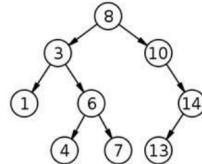


# Hash Table 原理

前情提要～

譬如說要來查詢資料，我們會想到用Binary Search Tree來尋找，而時間複雜度就為 $O(\log N)$ ，N為node數量， $O(\log N)$ 樹高。但是若資料量非常龐大(例如社交平台的註冊會員資料庫)，即使是 $O(\log N)$ 也非常可觀，如右圖。



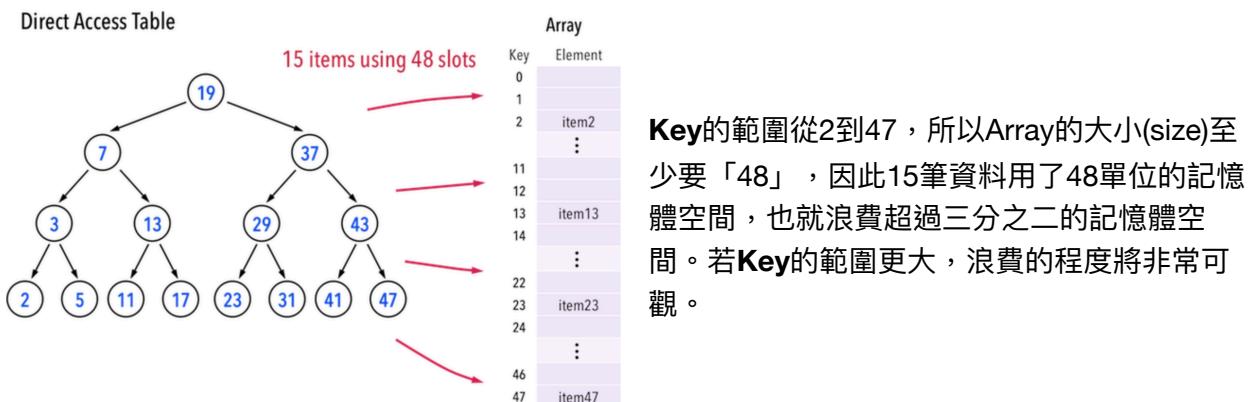
	Average	Worst Case
Space	$O(n)$	$O(n)$
Search	$O(\log n)$	$O(n)$
Insert	$O(\log n)$	$O(n)$
Delete	$O(\log n)$	$O(n)$

那如果用Array實現Direct Access Table呢？

就是把Key當作Array的index，並將Value存放進Array，這樣的實作稱為Direct Access Table。

首先有兩個限制：

1. Key一定要是「非負整數」，才能作為Array的index。
2. 若**Key**的「範圍」非常大，可是**Key**的「數量」相對很少，那麼會非常浪費記憶體空間。



而Hash table就能在時間複雜度為常數的 $O(1)$ 完成查詢且又能避免記憶體空間浪費。

## 雜湊表 (Hash table)

- 是根據鍵 (Key) 而直接查詢在內存存儲位置的資料結構。也就是說，它通過計算一個關於鍵值的函數，將所需查詢的數據映射到表中一個位置來查詢記錄，這加快了查找速度。
- 這個映射函數稱做雜湊函數，存放記錄的數組稱做雜湊表。
- 希望能夠將存放資料的「Table」的大小(size)降到「真正會存放進Table的資料的數量」，也就是「有用到的**Key**的數量」。
- 若有用到的**Key**之數量為n，Table的大小為m，那麼目標就是 $m=\Theta(n)$ 。
- 要達到這個目標，必須引入**Hash Function**，將Key對應到符合Table大小m的範圍內， $index=h(Key)$ ，即可成為Hash Table的index，而Hash Function設計不易，所以很可能發生Collision。（下方會說明**Hash Function**原理）
- Hash Table和Direct Access Table的差別在於Hash Function。

	Bea	Tim	Leo	Sam	Mia	Zoe	Jan	Lou	Ada
	0	1	2	3	4	5	6	7	8
Mia	M	77	i	105	a	97	279	4	
Tim	T	84	i	105	m	109	298	1	
Bea	B	66	e	101	a	97	264	0	
Zoe	Z	90	o	111	e	101	302	5	
Jan	J	74	a	97	n	110	281	6	
Ada	A	65	d	100	a	97	262	8	
Leo	L	76	e	101	o	111	288	2	
Sam	S	83	a	97	m	109	289	3	
Lou	L	76	o	111	u	117	304	7	

假如這裡有9位人名要儲存，擁有辨識度的稱為key（人名），而Value代表著較為廣義的「資料」，例如電話號碼、學籍資料、IP位置等等。同學Mia透過Hash Function，來找出他應該儲存的地點，所以如果要搜尋Ada同學，只要透過Hash Function一樣就能輕鬆地找出他的儲存地點，但是這也會遇到問題，如果有兩位同學經過Hash function後的數字一樣，該要怎麼處理這就是所謂的碰撞（Collision），會在下方說明。

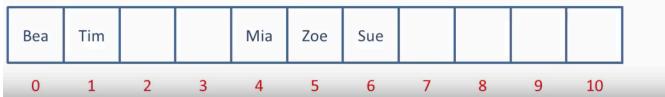
**Hash Table**大概是所有資料結構中應用最廣泛的  
以下解釋當遇到碰撞（Collision）的狀況

Hash Table主要可以分作兩大類：Separate Chaining以及Open Addressing。

Separate Chaining：Hash到array之後，如果多個key被映射到同個位置，就使用Linked List來裝衝突的資料。如右圖



Mia	M	77	i	105	a	97	279	4
Tim	T	84	i	105	m	109	298	1
Bea	B	66	e	101	a	97	264	0
Zoe	Z	90	o	111	e	101	302	5
Sue	S	83	u	117	e	101	301	4



Open Addressing處理衝突時，則是將其中一個key放到別的位置去。根據放的邏輯又分成兩大流派：linear probing及quadratic probing兩種。許多較快的Hash Table會採用Open Addressing這種實作，因為Linked List的指標會對記憶體快取不友善。如作圖可見到Mia與Sue同學都為4，但是4已被Mia佔據，所以往右一格有Zoe再往右一格為空，所以就為Index 6。

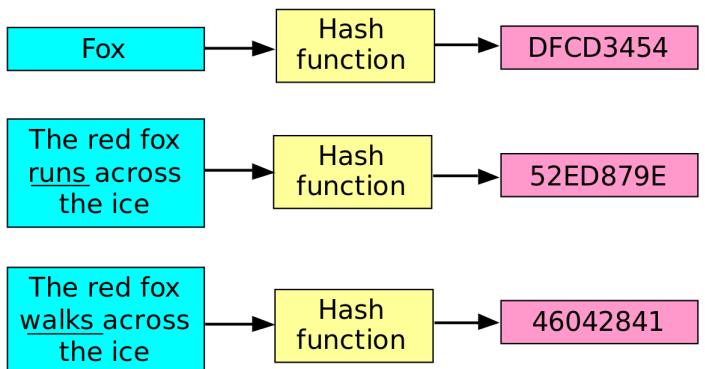
## Hash Function 原理

## 雜湊函式 (Hash function)

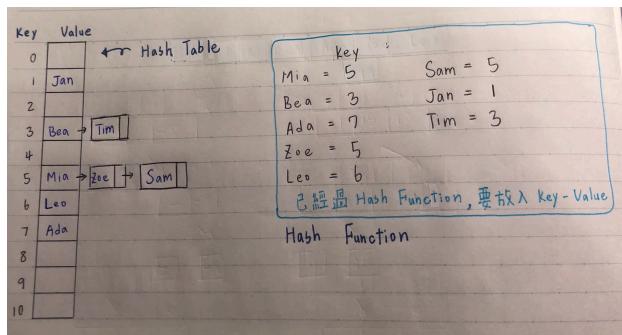
- 雜湊值通常用一個短的隨機字母和數字組成的字串來代表。
  - 該函式將資料打亂混合，重新建立一個叫做雜湊值的指紋。

- 雜湊不是加密！雜湊是因為他的特性很適合來做加密的運算，但真的不等同於加密。
- 如果兩個雜湊值是不相同的（根據同一函式），那麼這兩個雜湊值的原始輸入也是不相同的。這個特性是雜湊函式具有確定性的結果，具有這種性質的雜湊函式稱為單向雜湊函式。
- 雜湊函式的輸入和輸出不是唯一對應關係的，如果兩個雜湊值相同，兩個輸入值很可能是相同的，但也可能不同，這種情況稱為雜湊碰撞（collision），上方有說明。

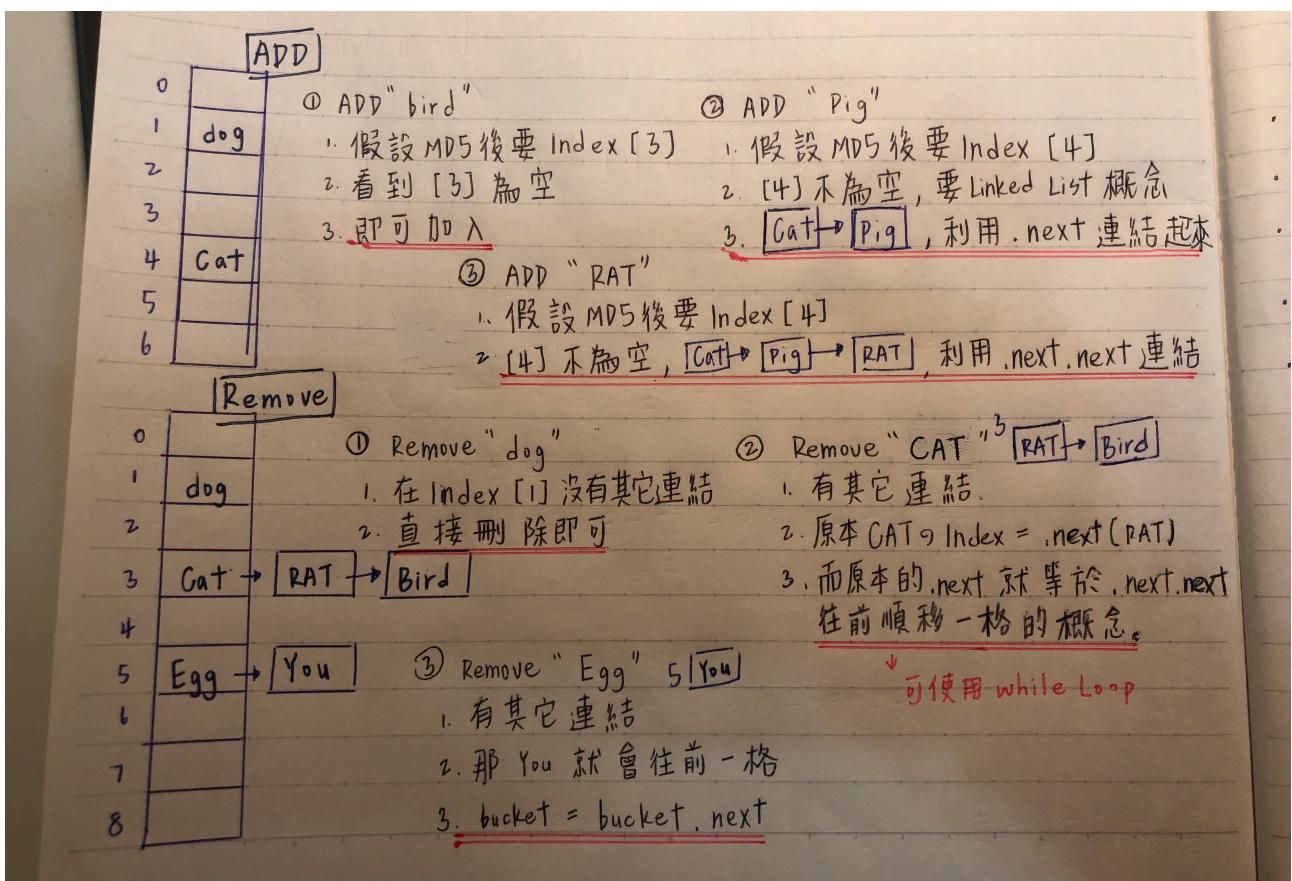
## Input                          Hash sum



## Hash Table 流程圖



假設我今天有任意長度KEY1 跟KEY2，因為太長了，我不想每個都檢查，只要把它們透過HASH轉換的數值相同，就可以確認它們是不是同樣的東西。如果我今天有N個KEY HASHTABLE就是幫我把不同的KEY放進TAB:E裡面，這樣我就可以確認那些有出現過，HASHTABLE:對面多筆資料中，每一種資料只保留一份，且可快速查詢功能。



上圖為，我寫程式碼的流程圖，程式碼幾乎都是照著此流程圖的規則走，以下還有說明。

