資訊管理學系 陳士杰老師

資料庫系統管理
Database System Management

檔案組織
File Organization

國立聯合大學
NATIONAL UNITED UNIVERSITY

Outlines

- 資料儲存格式
- 檔案組織

【講義：Ch. 3】

【原文：Ch. 13】

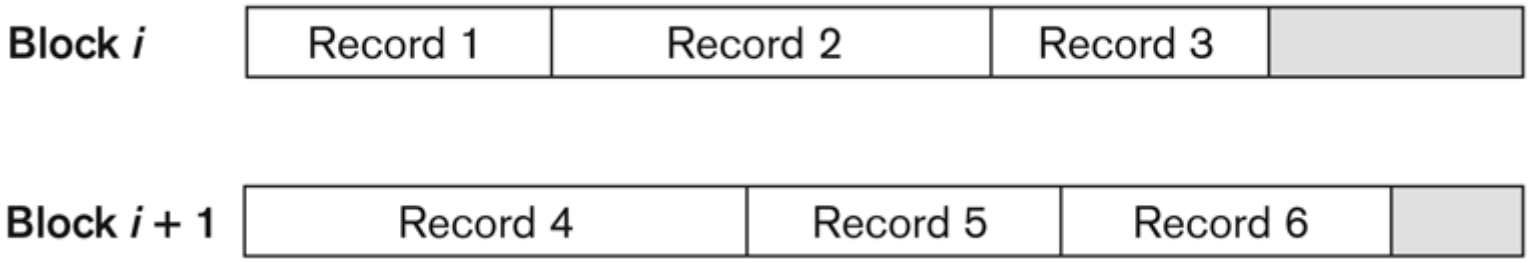
資料儲存格式

- 資料庫儲存資料的階層：
  - 字元(Character)：為資料庫中資料表示的最小單位。
  - 欄位(Field)：又稱屬性(Attribute)、欄(Column)，用來描述個體的某一個屬性。
  - 記錄(Record)：又稱值組(Tuple)、列(Row)，由多個欄位組成，用以描述一個個體。
  - 關聯(Relation)：又稱表格(Table)，為相關記錄的集合。
  - 資料庫(Database)：電腦化的記錄儲存軟體系統，為相關關聯的集合，透過資料庫管理系統(DBMS)來管理。

- **例：員工關聯表格。**

員工編號	姓名	職稱	性別	出生日期	任用日期	區域號碼	地址	分機號碼	報告人
8111131	陳明明	總經理	男	1966/7/15	1992/11/13	114	台北市內湖區康寧 路23巷	1888	Null
8111261	黃謙仁	工程師	男	1969/3/22	1992/11/26	407	台中市西屯區工業11路	3087	8201141
8112061	林其達	工程助理	男	1971/6/6	1992/12/6	235	台北縣中和市大勇街25巷	2138	8111261
8201141	陳森耀	工程協理	男	1968/11/14	1993/1/14	106	台北市大安區忠孝東路4段	3085	8111131
8203161	徐沛汶	業務助理	女	1963/9/30	1993/3/16	330	桃園縣桃園市縣府路	2234	8312261
8205231	劉逸萍	業務	女	1958/9/15	1993/5/23	111	台北市士林區士東路	2230	8308271
8209241	朱辛傑	業務協理	男	1955/4/3	1993/9/24	114	台北市內湖區瑞光路513巷	2247	8111131
8210171	胡琪偉	業務	男	1963/8/12	1993/10/17	220	台北縣板橋市中山路一段	2238	8308271
8307021	吳志梁	業務	男	1960/5/19	1994/7/2	406	台中市北屯區太原路3段	2236	8308271
8308271	林美滿	業務經理	女	1958/2/9	1994/8/27	104	台北市中山區 一江街	2344	8209241
8311051	劉嘉雯	業務	女	1968/2/7	1994/11/5	111	台北市士林區福志路	2234	8308271
8312261	張懷甫	業務經理	男	1952/9/16	1994/12/26	106	台北市大安區仁愛路四段	2342	8209241
8411151	張若蘭	業務助理	女	1969/1/2	1995/11/15	220	台北縣板橋市五權街32巷	2232	8312261

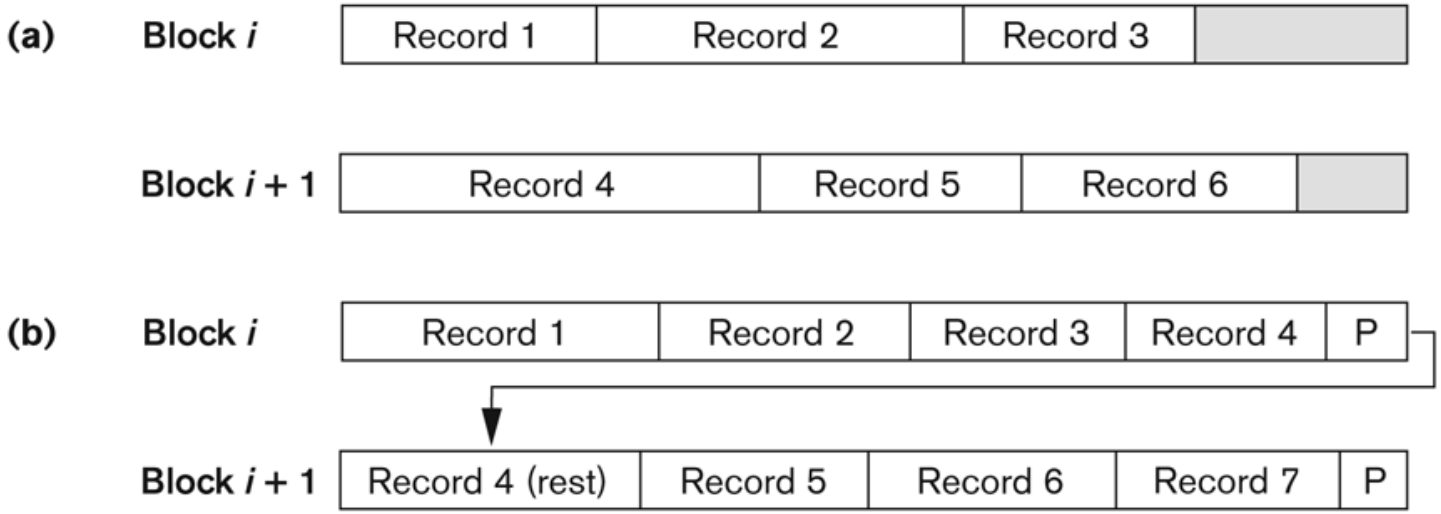
- **磁碟儲存資料的階層：**
  - **位元(Bit)**：電腦儲存資料的最小單位，以二進位表示。
  - **位元組(Byte)**：1 byte = 8bits，為一般電腦從事資料處理的最小單位。
  - **區塊(Block)**：磁碟與記憶體間儲存與傳送資料的單位。磁碟與記憶體的儲存空間皆可切割成許多區塊。若資料記錄的大小較區塊小，則每個區塊可以容納數筆記錄。



記錄儲存的方法

- 固定長度(Fixed Length)
  - 在一個Block中，每一筆記錄長度皆相同
- 可變長度(Variable Length)
  - 在一個Block中，每一筆記錄長度不完全相同
- 不可分割(Unspanned)
  - 一筆記錄於儲存時，不允許被分割與跨越區塊邊界
- 可分割(Spanned)
  - 一筆記錄於儲存時，允許被分割與跨越區塊邊界
  - 做法：在前一區塊結尾處，以指標指向記錄其餘部份之區塊

Types of record organization. (a) Unspanned. (b) Spanned.



- Fixed Length與Unspanned：
  - 優點：容易維護與管理，存取效率佳
  - 缺點：浪費空間
- Variable Length與Spanned：
  - 優點：節省空間
  - 缺點：不易維護與管理，存取效率差
- 實務上，目前的Relational DB較偏好Fixed Length與Unspanned

- 可變長度的使用時機：
  - 欄位長度大
  - 內容值變化大
  - 空間較時間珍貴

檔案組織

- 以檔案資料的存取方式，區分成以下三種：
  - 循序檔(Sequential File)
  - 雜湊檔(Hashing File)
  - 索引檔(Index File)

循序檔(Sequential File)

- 循序檔的每一筆記錄，是依**加入順序**儲存，記錄本身無順序關係。
  - 另外有一種稱為**排序檔(Ordering File)**的檔案組織，其每一筆記錄是依照**鍵值(Key Value)**加以排序而存入檔案中。有些書籍將此檔案組織稱為循序檔。

- **特性：**
  - 依照檔案加入的順序，將記錄存入檔案的**結尾處**。
  - 新記錄直接插入檔案的結尾，時間複雜度為**O(1)**。
  - 記錄搜尋方式採線性搜尋(Linear Search)，時間複雜度為O((1+n)/2) = **O(n)**。
  - 刪除做法有以下兩種，但皆會造成磁碟空間的浪費，且需要資料重組的動作。
    - 複製含有欲刪除記錄之區塊至緩衝區(Buffer)，自緩衝區刪除記錄，再寫回磁碟區塊中。
    - 每一筆記錄均加入一個刪除標記，以表示記錄有效或已被刪除。

- 優點
  - 適合批次作業
  - 程式設計簡單
  - 可儲存於循序性媒體，如：磁帶
  - 加入資料十分有效率(插入尾端)
  - 若檔案很少刪除及更新動作時，十分節省空間
- 缺點
  - 搜尋速度慢(線性搜尋)
  - 處理速度慢，不適合即時性應用
  - 記錄更新、刪除時，必須產生額外的暫時檔案

雜湊檔(Hashing File)

- 利用**雜湊函數(Hash Function)**將每筆記錄之鍵值(Key Value)或雜湊欄位(Hash Field)轉換成相對應之**磁碟儲存位址**。
- 將欲插入資料的鍵值或雜湊欄位帶入雜湊函數，轉換成儲存位址，再將資料插入至此位址。若此位址已有資料，則採用碰撞(或稱Overflow)解決方法加以處理。
- 將欲搜尋資料的鍵值或雜湊欄位帶入雜湊函數，轉換成儲存位址，再將資料取出。若此位址的資料並非欲搜尋的資料，則亦採用碰撞(或稱Overflow)解決方法加以處理。

- 優點：
  - 存取效率高 (前提：雜湊函數須設計的好，碰撞少)
  - 適合即時性、線上應用
  - 一般情況下，插入、搜尋、刪除效率皆十分高
- 缺點：
  - 雜湊函數的設計十分重要，否則可能產生嚴重的碰撞問題，造成整體效率的急速下降
  - 演算法設計較為複雜
  - 不適合循序性的媒體

索引檔(Index File)

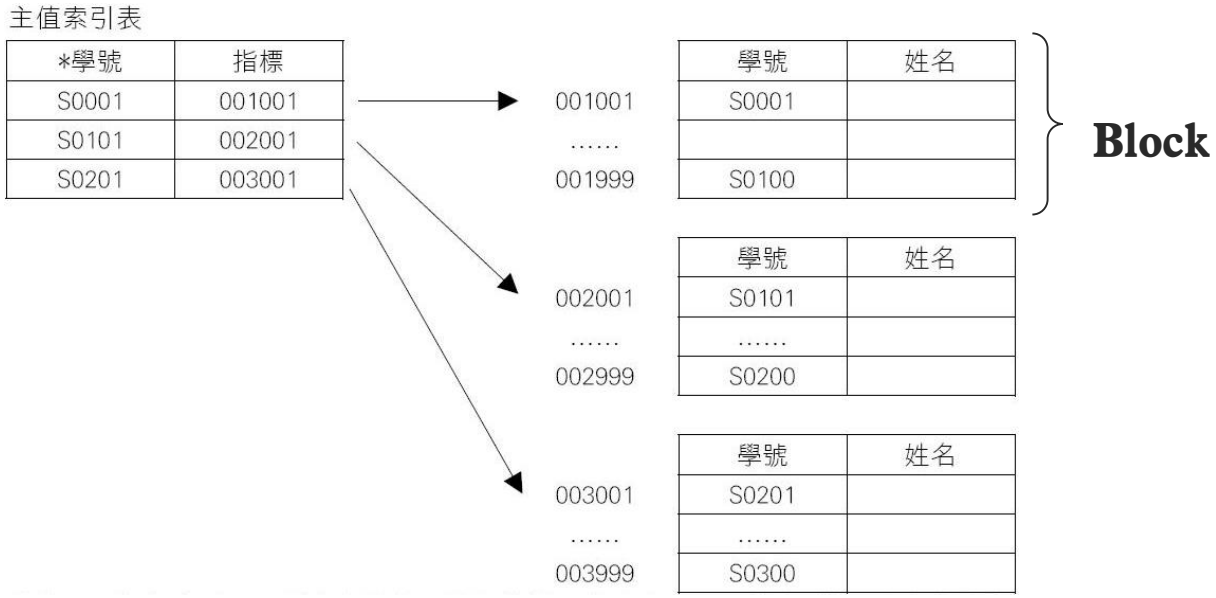
- 索引(Index)是一種資料存取的結構，其目的是在特定的搜尋條件下，用來**加速擷取資料的速度**。建立索引的主要好處是用來**加速查詢**。
- 可分成以下兩種：
  - **單層索引 (Single-level Index)**：依照**有序之索引欄位**建立索引。
    - 主索引 (Primary Index)
    - 次索引 (Secondary Index)
    - 叢集索引 (Clustering Index)
  - **多層索引(Multi-level Index)**：以**樹狀結構**建立索引。
    - B Tree
    - B^+ Tree

- **特性：**
  - 資料實體儲存不一定要依照特定順序，另以索引(Index)表達資料間的順序關係。
  - 插入、更新、刪除皆不需要產生新檔，利用索引可快速地找到所需之記錄。
- **優點：**
  - 存取效率高 (介於循序、雜湊之間)
  - 適合即時性(Real-time)、線上(Online)應用
  - 資料不需事先排序，但循序存取時亦十分有效率
  - 使用廣泛，多數系統皆支援索引結構

- 缺點：
  - 額外的索引佔空間
  - 需耗費額外時間建立索引
  - 程式設計較複雜
  - 資料新增、刪除動作過多時，可能常需要資料重組
  - 不適合循序性的媒體

單層索引(1)：主索引(Primary Index)

- 以主鍵 (Primary Key)為索引欄位
- 記錄依主鍵值做排序
- 為**非密集索引**
- 索引記錄數 = 資料檔區塊(Block)數
  - 索引記錄 = Key value(索引欄位值) + Block Pointer(區塊指標)



- **密集索引(Dense Index)**
  - 每一筆記錄皆有一筆相對應的索引欄位值及區塊指標
- **非密集索引(Non-dense Index)**
  - 並非每一筆記錄皆有一筆相對應的索引欄位值及區塊指標

單層索引(2)：次索引(Secondary Index)

- 分成兩種：
  - 以次鍵 (Secondary Key)為索引欄位
  - 以非鍵值欄位(Non-key Field)為索引欄位
- 上述兩種次索引於使用時，其資料記錄皆未排序。

以次鍵 (Secondary Key)為索引欄位

- 為密集索引
- 索引記錄數 = 資料記錄數

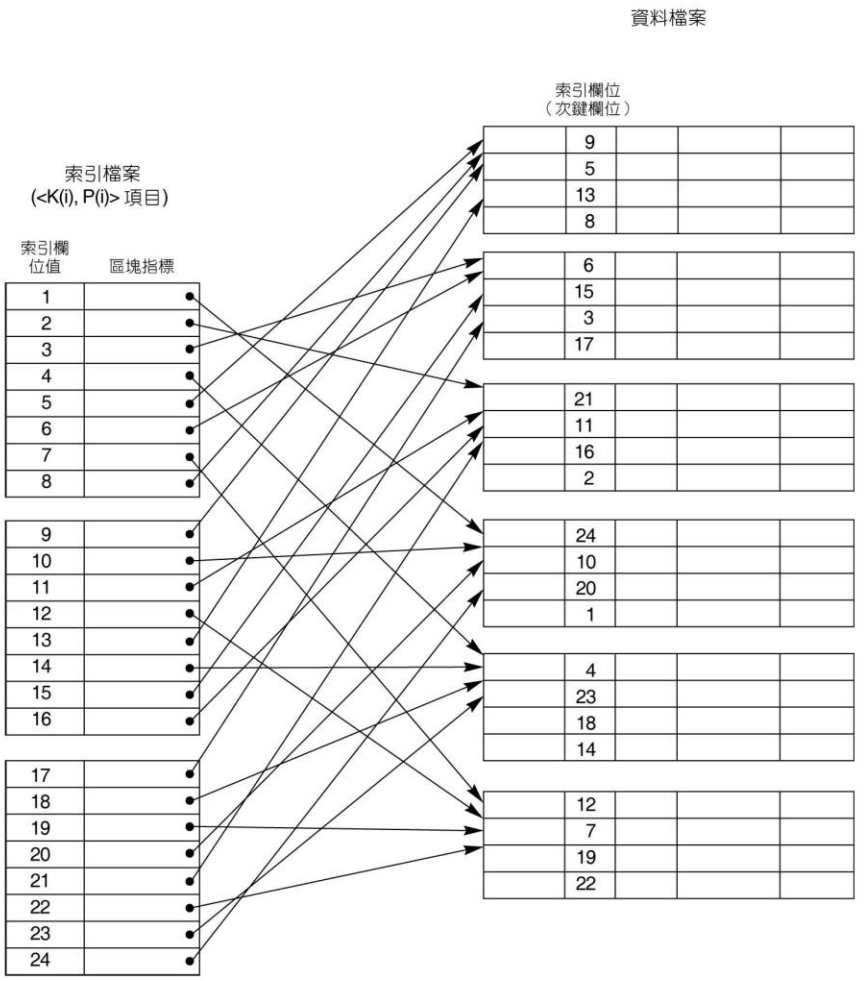


圖 12.10 針對檔案的非序鍵值欄位的緊密次索引（有區塊指標）

以非鍵值欄位 (Non-key Field)為索引欄位

- 為非密集索引
- 索引記錄數 = 相異索引欄位數

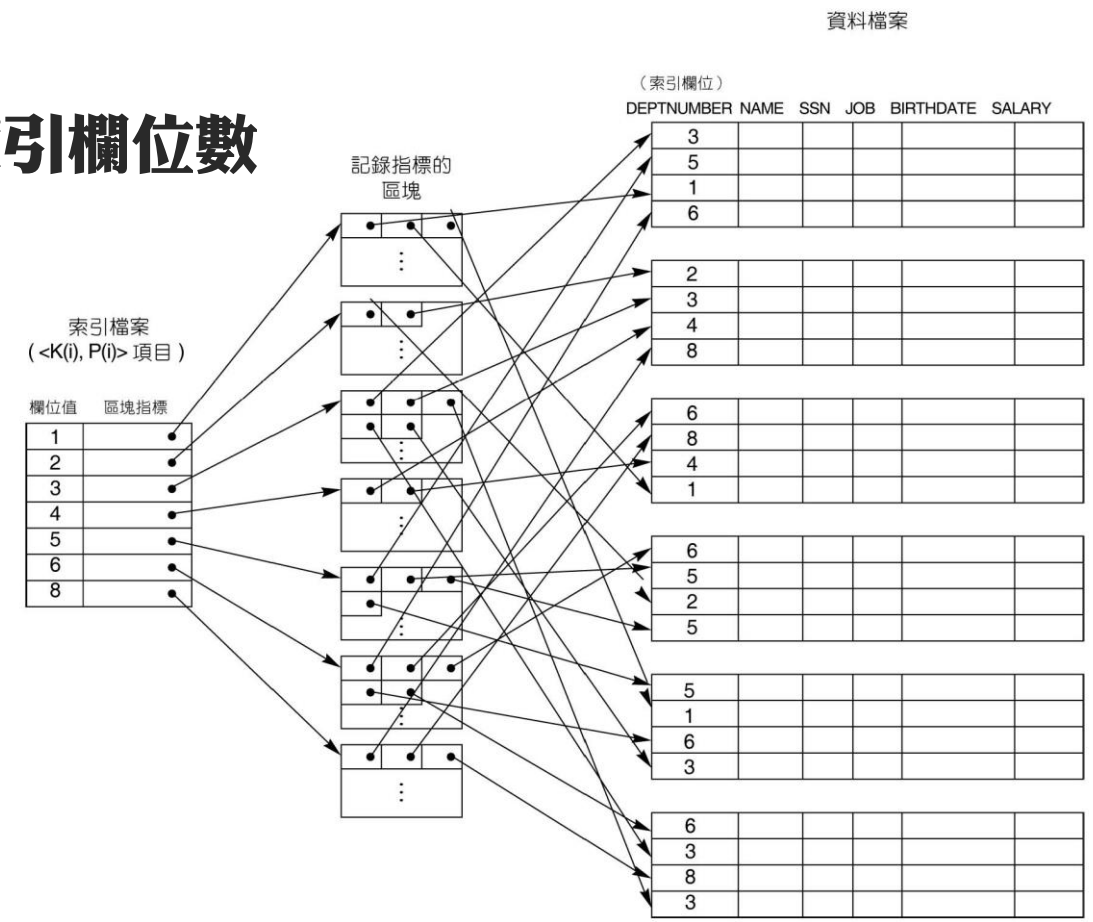


圖 12.11 使用一個間接層來建置在非鍵值欄位上的次索引（具有區塊指標），使得索引項目是固定長度，而且具有唯一的欄位值

單層索引(3)：叢集索引(Clustering Index)

- 以非鍵值欄位 (Non-key Field)為索引欄位
- 記錄依非鍵值之索引欄位做排序
- 為非密集索引
- 索引記錄數 = 相異索引欄位數

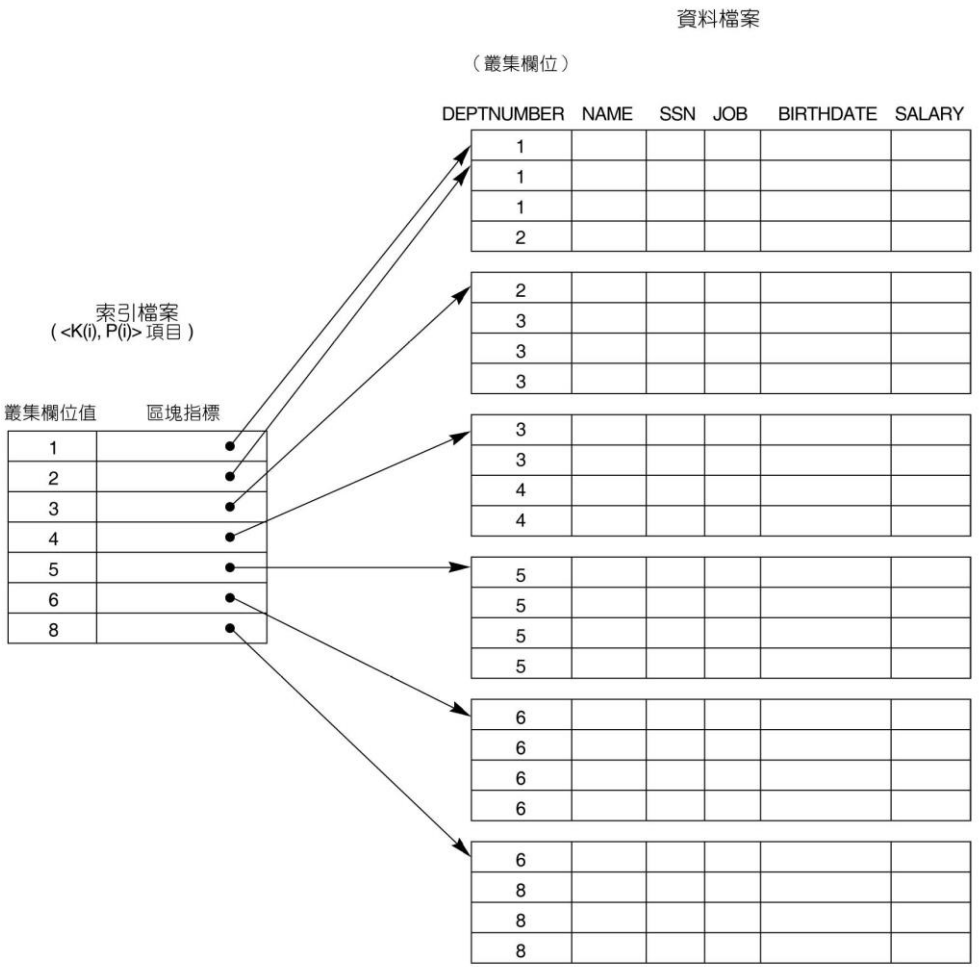


圖 12.8 針對EMPLOYEE檔案的非鍵值有序欄位DEPTNUMBER的叢集索引

	索引欄位有用來排序檔案	索引欄位沒有用來排序檔案
索引欄位是鍵值	主索引	次索引（鍵值）
索引欄位不是鍵值	叢集索引	次索引（非鍵值）

多層索引(Multi-Level Indexes)

- 多層索引是針對單層索引的索引表本身再建立一層主索引；此時原始的索引檔案稱作第一層索引(First-level Index)，而索引的索引則稱作第二層索引(Second-level Index)，以此類推。
- 可適用於各種類型的索引，不管是主索引，叢集索引或次索引，只要第一層索引的每個索引項目都有唯一的K(i)且為固定長度即可。
- 為了要保有使用多層索引好處，同時也降低索引插入與刪除的問題，通常會採用B樹和B^+樹資料結構來實作，這種方法稱為動態多層索引。

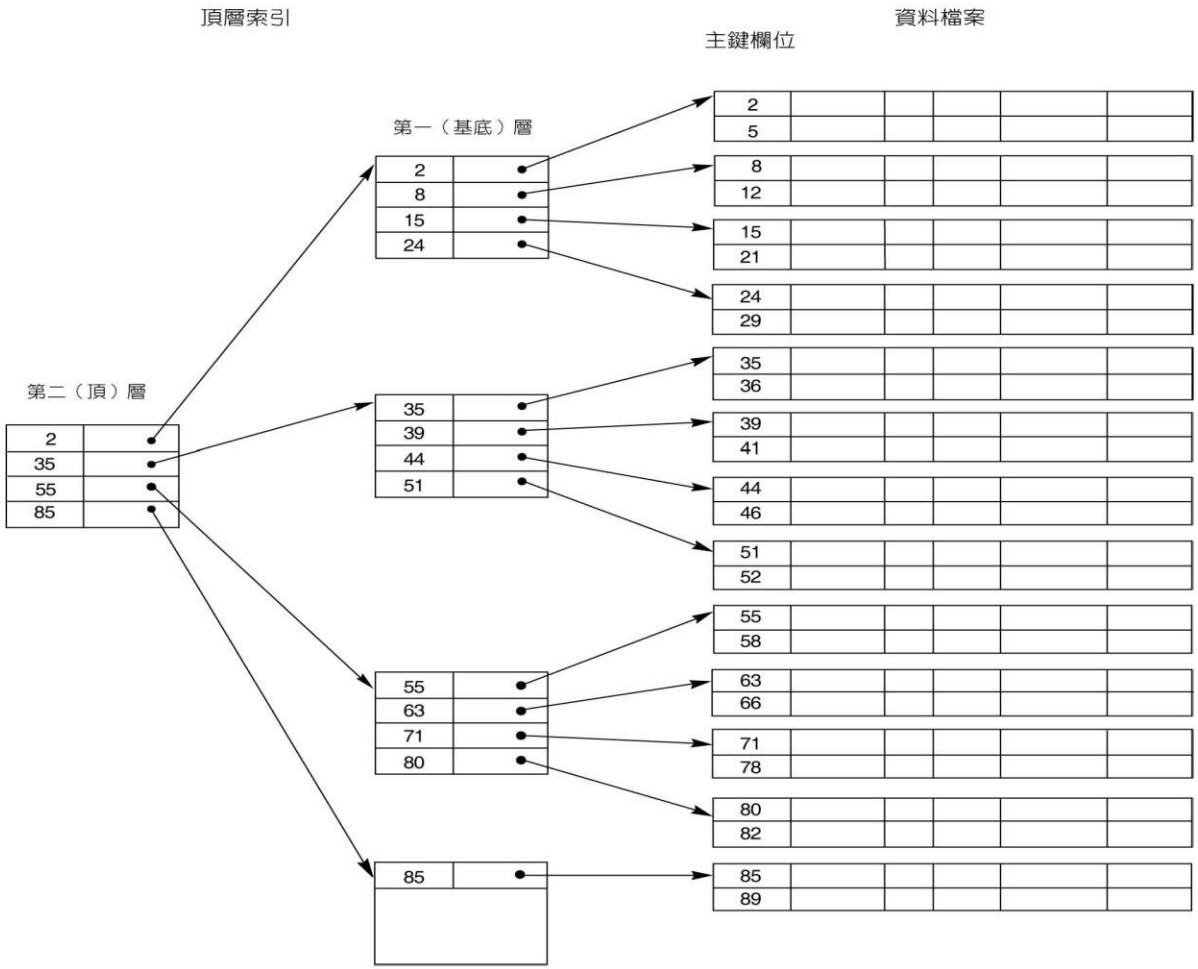
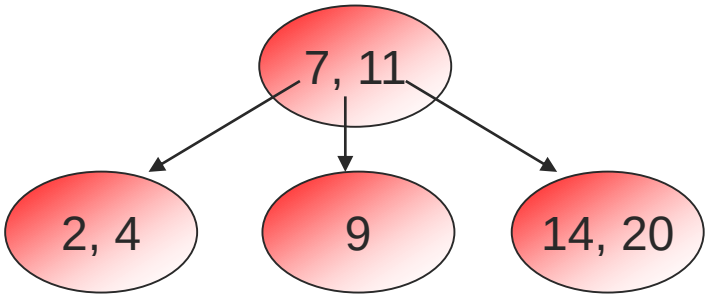
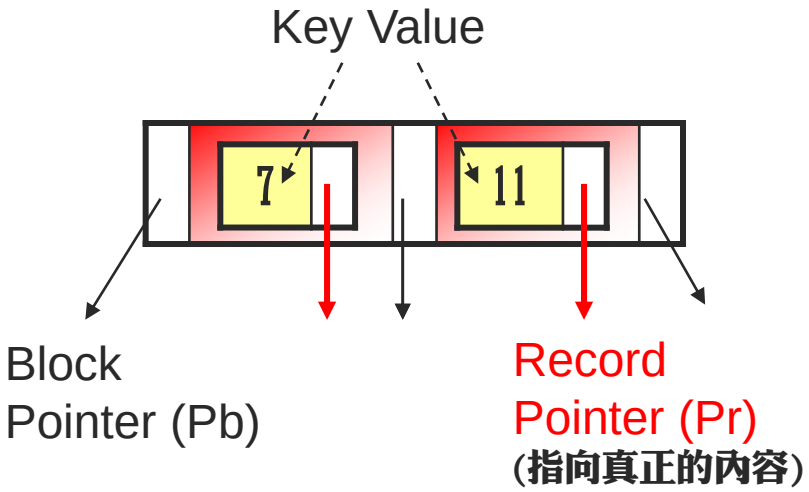


圖 12.12 一種兩層的主索引架構

動態多層索引：B tree

- B tree中的每個節點即為磁碟中的一個區塊 (Block)
- 以下為分支度3的B-tree節點結構：



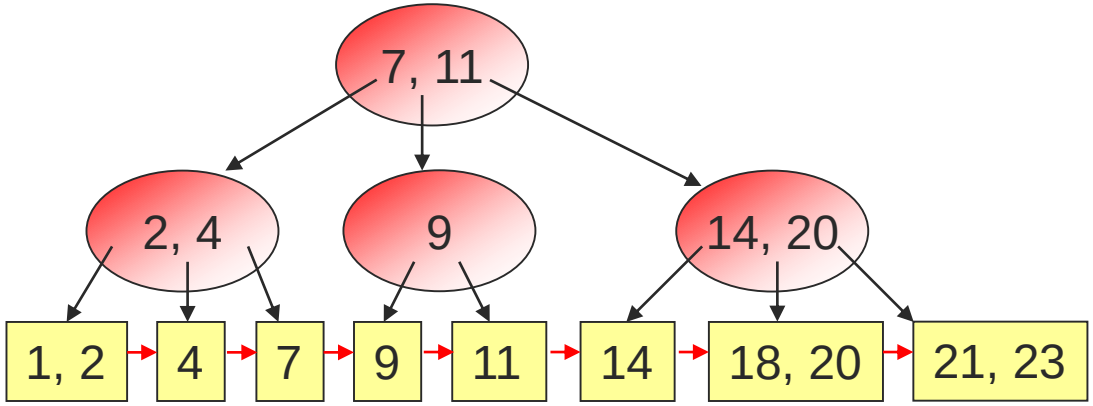


- 假設每個區塊的大小為B，區塊指標的大小為Pb，記錄指標的大小為Pr，鍵值大小為K，每個區塊的分支度最多為p，則：

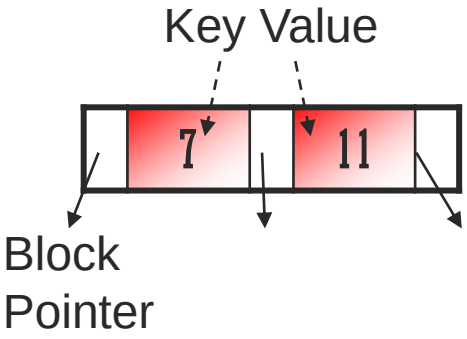
$$(p \times Pb) + (p-1) \times (Pr+K) \leq B$$

動態多層索引：B+tree

- B+ tree中的每個節點即為磁碟中的一個區塊 (Block)
- 以下是分支度為3的B+tree：

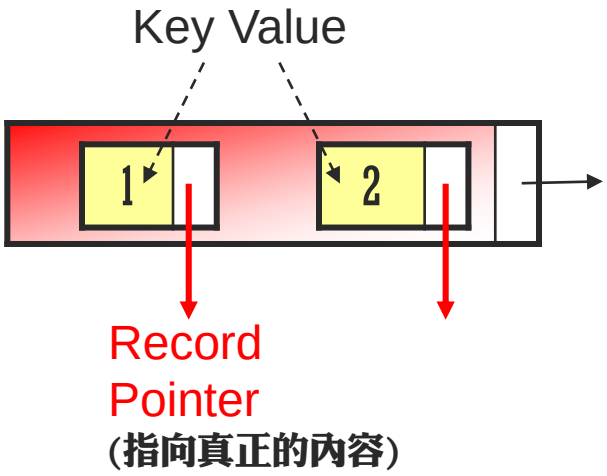


- 分支度為3之B+tree的內部節點(Internal Node)：



■B+tree的內部節點所存放的Key Value，僅為搜尋用的索引值，用以往下搜尋找資料。

- 分支度為3之B+tree的葉節點(Leaf Node)：



■B+tree的Block Pointer可用於資料的循序存取 (較B tree方便)

- 假設每個區塊的大小為B，區塊指標的大小為Pb，記錄指標的大小為Pr，鍵值大小為K，每個區塊的分支度最多為p，p_{leaf}為在Leaf Node內可放的鍵值數量，則：
  - (內部節點)　$p \times Pb + (p-1) \times K \leq B$
  - (葉節點)　$Pb + p_{leaf} \times (Pr+K) \leq B$

比較

	B tree	B^+tree
資料所在	樹中所有節點	Leaf Node
分支度	較小	較大
樹的深度	較大	較小
適合的資料量	小	大
搜尋次數	1(最佳)~樹深(最差)	固定為樹深
循序存取	不適用	適用

❖練習範例 1❖

- 有50000筆員工記錄，員工編號(鍵值)長度為10bytes，區塊大小為1024bytes，區塊指標為6bytes，記錄指標長度為8bytes，且每個節點皆約80%滿，請問：
  - 若採B tree，最多需幾次區塊存取?

Ans:

- 先求出tree的分支度p

  $(p \times Pb) + (p-1) \times (Pr+K) \leq B$

  ⇨ $(p \times 6) + (p-1) \times (8+10) \leq 1024$

  ⇨ $p \leq 43.42$，取$p = 43$ (分支度)
- 因為每個node只存80%，故每個node內區塊指標數量 = 43×80% = 34，鍵值數量 = 33

	Node	Pointer數量	存放鍵值數量
Root	1	34	33
Level 1	34	34×34 = 1156	34×33=1122
Level 2	1156	1156×34 = 39304	1156×33 = 38148
Level 3	39304	…	1297032

合計大於50000

- 因為樹深為4，索引的存取次數為1~4次，若再加上一次記錄存取(即：利用記錄指標將真正資料取出)，故 “最多” 存取5次。

- 續前題，若採B^+tree，最多需幾次區塊存取?

Ans:

- 先求出tree的內部節點與葉節點的分支度：

Internal Node: $(p \times 6) + (p-1) \times 10 \leq 1024 \Rightarrow p = 64$

因node只存80%，故區塊指標數量 = $64 \times 80\% = 51$

Leaf Node: $p_{leaf} \times (10+8) + 6 \leq 1024 \Rightarrow p_{leaf} = 56$

因node只存80%，故Node內鍵值數量 = $56 \times 80\% = 44$

	Node	Pointer數量	存放鍵值數量
Root	1	51	50
Level 1	51	51×51 = 2601	51×50=2550
~~Level 2~~	~~2601~~	~~2601×51~~	~~2601×50 = 130050~~
Leaf Level	2601	…	2601×44=114444

此層內所能存放的鍵值數已大於50000，故不適合再建立此一內部層，應改設為Leaf層。

- 因為樹深為3 (包含leaf)，索引的存取次數為3次，若再加上一次記錄存取(即：利用記錄指標將真正資料取出)，故 “最多” 存取4次。

❖練習範例 2❖

- A file has r = 20,000 STUDENT records of fixed length. Each record has the following fields: NAME(30 bytes), SSN(9 bytes), ADDRESS(40 bytes), PHONE(9 bytes), BIRTHDATE(8 bytes), SEX(1 bytes), MAJORDEPTCODE(4 bytes, integer), and DEGREEPROGRAM(3 bytes). An additional byte is used as a deletion marker. The file is stored on a disk whose block size is 512 bytes. Assuming an unspanned organization, what is the blocking factor?

Ans:

- Blocking Factor(BFR)：每個區塊所能夠存放的記錄數目
- 區塊大小 (B) = 512 bytes
- 每筆記錄長 (R) = 30+9+40+9+8+1+4+4+4+3+1 = 113 bytes
- BFR = $\lfloor B/R \rfloor$ =4
  - BFR為區塊大小除每筆記錄長，若有餘數則無條件捨去到整數(餘數表示無法塞下一筆完整記錄的剩餘空間)。

❖練習範例 3❖

- Consider a disk with block size B =512 bytes. A block pointer is P = 6bytes long, and a record pointer Pr = 7 bytes long. A file has r =30,000 EMPLOYEE records of fixed length. Each record has the following fields: NAME (30 bytes), SSN(9 bytes), DEPARTMENTCODE(9 bytes), ADDRESS(40 bytes), PHONE(9 bytes), BIRTHDATE(8 bytes), SEX(1 bytes), JOBCODE(4 bytes), SALARY(4 bytes, real number). An additional byte is used as a deletion marker. Suppose that the file is ordered by the key field SSN and we want to construct a primary index on SSN. What is the index blocking factor?

Ans:

- Index Blocking Factor (BFRi)：每個區塊所能存放之索引記錄數目
- 索引記錄的組成 = Key Value + Block Pointer
- 每筆索引記錄的長度 (Ri) = SSN+P = 9+6 = 15 bytes
- Block Size (B) = 512 bytes
- Index Blocking Factor = $\lfloor B/Ri \rfloor = 34$

反轉檔(Inverted File)

- 特性：
  - 表格中每個欄位都可建立次索引，各擁有並維護其索引表。
  - 根據任何欄位來做檢索，皆可容易地搜尋到相對應的記錄。
  - 每建立出一個索引表時，其原資料表格中的相對應欄位可以刪除，以節省空間。

- **範例：**
  - **原始學生成績資料表**

記錄所在位址	學號	姓名	物理成績	化學成績
0010	S001	陳一	100	90
0013	S002	林二	72	63
0210	S003	張三	100	53
1005	S004	李四	45	20
0021	S005	王五	15	53
0120	S006	周六	94	100

- 以姓名為索引

姓名索引表

姓名	指標
陳一	0010
林二	0013
張三	0210
李四	1005
王五	0021
周六	0120

學生成績資料表

記錄所在位址	學號	物理成績	化學成績
0010	S001	100	90
0013	S002	72	63
0210	S003	100	53
1005	S004	45	20
0021	S005	15	53
0120	S006	94	100

以物理成績為索引

物理成績索引表

物理成績	指標
100	0010, 0210
72	0013
45	1005
15	0021
94	0120

學生成績資料表

記錄所在位址	學號	姓名	化學成績
0010	S001	陳一	90
0013	S002	林二	63
0210	S003	張三	53
1005	S004	李四	20
0021	S005	王五	53
0120	S006	周六	100

- 以化學成績為索引

化學成績索引表

化學成績	指標
90	0010
63	0013
53	0210, 0021
20	1005
100	0120

學生成績資料表

記錄所在位址	學號	姓名	物理成績
0010	S001	陳一	100
0013	S002	林二	72
0210	S003	張三	100
1005	S004	李四	45
0021	S005	王五	15
0120	S006	周六	94

- 以姓名、物理成績、化學成績為索引

學生成績資料表

記錄所在位址	學號
0010	S001
0013	S002
0210	S003
1005	S004
0021	S005
0120	S006

姓名索引表

姓名	指標
陳一	0010
林二	0013
張三	0210
李四	1005
王五	0021
周六	0120

物理成績索引表

物理成績	指標
100	0010, 0210
72	0013
45	1005
15	0021
94	0120

化學成績索引表

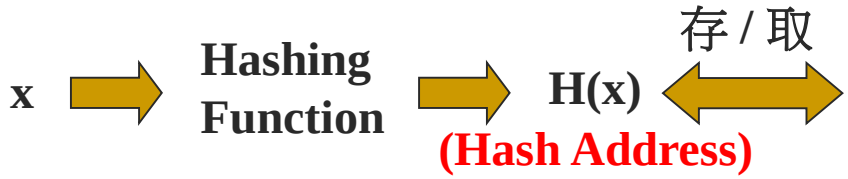
化學成績	指標
90	0010
63	0013
53	0210, 0021
20	1005
100	0120

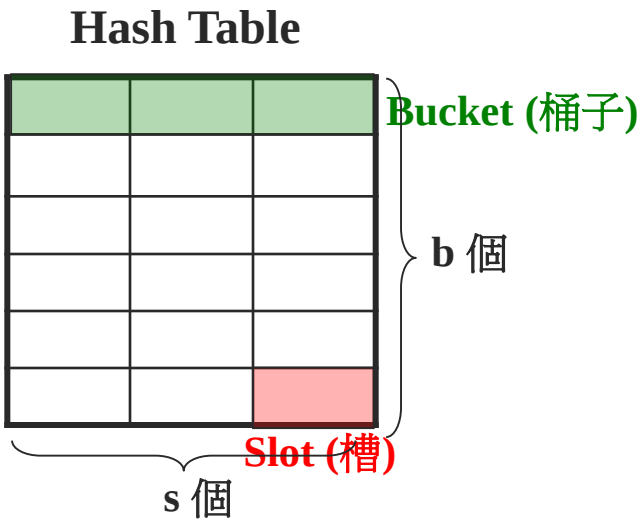
- **完全反轉檔**
  - 關聯表格中有幾個欄位，就建幾個索引檔。
  - 理論上，表格中的每個欄位皆可建立其各自的索引表。然而，在實務上礙於一個資料表格至少要有一個欄位存在的系統要求，所以實作上無法將所有欄位皆刪除掉。
- 優點：
  - 根據特定欄位的某個搜尋條件進行存取時，十分迅速。
  - 適用於多重鍵之索引。
- 缺點：
  - 當使用許多欄位建立多個索引表時，若欲得到一筆完整記錄，必須查詢所有次索引，十分耗時。

補 充

Hashing (雜湊)

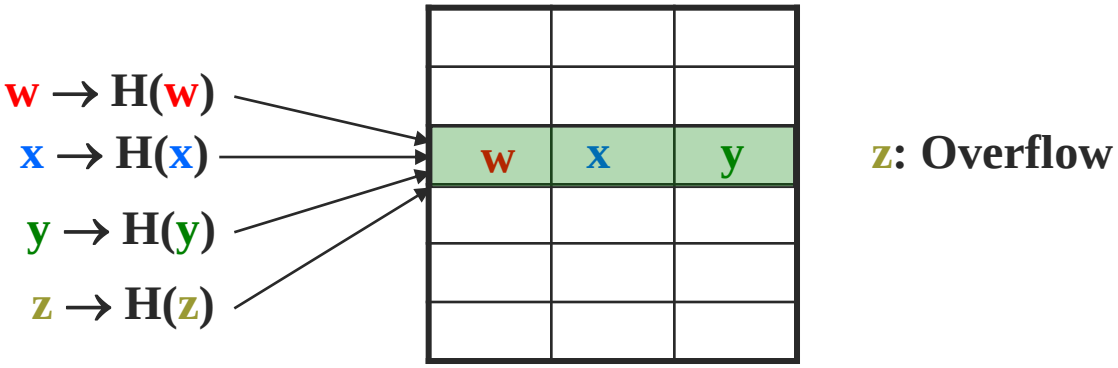
- Def: 為一種資料貯存與搜尋的技術。若要存取某筆資料x，則先將x經過Hashing Function計算，得出Hashing Address，再到Hash Table對應的Bucket中進行存取x的動作。
- Hash Table的結構
  - 由一組Buckets所組成，每個Buckets由一組Slot所組成，每個Slot可存一筆記錄。
  - 圖示:
    - ⇨ Hash Table Size = b × s





相關術語

- **Collision**
  - Def: 不同的資料 (e.g., x與y) 在經由Hashing Function計算，竟得出**相同的Hashing Address** (即 H(x) = H(y)) 稱之。
- **Overflow**
  - Def: 當Collision產生，且**Bucket中無多餘的Slot**可存資料稱之。



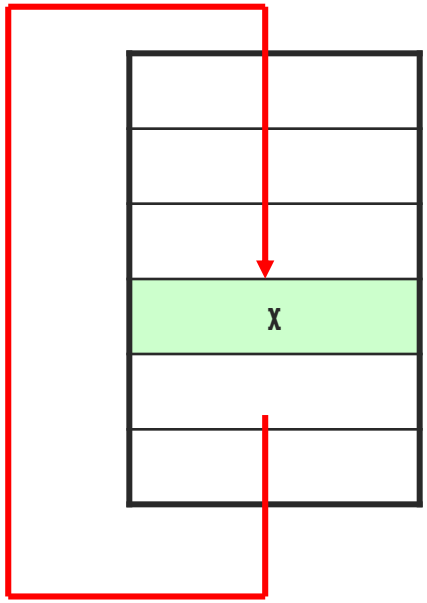
  - 有Collision並不一定有Overflow，但有Overflow，則必有Collision發生。
  - 若Bucket只有一個Slot，則Collision = Overflow

4種常見的Overflow處理方式

- Linear Probing (線性探測)
- Quadratic Probing (二次方探測)
- Rehashing (再雜湊)
- Link List (鏈結串列，或稱Chain)

Linear Probing (線性探測)

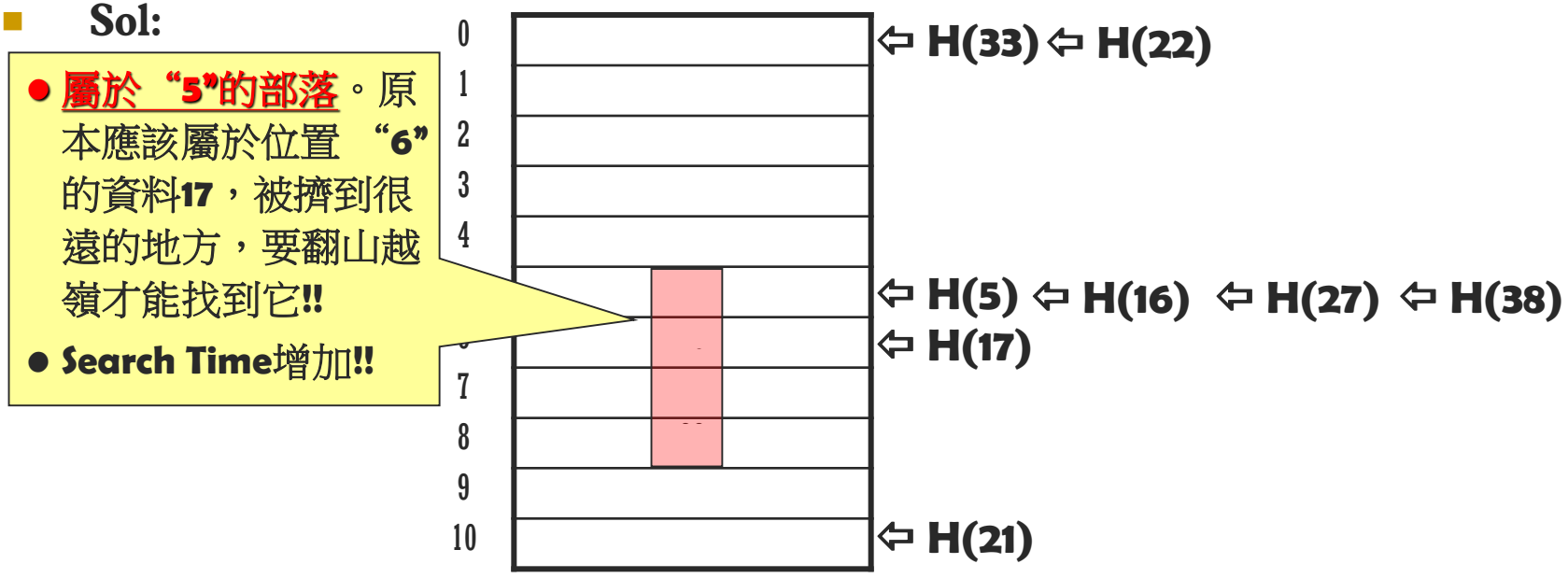
- Def: 又稱Linear Open Addressing。當H(x)發生overflow，則循著H(x)+1, H(x)+2, …, H(x)-1順序，逐步搜尋，直到:
  - 遇見有空的Bucket
  - 已搜尋完一圈為止 (表示Hash Table Full，無法store)
- 圖示:



- Hash Table有11個buckets (編號: 0~10)，每個bucket只有一個slot，假設Hashing Function = **x mod 11**，並採取 “Linear Probing”處理overflow。試依照下列資料次序存入Hash Table，會得到什麼結果?

  5, 16, 33, 21, 22, 27, 38, 17

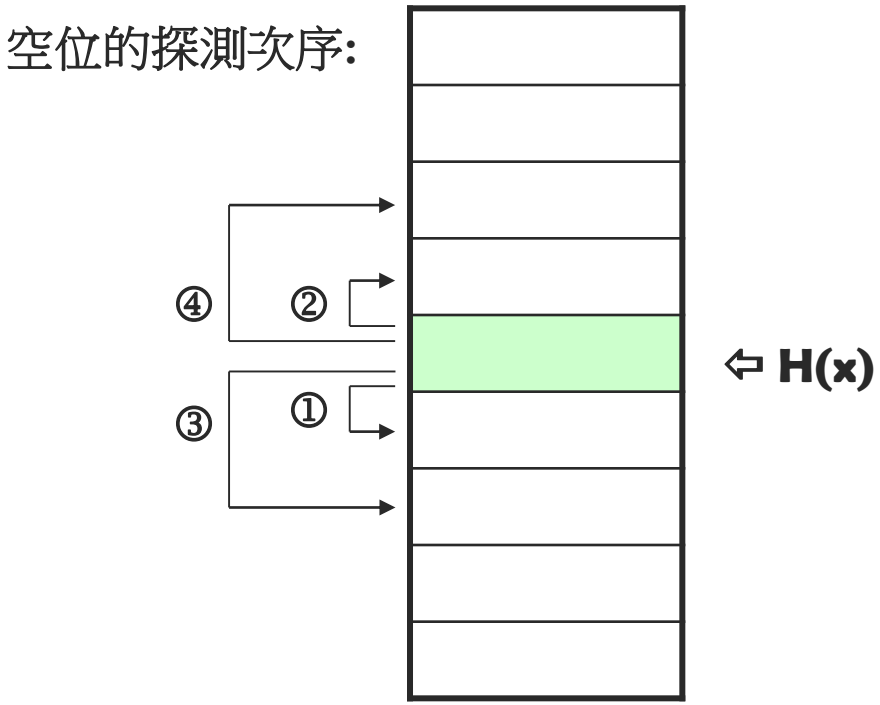
- Sol:



- 缺點: 易形成**資料群聚 (Clustering)**現象，增加Searching Time

Quadratic Probing (二次方探測)

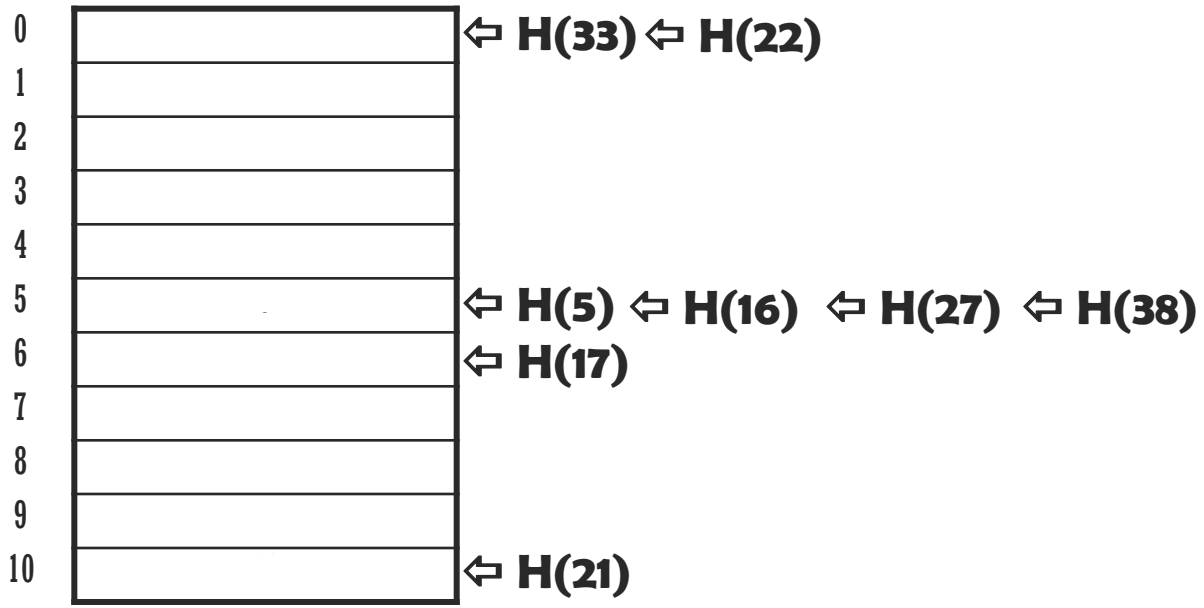
- Def: 為改善Clustering現象而提出。當H(x)發生overflow時，則探測 $(H(x) \pm i^2) \bmod b$，b為bucket數，$1 \le i \le (b-1)/2$
- 圖示:



- 承接上題，並改採 “Quadratic Probing”處理overflow。則Hash Table內容為何?

5, 16, 33, 21, 22, 27, 38, 17

- Sol:



Rehashing (再雜湊)

- Def: 提供一系列的Hashing Functions: $f_1, f_2, f_3, \ldots f_n$。若使用 f_1 發生overflow，則改用 f_2; 以此類推，直到:
  - 沒有overflow發生
  - 全部function用完

Link List (鏈結串列，或稱Chain)

- 將具有相同Hashing Address的資料，以Link list方式串連在同一Bucket中。
  - 承接上題，並採 “Link List”處理overflow。則Hash Table內容為何?

    5, 16, 33, 21, 22, 27, 38, 17
  - Sol: