

第八章

資料搜尋

Searching

版權屬作者所有，非經作者
同意不得用於教學以外用途

本章內容

8-1 搜尋及定義

8-2 在循序結構上的搜尋

線性搜尋法

二分搜尋法

內插搜尋法

8-3 索引結構的搜尋

直接索引、二元搜尋樹索引、B樹索引

8-4 雜湊法

雜湊函數

解決碰撞的方法

可擴充雜湊

「**搜尋引擎**」(search engine)：根據使用者輸入的「**關鍵字**」(key words)當作搜尋鍵，藉助於事先整理好的索引結構，搜尋出與輸入關鍵字相關的網頁列表作為**搜尋結果**。

The screenshot shows a Google search results page in Chinese. The search bar at the top contains the text '細談資料結構第七版'. Below the search bar, there are several search results. A red box highlights the search bar, and a red arrow points from the text '關鍵字' in the definition above to the search bar. A large red bracket on the right side of the page groups the search results under the label '搜尋結果'. The search results include links to various websites, such as '旗標學校服務網', '天瓏網路書店', and '博客來-探索資料結構'. The Windows taskbar is visible at the bottom, showing the search bar and several application icons.

Google 細談資料結構第七版

全部 影片 圖片 新聞 地圖 更多 設定 工具

約有 268,000 項結果 (搜尋時間: 0.47 秒)

旗標學校服務網：細談資料結構第七版
www.flag.com.tw/book/bookinfo.asp?bokno=F7801D
細談資料結構第七版 作者：謝樹明書號：F7801D ISBN：9789863124290 定價：420 元附件：附1片光碟片, Arrow 教學資源：習題-解答-投影片, Arrow 內容介紹! 內容 ...

旗標學校服務網：細談資料結構第六版
www.flag.com.tw/book/bookinfo.asp?bokno=F7801C
細談資料結構第六版 ... 本書將資料結構最重要的精神融入簡明易懂的實例當中, 讓學習者免於在龐雜的抽象文字中摸索, 而能透過範例與豐富圖表的引導, 輕鬆認識每 ... 5. 圖形Graphs 6. 樹狀結構Tree 7. 資料排序Sorting 8. 資料搜尋Searching 附錄

天瓏網路書店-細談資料結構(第七版)(附光碟)
<https://www.tenlong.com.tw/products/9789863124290>
書名：細談資料結構(第七版)(附光碟), ISBN：986312429X, 作者：謝樹明, 出版社：旗標, 出版日期：2017-06-30.

【大享】細談資料結構第六版 9789863120148 旗標 謝樹明著 F7801C ...
goods.ruten.com.tw > 書籍、文創、科學、電腦、程式語言、其他
★★★★★ 評分：5 - 25,273 則評論
2013年1月6日 - 【大享】細談資料結構第六版 9789863120148 旗標 謝樹明著 F7801C 450. ... 本賣場採合併計費~不論本數65元(郵費) 7-11取貨付款60元(7-11限 ...

的賣場在露天拍賣
class.ruten.com.tw/user/index00.php?s=mini250049
... 【資管必備】《細談資料結構-使用Visual Basic》ISBN:9574428605| 旗標| ... 【資管系必備】《商用微積分(第7版)》ISBN 9867138635| 新加坡商湯姆生亞洲私人 ...

博客來-探索資料結構(附光碟)
www.books.com.tw, 中文書, 電腦資訊, 程式設計/APP開發, 程式邏輯/演算法

搜尋 Windows

3 上午 11:30 2017/7/20

當眾多玩家使出看家本領，追分搶分灌分後，遊戲設計者如何按照分數將各家英雄好漢定出高下，以找出冠軍得主？

是的，答案是使用「**排序**」演算法。

排序被廣泛用在各種場合，例如：

1. 各種考試後的分發，按照成績高低順序媒合志願
2. 使用 Excel 試算表時，按照選定的欄位將資料由小到大或由大到小排序。

8-1 搜尋及定義

※用某些欄位為依據來找出記錄，進而得到同一筆記錄的其他屬性，
這個動作稱為**搜尋**（search），搜尋首要的考慮是速度。

◎ 例如：我們可以在通訊錄檔案中，根據姓名來找出某一筆記錄，進而
獲知其他屬性如年齡、電話、地址

記錄編號	姓名	年齡	電話	住址
1	張四	31	(02)2334----	台北市XXX
2	李五	16	(02)2999----	台北縣XXX
3	王六	25	(03)3444----	桃園縣XXX
4	趙大	41	(02)2887----	台北市XXX
5	陳文	53	(04)333----	台中市XXX
6	江水	28	(03)534----	新竹市XXX

根據搜尋鍵“趙大”
找到第四筆記錄，
得到他的電話、
地址等欄位

◎ 在符號表(symbol table)上作搜尋，也是根據搜尋鍵來獲得此符號在符號表中的位置，進而得到此符號的其它屬性值，例如符號在記憶體中的位址，或符號的內含值等

「鍵—值配對」(key-value pairs)：根據 *key* 找出對應的 *value*

若 key 為 "banana"
則其 value 為 61

<i>key</i>	<i>value</i>
apple	57
banana	61
guava	43
mango	124

※整理檔案或符號表的方式基本上有兩種：

- ◆ **循序結構**：一筆一筆記錄或一個一個符號循序排列下來，除此之外沒有其它輔助，循序結構又分為**沒有排序過**和有根據鍵欄**排序過**兩種。
- ◆ **索引結構**：另外加上的輔助工具，用來協助搜尋，這些工具通稱為**索引** (index)。例如一本書除了課文 (循序結構) 之外，另外會有章節索引或關鍵字索引。利用索引可以比較有效率地找到記錄或符號所在的位置。索引結構有很多，例如二元搜尋樹索引、B樹、B+樹、以及雜湊表

※對檔案或符號表主要的處理動作包括：

- 排序。
- 插入新記錄或新符號。
- 刪除記錄或符號。
- 搜尋某特定記錄或符號是否存在。
- 取得該特定記錄或符號的其他屬性。

※如果一個檔案或符號表的資料儲存妥當之後，就不會再有插入及刪除的動作，則稱這個檔案或符號表是「**靜態**」(static)的結構。如果可能會有插入或刪除的動作，則它們屬於「**動態**」(dynamic)的結構。動態的結構除了和靜態的結構一樣重視**搜尋的速度**，它同時還要考慮**動態維護** (插入及刪除資料)的效率。

8-2 在循序結構上的搜尋

循序搜尋法

```
int Sequential_Search(int a[], int n, int key)
{
    int i;
    for( i=0 ; i < n ; i++)
        if (a[i] == key)
            return(i);    /*成功傳回位置*/
    return(-1);           /*失敗傳回-1*/
}
```

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
39	81	6	78	69	41	52	33	55	77

key = 78 : 比較 4 次後成功 (39, 81, 6, 78) 。

key = 68 : 比較 10 次後失敗 (全部) 。

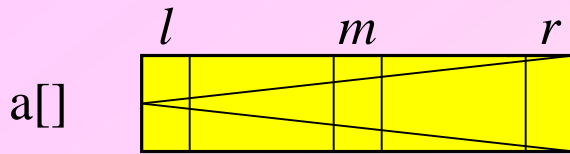
特性： 從第一個資料找起，直到找到為止，或是比對了所有資料都與 **key** 不相等而宣告失敗。

搜尋效率： $O(n)$ 。

整理效率： 資料不需要整理。

應用場合： 通常應用在未經排序的資料。或是資料經常變動，以致不值得花代價排序的時候。

二分搜尋法



由小到大排序好的資料

循序搜尋從頭比起，
二分搜尋比中間

$$m = (l + r) / 2$$

搜尋鍵 key 和 中間元素 $a[m]$ ($m = (l + r) / 2$) 作比較，只有三種結果之一：

1. $key = a[m]$ ：那麼搜尋已經成功。
2. $key > a[m]$ ：那麼如果在陣列中，只可能出現在右半部。 $(a[m+1] \sim a[r])$
3. $key < a[m]$ ：那麼如果在陣列中，只可能出現在左半部。 $(a[l] \sim a[m-1])$

* 如果是 2. 或 3.，則搜尋範圍已經減半，我們再重複選取新範圍的中間元素做比較，直到搜尋成功

* 或是範圍內已經無法再二分為止，亦即搜尋失敗

```
1.      int Binary_Search(int a[], int n, int key)
2.      {          int l = 0, r = n-1, m;
3.              while (l <= r)
4.              {          m = (l+r)/2;
5.                      if (key == a[m])
6.                          return(m);    //找到,傳回位置
7.                      if (key > a[m])    //右半部,改變左限
8.                          l = m+1;
9.                      else                //左半部,改變右限
10.                         r = m-1;
11.              }
12.          return(-1);                //失敗傳回-1
13.      }
```

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
6	33	39	41	52	55	69	77	78	81

key=41

第 i 次比較	l	r	m	key 與 $a[m]$	新範圍
1	0	9	4	$41 < 52$	左半部
2	0	3	1	$41 > 33$	右半部
3	2	3	2	$41 > 39$	右半部
4	3	3	3	$41 = 41$	成功

共作4次比較，搜尋成功

搜尋效率： $O(\lg n)$

整理效率： 資料需要排序

應用場合： 通常應用在資料極少變動，亦即插入和刪除極少發生的情況。因此一次排序，就可以做多次的搜尋

補充：二分搜尋法搜尋效率

對 n 個鍵值
作搜尋所需
的時間

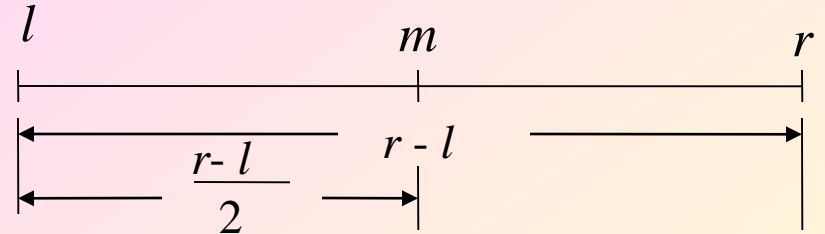
每作 1 次比較，就將
問題的規模減半為 $n/2$

$$\begin{aligned}T(n) &\leq T(n/2) + 1, \quad T(1)=1 \\&\leq (T(n/4) + 1) + 1, \quad \because T(n/2) \leq T(n/4) + 1 \\&\rightarrow T(n) \leq T(n/2^2) + 2 \\&\quad \vdots \\&\rightarrow T(n) \leq T(n/2^k) + k \\&\quad \text{當 } n \doteq 2^k \text{ (} \lg n \doteq k \text{) 時} \\&\rightarrow T(n) \leq T(n/2^k) + k \\&\rightarrow T(n) \leq T(1) + k \\&\quad \leq 1 + k \\&\quad = 1 + \lg n \\&\text{因此 } T(n) = O(\lg n)\end{aligned}$$

內插搜尋法

◎二分搜尋法預測 m 值

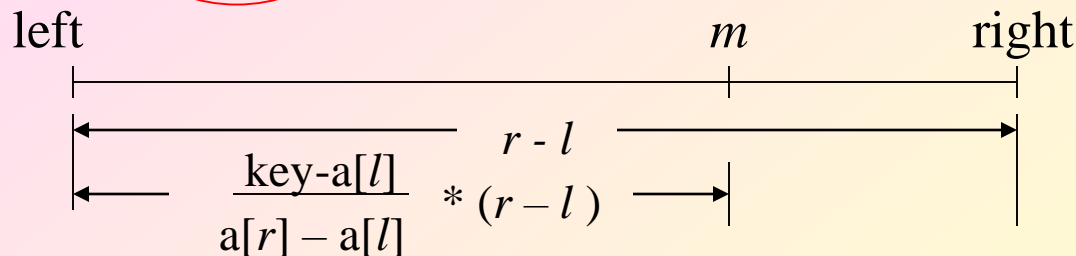
$$m = \frac{l+r}{2} = l + \frac{1}{2} * (r-l)$$



◎內插搜尋法預測 m 值

$$m = l + \frac{key - a[l]}{a[r] - a[l]} * (r - l)$$

由固定的1/2變成一個比例



◎除了預測 m 值的方法不同，其他都和二分搜尋法相同

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
6	33	39	41	52	55	69	77	78	81

key= 78

第i次比較	l	r	m	key與 a[m]	新範圍
1	0	9	$0 + (\frac{78-6}{81-6}) * 9 = 8$	$78 = 78$	成功

共作 1 次比較，搜尋成功

搜尋效率： $O(\lg(\lg n))$

整理效率： 資料需要排序

應用場合： 與二分搜尋法相同，但如果鍵值分佈極端不平均，則效率會變差

```

1.  int Inter_Search(int a[], int n, int key)
2.  {      int l = 0, r = n-1, m;
3.          float x;
4.          while (l <= r)
5.          {      if (a[r] - a[l] != 0)
6.                  x = (float) (key - a[l]) / (a[r] - a[l]);
7.                  else
8.                      x = 0;
9.                  m = l + (int)(x * (r-l));
10.                 if (key == a[m])
11.                     return (m);           //找到,傳回位置
12.                 if (key > a[m])           //右半部,改變左限
13.                     l = m + 1;
14.                 else                     //左半部,改變右限
15.                     r = m - 1;
16.             }
17.         return(-1);                       //失敗傳回-1
18.     }

```


費氏搜尋法

- 費氏搜尋法的好處，是避免二分搜尋法及內插搜尋法計算搜尋目標時所必須作的乘法及除法。
- 費氏搜尋法仍然必須作用在已經排序好的資料，並且其搜尋效率也是 $O(\lg n)$ 。
- 費氏數列的定義： $F[0] = 0, F[1] = 1$ ； $F[i] = F[i-1] + F[i-2]$ ， $i \geq 2$
因此 $F[i] - 1 = (F[i-1] - 1) + (F[i-2] - 1) + 1$

費氏搜尋法
的核心

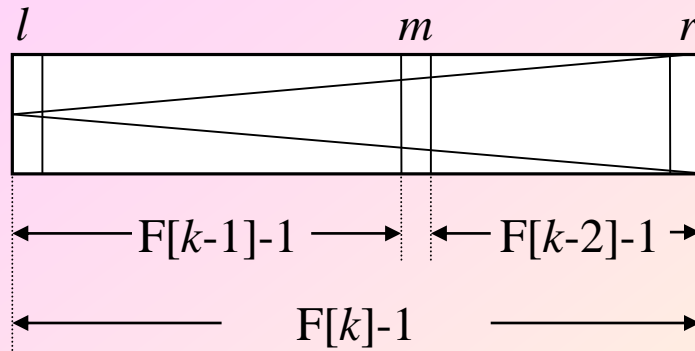
假設鍵值個數 n 為某個費氏數減 1 ($F[k] - 1$ ，否則只要在後面全部加上最大的鍵值作為虛擬鍵值，dummy key)，並且整個搜尋範圍為 $a[l] \sim a[r]$ ，

假設鍵值個數 n 為某個費氏數減1 ($F[k]-1$)，並且整個搜尋範圍為 $a[l] \sim a[r]$ ，選取 $a[m]$ ($m = l + F[k-1] - 1$) 與鍵值 (key) 作比較，比較只有三種結果：

$key = a[m]$ ：那麼搜尋已經成功。

$key < a[m]$ ：那麼新範圍是第 l 個到第 $m-1$ 個 ($a[l]$ 到 $a[l + F[k-1] - 2]$)，此範圍的個數為 $F[k-1]-1$ 個，仍為某個費氏數減1

$key > a[m]$ ：那麼新範圍是第 $m+1$ 個到第 r 個 ($a[l + F[k-1]]$ 到 $a[r]$)，此範圍的個數為 $F[k-2]-1$ 個，仍為某個費氏數減1



由小到大排序好的鍵值

$$F[k] - 1 = (F[k-1] - 1) + (F[k-2] - 1) + 1$$

費氏搜尋法
的核心

這種搜尋法與二分搜尋法類似，都屬於「個個擊破法」。但是二分搜尋法的 m 固定為中間位置，而費氏搜尋法則固定以費氏數列作分割。

假設鍵值數目 n 為12 ($= F[7]-1$)

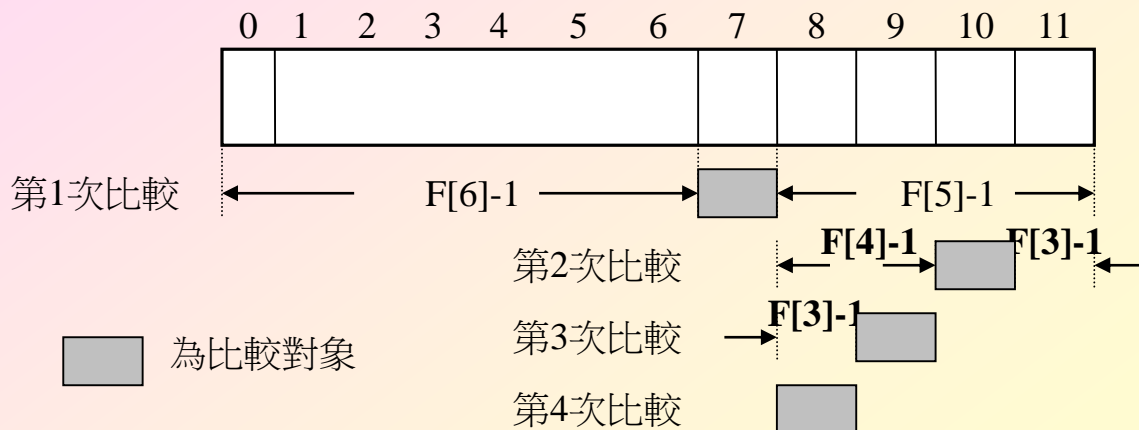
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
6	33	39	41	52	55	69	77	78	81	88	92

且假設搜尋鍵值 $key = 78$

第 <i>i</i> 次比較	l	r	m	key 與 $a[m]$ 比較	新範圍
第1次比較	0	11	$7(=0+F[6]-1)$	$78 > 77$	右半部
第2次比較	8	11	$10(=8+F[4]-1)$	$78 < 88$	左半部
第3次比較	8	9	$9(=8+F[3]-1)$	$78 < 81$	左半部
第4次比較	8	8	$8(=8+F[2]-1)$	$78 = 78$	成功

共作4次比較，搜尋成功

比較過程可以用下圖圖示：



8-3 索引結構的搜尋

直接索引

使用
Number
的索引(未
排序)

Number	Record#	Record#	Number	Name	Address
70	1	→ 1	70	王五	台北市XXX
81	2	→ 2	81	張三	台北縣XXX
42	3	→ 3	42	江水	新竹市XXX
31	4	→ 4	31	陳文	台北市XXX
72	5	→ 5	72	李四	台北市XXX
51	6	→ 6	51	趙大	新竹市XXX
90	7	→ 7	90	黃扁	台北縣XXX
29	8	→ 8	29	李九	新竹市XXX
63	9	→ 9	63	陳七	台北市XXX
98	10	→ 10	98	張林	新竹市XXX

索引

檔案

在未排序的直
接索引上只能
作循序搜尋

根據
Number
排序過的
索引

Number	Record#	Record #	Number	Name	Address
29	8	→ 1	70	王五	台北市XXX
31	4	→ 2	81	張三	台北縣XXX
42	3	→ 3	42	江水	新竹市XXX
51	6	→ 4	31	陳文	台北市XXX
63	9	→ 5	72	李四	台北市XXX
70	1	→ 6	51	趙大	新竹市XXX
72	5	→ 7	90	黃扁	台北縣XXX
81	2	→ 8	29	李九	新竹市XXX
90	7	→ 9	63	陳七	台北市XXX
98	10	→ 10	98	張林	新竹市XXX

索引

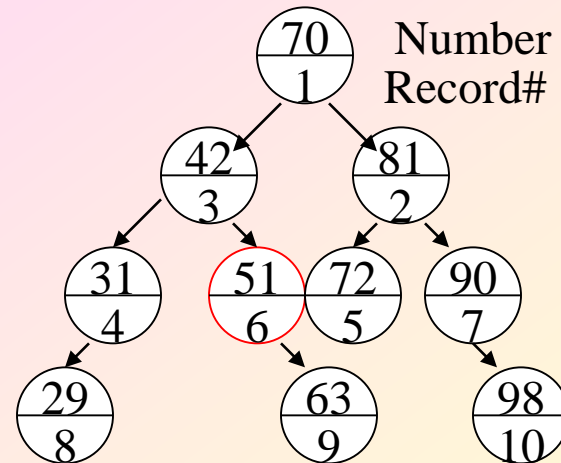
檔案

在已排序的索引
上用Number 90
為鍵值作二分搜
尋，再根據
Record# 7找到
檔案中對應的記
錄

二元搜尋樹的索引

檔案			
Record#	Number	Name	Address
1	70	王五	台北市XXX
2	81	張三	台北縣XXX
3	42	江水	新竹市XXX
4	31	陳文	台北市XXX
5	72	李四	台北市XXX
6	51	趙大	新竹市XXX
7	90	黃扁	台北縣XXX
8	29	李九	新竹市XXX
9	63	陳七	台北市XXX
10	98	張林	新竹市XXX

索引



要搜尋鍵值為 51 的記錄，根據二元搜尋樹的搜尋法則，

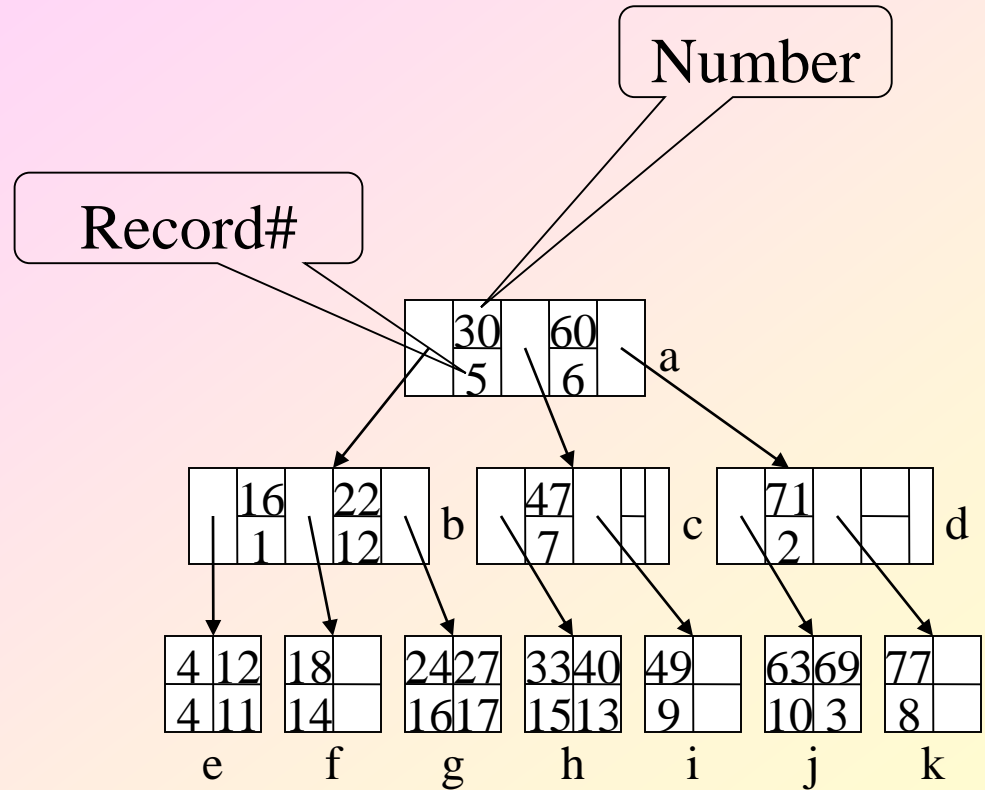
1. 比較 3 次可得到鍵值 51 的記錄在二元搜尋樹中的位置。
2. 再根據這個節點的Record# 屬性，得到鍵值 51 的記錄在檔案的第 6 筆，
3. 再到檔案中抓取第6筆紀錄即可

B樹的索引

檔案

Record#	Number	Name
1	16	王五
2	71	張三
3	69	江水
4	4	陳文
5	30	李四
6	60	趙大
7	47	黃扁
8	77	李九
9	49	陳七
10	63	張林
11	12	湯姆
12	22	捷克
13	40	約翰
14	18	西門
15	33	保羅
16	24	雅各
17	27	馬可

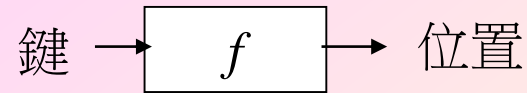
索引



1. 在B樹中作搜尋找到key(Number)的位置。
2. 再根據這個節點的Record# 屬性，得到此記錄在檔案的紀錄編號，
3. 再到檔案中抓取這筆紀錄即可

8-4 雜湊法 (Hashing)

位置 = $f(\text{鍵})$



雜湊函數

- ◎ 雜湊函數：把鍵轉換成位置的函數。因此理想狀況下效率為 $O(1)$ 。
- ◎ 碰撞：把不同鍵轉換成同一位置的現象(現實狀況)
- ◎ 雜湊函數的設計考量：在 “減少碰撞” 和 “計算簡單” 之間作權衡
- ◎ 常見的雜湊函數設計方式：
 - ★ 除法 (division method)
 - ★ 平方去中法 (midsquare method)
 - ★ 折疊法 (folding method)
 - ★ 位數分析法 (digit analysis method)

★除法 (division method)

- 如果資料量大小（鍵值數目）為 n ，我們可以取一個大於 n 的質數 m ，而令 位置 = 鍵值 MOD m 。
- 因此儲存資料的表格大小也必須為 m 。
- 例如 $m = 13$ ，則鍵值為57的記錄將被放在表格的第5個位置（因為 $57 \text{ MOD } 13 = 5$ ）。
- 為什麼表格大小必須為質數呢？原因在於如果 m 不是質數，鍵值 MOD m 得到的數值會較容易碰撞。

如果鍵值數目 $n = 6$ ，鍵值為57, 8, 62, 26, 77, 42，而表格大小 $m = 13$ ，則可以將這些鍵值雜湊為：

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]
26			42		57			8		62		77


$$42 \% 13 = 3$$

★平方取中法 (midsquare method)

我們可以將鍵值平方後，再取中間 k 位數當作位置。例如鍵值為5762，5762的平方為33200644，取任意3位數，如千百十3位數作為位置，得到064。如果鍵值為2642，2642的平方為6980164，獲得016。

★折疊法 (folding method)

如果鍵值的位數很多，我們可以將鍵值折成若干段，再相加之後MOD表格大小。例如 $m = 101$ ，而鍵值為381231596，將鍵值折成3段：381, 231, 596，再相加得到：

$$\begin{array}{r} 381 \\ 231 \\ + 596 \\ \hline 1208 \end{array}$$

MOD 101 = 97
因此得到位置為 97

折疊的方式很多，我們也可以將鍵值折成：38, 1231, 596，再相加得到：

$$\begin{array}{r} 38 \\ 1231 \\ + 596 \\ \hline 1865 \end{array}$$

MOD 101 = 47
因此得到位置為 47

另外相加的方法也很多，可以向左對齊、向右對齊，甚至可以翻轉過來：

$$\begin{array}{r} 38 \\ 1231 \\ + 596 \\ \hline 10991 \end{array}$$

向左對齊

$$\begin{array}{r} 83 \\ 1321 \\ + 695 \\ \hline 2099 \end{array}$$

全部翻轉

★位數分析法 (digit analysis method)

位數分析法是分析所有已知鍵值每一個位數出現的分佈狀況，並且挑選分佈較為均勻的

百萬	十萬	萬	千	百	十	個
5	8	1	1	2	1	1
5	8	0	1	1	5	3
5	7	9	3	2	3	7
2	8	3	2	2	3	9
5	8	1	3	3	1	8
5	8	0	4	1	3	2
5	7	9	5	2	5	4
5	7	9	5	3	2	5

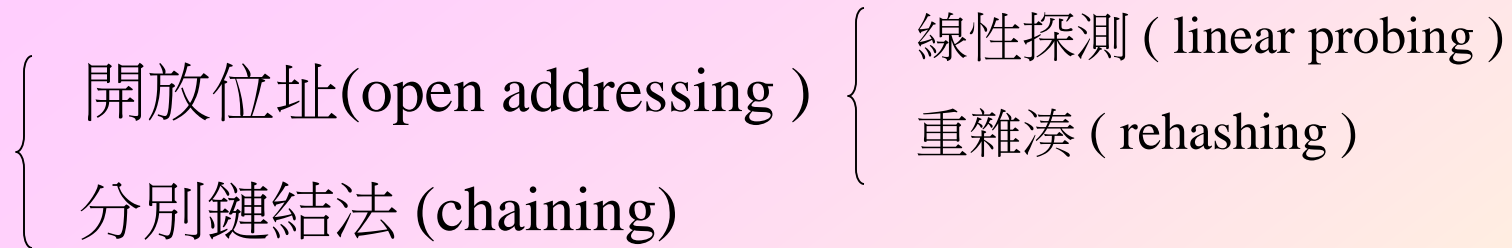
很明顯地百萬位數出現只有 2 和 5，十萬位數出現只有 7 和 8，而百位數只有 1 和 2 和 3。因此我們挑出其餘較為均勻的「萬、千、十、個」位數再 MOD 101。因此：

鍵值5811211 \rightarrow 1111, $1111 \text{ MOD } 101 = 0$

鍵值5801153 \rightarrow 0153, $0153 \text{ MOD } 101 = 42$

鍵值5793237 \rightarrow 9337, $9337 \text{ MOD } 101 = 45$

解決碰撞的方法



◎ 線性探測 (linear probing)

一旦發生碰撞，就往下探測下一個位置。在安排資料時，一旦雜湊出來的表格位置已被佔用，就往下找空格將鍵值放入

如果表格大小 $m = 13$, 鍵值為 62, 42, 13, 57, 8, 60, 73, 雜湊函數為 " 鍵值 MOD m "

1. 加入 62 , $62 \text{ MOD } 13 = 10$

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]
										62		

2. 加入42 , $42 \text{ MOD } 13 = 3$

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]
			42							62		

3. 加入13 , $13 \text{ MOD } 13 = 0$

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]
13			42							62		

4. 加入57 , $57 \text{ MOD } 13 = 5$

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]
13			41		57					62		

5. 加入 8 , $8 \text{ MOD } 13 = 8$

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]
13			41		57			8		62		

6. 加入 60 , $60 \text{ MOD } 13 = 8$, 碰撞 , 往下探測得 table[9] 為空格

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]
13			41		57			8	60	62		

7. 加入 73 , $73 \text{ MOD } 13 = 8$, 碰撞 , 往下探測得 table[11] 為空格

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]
13			41		57			8	60	62	73	

◎重雜湊 (rehashing)

重雜湊和線性探測不同之處，在於一旦發生碰撞，不是往下探測，而是再利用第二個雜湊函數hash_2。如果第二個雜湊值也碰撞，就再利用第三個雜湊函數hash3，以此類推。重雜湊在平均狀況下，會比線性探測所探測的次數要少。

例 8.1

若表格大小 $m = 13$ ，鍵值為62, 42, 13, 57, 8, 60, 73。

而第一個雜湊函數為 $n_1 = \text{key MOD } m$ 。

第二個雜湊函數為 $n_2 = n_1 * \text{key MOD } m$ 。

第三個雜湊函數為 $n_3 = n_2 * \text{key MOD } m$ 。

第 i 個雜湊函數為 $n_i = n_{(i-1)} * \text{key MOD } m$ 。

則安排資料的過程為：

1.加入62, 42, 13, 57, 8之後表格為：

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]
13			42		57			8		62		

2.加入60， $60 \text{ MOD } 13 = 8$ ，碰撞，
重雜湊hash2

$8 * 60 \text{ MOD } 13 = 12$

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]
13			42		57			8		62		60

3. 加入73， $73 \text{ MOD } 13 = 8$ ，碰撞，重雜湊hash2

$(8 * 73) \text{ MOD } 13 = 12$ ，碰撞，重雜湊hash3

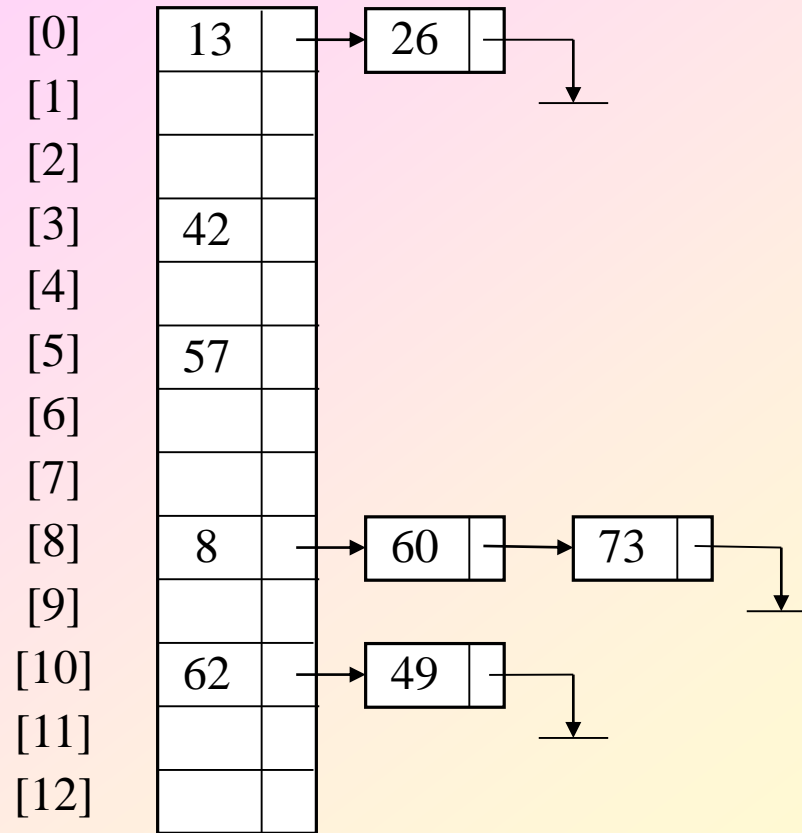
$(12 * 73) \text{ MOD } 13 = 5$ ，碰撞，重雜湊hash4

$(5 * 73) \text{ MOD } 13 = 1$

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]
13	73		42		57			8		62		60

◎分別鏈結法 (chaining)

一旦發生碰撞的時候，就將雜湊值相同的鍵值串在同一個串列



可擴充雜湊

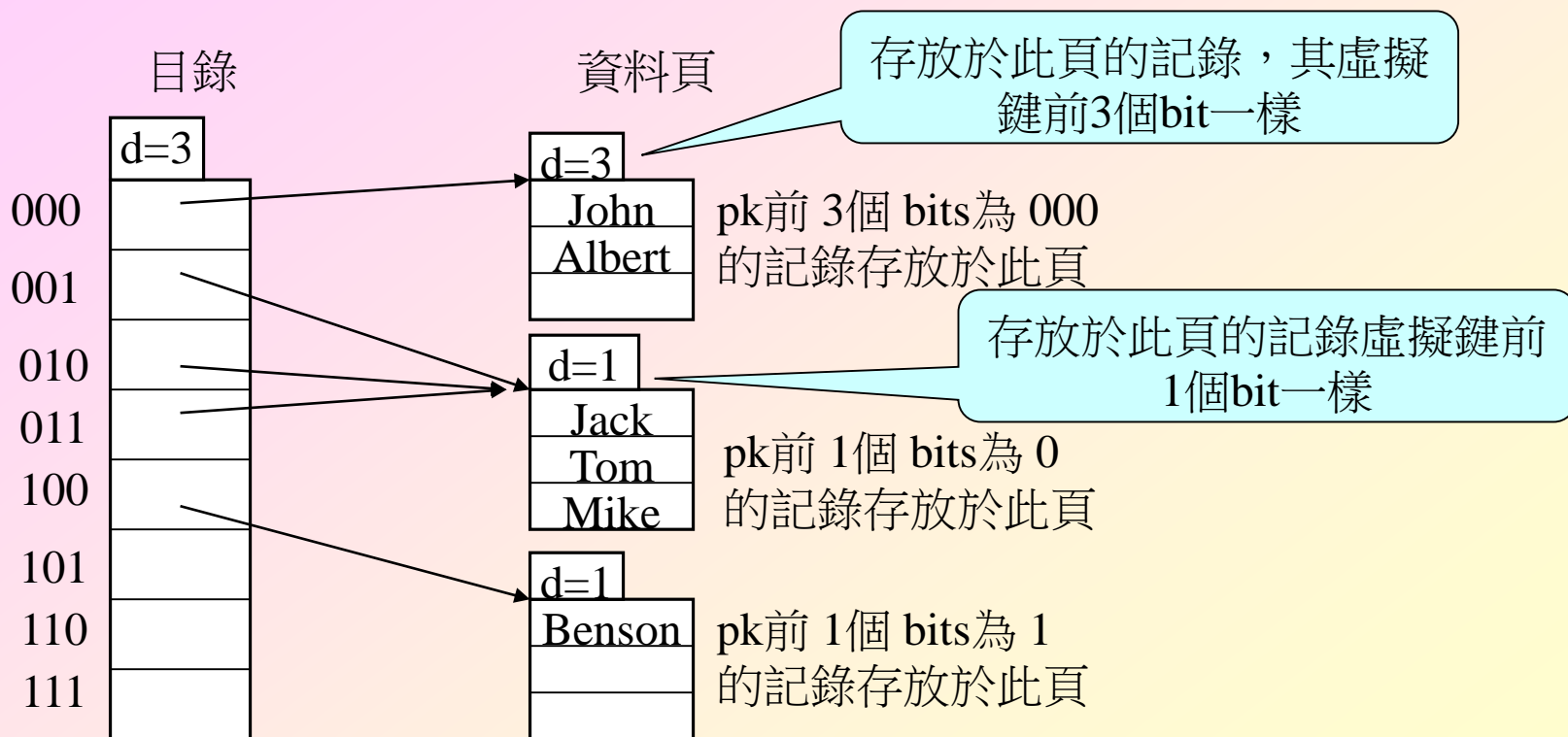
特性

1. 雜湊鍵值可以動態擴充，不需將原有資料進行重新雜湊。
2. 保證磁碟存取次數不超過2次。因為記錄通常儲存在輔助記憶體例如磁碟中，因此減少磁碟存取次數就是減少搜尋時間。

方法

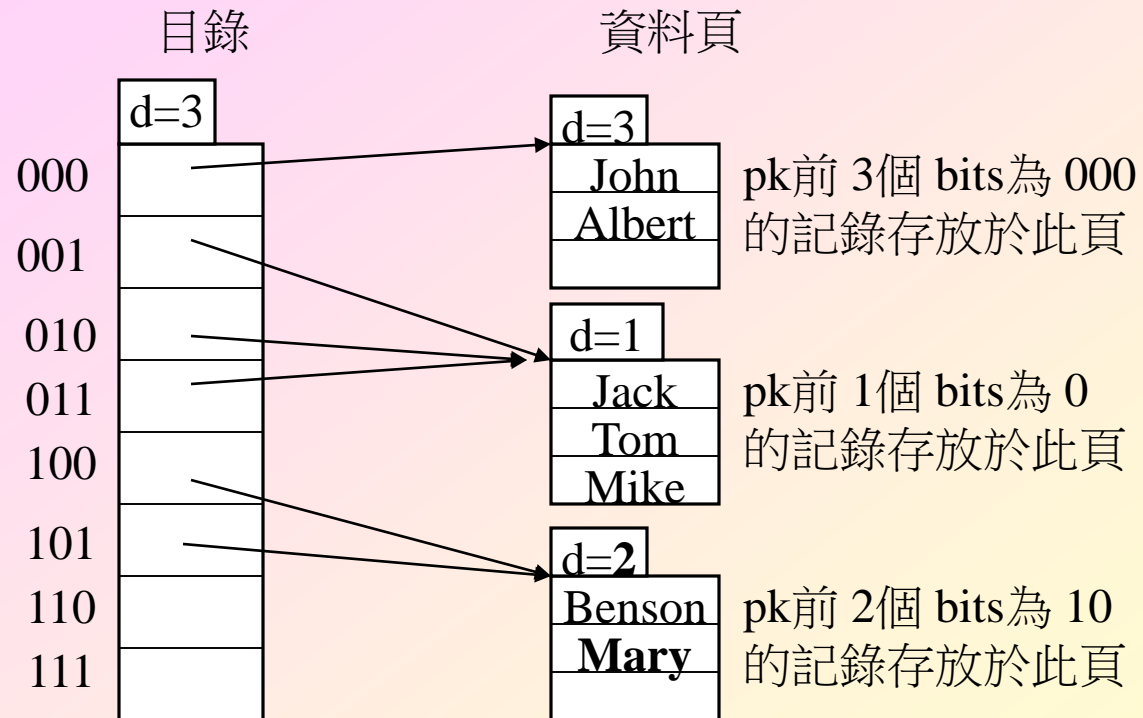
1. 將記錄的鍵值 key ，經過雜湊函數 h 計算得到雜湊值 pk ，稱為虛擬鍵 (pseudo key)
2. 取 pk 的前面 d 個位元 (bits) 構成目錄，目錄中有指標指向資料位址

假設有6筆記錄，鍵值分別為 Tom, Jack, John, Mike, Benson, Albert。
經過雜湊函數計算，取3個位元後分別為 010, 001, 000, 011, 100, 000。
同時假設一個資料頁能存放3筆記錄，則目錄和資料頁將變為



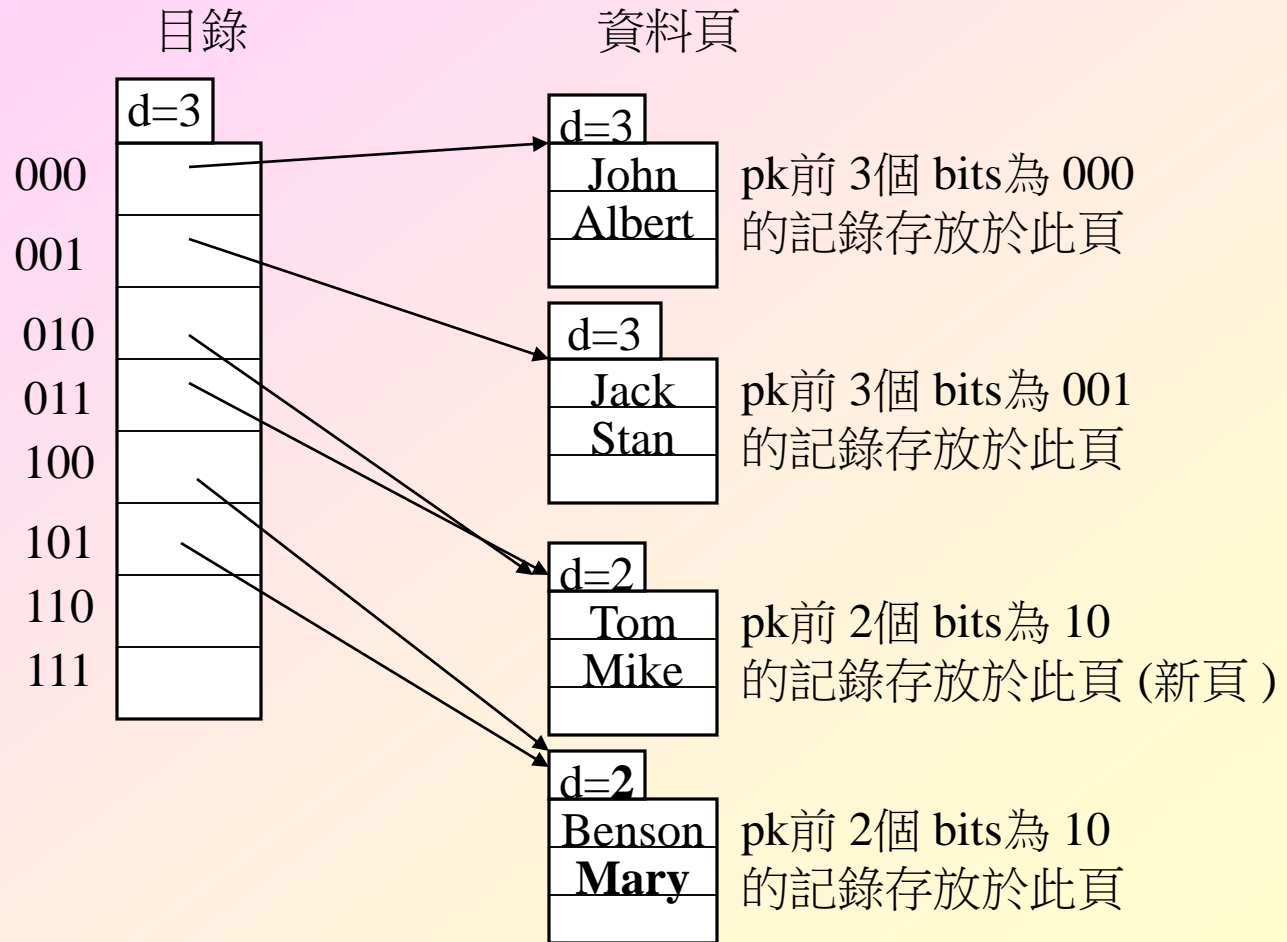


加入新記錄的鍵值為**Mary**，雜湊後取前 3 個bits為101



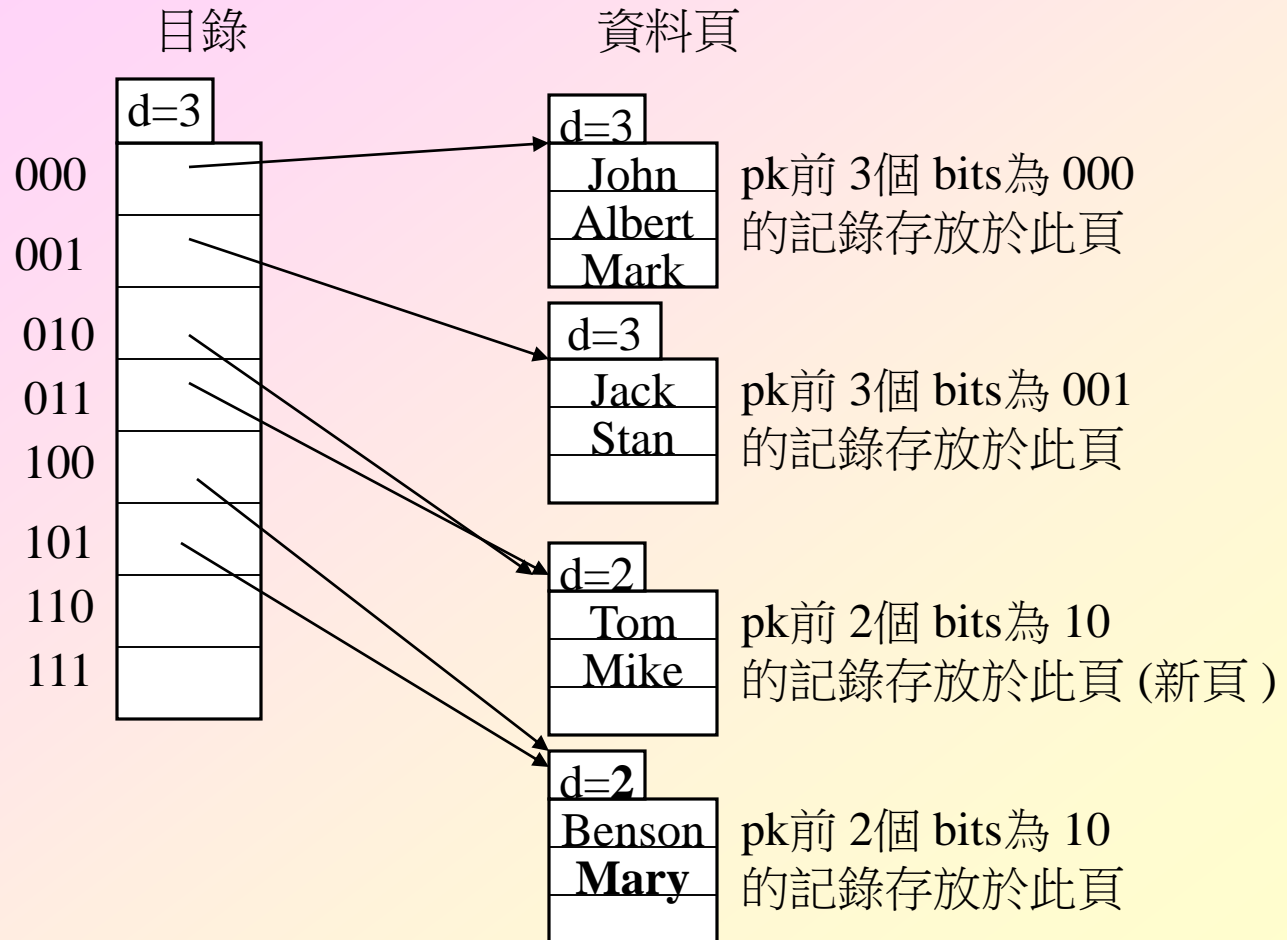


加入新記錄的鍵值為Stan，雜湊後取前 3 個bits為001





加入新記錄的鍵值為**Mark**，雜湊後取前 3 個bits為000





加入新記錄的鍵值為Luke，雜湊後取前 3 個bits為000

