

演算法期中報告

Hash Function

姓名:趙家逸

學號:C110154255

班級:四電二乙

一、設計目標

一個好的 Hash Function 必須具備以下條件:

1.1. 均勻分布:

希望從多個輸入進去 Hash Function 得到的多個輸出會是非 normal distribution 的，換句話說，就是得出來的 hash 值是什麼值的機率會是相同的。

1.2. 雪崩測試(Avalanche test):

當我的輸入極為相近(例如:" abc" 和" abd")時，希望得到的 hash 值會是非常分散的。

1.3. 運算速度:

希望能把運算速度壓得越低越好

1.4. 如何解決碰撞 collision 和 overflow:

Collision:當得到的 hash 值已經被使用過了，必須放入其他位置或額外配置。

Overflow:當 hash 值已超過所配置的陣列大小，必須放入其他位置或額外配置。

二、 設計動機與想法

2.1. 均勻分布:

一開始我想到如果使用位元位移的話是否可以做到均勻性，可是試了幾次發現效果並不好，後來找了很多資料發現了 murmurhash 的方式，也就是透過大量的位元位移以及相乘和做 XOR 做到混合位元樣式，然而我發現原版的 murmurhash 值用在這個 case 上還是會出現很多碰撞，尤其是同一個 hash 值，這樣會導致我的 linked list 被拉很長，所以我捨棄了一點點的運算時間，做了比較多的 mix 來讓最終的 hash 值能均勻分布以及把長度變數考慮進來。

2.2. 雪崩測試(Avalanche test):

透過位移位元來拉大相似位元的相似度，並透過將字串分割成多個等份來做累加，做到隨機性。

2.3. 運算速度:

不同字串長度會做不同次迴圈次數，而迴圈內做的只是單純的位元位移及乘法，所以不會太花時間。

2.4. 如何解決碰撞 collision 和 overflow:

Collision: 在 array 後面串上 linked list，原本是打算用平方探測等可以避開區域集中的問題，可是發現這樣可能會導致更多得碰撞連鎖反應，所以最後選擇使用 linked list。

Overflow: 在大數字上，等比例的轉成 array 的大小(函式: scaling)。

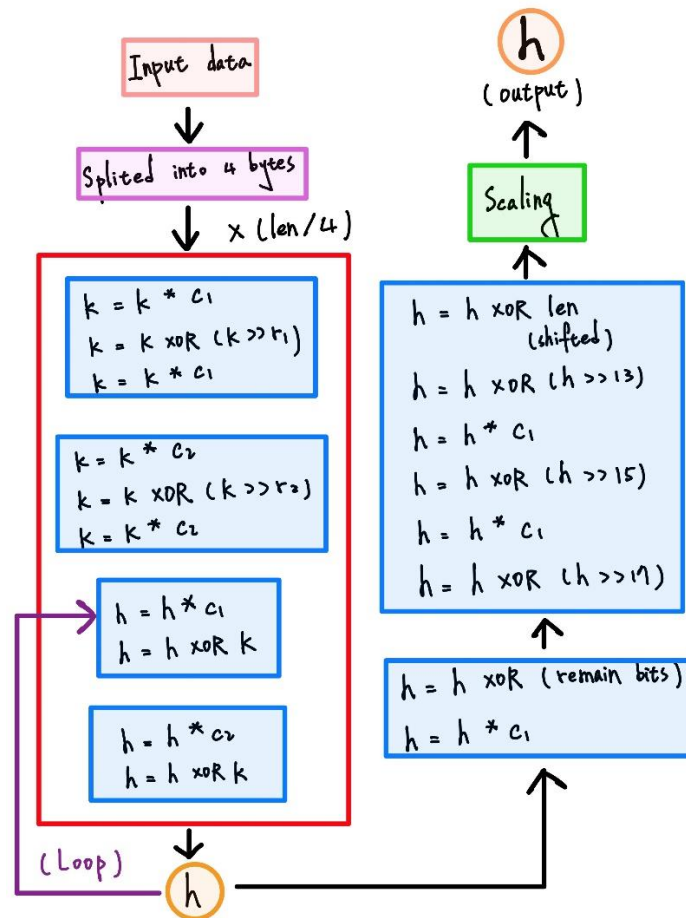


Fig.1 hash function 流程圖

三、 程式說明

3.1. 測試資料:

函式(rand_generate_string)，會回傳 x 筆隨機長度、隨機字元(將它限制在 0~9 和 A 到 Z 和 a~z 之間)的二維陣列，作為測試資料，皆可在輸出看到。

3.2. Hash table:

基礎的 hash function 往往會因原始資料本身相似或有同樣處理方式而 hash 完發現局部群聚，所以使用位元位移和相乘數據混合，並且用 uint32_t(有 4294967295 個數字，也就是 int 的無符號版)，讓 32 個 bit 充分混和，也都會處理到，而不會造成有一些位元總是處理不到，來達到均勻分布的效果。

3.3. output 說明:

- (I) 顯示各個原始字串和它的 hash 值
- (II) x 筆資料經過 hash function 並加入 array 所需時間
- (III) array(Hash table)的資料，若有碰撞，會串接在它的後面(-> 符號)
- (IV) 紀錄 x 筆資料各個的 hash 值(hash_record)
- (V) 碰撞次數(collision number)
- (VI) 找字串的 hash 值 (示範範例:測試資料中的第一筆)

```
Input n and x: 10 8
NO.1 string : j9UwiH
(unscaling = 3631238365 scaling = 8)
NO.2 string : 2TH
(unscaling = 2939508417 scaling = 6)
NO.3 string : 1IR5
(unscaling = 2013964593 scaling = 4)
NO.4 string : m5
(unscaling = 288825114 scaling = 0)
NO.5 string : 0j4Me
(unscaling = 2947539134 scaling = 6)
NO.6 string : QwLeOiRY
(unscaling = 1648468496 scaling = 3)
NO.7 string : WF
(unscaling = 3702037056 scaling = 8)
NO.8 string : KQkZWwf6
(unscaling = 784510301 scaling = 1)
-----
8 data take 0.003 second.
-----
array[0] = m5
array[1] = KQkZWwf6
array[3] = QwLeOiRY
array[4] = 1IR5
array[6] = 2TH -> array[6] = 0j4Me
array[8] = j9UwiH -> array[8] = WF
-----
hash_record = [8,6,4,0,6,3,8,1,]
-----
Collision number = 2
-----
String : "j9UwiH" is at linked list number [1] of the array[8].
```

Fig2. Output 範例

四、實驗記錄

X	collision	N = 100 時
10		0(%)
20		1(%)
30		3.5(%)
40		6(%)
50		11(%)
60		13.5(%)
70		18.5(%)
80		22.5(%)

五、負載係數

經實驗發現，當負載係數到達 0.85 時，碰撞率到了 25%，linked list 會疊加太多，所以可能需要 resize 陣列。

六、分析數據\實驗結果

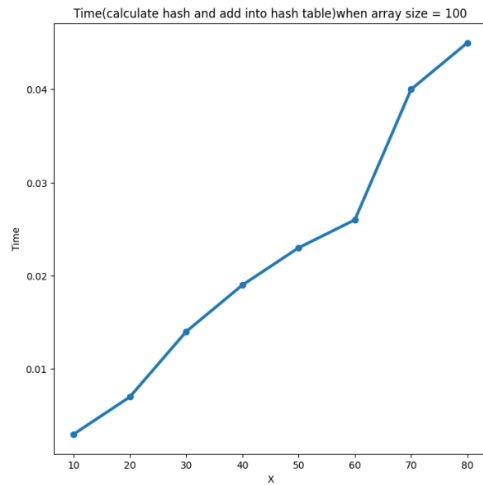


Fig3. Running Time

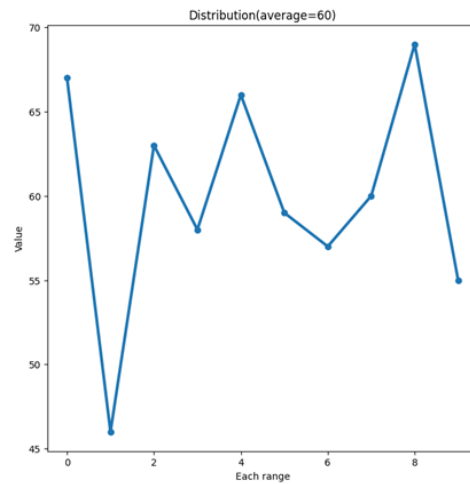


Fig4. Distribution

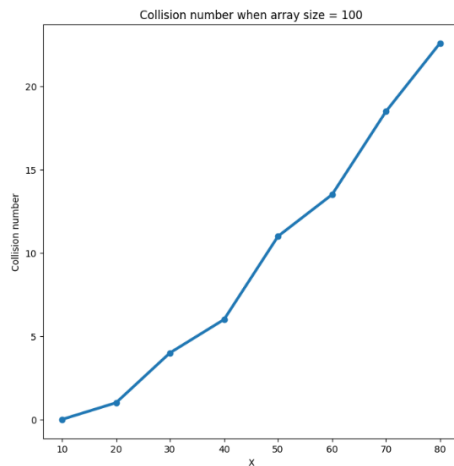


Fig5. Collision number

6.1. 運算速度:

當有較多碰撞可能發生時，時間會比較久(by Fig.3)。

6.2. 均勻分布:

雖然能有特定區域會出現較少或較多的次數，但大多數仍維持在平均值，從而達到均勻分布(by Fig.4)。

6.3. 如何解決碰撞 collision 和 overflow:

Collision: 只會有極少數的 linked list 在後面串 2 個(含)以上，大多皆為 1 個。

Overflow: 已經將 h 值縮放到 array size 了，所以不會發生 overflow。

七、結論

在一個長度和數字字元皆不固定的情況下，做的一個 $O(1)$ 的 hash function，一些參數參考了 murmurhash 裡的參數(例如 m1、r1，但這參數不代表數學運算，僅代表這樣相乘會比較均勻分布而已)，希望未來能用機器學習去訓練，要 minimize 的 loss 就是我碰撞的次數，就有機會找到最好的參數。最後，最重要的還是理解了如何克服均勻分布、雪崩測試、運算速度和碰撞處理，每個都有每個的困難之處，成就感很大。