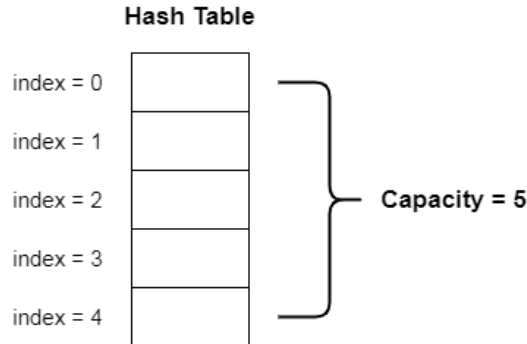


Hash Table 流程圖、學習歷程與 Hash Table 與 Hash Function 原理

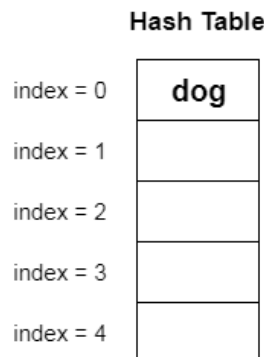
一、Hash Table 流程圖

1. 建立 Hash Table 並決定其長度

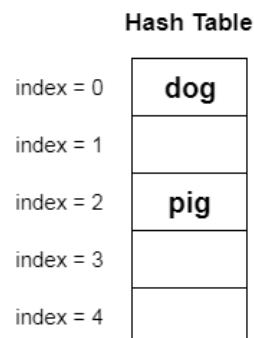


2. 新增資料進入 Hash Table

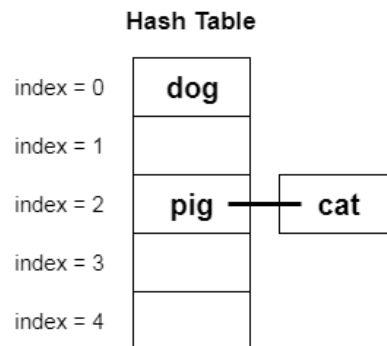
- 新增非數值型態資料「dog」進入 Hash Table 中。
- 計算該資料須存放的位置: 以 MD5 編碼並將編碼結果轉成十進位，並將該數值除以 Hash Table 的長度求餘數。
「dog」以 MD5 編碼並將編碼結果轉成十進位後為 9097202055026264535080901219663267845，除以 Hash Table 的長度求餘數為 0。
- 將「dog」存放至 Hash Table 的 index=0 的位置。



- 新增非數值型態資料「pig」進入 Hash Table 中。
- 「pig」以 MD5 編碼並將編碼結果轉成十進位後為 9097202055026264535080901219663267845，除以 Hash Table 的長度求餘數為 2。
- 將「pig」存放至 Hash Table 的 index=2 的位置。

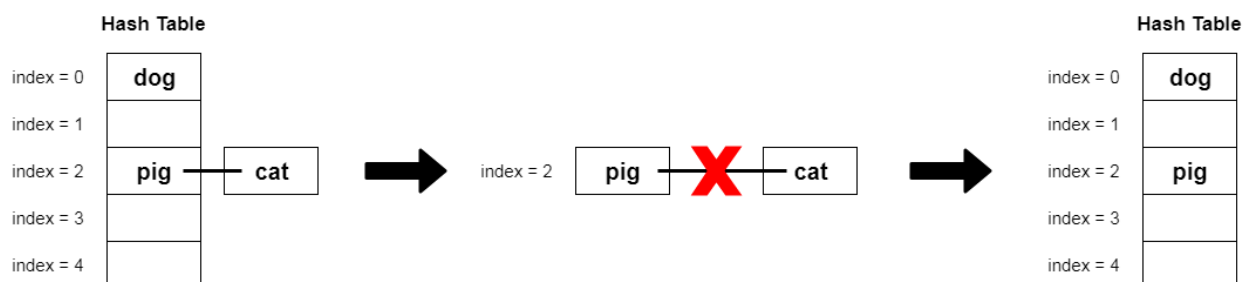


- 新增非數值型態資料「cat」進入 Hash Table 中。
- 「cat」以 MD5 編碼並將編碼結果轉成十進位後為 277102220249073555409885156483852860632，除以 Hash Table 的長度求餘數為 2。
- 將「cat」存放至 Hash Table 的 index=2 的位置。
- 在 index=2 的位置發生 Collision，利用 Chaining 的方法，使用 Linked list 將分在 Hash Table 中同一個 index 的資料串起來，將「cat」接在「pig」後面。



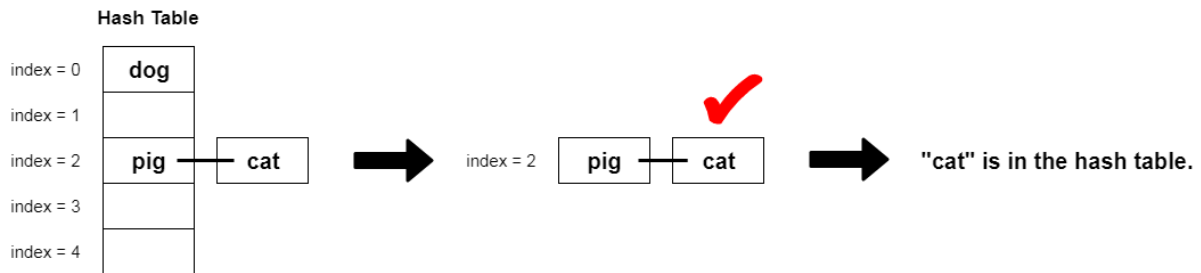
3. 刪除 Hash Table 中資料

- 在 Hash Table 中刪除新增非數值型態資料「cat」。
- 計算該資料須存放的位置：以 MD5 編碼並將編碼結果轉成十進位，並將該數值除以 Hash Table 的長度求餘數。
「cat」以 MD5 編碼並將編碼結果轉成十進位後為 277102220249073555409885156483852860632，除以 Hash Table 的長度求餘數為 2。
- 訪尋位於 Hash Table 中 index=2 位置的 Linkedlist，並刪除含有「cat」的 node。



4. 查找 Hash Table 中的資料

- 在 Hash Table 中查找非數值型態資料「cat」。
- 計算該資料須存放的位置：以 MD5 編碼並將編碼結果轉成十進位，並將該數值除以 Hash Table 的長度求餘數。
「cat」以 MD5 編碼並將編碼結果轉成十進位後為 277102220249073555409885156483852860632，除以 Hash Table 的長度求餘數為 2。
- 訪尋位於 Hash Table 中 index=2 位置的 Linkedlist，並查找含有「cat」的 node。
如找到符合條件的 node 則返回 "cat" is in the hash table.。



二、 Hash Table 學習歷程

1. 嘗試加入 MyLinkedList 的寫法

加入 MyLinkedList 的 class，該 class 的原理是能在一條 Linked List 中做新增、刪除、查詢的功能，因此在 MyHashSet 的 class 中只需先在 Hash Table 的每個 Index 放入空的 Linked List，然後在做新增、刪除、查詢時，只需先找到輸入的 key 值在 Hash Table 的位置後，再呼叫位於該位置的 MyLinkedList，並利用該 class 的功能做新增、刪除、查詢。程式碼變得更清楚簡潔。

第 57 行的 self.data 不能寫：

`self.[MyLinkedList()]*capacity`

這樣會讓位於每個 Hash Table 位置的 MyLinkedList 都一樣，這樣會導致在某個 Hash Table 位置做新增、刪除、查找時，對每個位置的 MyLinkedList 都做上述的動作，而不只是在輸入的 key 值在 Hash Table 的位置做而已。

要寫：

`for i in range(capacity):`

`self.data[i]=MyLinkedList()`

或是

`self.data=[MyLinkedList() for x in range(0, capacity)]`

一個一個位置去索引並賦值 MyLinkedList()。

```

1 class ListNode:
2     def __init__(self, val):
3         self.val = val
4         self.next = None
5
6 class MyLinkedList:
7     def __init__(self):
8         self.head = None
9         self.size=0
10
11     def order(self):
12         currnode = self.head
13         while currnode!=None:
14             print(currnode.val,end=' ')
15             currnode = currnode.next
16         print('\n')
17
18     def addAtTail(self,val):
19         if self.head == None:
20             self.head = ListNode(val)
21             self.size+=1
22             return
23         currnode = self.head
24         while currnode.next != None:
25             currnode = currnode.next
26         currnode.next = ListNode(val)
27         self.size+=1
28
29     def deleteValue(self,val):
30         while self.searchValue(val)==True:
31             if self.head.val==val:
32                 tmp=self.head.next
33                 self.head=tmp
34                 self.size-=1
35             else:
36                 pre=self.head
37                 currnode=self.head
38                 while currnode!=None:
39                     if currnode.val==val:
40                         pre.next=currnode.next
41                         self.size-=1
42                         break
43                 pre=currnode
44                 currnode=currnode.next
45
46     def searchValue(self,val):
47         currnode=self.head
48         while currnode!=None:
49             if currnode.val==val:
50                 return True
51             currnode=currnode.next
52         return False
53
54 class MyHashSet:
55     def __init__(self, capacity=5):
56         self.capacity = capacity
57         self.data = [MyLinkedList() for x in range(0, capacity)]
58
59     def add(self, key):
60         index = self.hash(key)
61         linkedlist = self.data[index]
62         linkedlist.addAtTail(key)
63
64     def remove(self, key):
65         index = self.hash(key)
66         linkedlist = self.data[index]
67         linkedlist.deleteValue(key)
68
69     def contains(self, key):
70         index = self.hash(key)
71         linkedlist = self.data[index]

```

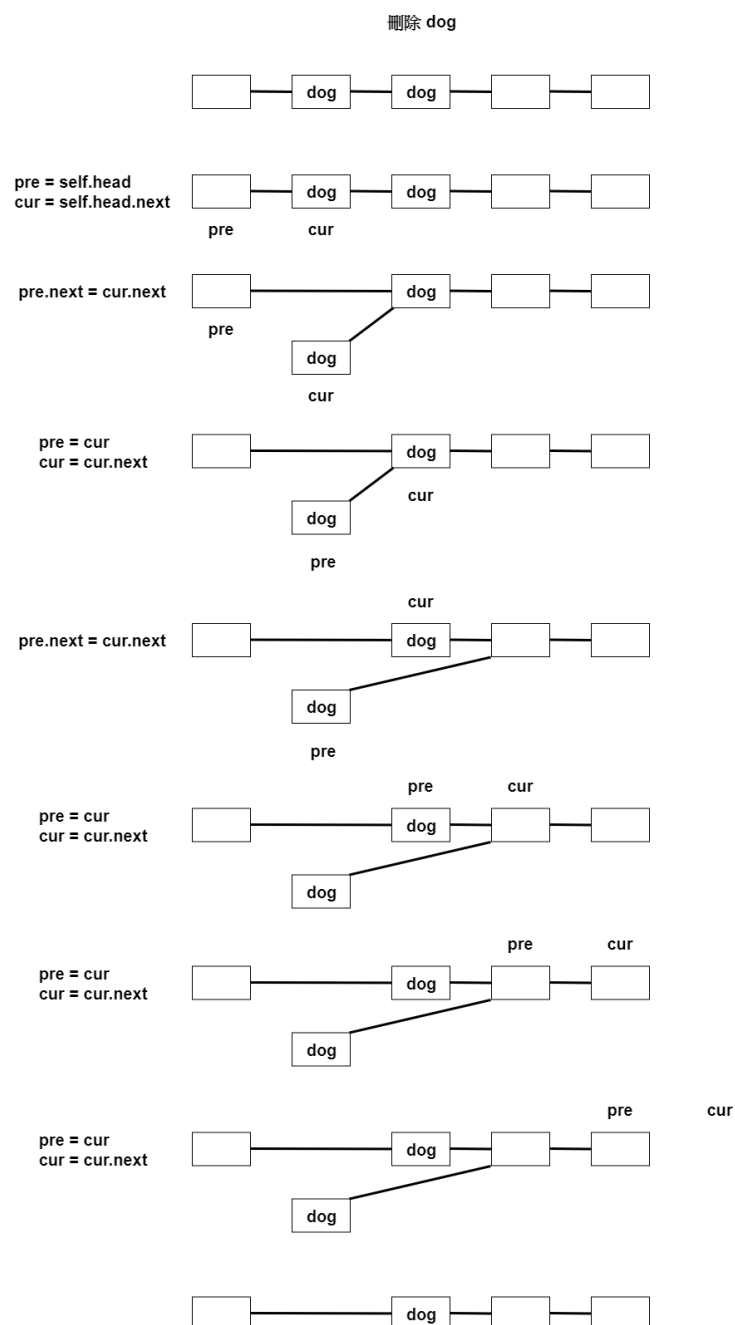
另外在第 29 行 deleteValue 不能寫:

```

1 def deleteValue(self, val):
2     pre=self.head
3     curnode=self.head.next
4     while curnode!=None:
5         if curnode.val==val:
6             pre.next=curnode.next
7             pre=curnode
8             curnode=curnode.next
9         if self.head.val==val:
10            tmp=self.head.next
11            self.head=tmp

```

上述的方法是打算一次將所有欲刪除的值刪除，但會發生以下問題，導致無法刪完所有的值：



2. Remove 寫法注意

在第 57 行的 return 一定要加，或是改成 break，不然會有無窮迴圈的情況。原因在於在第 56 行讓 self.data[index]=None 時並不會讓 node 變成 None，只會讓 self.searchValue 重複返回 True，然後進入 node.next==None 的條件中，然後不斷循環。

```
25 class ListNode:
26     def __init__(self, val):
27         self.val = val
28         self.next = None
29
30 class MyHashSet:
31     def __init__(self, capacity=5):
32         self.capacity = capacity
33         self.data = [None] * capacity
34
35     def add(self, key):
36         index = self.hash(key)
37         if self.data[index]==None:
38             self.data[index]=ListNode(None)
39         node = self.data[index]
40         if node.val==None:
41             node.val=key
42         else:
43             currnode = node
44             while currnode.next != None:
45                 currnode = currnode.next
46             currnode.next = ListNode(key)
47
48     def remove(self, key):
49         index = self.hash(key)
50         if self.data[index]==None:
51             return
52         node = self.data[index]
53         while self.searchValue(key,node)==True:
54             if node.val==key:
55                 if node.next==None:
56                     self.data[index]=None
57                     return #注意! 一定要加，不然會無窮迴圈，因為no
58                 else:
59                     node.val=node.next.val
60                     node.next=node.next.next
61             else:
62                 pre=node
63                 currnode=node
64                 while currnode!=None:
65                     if currnode.val==key:
66                         pre.next=currnode.next
67                         break
68                     pre=currnode
69                     currnode=currnode.next
70
71     def contains(self, key):
72         index = self.hash(key)
73         if self.data[index]==None:
74             return False
75         node = self.data[index]
76         outcome=self.searchValue(key,node)
77         return outcome
78
79     def hash(self,key):
80         from Cryptodome.Hash import MD5
81         h = int(MD5.new(key.encode('utf-8')).hexdigest(),16)
82         index = h % self.capacity
83         return index
84
85     def orderAll(self):
86         for i,j in zip(self.data,range(0, self.capacity)):
87             print(j,end=' ')
88             self.order(i)
89
90     def searchValue(self,key,node):
91         currnode=node
92         while currnode!=None:
93             if currnode.val==key:
94                 return True
95             currnode=currnode.next
96         return False
97
98     def order(self,node):
99         currnode = node
100         while currnode!=None:
101             print(currnode.val,end=' ')
102             currnode = currnode.next
103         print('\n')
```

3. MD5 原理

星期二實習課已請教過助教。

data: 10^6 MD5 encode

$$\text{partition: } 10^6 = \underbrace{512 \times N}_{+n} + \underbrace{448}_{512} + \underbrace{64}_{64} = (N+1) \times 512$$

\downarrow
 $\ln(\text{data})$ \downarrow
 64×16
 1024

\parallel $R = \begin{matrix} 1 & 0 & 1 \\ 1 & 0 & 0 \end{matrix} = 5 \sim 10 / 6 = 10$
 \parallel $\begin{matrix} 1 & 0 & 0 \\ 1 & 0 & 0 \end{matrix} = 4$
 \parallel $\begin{matrix} 1 & 1 & 1 \\ 1 & 1 & 0 \end{matrix} = 6$

4. 程式碼說明

def __init__(self, capacity=5):

- 決定 Hash Table 長度。
- 建立 Hash Table，Hash Table 中每個位置的初始值為 None。

def add(self, key):

- 透過 hash 函式找出 key 值在 Hash Table 的位置。
- 如果該位置的值尚為 None，則賦值為一個空的 node。
- 接下來的動作與 LinkedList 中 addAtTail 函式相似

def remove(self, key):

- 透過 hash 函式找出 key 值在 Hash Table 的位置。
- 採用一次只刪除一個值，並透過 while 迴圈，每刪完一次就從頭到尾訪尋一次 Linked List，刪到 Linked List 中沒有該值為止。
- 分兩種情況討論：1. 欲刪除的值位於 Linked List 的首位 2. 欲刪除的值位於 Linked List 其他位置。刪除方法與 LinkedList 中 deleteAtIndex 函式相似，只是這次刪除的條件是 curnode.val==key。

def contains(self, key):

- 透過 hash 函式找出 key 值在 Hash Table 的位置。
- 利用 searchValue 函式查找 LinkedList 中是否有欲查找的 key。查找方式與 LinkedList 中 get 函式相似，只是這次查找的條件是 curnode.val==key。

```
25 class ListNode:
26     def __init__(self, val):
27         self.val = val
28         self.next = None
29
30 class MyHashSet:
31     def __init__(self, capacity=5):
32         self.capacity = capacity
33         self.data = [None] * capacity
34
35     def add(self, key):
36         index = self.hash(key)
37         if self.data[index]==None:
38             self.data[index]=ListNode(None)
39         node = self.data[index]
40         if node.val==None:
41             node.val=key
42         else:
43             curnode = node
44             while curnode.next != None:
45                 curnode = curnode.next
46             curnode.next = ListNode(key)
47
48     def remove(self, key):
49         index = self.hash(key)
50         if self.data[index]==None:
51             return
52         node = self.data[index]
53         while self.searchValue(key,node)==True:
54             if node.val==key:
55                 if node.next==None:
56                     self.data[index]=None
57                     return #注意! 一定要加，不然會無窮迴圈，因為no
58                 else:
59                     node.val=node.next.val
60                     node.next=node.next.next
61             else:
62                 pre=node
63                 curnode=node
64                 while curnode!=None:
65                     if curnode.val==key:
66                         pre.next=curnode.next
67                         break
68                     pre=curnode
69                     curnode=curnode.next
70
71     def contains(self, key):
72         index = self.hash(key)
73         if self.data[index]==None:
74             return False
75         node = self.data[index]
76         outcome=self.searchValue(key,node)
77         return outcome
78
79     def hash(self,key):
80         from Cryptodome.Hash import MD5
81         h = int(MD5.new(key.encode('utf-8')).hexdigest(),16)
82         index = h % self.capacity
83         return index
84
85     def orderAll(self):
86         for i,j in zip(self.data,range(0, self.capacity)):
87             print(j,end=' ')
88             self.order(i)
89
90     def searchValue(self,key,node):
91         curnode=node
92         while curnode!=None:
93             if curnode.val==key:
94                 return True
95             curnode=curnode.next
96         return False
97
98     def order(self,node):
99         curnode = node
100         while curnode!=None:
101             print(curnode.val,end=' ')
102             curnode = curnode.next
103         print('\n')
```

三、 Hash Table 與 Hash Function 原理

I. Hash Table 與 Hash Function 概念

根據鍵 (Key) 而直接查詢在內存存儲位置的資料結構。通過一個關於計算鍵值的函數，將所需查詢的數據映射到表中一個位置來查詢記錄，該方法也加快了查找速度。此映射函數稱做雜湊函數 (Hash Function)，存放記錄的數組稱做雜湊表 (Hash Table)。

以查找與儲存姓名為例。在未使用 Hash Table 與 Hash Function 的情況下，如欲儲存一百位人名就直接將一個個人名儲存起來，假設當要查找「Albert」該人名時，就只能將「Albert」這三個字一個個與一百個人名比對，共做一百次比對，該方法相當耗時。當使用 Hash Table 與 Hash Function 的方法後，在儲存資料時，可以先透過 Hash Function 將欲儲存的人名轉換成人名字首，如將「Albert」轉換成「A」儲存、將「Alice」轉換成「A」儲存、將「Charles」轉換成「C」儲存.....。經過轉換後，最終儲存的種類只會有 26 種可能 (英文有 26 個字母)，該 26 種儲存的種類構成了一個數組，該數組即為 Hash Table，長度為 26，同樣被轉成「A」的人名被視為一類，並儲存在「A」類別中.....。如欲查找「Albert」該人名時，先透過 Hash Function 將「Albert」轉換成「A」。此時只需查找同樣是「A」的儲存種類即可，其他種類則不必查找，並在該種類下找到「Albert」這個人名。該方法省下了許多不必要的比對。以上是 Hash Table 與 Hash Function 來儲存和查找資料的例子。

II. Hash Table 介紹

因此在 Hash Table 這個數組中，每個位置代表資料經 Hash Function 轉換後的數值，原本的資料稱為鍵值(Key)，經 Hash Function 轉換後的數值則為 Hash Table 的索引值(Index)。若用到的 Key 的數量為 n ，Table 的大小為 m ，Hash Function 為 h ，那麼 m 等於 $h(n)$ 的種類； $\text{index}=h(\text{Key})$ ， $h(\text{Key})$ 即為 Hash Table 的 index。然而很可能發生 Collision 的情況，很可能有多個 Key 值經 Hash Function 轉換後被分在 Hash Table 中同一個 index，此時就可以利用 Chaining 的方法，使用 Linked list 把被分在 Hash Table 中同一個 index 的 Key 值串起來。有了 Linked list 處理被分配到同一個 index 的 key 值，在 Hash Table 中每個 index 的 Linked list 有幾種資料處理：1. Add: 先利用 Hash Function 取得 Table 的 index，接著再利用 addAtTail 每個資料依序串起來。2. Contains: 先利用 Hash Function 取得 Table 的 index，接著再利用 Search 找到欲查找的資料。3. Remove: 先利用 Hash Function 取得 Table 的 index，接著再利用 Delete 刪除欲刪除的資料。

III. Hash Function 介紹

優秀的 Hash Function (h)應具備以下特徵：

1. 定義 h()的定義域(domain)為整個 Key 的字集合 U，值域(range)應小於 Table 的大小 m： $h:U \rightarrow \{0,1,\dots,m-1\}$, where $|U| \gg m$
2. 盡可能讓 Key 在經過 Hash Function 後，值域(也就是 Table 的 index)能夠平均分佈，如此才不會讓兩筆資料存進同一個 Table 的 index，而發生 Collision 的情況。

若把 Table 想像成「書桌」，index 想像成書桌的「抽屜」，那麼為了要能更快速找到物品，當然是希望「每一個抽屜只放一個物品」，如此一來，只要拿著 Key，透過 Hash Function 找到對應的抽屜，就能保證是該 Key 所要找的物品。反之，如果同一個抽屜裡有兩個以上的物品時，便有可能找錯物品。

Hash Function (h) 其中一種方法為 Division Method，假設 Table 大小為 m，經 Hash Function 轉換後的鍵值(key)為鍵值除以 Table 大小的餘數，該餘數即為該鍵值在 Hash Table 中的 Index:

$$h(\text{Key}) = \text{Key} \bmod m$$

例如選定 Table 大小為 $m=8$ ，的 Key 與 Table 之 index 將有對應關係如下：

$$h(14) = 14 \bmod 8 = 6$$

代表「鍵值 14」要儲存在 Hash Table 「index=6」的位置。

$$h(23) = 23 \bmod 8 = 7$$

代表「編號 23」要儲存在 Hash Table 「index=7」的位置。

$$h(46) = 46 \bmod 8 = 6$$

代表「編號 46」要儲存在 Hash Table 「index=6」的位置。

$$h(50) = 50 \bmod 8 = 2$$

代表「編號 50」要儲存在 Hash Table 「index=2」的位置。

如欲儲存和查找的資料非數值型態，可以透過編碼將該型態資料轉換成數值型態資料，如 MD5 編碼。

四、 參考資料

1. Hash Table 與 Hash Function 原理

- <https://zh.wikipedia.org/wiki/%E5%93%88%E5%B8%8C%E8%A1%A8>
- <http://alrightchiu.github.io/SecondRound/hash-tableintrojian-jie.html#ht>
- <http://alrightchiu.github.io/SecondRound/hash-tablechaining.html>

2. 程式碼

- 參考自己原創的 LinkedList 程式碼:
Github/DSA/Linked List 資料夾/ linked list_原創.py
<https://github.com/albert0796/DSA.git>