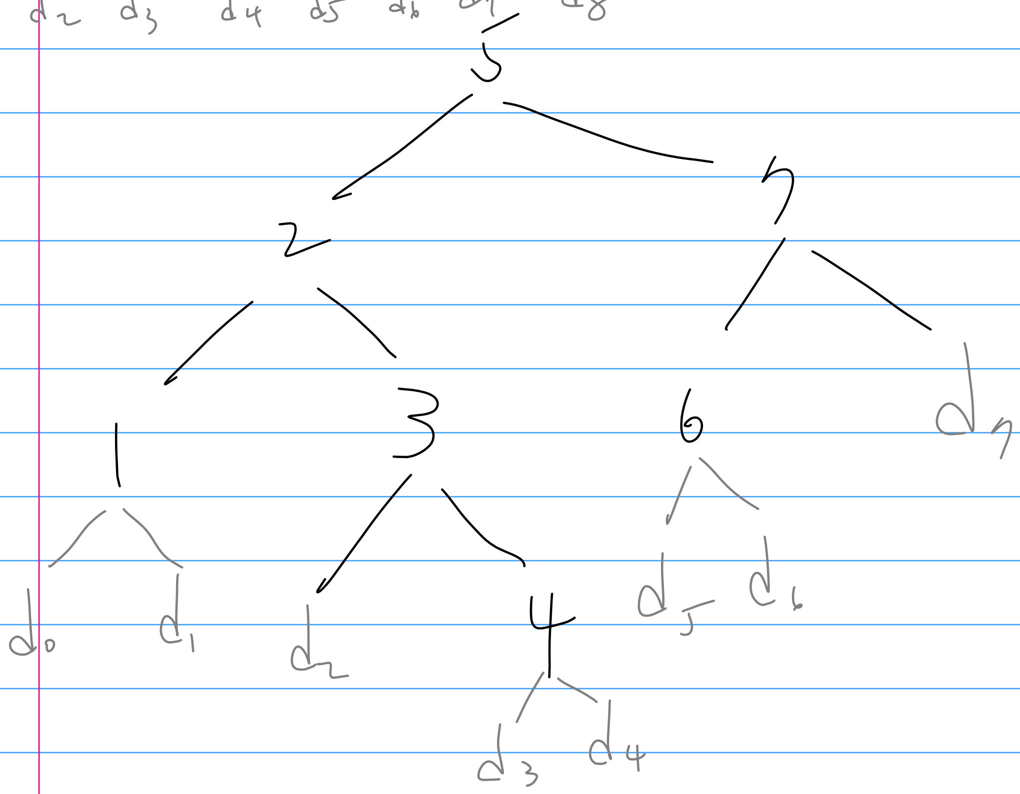
In advance, when we calulate c[i, j], what we need is c[i-1, j] and c[i-1, j-1], so we can only use one row to finish the calculation, which reduce the space to min(m, n) + O(1)

14.5-2

Determine the cost and structure of an optimal binary search tree for a set of $n = 7$ keys with the following probabilities:

i	0	1	2	3	4	5	6	7
p_i		0.04	0.06	0.08	0.02	0.10	0.12	0.14
q_i	0.06	0.06	0.06	0.06	0.05	0.05	0.05	0.05

d_1 d_2 d_3 d_4 d_5 d_6 d_7 d_8



node	depth	prob.	contrib.
1	2	0.04	0.12
2	1	0.06	0.12
3	2	0.08	0.24
4	3	0.02	0.08
5	0	0.10	0.10
6	2	0.12	0.36
7	1	0.14	0.28
d_0	3	0.06	0.24
d_1	3	0.06	0.24
d_2	3	0.06	0.24
d_3	4	0.06	0.30
d_4	4	0.05	0.25
d_5	3	0.05	0.20
d_6	3	0.05	0.20
d_7	2	0.05	0.15

3.12 #