1 Hash function

1.1 Introduction

在解題甚至是一般應用中，我們常常需要執行下列這些操作：

1. 檢索 (ex. 詢問某元素是否在某集合內)
2. 比較 (ex. 比較兩字串是否相同)
3. 映射 (ex. 把一個集合 S 映射到 $\{1 \cdots n\}$)

在很多情況下如果元素的結構非常複雜，以上的操作難度就會大幅增加。舉例來說，許多人常常用 BitTorrent 等 P2P 界面下載影片，影片來源這麼多，到底要怎麼確定兩個巨大的影片檔案是不是同一個檔案呢？一個檔案大小動輒數 GB，如果對於所有檔案都還要 $O(n)$ 比較一下位元，將會非常耗費時間。類似這樣的情形，我們通常會需要把原先複雜的資料型態轉換為較簡單的資料型態，而轉換過程中使用的函數我們就稱為 hash function (雜湊函數)，映射後的所有可能形成的狀態空間則稱為 hash table。

對於一個資料型態 T，一個好的雜湊函數 f 通常需要滿足以下條件：

1. 對於一個 T 型態的元素 x ，$f(x)$ 為一個 S 型態的元素，且比較兩 S 型態元素是否相等的複雜度低於比較 T 型態。
2. 對於兩個 T 型態的元素 x, y ，若 $x = y$ ，則 $f(x) = f(y)$ 。
3. 對於兩個 T 型態的元素 x, y ，若 $x \neq y$ ，則 $f(x) \neq f(y)$ 成立的機率應盡量高。

注意到我們並不要求第三點的機率等於 1 ，因為將複雜的型態轉為簡單的型態往往會丟失部份訊息，從而降低代表性。舉例而言，如果我們想要針對僅含有小寫字母，長度為 L 的字串設計一個雜湊函數將之映射到 $[0, 1000000)$ 內的整數，則原始輸入資料有 26^L 種可能，映射到的對象卻只有 1000000 種可能，根據鴿籠定理必定會有許多元素映射到同一個整數。然而這並不影響雜湊函數的作用，至少雜湊函數已經減少許多不必要的比較。

底下我們介紹一些雜湊函數設計的例子，注意到雜湊函數設計的方式有很多種，很多時候優劣性也不容易直觀判斷，所以在實務上使用雜湊函數時很多時候是需要一些經驗與靈感的。

例題 1

給定 C 語言中 `int` 範圍內的整數 k ，請設計雜湊函數 f 將 k 映射到 $[0, M)$ 內的整數。

解答

令 $f(k) = k \mod M$。

這是一個很常見的問題，例如排列組合問題中，解答常常是個很大的數值，為了避免大家要寫大數的困擾，我們當然會希望輸出會是一個範圍內的數值。最常見的作法就是取除以 M 的餘數，而這顯然滿足雜湊函數的前兩個要求。而第三個要求效果是否良好則仰賴於 M 的選擇，根據數學性質我們知道 M 數值較大時，映射到同一個元素的情形會較不嚴重，雜湊函數的正確性較高。

例題 2

給定字串 $s = s_1 \cdots s_n$ ，字元集合大小為 C ，試設計雜湊函數 f 將 s 映射到一個整數。

解答

令 $f(s) = \sum_{i=1}^{n} s_i \times C^{n-i}$ 。

此函數在字串演算法中扮演著重要的角色，稱為 rolling hash 。該函數實際上等價於把字串作為一種 C 進位的數字系統，除了雜湊函數的前兩要素都符合外，連第三個要素都能滿足機率為 1，理論上是個絕佳的雜湊函數。然而由於映射過後得到的整數往往非常龐大，實務上我們往往還是需要如例題 1 一樣，將運算結果除以 M 取餘數，導致雜湊函數的精確度瓶頸仰賴於設計者選取的數字 M ，在這裡為了希望數字乘上 C 之後盡量有不同的結果（如不希望發生 $1 \times 2 \equiv 4 \times 2 \equiv 2 \pmod 6$），基於數學原理會將 M 選成一個質數。

例題 3

給定 k 個長度為 n 的序列 $A_1 \cdots A_k$，每次詢問其中兩序列是否為相同的集合(即包含的元素集合完全相同)。

解答

令 $f(A) = A[1] \oplus A[2] \oplus \cdots \oplus A[n]$ 。($\oplus$ 表示布林運算的 XOR)

如果我們直接每次都比較兩個序列，那麼複雜度是 $O(n)$。由於我們想知道的是元素組成是否相同，元素的順序並不重要，因此可以設計如上的雜湊函數 f，並且先 $O(kn)$ 預處理所有序列的 f 函數，之後對於每組詢問直接檢查兩序列的 f 函數值是否相等即可，詢問複雜度為 $O(1)$。當然，這犧牲了一些正確性。

例題 4

給定 k 個長度為 n 的序列 $A_1 \cdots A_k$，每次詢問其中兩序列是否為相同的序列(即元素順序也須完全相同)。

解答

接續例題 3，先令序列 $B = \{A[1], A[2] - A[1], \cdots, A[n] - A[n-1]\}$，即 A 序列中相鄰兩數的差，再令雜湊函數 $g(A) = f(A) + f(B)$。

相較於上題，本題的要求更嚴格，連元素之間的順序關係也是需要考慮的部份。一個解決的辦法是把前一題的想法強化，先定義序列 B ，再將兩序列的 f 函數值相加。如此一來如果兩序列 A_1, A_2 元素組成相同但順序不同，則 $f(A_1)$ 和 $f(A_2)$ 會相等，但 $f(B_1)$ 和 $f(B_2)$ 很可能不相等，導致 $g(A_1) \neq g(A_2)$ 的機率較高，達到雜湊函數的需求。

1.2 Conflict

在上一小節中，我們已經提到 hash function 的侷限性：由於丟失部份資料，且必須滿足映射到的型態較原型態更易操作，難免會有兩個不同的元素在經過轉換後變成相同的元素。我們稱這種情形為「碰撞」(hash conflict)。為了避免碰撞造成的誤判，我們最常見的處理方式有如下四種：

1. 視而不見：
   總是假設碰撞不存在。無視並不代表消極處理，而是代表我們沒必要或者沒辦法處理此時的碰撞情形。舉例而言，前面提到的 P2P 檔案辨識問題，如果兩個檔案經過 hash function 得到的驗證碼是相同的，理論上並不代表兩個檔案就是相同的檔案，但在實務上我們會如此默認，因為確認的成本過於高昂且發生的機率過低。

2. 閉合雜湊 (closed hashing)：
   如果把映射後的結果看成座位，則碰撞可以看成是「有人搶了自己的座位」。這時候閉合雜湊的解決方法是，如果真的在自己座位上的人不是自己 (即碰撞確實成立)，那麼當前元素就按照某些特定的規則再去搶別人的空位。這樣做的好處是使用的空間量較固定，壞處則是好的搶奪規則不好設計，且在碰撞嚴重時效率不佳。

3. 開放雜湊 (open hashing)：
   相較於閉合雜湊，開放雜湊在遭遇碰撞時的處理方式是「既然命運決定我們同個位子，那只好擠一擠了」。開放雜湊的實做方式是在 hash table 的每個狀態裡都維護一個 list，記錄所有同屬此狀態的相異元素。如果有某個元素已知映射到某個狀態 S ，則為了避免誤判，我們可以進一步檢查 S 內 list 中的所有相異元素來確定元素是否相等。由於實做相對方便，且可配合諸多優化方式 (如將 list 改為二元平衡樹)，在解題實務上我們相較於閉合雜湊更常使用開放雜湊。

4. 多重雜湊：
   一個便當吃不飽，可以吃兩個；一個雜湊函數不夠準確，我們當然也可以用兩個！假如兩個雜湊函數 f, g 互相獨立，f 的誤判率為 p_1 ，g 的誤判率為 p_2 ，則兩者同時誤判的機率就大幅下降為 $p_1 \times p_2$。類似地，若有更多彼此獨立的雜湊

函數，則誤判的機率為各個雜湊函數誤判率的乘積。多重雜湊的好處在於誤判率足夠低後，我們甚至可以略去判斷碰撞的過程 (即視而不見)，壞處則為運算常數增加，且設計許多獨立的雜湊函數並不容易。

※ 例題 4 實際上就是多重雜湊的一個應用，只是為了方便起見將兩個雜湊函數直接相加起來。

雜湊函數本身並不穩定，相較於其他資料結構效能分析也更困難，因此一開始會不容易掌握；但在平均效率上，雜湊函數卻有相當驚人的表現，實務上也隨處可見。希望大家可以不要太過害怕其不穩定性，多多研究與嘗試，累積一些經驗後將會成為好使又強力的工具。

習題

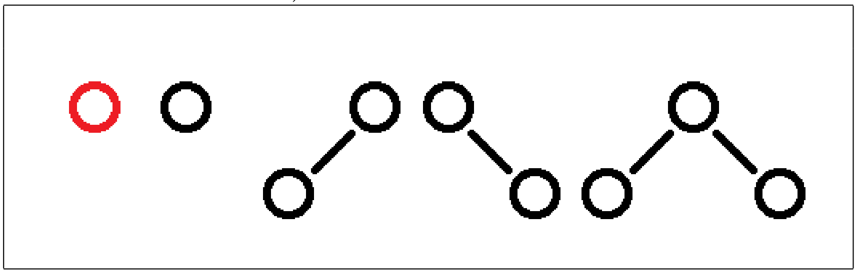
1. 請依照需求設計雜湊函數。請注意，以下題目並沒有標準答案，也不需要太在意運算的時間複雜度，但是請大家盡量避免碰撞，例如令 $\forall x,\ f(x) = 1$ 是不會得到任何分數的 (笑)。

   (a) (6 pts) 請設計一個雜湊函數 f 將一張有花色的撲克牌 (可能是鬼牌) 映射到一個整數，以判斷兩張牌是否相同。請保證對於一副 54 張完整撲克牌內，沒有任兩張牌的雜湊值相同。

   (b) (13 pts) 我們說兩棵 (節點和邊皆沒有權重的) 有向二元樹 T_1, T_2「同構」($T_1 = T_2$)，若且唯若 T_1, T_2 滿足以下條件其中之一：

      i. $T_1 = \text{NULL}$ 且 $T_2 = \text{NULL}$
      ii. $T_1 \neq \text{NULL}$ 且 $T_2 \neq \text{NULL}$ 且 $T_1.left = T_2.left$ 且 $T_1.right = T_2.right$

   請設計一個雜湊函數 f 將一棵二元樹映射到一個整數，以判斷兩棵二元樹是否同構，並滿足所有深度不大於 2 的二元樹的雜湊值都不相同 (共五種如下圖，紅色圓圈表示 NULL)。



2. 在密碼學或資訊安全的領域中，Hash Function 常被應用來進行加密。一個好的加密用雜湊函數 (Cryptographic Hash Function)H 會滿足以下幾個性質：

   (a) *One-wayness*: 給定一個雜湊值 y，我們很難找出原始的 x，使得 $H(x) = Y$，意即，這個函式是接近不可逆的。

   (b) *Weak Collision Resistance*: 給定一個值 x，我們很難找到一個相異的 x'，使得 $H(x) = H(x')$。

   (c) *Strong Collision Resistance*: 我們很難找到一組 x_1, x_2，使得 $H(x_1) = H(x_2)$。

   (a) (6 pts) 請問如果我們要加密一個長度至少為 1，至多為 6，其中僅包含英文大小寫字母的密碼，為了達成完全 *Strong Collision Resistance*，使得任兩組密碼雜湊後均不相同，請問你的雜湊函數值域至少要多大？為什麼？

   (b) (6 pts) 承上題，請問若你可以任意使用該雜湊函數，請問 *One-wayness* 性質還會存在嗎？意即，你如果知道一組密碼的雜湊值後，你能找出原始的密碼明文嗎？為什麼？

   (c) (10 pts) 伺服器上通常都不存有使用者密碼的明文，通常都是儲存密碼經過雜湊函數後產生的雜湊值，而後當使用者輸入密碼後，則將輸入的字串再經過原本的雜湊函數後，比較兩個雜湊值是否相同。若今天有一個網路服務，雜湊函

數位於使用者瀏覽器，將輸入的字串通過該雜湊函數後，將雜湊值透過網路傳輸給伺服器端。請問若該雜湊函式不滿足 *One-wayness* 性質，對於攻擊者而言，他可以如何取得使用者的權限呢？但攻擊者嘗試登入時仍必須經過瀏覽器。除此之外，你可以做出一些假設來輔助你的答題，例如這個服務使用的通訊協定 (HTTPS/HTTP/SFTP/...)。

(d) (10 pts) 如前課文所提到，Open Hashing 理想上可以在常數時間完成查詢 (query)。但是攻擊者常會利用這個特性，使得查詢所消耗的時間大幅上升。若目前的雜湊函數 f 定義為 $f(x) = x \mod 1000000007$，請你設計一組有 $N = 20000$ 筆正整數的輸入，且資料內沒有重複的數字，使得他們輸入進 Open Hashing 的 Hashing Table 後，接下來的每一次查找最差會花到 $O(N)$ 的時間。

3. 在例題中，我們提到一個字串 $s = s_1s_2\cdots s_n$ 的 rolling hash 為

$$H(s) = \sum_{i=1}^{n} s_i \times C^{n-i} \mod M$$

其中 C 為字元集合大小，M 為設計者選擇的一個常數。

(a) (10 pts) 請描述如何在 $O(n)$ 的時間內得到 $H(s)$ 的值。

(b) (15 pts) 令字元集合為英文小寫字母，$a = 0, b = 1, \cdots, z = 25$，$C = 26$，$M = 1000007$。請列出三個互不相同，但長度相等且 ≤ 6 的字串 s_1, s_2, s_3，滿足 $H(s_1) = H(s_2) = H(s_3)$。(你可能會想要寫點 code。)

(c) (12 pts) 令 $s[l:r] = s_ls_{l+1}\cdots s_r$ 為 s 的一個長度為 $k = (r-l+1)$ 的子字串。已知 $s[l:r]$ 的 rolling hash 值 $H(s[l:r]) = x$ ，且 $H(s[l+1:r+1]) = y$ ，請以 x, s_i, k, C, M 表示 y 。(以數學形式表達即可，不需要考慮程式上的細節)

(d) (12 pts) 給定兩個長度分別為 n 和 m $(n \leq m)$ 的字串 s, t ，請描述如何使用 rolling hash 在 $O(n+m)$ 的時間內，判斷 s 是否為 t 的子字串。為了方便起見，可以假設 hash value 不會發生碰撞。