Announcements

- HW#2 will be due on 23:59 4/27.
- We will have a midterm on 4/18 (next Tuesday).
- The midterm will take place at S101 and S102.

Review

- Dynamic programming
  - Rod cutting
  - Matrix chain multiplication
  - Longest common subsequence

BCT

4/11 (二)
START

Edit Distance

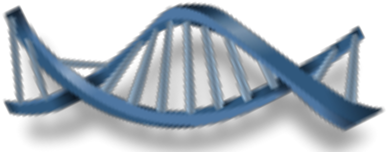
Q 要算出最少需要幾個operation可以把X改成Y？
這個最少的步驟就是 X & Y 的 edit distance

- Given the following operations, the edit distance from X and Y is the minimal number of operations that transform X to Y.

可以選三個指令

  - **Replace:** set x_i to some other character
  - **Insert:** add a character into the sequence
  - **Delete:** remove a character into the sequence

X = GATCG
Y = CAATG
Edit distance (X, Y) = ?



Example (1)

Operations:
- Replace
- Insert
- Delete

C A

X = ~~G~~ A T ~~C~~ G

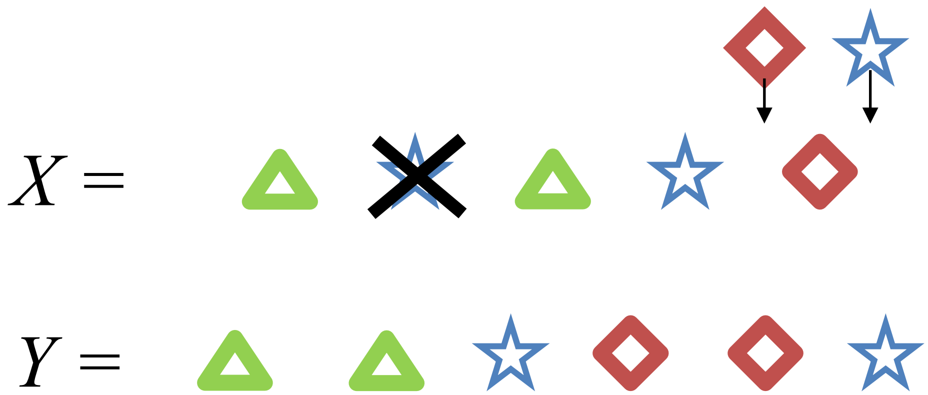
Y = C A A T G

Edit distance (*X*, *Y*) = 3

1. Replace G with C
2. Insert A
3. Delete C

一個取代
一個插入
一個刪除
3步

Example (2)



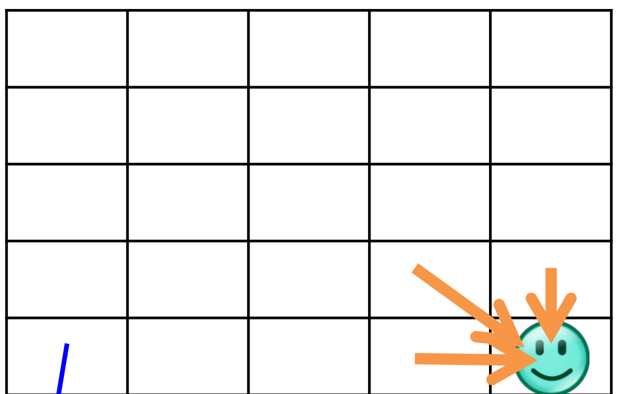
Edit distance $(X, Y) = 3$

1. Delete ☆
2. Insert ◇
3. Insert ☆

一個刪除
一個插入
一個插入
3步

Computing the edit distance

- Recurrence? (大問題解與小問題解的關係?)



相當於把整串測資都算過，所以答案在右下角

stores the edit distance
between two prefix strings

d(GATCG, CAATG) = 3

d(GATC, CAAT) = 3

d(GATC, CAATG) = $\cancel{4}$ 3

d(GATCG, CAAT) = 4

$d[i, j]$ vs. $d[i\text{-}1, j\text{-}1]$

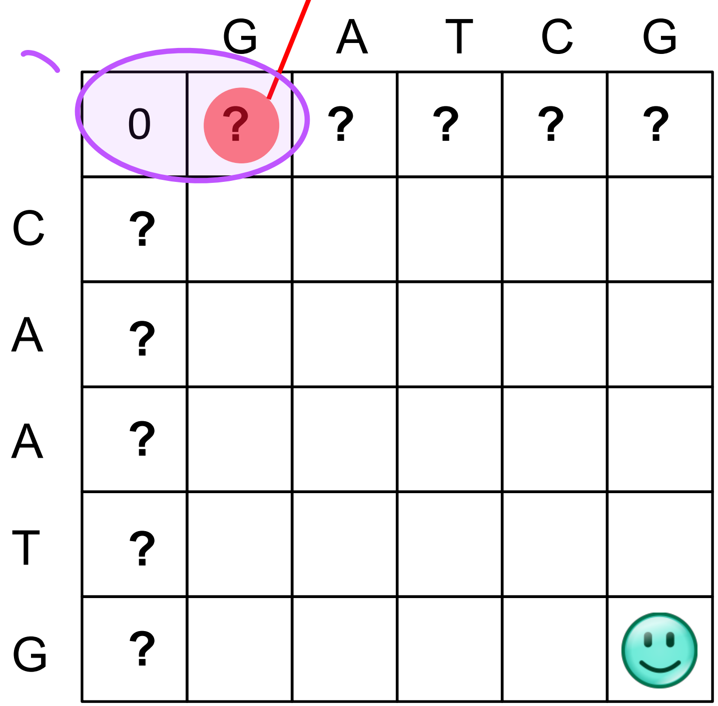
$d[i\text{-}1, j]$

$d[i, j\text{-}1]$

$d_{edit}(G, \{\}) = 1$

需要填入初始值

空的字串跟G
的edit distance 是1



從空字串到變成G A 需要兩步驟
所以edit distance 是 2
（他不是從G 變到A，他是從空的0 變到A）

		G	A	T	C	G
	0	1	2	3	4	5
C	1					
A	2					
A	3					
T	4					
G	5					

$d[i, j]$ vs. $d[i-1, j-1]$

$d[i-1, j]$

$d[i, j-1]$

$$d(i,j)=\begin{cases} i & \text{if } j=0 \\ j & \text{if } i=0 \\ \min(\, d(i-1,j-1)+(x_i \neq y_j),\, d(i-1,j)+1,\, d(i,j-1)+1) & \text{otherwise} \end{cases}$$

從左上角過來　從左邊過來　從上面過來

從上面和從左邊過來都需要2個步驟
從左上過來只需要1個步驟
選最少的，故為1

		G	A	T	C	G
	0	1	2	3	4	5
C	1	1	2	3	3	4
A	2	2	1	2	3	4
A	3	3	2	2	3	4
T	4	4	3	2	3	4
G	5	4	4	3	3	3

d(GATCG, CAATG)

min　+1

檢查三個位置，選最小的值+1，
就會是右下角那格的數字。

如果兩個字母相同的情況，
就直接把左上角的數字抓下來即可。

暴力法的時間複雜度是O(3^n)

replace	insert
delete	you are here

令 A字串長度為 a, B字串長度為 b
worst case：O(ab)

$$d(i,j)=\begin{cases} i & \text{if } j=0 \\ j & \text{if } i=0 \\ \min(\,d(i-1,j-1)+(x_i \neq y_j),\, d(i-1,j)+1,\, d(i,j-1)+1) & \text{otherwise} \end{cases}$$

		△	☆	△	☆	◇
	0	1	2	3	4	5
△	1	0	1	2	3	4
△	2	1	1	1	2	3
☆	3	2	1	2	1	2
◇	4	3	2	2	2	1
◇	5	4	3	3	3	2
☆	6	5	4	3	3	3

Longest increasing subsequence

LIS problem

Given a sequence of numbers X = $x_1, x_2, \ldots, x_N$, find the longest increasing *subsequence* ($i_1, i_2, \ldots, i_k$) where numbers in the sequence increase.

找到一個最長的子序列，數字要是遞增的

5 2 8 6 3 6 9 7

Longest increasing subsequence

Given a sequence of numbers X = $x_1, x_2, \ldots, x_N$, find the longest increasing *subsequence* $(i_1, i_2, \ldots, i_k)$ where numbers in the sequence increase.

暴力解的時間複雜度是O(2^n)

5 2 8 6 3 6 9 7

不是唯一解

Computing LIS

- Let $L[i]$ be the length of the LIS ending at index i such that X[i] is the last element of the LIS.

X = 5 2 8 6 3 6 9 7

L = ?

Computing LIS

- Let $L[i]$ be the length of the LIS ending at index i such that X[i] is the last element of the LIS.

X = 5 2 8 6 3 6 9 7

存在L 表格的元素 L = 1 1 2 2 2 3 4 4

Computing LIS

- Let $L[i]$ be the length of the LIS ending at index i such that X[i] is the last element of the LIS.

1. 在 0~i 之間找一個最大的L[j]，拿來+1

2. L[j] 對應到的 x[j]值，不能大過我現在在的x[i]

※ 如果沒有第2個條件，的 L[j] 會找到 x=9 的 L[j]=4，這樣就不遞增了。

input $X = 5\ \ 2\ \ 8\ \ 6\ \ 3\ \ 6\ \ 9\ \ 7$

index 0 找 j i

額外開的 輔助array $L = 1\ \ 1\ \ 2\ \ 2\ \ 2\ \ 3\ \ 4\ \ 4$

L代表了這個數放在 LIS 裡會是第幾個位置

- Recurrence?

第 i 個的 L 在LIS中排第幾

在0~i之間找一個最大的L[j]拿來加1

L[j]對應到的X[j]要比我現在在的這個數字小

$L[i] = 1 + \max(L[j])$ where $0 < j < i$ and $X[j] < X[i]$

j 的位置要在 i 之前

or 1, if no such j exists.

上面那行沒有達成，就直接放1

Computing LIS

- Let *L*[*i*] be the length of the LIS ending at index *i* such that X[*i*] is the last element of the LIS.

LIS(x) = 4 ⇒ 2, 3, 6, 7 #

X = 5 2 8 6 3 6 9 7

這個L不會是遞增的，這裡只是巧合

L = 1 1 2 2 2 3 4 4

按照這個代表位置的號碼，照順序從4填回1

- Recurrence?

$L[i] = 1 + \max(L[i])$ where 0 < *j* < *i* and X[*j*] < X[*i*]
or 1, if no such *j* exists.

Another example of LIS

O(n^2)是來自，每個數字都要掃一次，掃完之後要再回頭看要排在誰後面。
兩個動作各要做n次，所以O(n^2)。

X = 50 3 10 7 40 80 8

L = 1 1 2 2 3 4 3

50 3

只有3比他小，3的L裡存的是1，再+1=2

BCT

$O(N^2)$ ✓

Q. 裡面的那層for loop能不能用更聰明的search 來降低他的時間複雜度變O(nlogn)？要怎麼做？

A. binary search的時間複雜度是O(logn)，做了n次，所以時間複雜度可以從O(n^2)降成O(nlogn)

Another solution

Can we use LCS to solve this problem?

x = 5 2 8 6 3 6 9 7

LCS O(n^2)

Sorted x = 2 3 5 6 6 7 8 9

找 X和 sorted X 的 LCS，就會得到LIS

如何利用 LCS 去解決 LIS 問題？

Another solution

Can we use LCS to solve this problem?

5 2 8 6 3 6 9 7

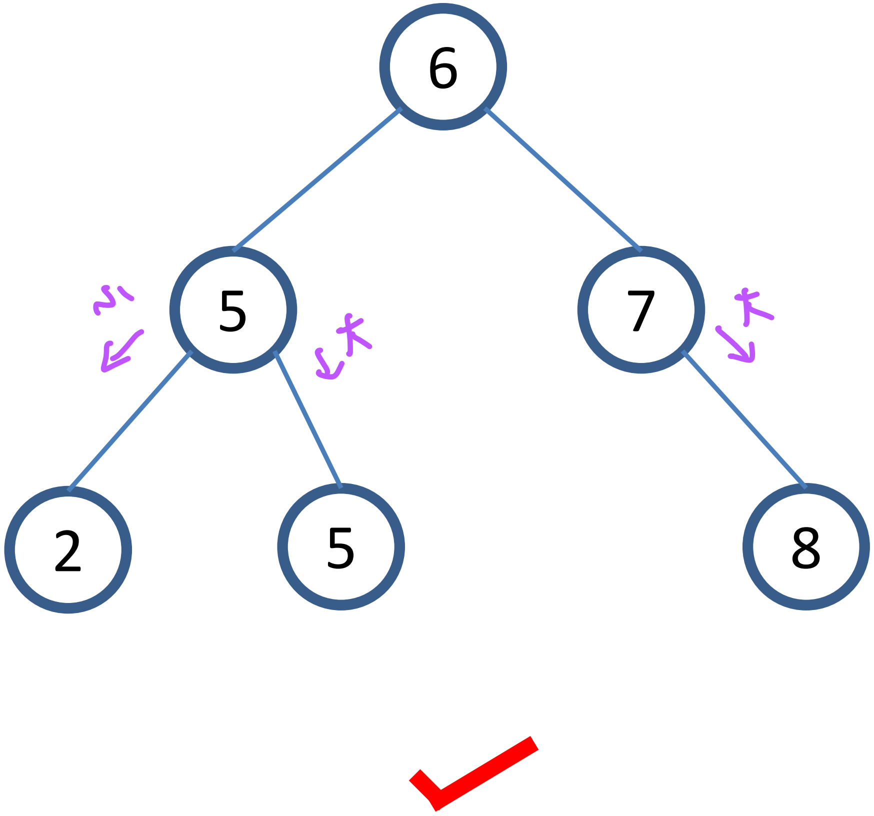
LCS

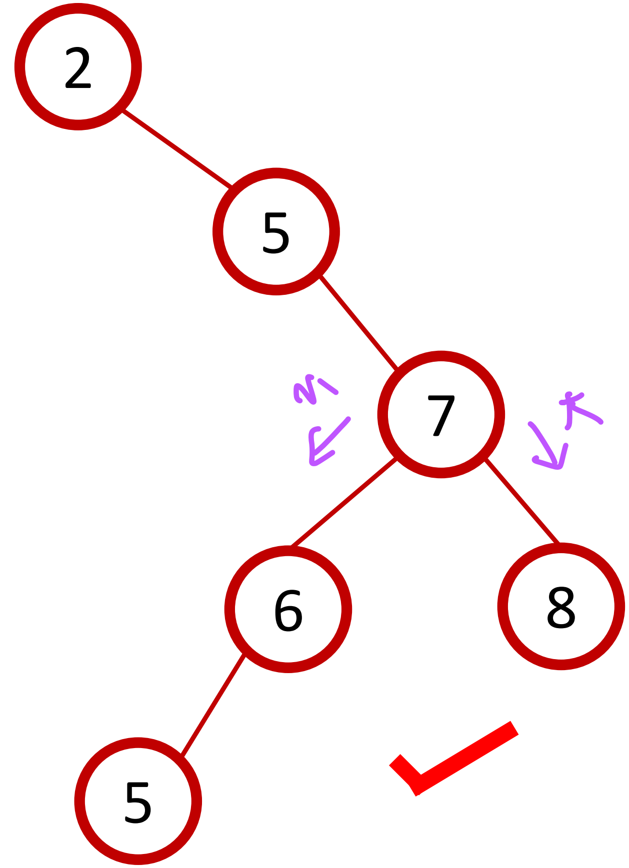
2 3 5 6 6 7 8 9

Optimal Binary Search Trees
Chapter 15.5

Mei-Chen Yeh

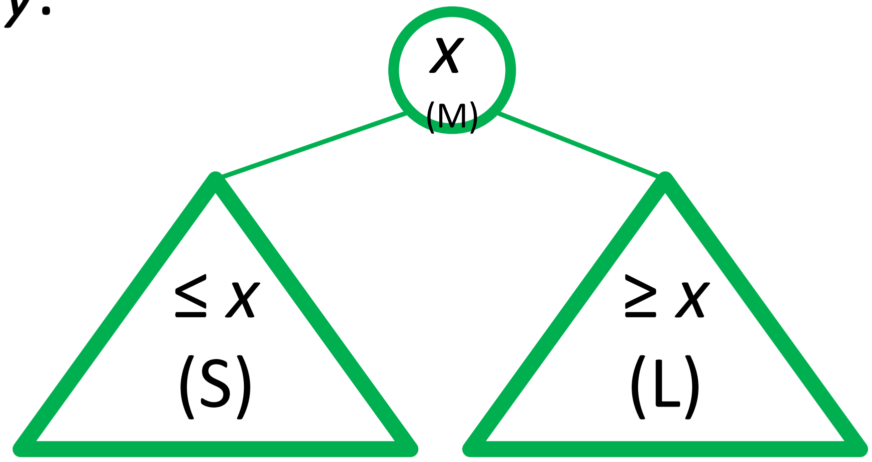
Binary search trees





Binary search trees

- Binary-search-tree property:
  - Let x be a node in a binary search tree.
  - If y is a node in the left subtree of x, then $y.key \leq x.key$.
  - If y is a node in the right subtree of x, then $y.key \geq x.key$.



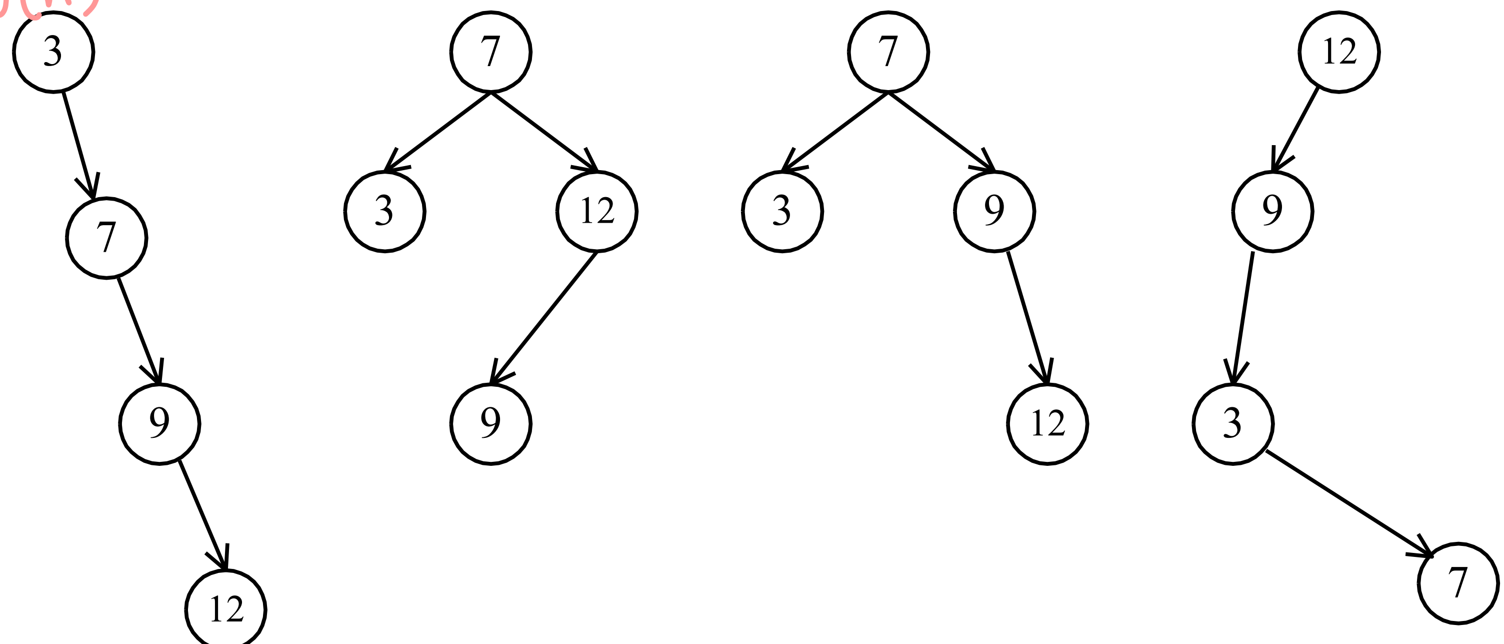
A *good* binary search tree?

會先看左邊能不能放，再看右邊

- Binary search trees for 3, 7, 9, 12

4個 key

$O(n^2)$



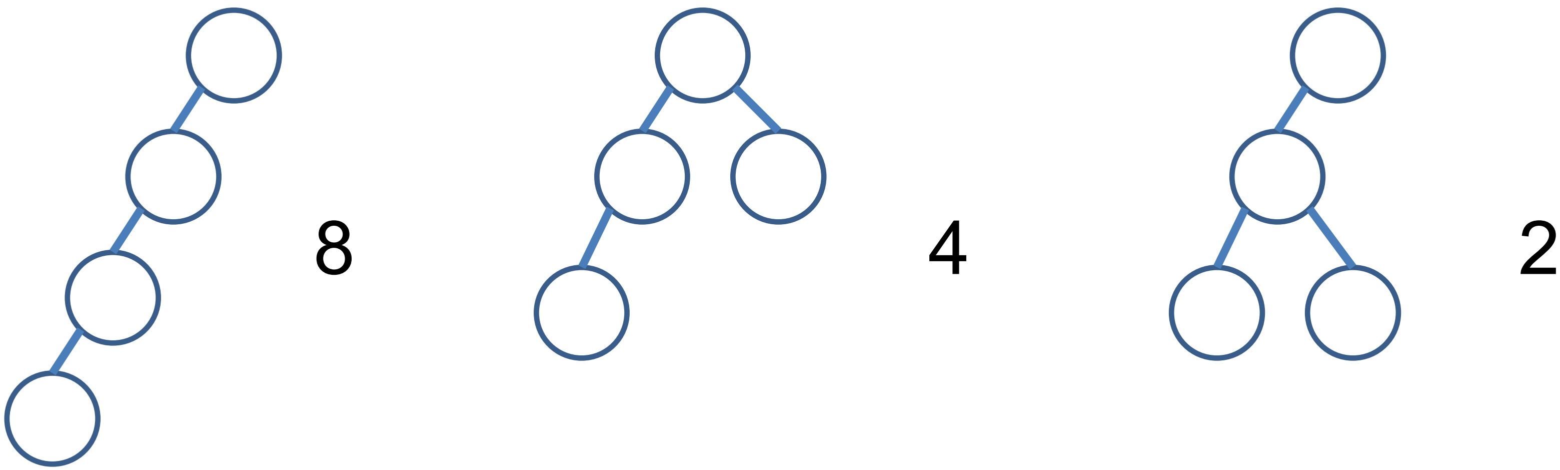
(a) (b) (c) (d)

How many trees in total?

Which one you prefer?

Binary trees for four nodes

存4 個值，會有14 種可能性（8+4+2）

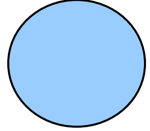


The total number of binary trees with *n* nodes is

$$c(n) = \frac{1}{n+1}\binom{2n}{n} = 4^n / n^{1.5}$$

大部分要造binary tree 需要的時間是O(nlogn)
最糟可能需要O(n^2)

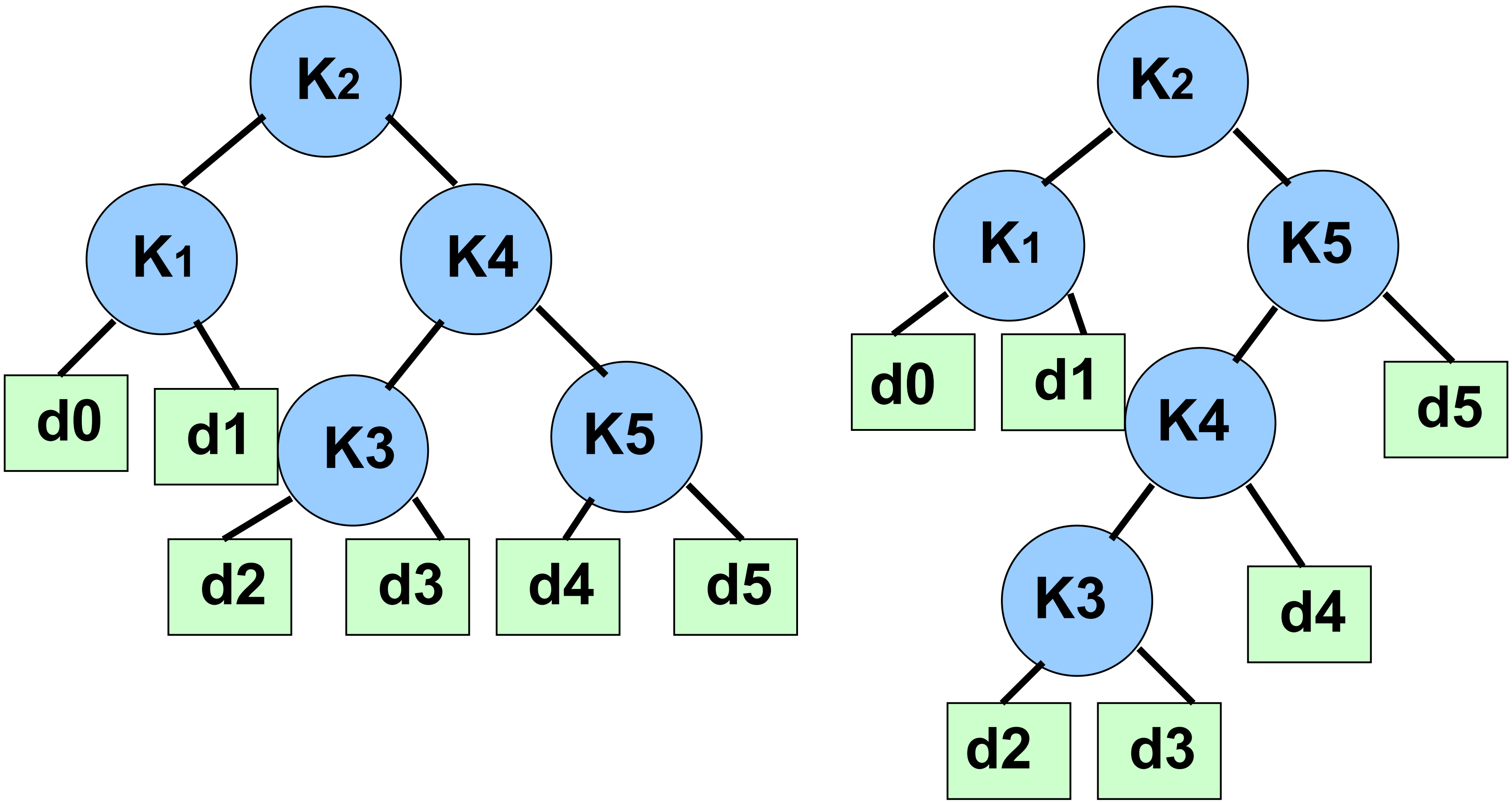
樹的種類數量跟節點的個數是呈指數成長



看到dummy keys 就代表到底了

Keys and dummy keys

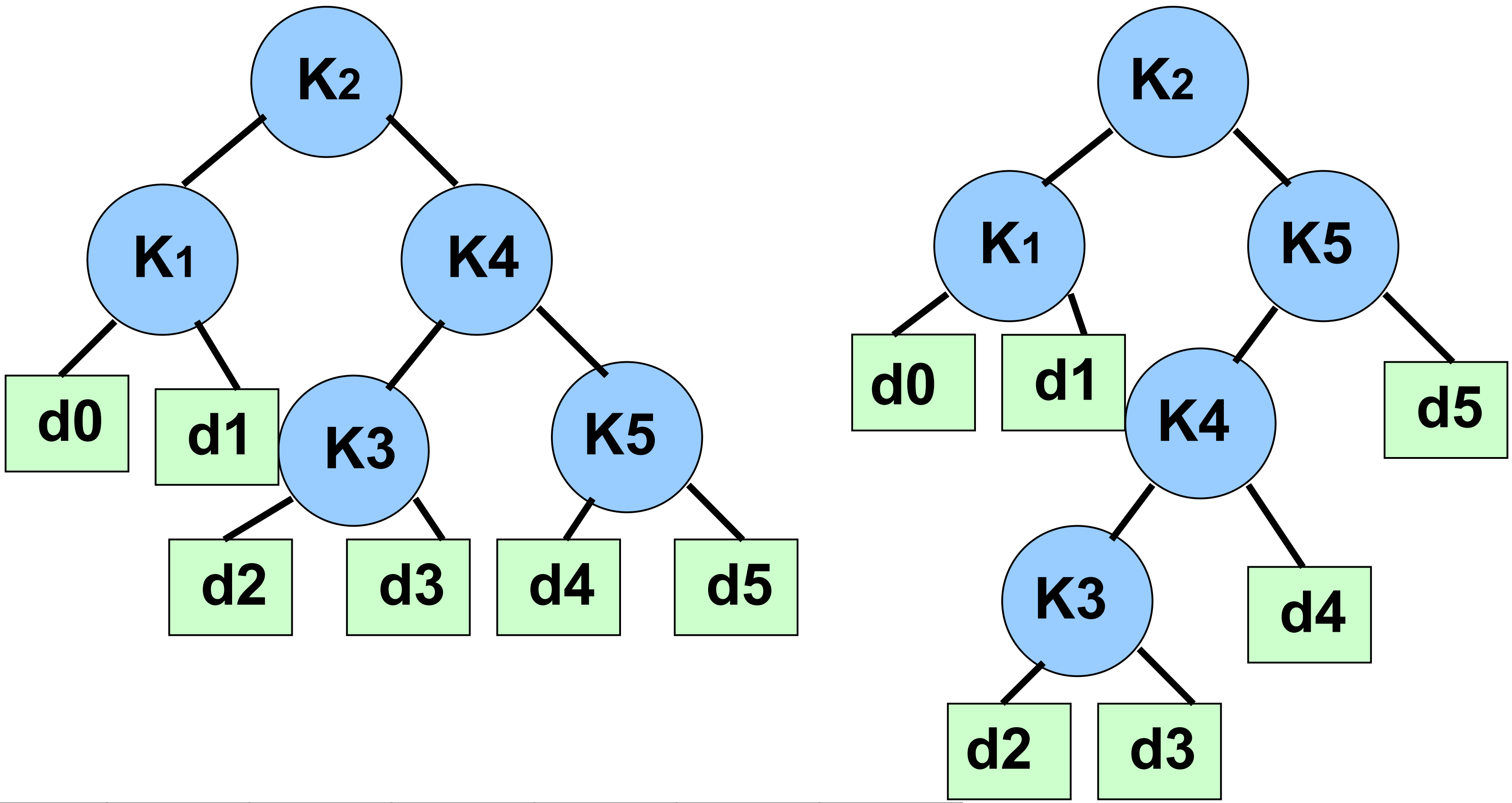
藍色的點代表數列裡的所有數字



就算樹形狀不同，5個節點都會有6個dummy keys

綠色的格子（dummy keys）代表，你要找的數字，在這個數列裡沒有，你看到我，你也不用找了，就是沒有

- p_i
  - the probability of searching for the key k_i
- q_i
  - the probability of searching for the key d_i



i	0	1	2	3	4	5
p_i key		0.15	0.1	0.05	0.1	0.2
dummy key q_i	0.05	0.1	0.05	0.05	0.05	0.1

$$\sum_{i=1}^{n} p_i + \sum_{i=0}^{n} q_i = 1$$

把所有節點被拜訪的機率都加起來會是1

機率表顯示每一個key和每一個dummy key，被拜訪的機率是多少

Optimal binary search trees

- Given that we know how often each key occurs, how do we build a binary search tree so as to *minimize the number of nodes visited in all searches*?

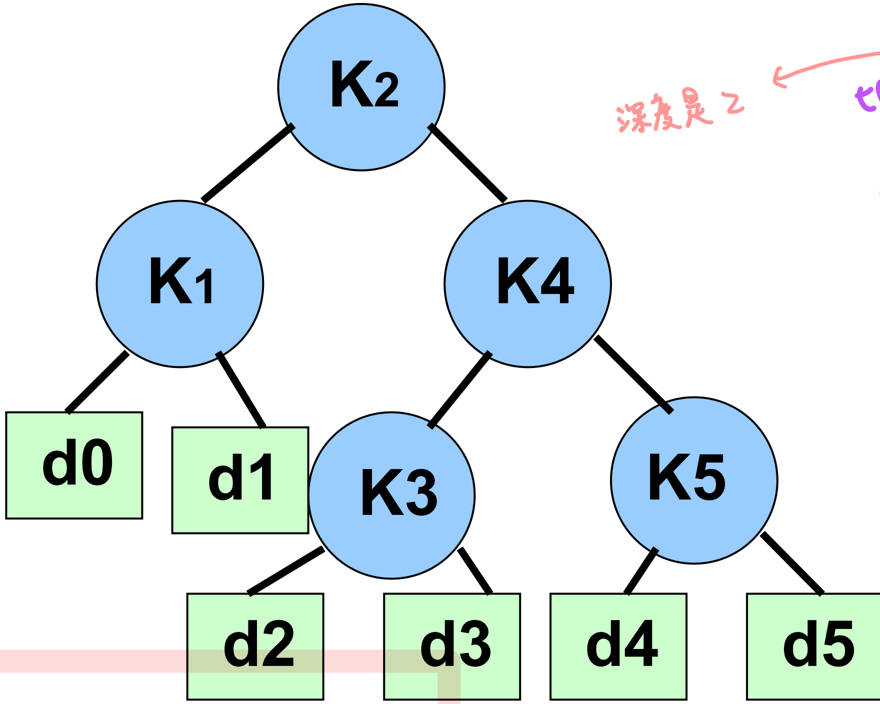
✓

這個節點的機率代表的是『今天會有多少人要來這裡』

i	0	1	2	3	4	5
p_i		0.15	0.1	0.05	0.1	0.2
q_i	0.05	0.1	0.05	0.05	0.05	0.1

熱門的節點如果擺上面一點，要找到特定的key所花費的次數就會比較少

Cost = Probability * (Depth+1)



深度是2

比較兩次 $K_1=2*0.15=0.3$

比較一次 $K_2=1*0.1=0.1$

$K_3=3*0.05=0.15$

$K_4=2*0.1=0.2$

$K_5=3*0.2=0.6$

$d_0=3*0.05=0.15$

$d_1=3*0.1=0.3$

$d_2=4*0.05=0.2$

$d_3=4*0.05=0.2$

$d_4=4*0.05=0.2$

$d_5=4*0.1=0.4$

all cost=2.8

這棵樹最後平均的期望值是 2.8次

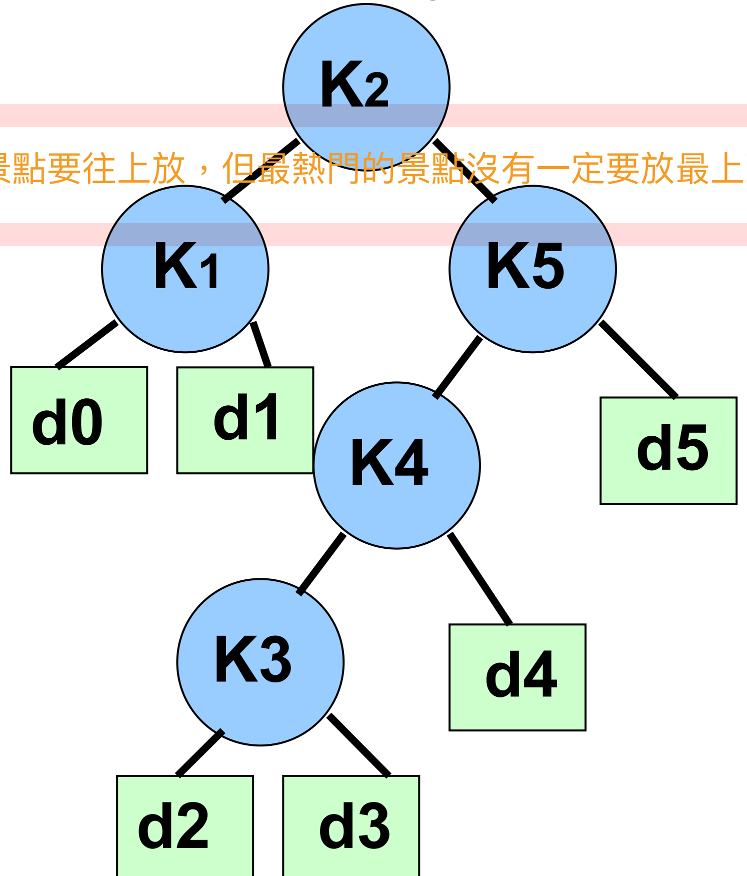
（平均的搜尋成本是 2.8次）

會有0.15的機率要找K1，
因為要找到K1要找兩層，
所以有0.15的機率要花費
這個兩層。
代表成本是0.15*2

i	0	1	2	3	4	5
p_i		0.15	0.1	0.05	0.1	0.2
q_i	0.05	0.1	0.05	0.05	0.05	0.1

期望值：帶有機率的情況下，去玩某個遊戲（彩券），產生的平均結果。

It is deeper!



熱門景點要往上放，但最熱門的景點沒有一定要放最上面（根）

i	0	1	2	3	4	5
p_i		0.15	0.1	0.05	0.1	0.2
q_i	0.05	0.1	0.05	0.05	0.05	0.1

這棵樹深度更深，但期望值（搜尋成本）比較低，所以比較好。
把熱門一點的節點往上放。

Cost = Probability * (Depth+1)

$K_1=2*0.15=0.3$

$K_2=1*0.1=0.1$

$K_3=4*0.05=0.2$

$K_4=3*0.1=0.3$

$K_5=2*0.2=0.4$

$d_0=3*0.05=0.15$

$d_1=3*0.1=0.3$

$d_2=5*0.05=0.25$

$d_3=5*0.05=0.25$

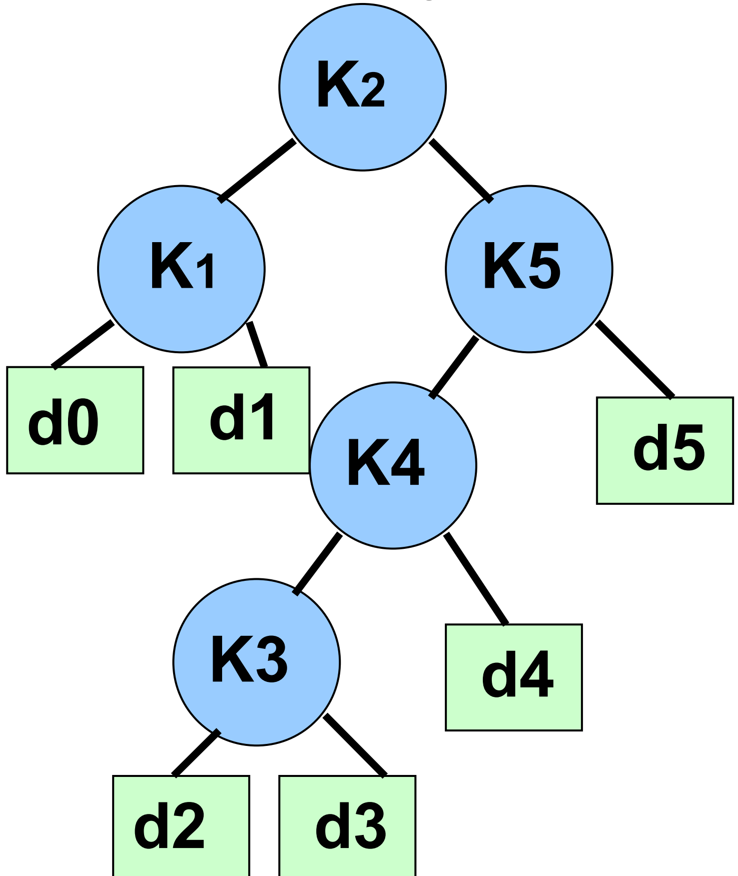
$d_4=4*0.05=0.2$

$d_5=3*0.1=0.3$

勝 **all cost=2.75**

But, optimal?

It is deeper!



i	0	1	2	3	4	5
p_i		0.15	0.1	0.05	0.1	0.2
q_i	0.05	0.1	0.05	0.05	0.05	0.1

Cost = Probability * (Depth+1)

這個式子算的是：給我一棵樹，要如何計算成本

E[Cost]

藍色的節點被拜訪的機率總和

$$= \sum_{i=1}^{n} p_i \left(depth(k_i)+1\right)$$

綠色的節點被訪的機率總和

$$+ \sum_{i=0}^{n} q_i \left(depth(d_i)+1\right)$$

要讓他是最小化成本，只需要考慮這邊是最小的

$$= \boxed{\sum_{i=1}^{n} p_i \cdot depth(k_i) + \sum_{i=0}^{n} q_i \cdot depth(d_i)}$$

$$+ \sum_{i=1}^{n} p_i + \sum_{i=0}^{n} q_i$$

$=1$

全部的key、dummy key加起來會是1

勝 **all cost=2.75**

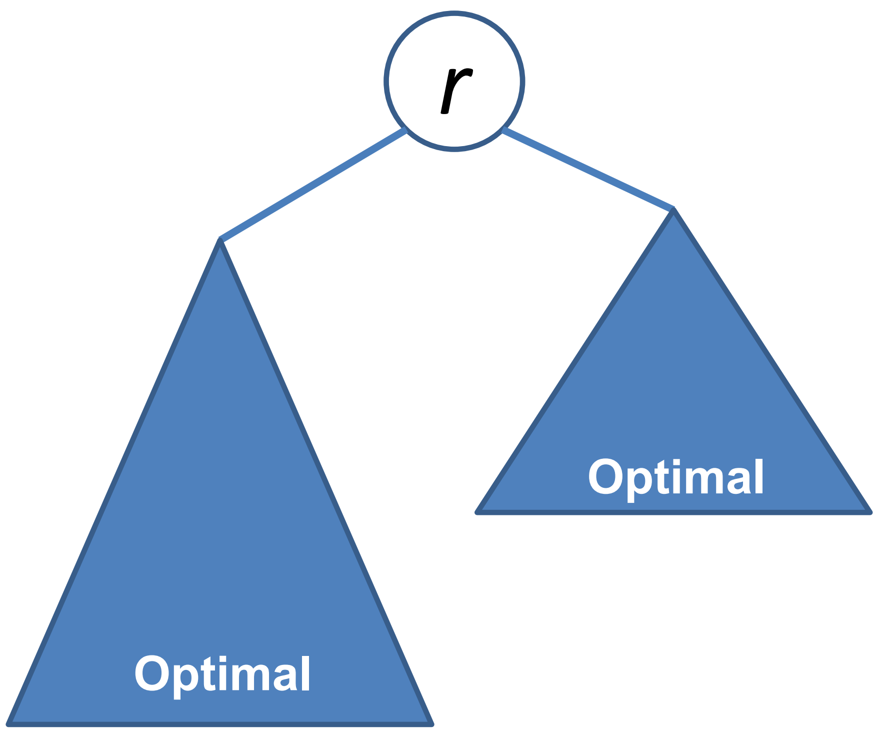
But, optimal?

E[search cost]

$$= \sum_{i=1}^{n} p_i \cdot depth(k_i) + \sum_{i=0}^{n} q_i \cdot depth(d_i) + \sum_{i=1}^{n} p_i + \sum_{i=0}^{n} q_i$$

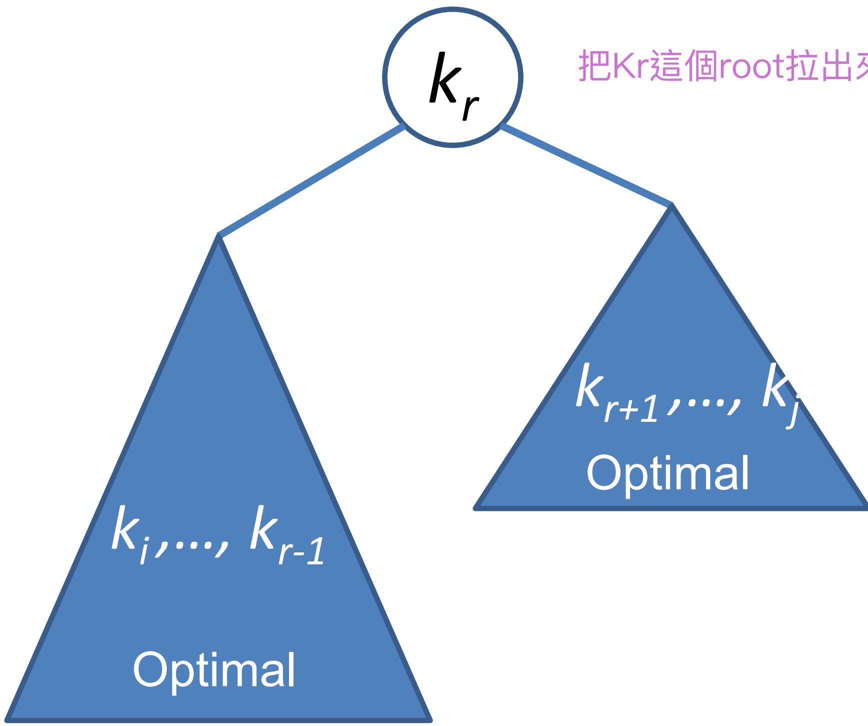
Dynamic programming

- Optimal substructure of BST?
  - The optimal binary search tree for 3, 7, 9, 12



Dynamic programming

$e[i, j]$: the expected cost of searching an optimal BST containing keys k_i, …, k_j.



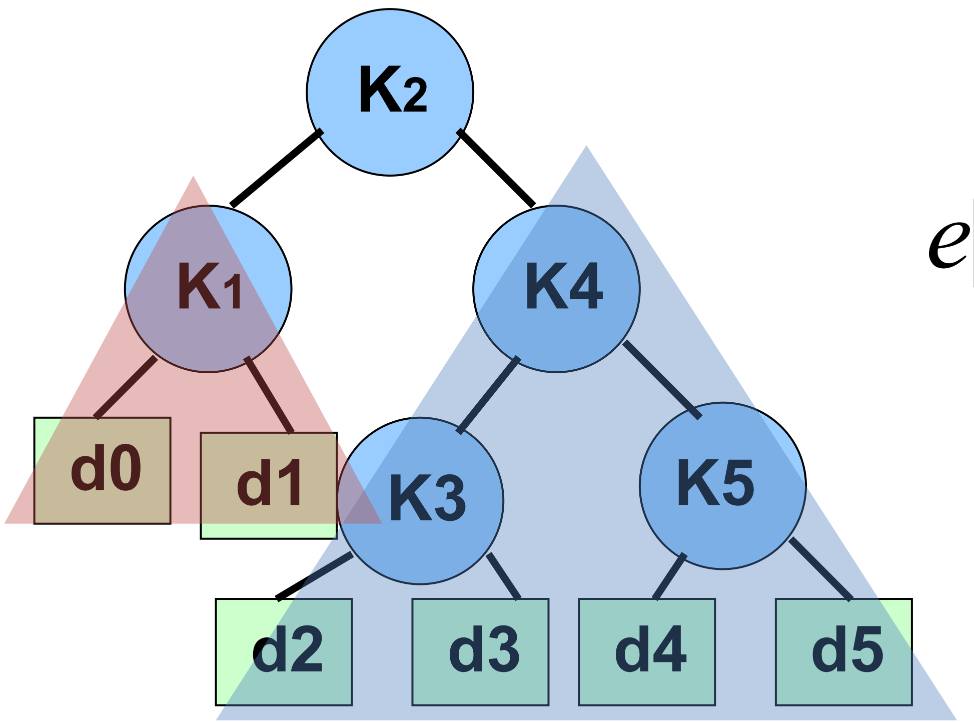
把Kr這個root拉出來，分左樹右樹

$$e[i, j]\ vs.\ e[i, r-1]$$

$$e[r+1, j]$$

Dynamic programming

$e[i, j]$: the expected cost of searching an optimal BST containing keys k_i, …, k_j.

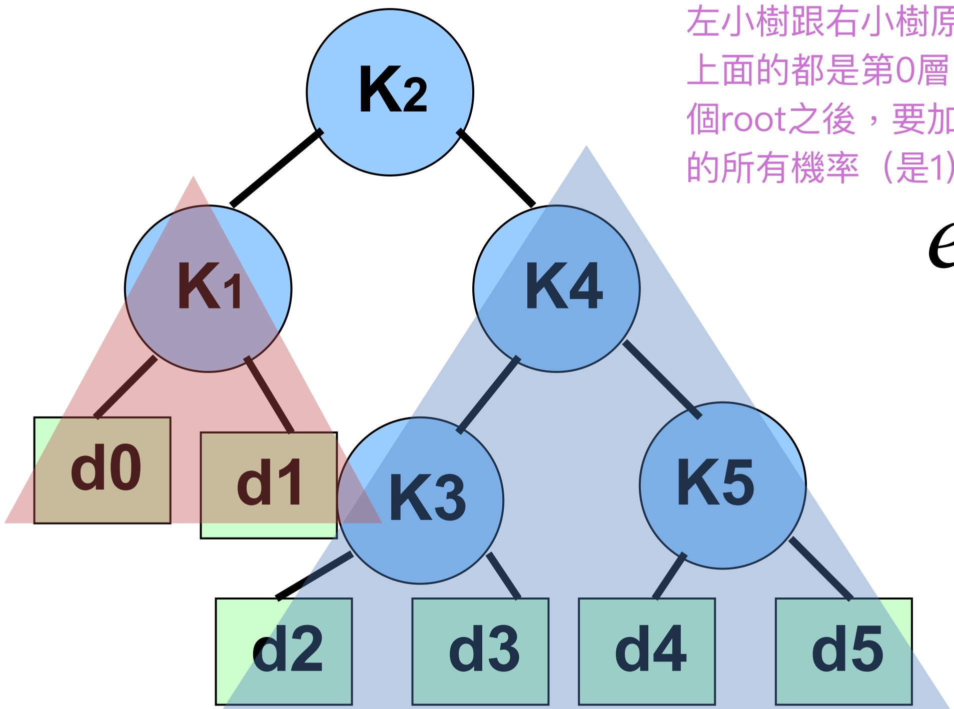


$$e[i, j] \; vs. \; e[i, r-1]$$

$$e[r+1, j]$$

$$e[i,j] = e[i,r-1] + e[r+1,j] + \sum_{l=i}^{j} p_l + \sum_{l=i-1}^{j} q_l$$

$e[i,j]$: the expected cost of searching an optimal BST containing keys k_i, ..., k_j.



左小樹跟右小樹原本是獨立的樹，最上面的都是第0層，所以現在拉出一個root之後，要加上左小樹跟右小樹的所有機率（是1）

$$e[i,j] \; vs. \; e[i,r-1]$$

$$e[r+1,j]$$

$$e[i,j] = \min_{i \le r \le j} \left\{ e[i,r-1] + e[r+1,j] + \sum_{l=i}^{j} p_l + \sum_{l=i-1}^{j} q_l \right\}$$

=

$w[i, j]$

計算的時候只需要先取前面兩個e的最小值，再把它加上後面的藍色部分，最後再存進e

這裡不會是1，因為1是整棵樹的機率。
我們這邊只是其中一棵子樹的機率總和。

- Build 3 tables 先把前處理做好
  - $e[i, j]$: keeps minimum cost
  - root[i, j]: keeps who is the root
  - $w[i, j]$: keeps $\sum_{l=i}^{j} p_l + \sum_{l=i-1}^{j} q_l$ 多存一個w表格

算好先放在w表格裡，
之後在算e的時候可以
直接查表

- $w[i, j] = w[i, j\text{-}1] + p_j + q_j$

查j 的前一格　　再把j放進去

$$w[i,j] = \sum_{l=i}^{j} p_l + \sum_{l=i-1}^{j} q_l = w[i,j\text{-}1] + p_j + q_j$$

建立w表，之後可以快速獲得這些機率的加總

input probability table

i	0	1	2	3	4	5
p_i		0.15	0.1	0.05	0.1	0.2
q_i	0.05	0.1	0.05	0.05	0.05	0.1

j

w	0	1	2	3	4	5
1	0.05	0.3				
2		0.1	0.25			
3			0.05	0.15		
4				0.05		
5					0.05	
6						0.1

i

0.05+0.15+0.1

0.1+0.1+0.05

0.05+0.05+0.05

左下角不管，因為左下角的關係是『i 比 j 還大』的關係

$$w[i, j] = \sum_{l=i}^{j} p_l + \sum_{l=i-1}^{j} q_l = w[i, j\text{-}1] + p_j + q_j$$

input probability table

i	0	1	2	3	4	5
p_i		0.15	0.1	0.05	0.1	0.2
q_i	0.05	0.1	0.05	0.05	0.05	0.1

j

w	0	1	2	3	4	5	
1	0.05	0.3	0.45	0.55	0.7	1	0.05+0.15+0.1
2		0.1	0.25	0.35	0.5	0.8	0.1+0.1+0.05
3			0.05	0.15	0.3	0.6	0.05+0.05+0.05
4				0.05	0.2	0.5	
5					0.05	0.35	
6						0.1	

i

$$e[i,j] = \min_{i \le r \le j} \left\{ e[i,r-1] + e[r+1,j] + \underbrace{\sum_{l=i}^{j} p_l + \sum_{l=i-1}^{j} q_l}_{= \, w[i,j]} \right\}$$

i	0	1	2	3	4	5
p_i		0.15	0.1	0.05	0.1	0.2
q_i	0.05	0.1	0.05	0.05	0.05	0.1

j

e	0	1	2	3	4	5
1	0.05	0.45				
2		0.1				
3			0.05			
4				0.05		
5					0.05	
6						0.1

i

e[1,0]+e[2,1]+w[1,1

=0.05+0.1+0.3

=0.45

$$e[i,j] = \min_{i \le r \le j} \left\{ e[i,r-1] + e[r+1,j] + \sum_{l=i}^{j} p_l + \sum_{l=i-1}^{j} q_l \right\}$$

$$\sum_{l=i}^{j} p_l + \sum_{l=i-1}^{j} q_l = w[i,j]$$

i	0	1	2	3	4	5
p_i		0.15	0.1	0.05	0.1	0.2
q_i	0.05	0.1	0.05	0.05	0.05	0.1

j →, i ↓

e	0	1	2	3	4	5
1	0.05	0.45	0.9	1.25	1.75	2.75
2		0.1	0.4	0.7	1.2	2
3			0.05	0.25	0.6	1.3
4				0.05	0.3	0.9
5					0.05	0.5
6						0.1

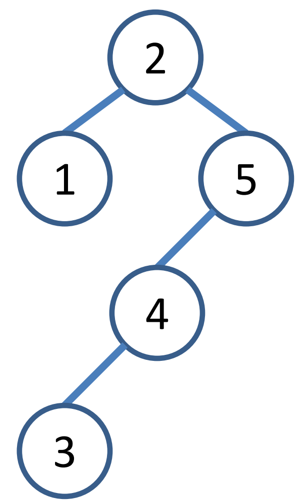
$$e[i,j]=\min_{i\le r\le j}\left\{e[i,r-1]+e[r+1,j]+\underbrace{\sum_{l=i}^{j}p_l+\sum_{l=i-1}^{j}q_l}_{= w[i,j]}\right\}$$

i	0	1	2	3	4	5
p_i		0.15	0.1	0.05	0.1	0.2
q_i	0.05	0.1	0.05	0.05	0.05	0.1

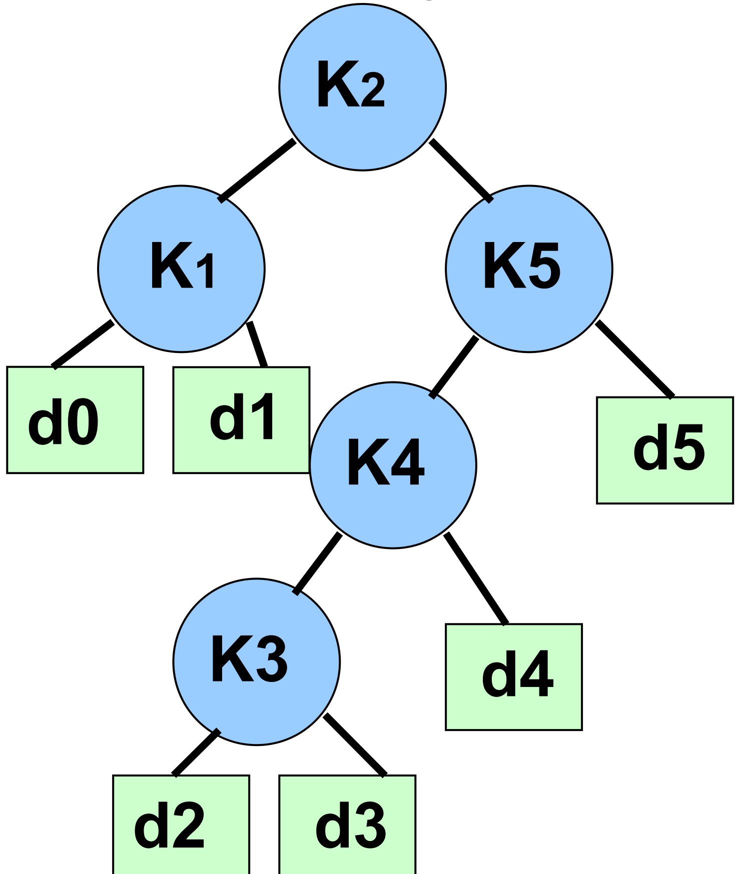
j

root	1	2	3	4	5
1	1	1	2	2	2
2		2	2	2	4
3			3	4	5
4				4	5
5					5

i



It is deeper!



i	0	1	2	3	4	5
p_i		0.15	0.1	0.05	0.1	0.2
q_i	0.05	0.1	0.05	0.05	0.05	0.1

Cost = Probability * (Depth+1)

$K_1=2*0.15=0.3$

$K_2=1*0.1=0.1$

$K_3=4*0.05=0.2$

$K_4=3*0.1=0.3$

$K_5=2*0.2=0.4$

$d_0=3*0.05=0.15$

$d_1=3*0.1=0.3$

$d_2=5*0.05=0.25$

$d_3=5*0.05=0.25$

$d_4=4*0.05=0.2$

$d_5=3*0.1=0.3$

勝 **all cost=2.75**

But, optimal?

- Time complexity: $O(n^3)$
  - Number of elements in a table: $O(n^2)$
  - Find the minimum cost for each element: $O(n)$

4/11(ㄷ)
End