Optimal Binary search Tree (OBST):

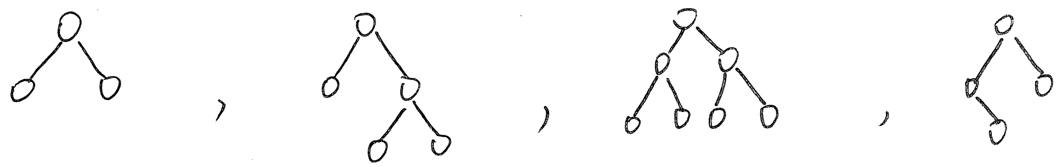
Binary Search Tree (BST) 是一個搜尋的資料結構。

因此，平均搜尋次數就是用來評估 BST 效能的衡量標準。(左子樹樹根鍵值小於樹根；右子樹樹根鍵值大於等於樹根)。

* 每個 node 的權重都相同時：

  構成平衡樹會使搜尋效能較好。

ex.



* 若是將 key node 和 null node 都各賦予權重時：

可將 key node 想成搜尋成功的機率、

null node 為搜尋失敗的機率。

平衡(相對對稱的)的 BST 不一定會使搜尋效能達到最佳。

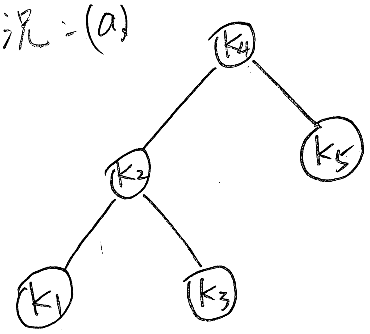
定義OBST：

（從小排到大，間隔一樣。）

給定 n 個相異鍵值 $K=<k_1,k_2,\ldots,k_n>$，不失一般性令其 $k_1<k_2<k_3<k_4<\cdots<k_n$。

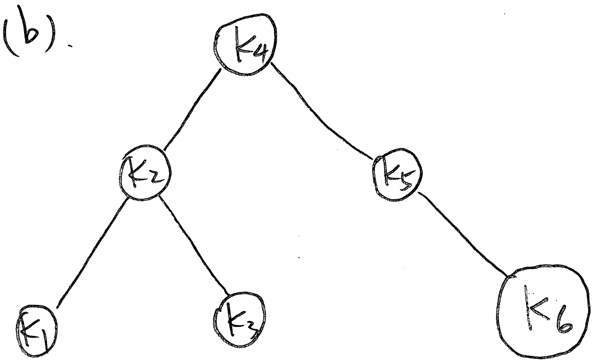
⇓

可能的狀況：(a)

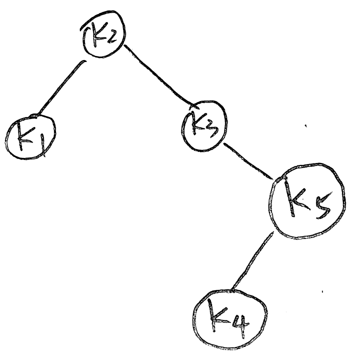


,

(b).



(c).



編號從小排到大，從左排到右，每次排鍵值比現在鍵值大的裡面排最小的逐一編號。

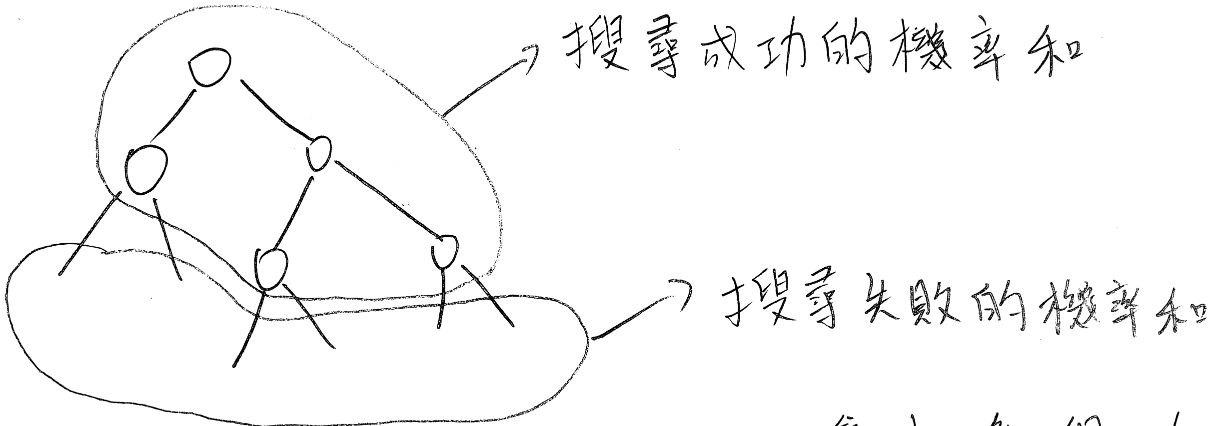
（想成頻率也可以）

其中，鍵值 k_i 發生的機率為 p_i。

如果以機率的角度來看，所有可能發生的情況機率為100%，記作1。舉例，今天有個資料庫是以OBST的方式排列資料，當我們需要搜尋某筆資料時（指的是找那個鍵值），並沒有找到我們想要的資料，所以單只有key的樹不可能包含所有情況。

也就要考慮搜尋失敗的情況。

ex.



會比 key 多一個 node

令每個搜尋失敗的點為 dummy key = $\langle d_0, d_1, \dots, d_n \rangle$

其中 d_i 代表 k_{i-1} 到 k_i 的所有值。

$\Downarrow$

指的是 d_i 這筆資料可能會在 k_{i-1} 到 k_i 之間。

不失一般性，dummy key 的編號是從左到右的所有 null node 開始填入 $d_0, d_1, d_2, \dots, d_n$。

鍵值大小：

$d_0 < k_1$, $k_n < d_n$

令 d_i 發生機率為 q_i。

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

評估搜尋效能的指標：平均搜尋次數
(越少越好)

(樹的level有稱樹高或深度，root點有時會是從0開始算或從1開始，隨定義而定。)

定義平均搜尋次數：(每棵樹有它的指標E)

$$E = \sum_{i=1}^{n}\left[depth_T(k_i)+1\right]\cdot p_i + \sum_{i=0}^{n}\left[depth_T(d_i)+1\right]\cdot q_i$$

其 $depth_T(n)$ 代表節點 n 在樹 T 的深度，其中 root 的深度為 0。

求哪個 Binary Search Tree，平均搜尋次數最小。

所以在給定key和dummy key權重時，將其安排成平衡的BST並不一定會使其平均搜尋次數的值較小。

Optimal Substructure

令 T' 為包含 $k_i \dots k_j$ 及 $d_{i-1} \dots d_j$ 的 OBST，令 k_r 為 root。則包含 $k_i \dots k_{r-1}$ 及 $d_{i-1} \dots d_{r-1}$ 的左子樹及包含 $k_{r+1} \dots k_j$ 及 $d_r \dots d_j$ 的右子樹均為 OBST。

在最佳解結構中，並沒有告訴我們要如何選擇 root。因此，實做上需要測驗所有的可能。將 $i=1 \sim j$ 都試過一次，才能知道哪個結點為 root 時，有最佳解。

令 $e[i,j]$ 為 optimal binary search tree 中包含 $k_i \sim k_j$ 和 $d_{i-1} \sim d_j$ 的 OBST 的平均搜尋次數，$w[i,j]$ 為 $k_i \sim k_j$ 及 $d_{i-1} \sim d_j$ 的發生機率和。

因為 $w[i,j] = \sum_{l=i}^{j} p_l + \sum_{l=i-1}^{j} q_l$，因此若 r 為 T' 的 root，則：

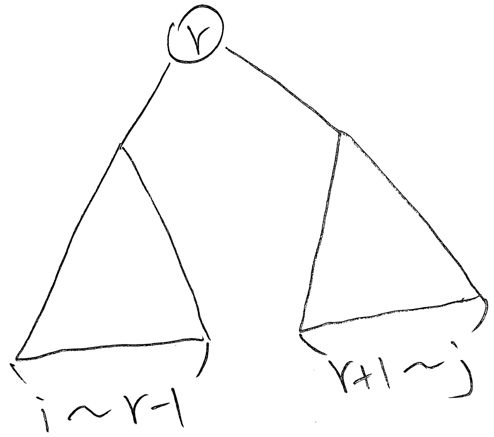
$$e[i,j] = p_r + (e[i,r-1] + w[i,r-1]) + (e[r+1,j] + w[r+1,j])$$

$$= e[i,r-1] + e[r+1,j] + p_r + w[i,r-1] + w[r+1,j]$$

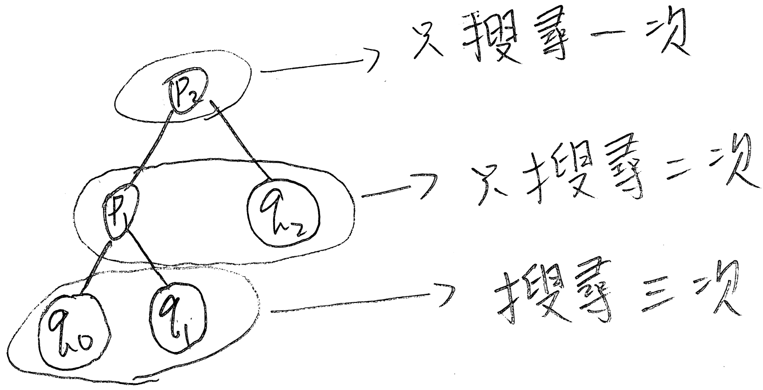
$$= e[i,r-1] + e[r+1,j] + w[i,j]$$

公式釐清：e[i,j] 指的是最小平均搜尋次數

在 i 點到 j 點中有一個 r 點當 root。

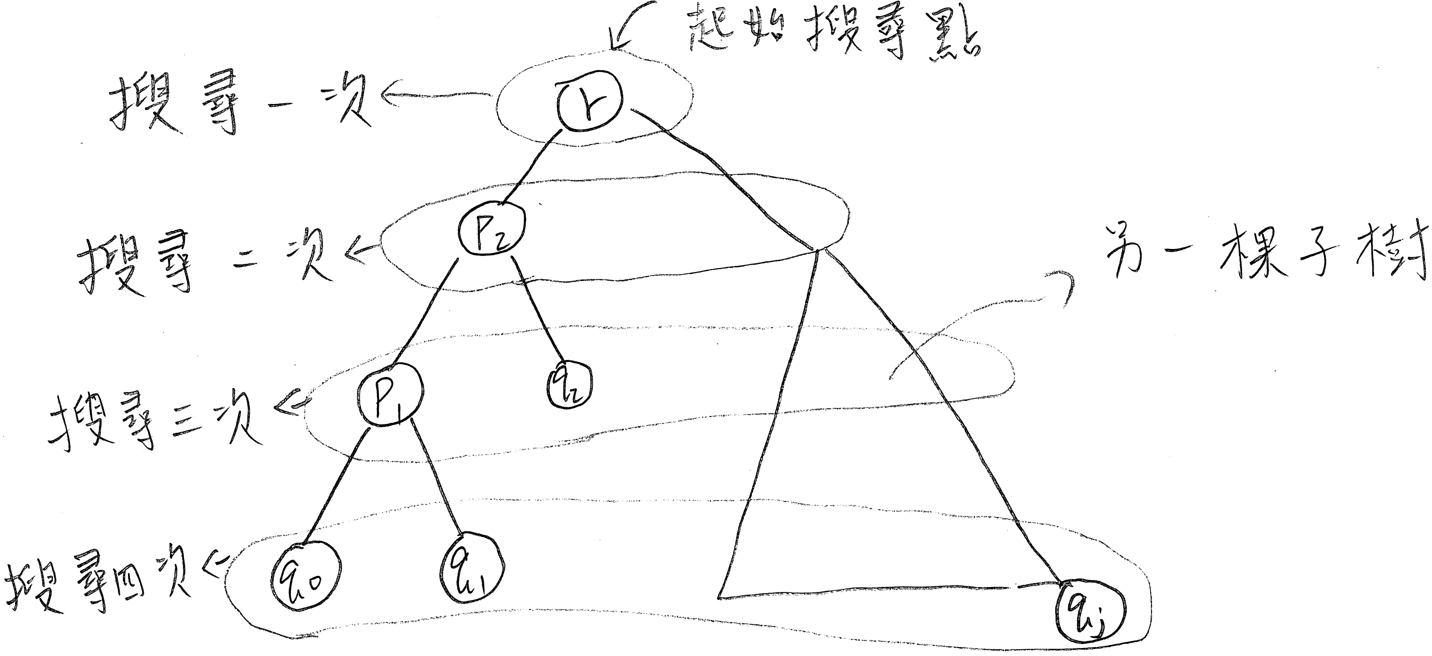


p_i 為 key 的發生機率，q_i 為 dummy key 的發生機率。



$$e[1,2] = (p_1 \times 2 + p_2 \times 1) + (q_0 \times 3 + q_1 \times 3 + q_2 \times 2)$$

那如果上圖是某樹根的一棵子樹，那每個點的搜尋次數會多加一次：



所以左、右子樹的最佳搜尋次數往下推，則左右子樹的每一點的機率需多加一次，r點發生的機率為 p_r，

∴公式為

左子樹

因為每點都往下推需所有在子樹的點再加一次

$$e[i,j] = p_r + (e[i,r-1] + w[i,r-1]) + (e[r+1,j] + w[r+1,j])$$

右子樹

應該寫成 $p_r \times 1$

搜尋一次

寫成 Recursion 表示：

令 $e[i,j]$ 為 optimal binary search tree 中包含 $k_i \cdots k_j$ 及 $d_{i-1} \cdots d_j$ 的 OBST 的平均搜尋次數，$w[i,j]$ 為 $k_i \sim k_j$ 及 $d_{i-1} \sim d_j$ 的發生機率和。則 $e[i,j]$ 定義如下：

$$e[i,j] = \begin{cases} q_{i-1} & , \text{if } j = i-1 \\ \displaystyle\min_{i \le r \le j} \{ e[i,r-1] + e[r+1,j] \} + w[i,j] & , \text{if } i \le j \end{cases}$$

Note：

$e[i,i-1]$ 為只含一個 q_{i-1} 之 OBST 的平均搜尋次數。

當只有一個 dummy key 時，其平均搜尋次數為：

$1 \times q_{i-1} = q_{i-1}$ 。

例 8.2

Given a sequence $K = < k_1, k_2, \ldots, k_n >$ of n distinct keys in sorted order such that $k_1 < k_2 < \ldots < k_n$,and we wish to build a binary search tree from thses keys. For each key k_i, we have a probability p_i that a search will be for k_i. Some searches may be for values not in K, and so we also have n+1 "dummy keys" d_0, d_1, d_2,…, d_n representing values not in K. In particular, d_0 represents all values less that k1, dn represents all values greater that k_n, and for $i = 1,2,\ldots,n-1$, the dummy keys di represents all values between k_i and k_{i+1}. For each dummy key d_i, we have a probability q_i that a search will correspond to di. Each key k_i is an internal node, and each dummy key d_i is a leaf. Every search is either successful (finding some key k_i) or unsuccessful (finding some dummy key d_i), and so we have $\sum_{i=1}^{n} p_i + \sum_{i=0}^{n} q_i = 1$. The expected cost of a search tree T is

$$E[\text{search cost in T}] = \sum_{i=1}^{n}(depth_T(k_i)+1)\cdot p_i + \sum_{i=0}^{n}(depth_T(d_i)+1)\cdot q_i$$
$$= 1 + \sum_{i=1}^{n} depth_T(k_i)\cdot p_i + \sum_{i=0}^{n} depth_T(d_i)\cdot q_i$$

Where $depth_T$ denotes a node's depth in the tree T. Given five keys with p_1=0.15, $p_2 = p_4 = q_5 = q_1 = 0.10$, $p_3 = q_0 = q_2 = q_3 = q_4 = 0.05$, $p_5 = 0.20$, compute the corresponding smallest search cost. 【95 年成大資工所】

Ans.
先整理發生機率的表格

index	0	1	2	3	4	5
key (p)		0.15	0.10	0.05	0.10	0.20
dummy key (q)	0.05	0.10	0.05	0.05	0.05	0.10

* 表格填法從左上到右下，從左到右。

* 開始填 e[i,j] 第2次時，就需填 root[i,j]。

需要畫三個表格：

(包含 dummy key)

1. i點到j點的機率和 (w[i,j])
2. i點到j點的平均最小搜尋次數
3. i點到j點誰當root點

1、2 — 表格形狀一樣

key 有幾點

dummy key 有幾點

從1開始到 n+1

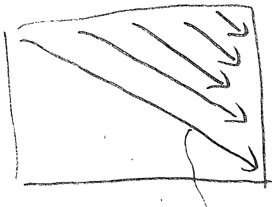
3 — 表格為方陣

x,y 軸為 key 的編號

w[i,j]	0	1	2	3	4	5
1	0.05	0.3	0.45	0.55	0.7	1
2		0.10	0.25	0.35	0.5	0.8
3			0.05	0.15	0.3	0.6
4				0.05	0.2	0.5
5					0.05	0.35
6						0.10

→ 可以想成右邊格為左邊格加 $\text{index}_{\text{右邊}}$ 的兩格

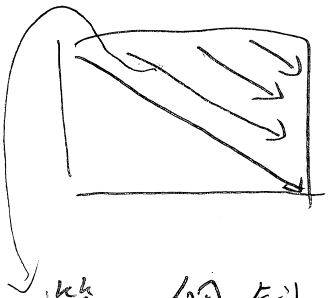
w[i,j] 填表技巧



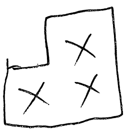
填第一個斜排只需逐步填 $q_0 \sim q_n$

---

i	0	1	2	3	4	...
key		×	×	×	×	...
dummy key	×	×	×	×	×	...

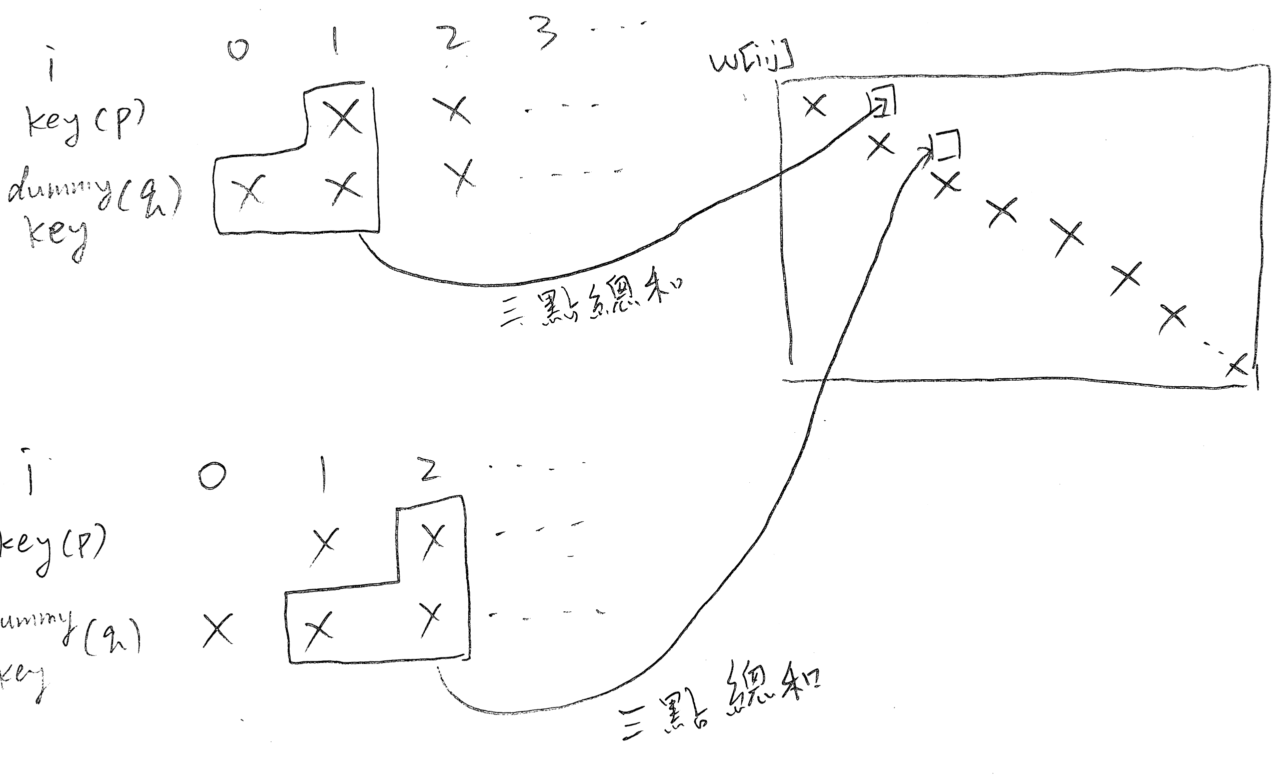


第二個斜排

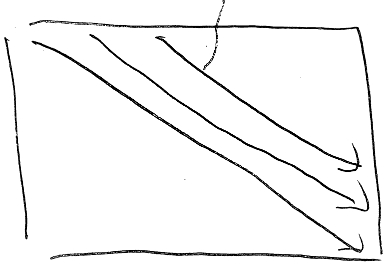
每次取  之值和填入 w[i,j].

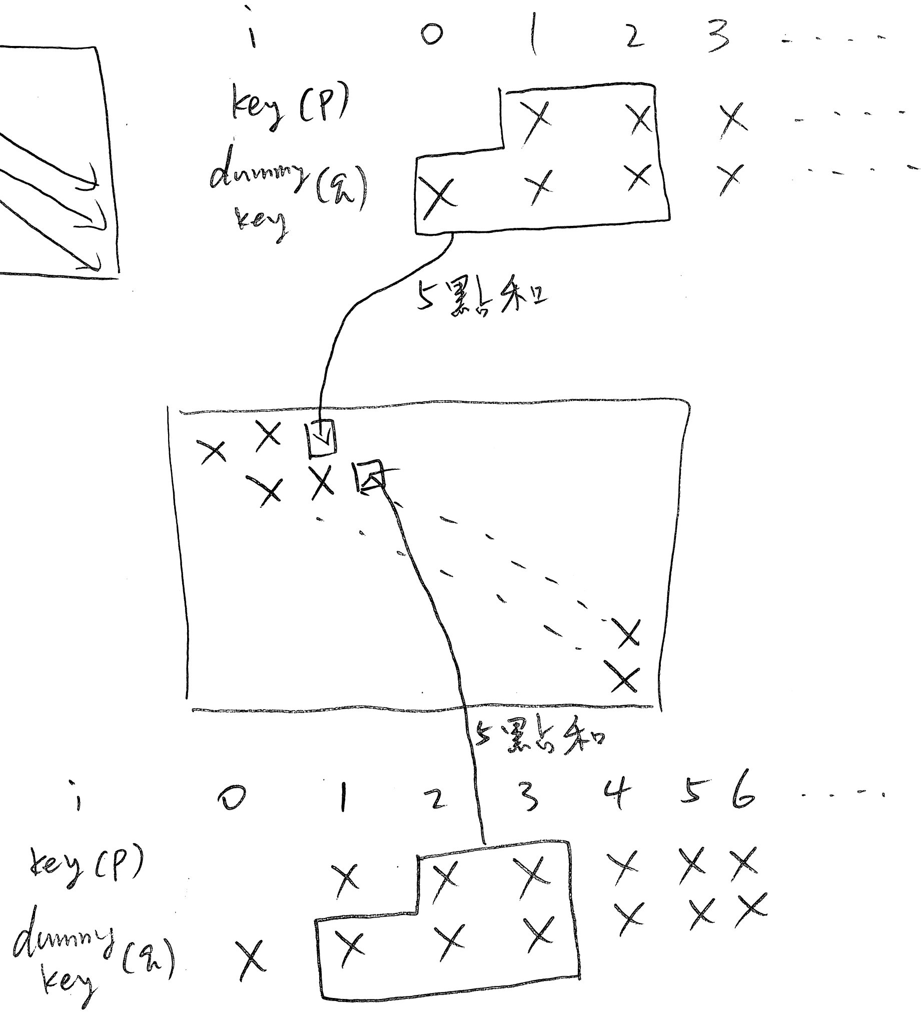
填完就往右一格位移

ex.



第三個斜排



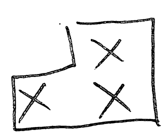


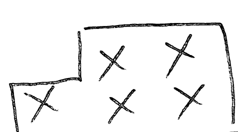
每次取5點和填入表格，再向右位移一格。

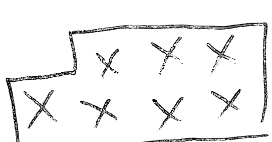
以此類推，每到新的斜排，取值和就多兩個要取

ex.

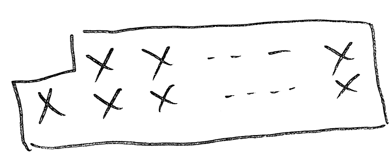
第一斜排： 

第二斜排： 

第三斜排： 

第四斜排： 

直到最後一斜排，
也就是最後一格，
即所有格加總：



除了 w[i,j] 還需 e[i,j] 和 root[i,j]

除了填 e[i,j] 的第一斜排不需填 root[i,j]，

e[i,j] 其餘，每次填一格就需填 root[i,j] 一格。

root[i,j] 為方陣的表格

e[i,j]	0	1	2	3	4	5
1	0.05	0.45	0.9	1.25	1.75	2.75
2		0.10	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

root[i,j]	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

e[i,j] 的第一斜排只需填 dummy key 的機率

↳ 為了方便看將 w[i,j] 寫到這。

w[i,j]	0	1	2	3	4	5
1	0.05	0.3	0.45	0.55	0.7	1
2		0.10	0.25	0.35	0.5	0.8
3			0.05	0.15	0.3	0.6
4				0.05	0.2	0.5
5					0.05	0.35
6						0.1

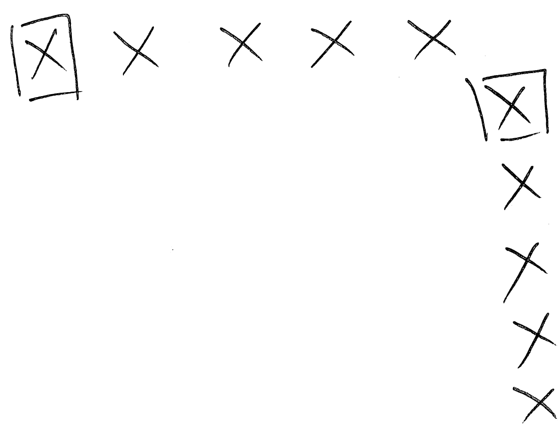
計算 e[i,j] 技巧：

因為 e[i,j] 為 $\underbrace{\min_{i\le r<j}}_{}\{e[i,r-1]+e[r+1,j]\}+w[i,j]$

r的可能為 i ~ j-1

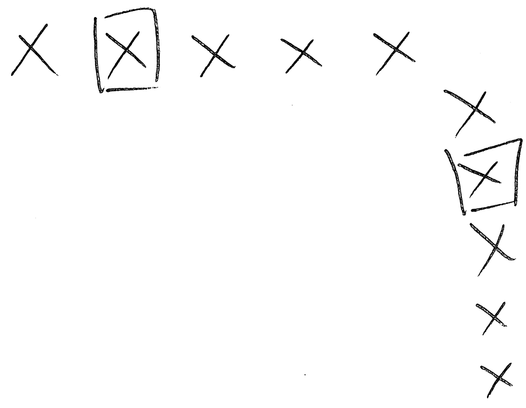
從表格看，假設要填 e[i,j] 那點，那就要一一比對某兩格和，找最小的，其順序範例。

比對的第一個： ×指的是填過的，⊠指合起的兩格。

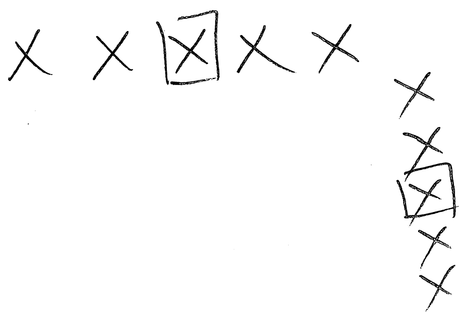


左邊的⊠為 r-1 點，右邊的⊠為 r+1 點

比對的第二個



比對的第三個



以此類推..

如果有比對出來最小值，那左邊那格所在的直行的index+1，為 root[i,j] 的 root 點。

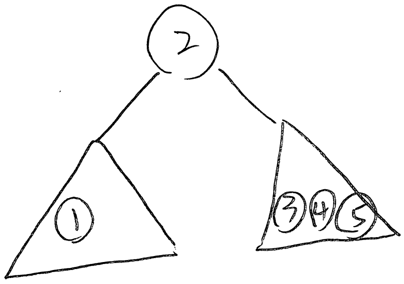
* 找出最小值後還需加 w[i,j] 才能填入 e[i,j]

從 root[i,j] 表格畫出 OBST

root[i,j]	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

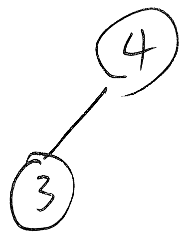
($k_1 \sim k_5$)

1. 現在有1~5點，找 root[1,5] 有最小搜尋



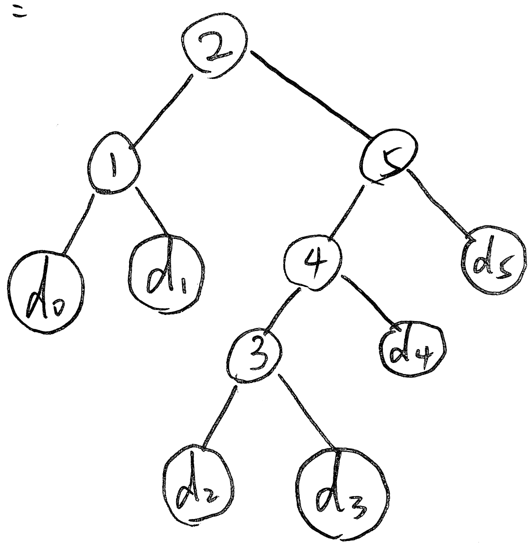
2. 看3~5點，找 root[3,5] 有最小搜尋

3. 看 3、4 點，root[3、4] 有最小搜尋



4. 整合 1~5 點為：

OBST =



#