Hash Table的介紹

在查詢資料時，以Dictionary來說就是一個key對一個value。但若直接把key當成陣列的index，並將Value存放進陣列裡，這樣的概念叫Direct Access Table。
但如果key的範圍很廣，那就會很浪費記憶體。
ex: 有兩個key->2跟7 ，那我的陣列就要用掉8個單位。(其中0,1,3,4,5,6皆為空值，但還是會佔空間)
Hash Table的概念就是為了節省這空間。

Hash Table的概念

Hash Table 透過Hash Function 將存放資料的大小降低到真正有用到的key的數量。

Hash Function

就像櫃子跟抽屜，每個抽屜只放一個東西(一個抽屜放兩個東西會發生拿錯東西的error)，Hash Function的功能就是指出「第幾個」抽屜，也就是抽屜的index，就能保證什麼key會拿到什麼value。且Hash Function具有不可逆的特性，無法從雜湊值推回原本的資料是什麼。

add(不重複):
先將key從字串轉成數字，空值就直接丟進去，非空值就跟linklist差不多。

```
def add(self, key):
    h = MD5.new()
    h.update(key.encode("UTF-8"))
    arr = int(h.hexdigest() , 16)%self.capacity
    #先加密，轉成10進位，type為int，除以capacity取餘數( % ->取餘數)
    if self.data[arr] is None:
        self.data[arr] = ListNode(h)
    #空值就直接丟進去
    else:
        n = ListNode(h)
        o = self.data[arr]
        while o.next is not None:
            o = o.next
        o.next = n
```

contains:

```
    def contains(self, key):
        h = MD5.new()
        h.update(key.encode("UTF-8"))
        arr = int(h.hexdigest() , 16)% self.capacity
        head = self.data[arr]
        if head is None:
            return False
        else:
            while head.val != arr: #用while迴圈，就可以持續往下一個找
                head = head.next
                if head is None:
                    return False  #head沒有接下來也會有
            return True           #剩下的就是Ans
#IndexError: cannot fit 'int' into an index-sized integer
```

多加一條 brr = int(h.hexdigest() , 16)

```
    def contains(self, key):
        h = MD5.new()
        h.update(key.encode("UTF-8"))
        arr = int(h.hexdigest() , 16)% self.capacity
        brr = int(h.hexdigest() , 16)
        head = self.data[arr]
        if head is None:
            return False
        else:
            while head.val != brr:
                head = head.next
                if head is None:
                    return False
            return True
```

```
hashset = MyHashSet()
hashset.add("dog")
```

```
rel = hashset.contains("dog")
print(rel)
```

```
False
```

Remove:

```
    def remove(self, key):
        h = MD5.new()
        h.update(key.encode("UTF-8"))
        arr = int(h.hexdigest() , 16)%self.capacity
        brr = int(h.hexdigest() , 16)

        if self.data[arr]:
            n = self.data[arr]
            while n.val != brr and n.next:
                f = n
                n = n.next
            f.next = n.next
        else:
            return
#UnboundLocalError: local variable 'f' referenced before assignment
```

把賦值語句放到外面一層就行了

```
    def remove(self, key):
        h = MD5.new()
        h.update(key.encode("UTF-8"))
        arr = int(h.hexdigest() , 16)%self.capacity
        brr = int(h.hexdigest() , 16)
        n = self.data[arr]
        if self.data[arr]:
            cur = n
            while n.val != brr and n.next:
                n = n.next
            cur.next = n.next
        else:
            return
```

參考資料：http://alrightchiu.github.io/SecondRound/hash-tableintrojian-jie.html
https://zh.wikipedia.org/wiki/哈希表