Hashing

一、 核心概念

在資料結構中，如果我們想在 Array 中找資料，通常需要:

- **循序遍歷搜尋：** O(n)

- **二分搜尋：** O(log n)

Hashing 的目標：實現平均常數時間 O(1) 的搜尋、插入與刪除。

Definition

1. **Hash Function（雜湊函數）：**將一個 Key（如字串或大數字）轉換成整數 Index 的函數。
2. **Hash Table（雜湊表）：**一種儲存「鍵值對」（Key-Value Pairs）的資料結構。它利用雜湊函數計算出的索引，將 Key 轉換為陣列中的 Index，藉此決定資料該存放在哪個位置 (稱為 Bucket，桶子)。

生活例子：字典, 透過首字母(Key)直接翻到該頁數(Index)，而不是從第一頁開始翻。

二、 Hash Function：設計與常見方法

一個好的雜湊函數應該具備：確定性（同一 Key 永遠得到同一 Index）、均勻分佈（減少衝突）、以及高效運算。

常見的計算方法

方法	公式	重點
Division Method	h(k) = k mod m	表大小 m 建議選擇質數， 以減少規律性輸入導致的衝突
Multiplication Method	h(k) = floor(m*(k*A mod 1))	0 < A < 1,優點是較不依賴 m 的選擇， 通常 A=0.618 取黃金分割數效果佳
Folding Method	將 Key 分段後相加， 再取餘數	適合處理很長的 Key， 能打亂原本的數值模式，使分佈更均勻
String Hashing	Polynomial rolling hash	$h(CAT)=(C*p^2 + A*p^1 + T* 1) \bmod m$

三、 Collision 衝突處理與特性分析

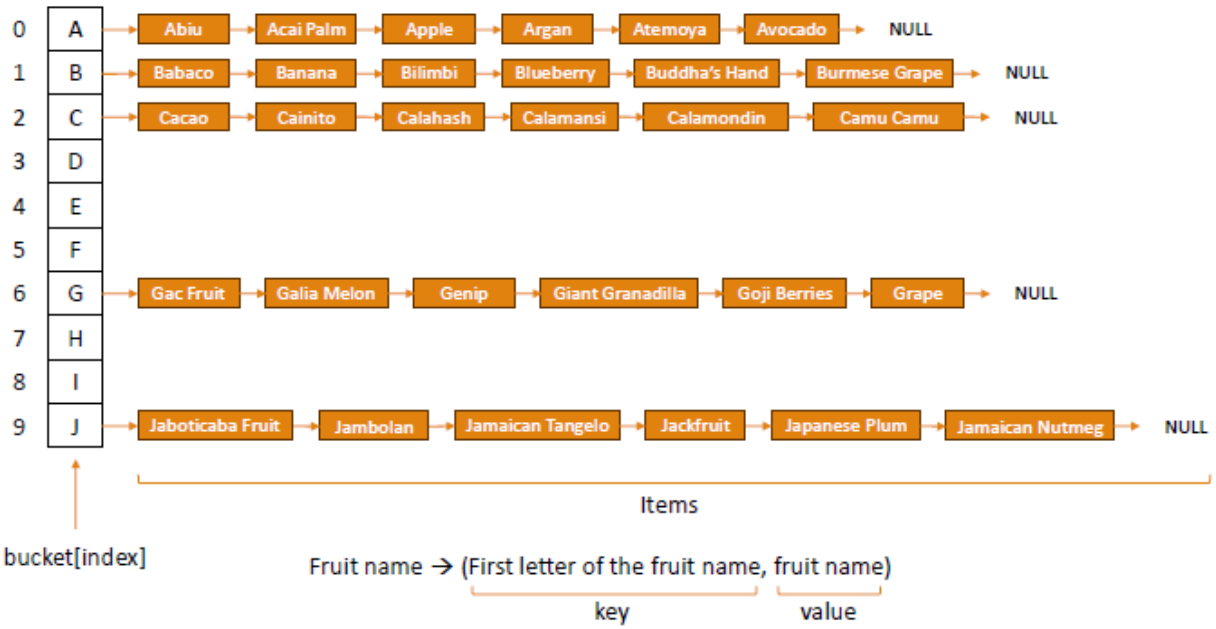
當不同的 Key 經雜湊函數計算後得到同一個 Index，稱為 **Collision (衝突)**。

1. 封閉定址法：單獨鏈結 (Separate Chaining)

核心觀念：每個 Bucket 維護一個鏈結串列。發生衝突時，將新資料掛在該 Index 的串列末端或前端。

- **優點：**實作簡單，且由於鏈結串列是動態的，即使資料量超過陣列大小 (n > m)，程式依然能運作 (只是變慢）。
- **缺點：**
  - 需要額外指標空間 (Pointer overhead)。
  - 如果所有資料都發生衝突，全部擠在同一個桶子裡，雜湊表就會退化成一個長長的**鏈結串列**，搜尋複雜度變成O(n)。
- **效能：** 如果雜湊函數分布均勻，每個桶子裡的鏈結串列會很短。搜尋的時間複雜度約為 O(1 + α)，接近**常數時間**。

  基本結構：Hash Table = Array + Linked List



- Array：作為桶子儲存空間 (Bucket Storage)

- **用途**：陣列提供了固定的連續記憶體空間，每個索引位置被稱為一個**桶子 (Bucket)**。
- **特性**：陣列最核心的優勢在於**索引(Indexing)**。只要知道索引編號，計算機可以在 O(1) 的時間內直接跳轉到該位置。
- **角色**：它是雜湊表的「大門」，透過雜湊函數算出的索引值，決定資料應該進入哪一個桶子。

- Linked List：處理衝突 (Handle Collisions)

- **用途**：當兩個不同的 Key 經過雜湊函數計算後得到相同的索引，就會發生**衝突 (Collision)** 。

- **機制**：雜湊表不會覆蓋掉舊資料，而是在該索引位置維護一個鏈結串列。
- **特性**：鏈結串列是動態的，可以根據需要新增節點，這解決了陣列大小固定、無法儲存多個相同索引資料的問題。

- 運作流程：從 Key 到 Value

a. **雜湊計算**：將 Key 丟入雜湊函數 h(k)，得到一個整數。
b. **取模運算**：使用公式 index = h(k) mod m (m為陣列大小)，將整數映射到陣列的合法索引範圍內。
c. **定位bucket**：直接跳轉到 Array [index]。
d. **操作串列**：
- **插入**：將新的鍵值對節點加到該位置的鏈結串列末端或前端。
- **搜尋**：遍歷該鏈結串列，逐一比對 Key，直到找到匹配的資料或抵達 NULL。

2. 開放定址法 (Open Addressing)

核心觀念： 所有的資料都必須存在陣列內。當衝突發生時，尋找下一個空位。

- 線性探測 (Linear Probing)：

- 公式：(h(k) + i) mod m。
- 缺點：容易產生**初級聚集 (Primary Clustering)**，因為每次只跳一格，只要有一段連續空間被佔據，後面的資料就會全部塞在那，導致效率下降。

- 平方探測 (Quadratic Probing)：

- 公式：(h(k) + c1*i + c2*i^2) mod m。
- 優點：緩解初級聚集，但可能產生**次級聚集 (Secondary Clustering)**，雖然跳的比較開，但初始索引相同的 Key 仍會走相同的探測路徑。

- 雙重雜湊 (Double Hashing)：

- 公式：(h1(k) + i*h2(k)) mod m。
- 優點：**效果最好**，跳躍步長由第二個雜湊函數決定，分布最均勻。

四、雜湊表的深度特性 (Characteristics)

- **排序 (Ordering): 無序，**資料儲存與鍵值大小無關。
- **索引 (Indexing): 支援，**透過 Key 計算 Index 直接存取。
- **重複性 (Duplicates): Key 不可重複**，Value 可重複。
- **記憶體佈局:**
  - 鏈結法：混合式（連續陣列 + 離散指標節點）。
  - 開放定址法：**連續記憶體**（Contiguous），快取效率 (Cache Locality) 較佳。
- **限制 (Limitations):**
  - 不支援「範圍查詢」(Range Query)。

- 需處理衝突與載重因子控制。
- 刪除操作較複雜（特別是開放定址法需標記為 Deleted）。

五、 性能與比較

1.Load Factor (α) : α = = n / m (元素個數/表的大小)，**α** 越高，衝突機率越大，搜尋時間越長 。

a. **鏈結法： α** 可大於 1
b. **開放定址法：α** 必須小於 1（通常建議 **α** < 0.7），否則效能會劇降。

2.Rehashing (重雜湊) : O(n)

當 α 過大時，為了維持 O(1) 效能，會進行：

a. 建立一個約為原表 2 倍大小 的新表 (通常選新的質數)。
b. 將舊表所有資料重新計算 Hash 並搬移至新表。

3.靜態 vs 動態雜湊

特性	靜態雜湊 (Static)	動態雜湊 (Dynamic)
Table Size	固定不變	可根據資料量增長或收縮
α	當負載過高時效能劇降	保持高效
Rehash	必須重建整張表	僅局部桶子分裂
Implementation	Sample	Complex

4.時間複雜度總結

操作	鏈結法 (平均)	開放定址法 (平均,α ≤ 0.7)	Worst
Insert	O(1)	O(1)	O(n)
Search	O(1+ α)	O(1)	O(n)
Delete	O(1)	O(1)	O(n)

六、優缺點與使用場景

優缺點 (Pros & Cons)

- 優點: 搜尋速度極快（常數時間）、適合海量資料。
- 缺點: 無序、最壞情況不穩定、Rehash 成本高。

使用場景 (Use Cases)

1. 快取系統 (Caching): 如 Redis，利用 Key 快速檢索資料。
2. 資料庫索引: 針對「唯一值」欄位建立 Hash Index。
3. 編譯器符號表: 快速查找變數名與函數定義。
4. 去重功能: 判斷某一元素是否已出現過。