CH8、Hashing

雜湊

目錄：

Hashing Method

Why Hashing？

Hash Table 的結構

Hash 中常見的問題

Hashing 優點

良好的 Hash Design 應滿足

Perfect Hashing Function、Uniform Hashing Function

常見的設計方法

Middle Equare、Mod(Divison)、Folding Adding(2)、Digits Analysis

Overflow 的處理方式

Linear Probing

Quadratic probing

Rehasing

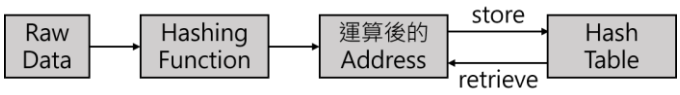
Linked List/Chaining

[補充]：Double Hashing(演算法)

Hashing Method(雜湊法)

Def：為一資料『儲存』及『搜尋擷取』之機制

作法：

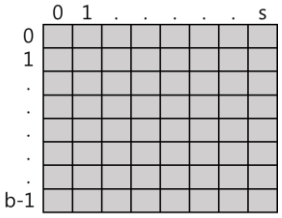


Why Hashing？

在 No Collision 下，Best Case：O(1)，快但需花額外的空間來存 Hash Tabl

Hash Table 的結構

Def：Hash Table 有 b 個 Bucket，各 Bucket 會有 s 個 Slot，各 Slot 可存放一筆記錄。共有：b*s 格子



令 T 為 Identifier 總數

n 為使用 Identifier 個數

b*s 為 Hash Table 的 size

Identifier Density = n/T

Loading Density(α) = n/(b*s)　　*//α 又稱 load factor*

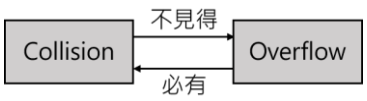
Note：α 愈大，Table 的利用度愈高，相對而言，Collision 機率會上升(缺點)

Hash 中常見的問題

1. Collision(碰撞)：指當 H(x)=H(y)，此時謂之
2. Overflow(溢位)：同 Collision，且 Bucket 中，已無可用的 Slot

Note：

1. Collision 與 Overflow



2. 當 each Bucket 只有一個 Slot，則 Collision = Overflow

Hashing 優點

1. Search 前，不需對 Data 做 Sorting
2. 沒有 Collision 之下，Search Time=O(1)
3. 可做為資料壓縮之用
4. 保密性高，不知道 Hashing Function 的人，無法取得資料

良好的 Hash Design 應滿足

1. 計算簡單
2. Collision 宜少
3. 不會造成 Hashing Table 局部偏重*(Data Cluster)*之情況=>Data 分佈『均勻』較好

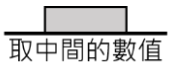
[補充]

1. Perfect Hashing Function
   Def：此函數保證不會發生"Collision"(Input 要知道，才能設計出)
2. Uniform Hashing Function
   Def：此函數之值對應到各 Bucket 的機率是相等的(Collision 機率下降)

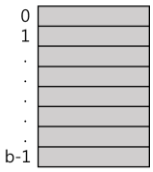
Hashin Function 常見設計方法

1. Middle Equare
2. Mod (or Division)
3. Folding Adding
4. Digits Analysis

1. Middle Equare(平均值取中間項)
   例：1234^2



2. Mod (or Division)(相除取餘數)
   例：有 b 個 Bucket，則 f(x)=x mod b



3. Folding Adding(折疊相加)
   例：有一值：x=123 | 203 | 241 | 112 | 20
   (1) 位移折疊
   123+203+241+112+20 = 699
   (2) 邊界折疊
   123+302+241+211+20 = 897

4. Digits Analysis(數值分析)
   從現有的 Data 分析，得出較好的 Hashing Function 的制訂方式

Overflow 的處理方式

1. Linear Probing
2. Quadratic probing
3. Rehasing
4. Linked List/Chaining
5. [補充]：Double Hashing(演算法)

1. Linear Probing

Def：當產生 Overflow 時，往下一個 Bucket 檢查：有空間，則 save & stop；無空間則往上一格

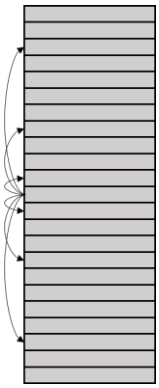
例：一 Hash Table 有 11 個 Bucket(0~10)，令採 Linear Probing 處理 Overflow，針對下列 Data：21, 1, 5, 16, 17, 22, 32, 27，問 Hashing Table 內容為何？

0	22	21 mod 11 ... 10
1	1	01 mod 11 ... 01
2	32	05 mod 11 ... 05
3		16 mod 11 ... 05 (1次)
4		17 mod 11 ... 06 (1次)
5	5	22 mod 11 ... 00
6	16	32 mod 11 ... 10 (3次)
7	17	27 mod 11 ... 05 (3次)
8	27	
9		
10	21	

問題：易形成 Data Cluster，使 Collision 機率、次數提升。故效能較差，O(n)

2. Quadratic probing(二次方探測)
可以改善上述 Data Cluster 之問題
當有 Overflow 發生，利用"$f(x)\pm i^2$" mod b 來找另一地方，其中：$1 \le i \le (b-1)/2$

概念：
先+、再-。最 i++(循環)

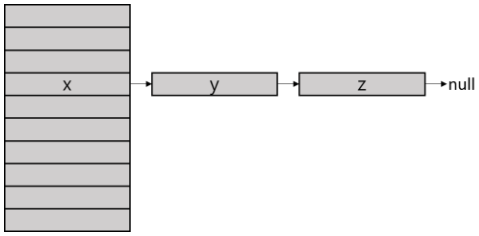


問題：解決了 Data Cluster(Primary)，但一互碰撞公式都一樣，造成愈後面進來者，碰撞愈多次(Second Cluster)

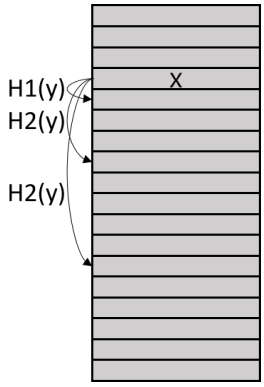
3. Rehasing(再散置)
採用多個 Hashing Function
當有 Collision 或 Overflow，採用別種 Hashing Function 求算

Note：現在較常採用 Double Hashing

4. Linked List/Chaining(鏈結法)
發生 Overflow 時，採 Linked List 將 Data 串接在後面即可
例：



5. [補充]：Double Hashing(演算法)
Def：令 $H_1(x)$ 為 Hashing Function，當 H1(x)發生 Overflow，則探測：
$(H_1(x)+i*H_2(x))\%b$
b：Bucket 數；i=1, 2, 3…(Collision 次數)



同時可解決 Primary Cluster、Second Cluster

例：$H_1(x)=x\%11$、$H_2(x)=R-(x\%R)$，Data：16, 27, …