Lab 10 : Hashing

วัตถุประสงค์ ฝึกเรื่อง hashing : hash table, hashing function, collision policy : separate chaining & open addressing-linear probing, rehashing by resizing, load factor, collission

ทฤษฎี hashing เป็นการเก็บข้อมูลในตาราง เพื่อให้การสืบค้นข้อมูลไม่ต้องไล่หาข้อมูลไปทีละตัว ดังนั้นในการเก็บข้อมูลจึง map key ของข้อมูลเข้ากับ index ของ array (ตาราง) เพื่อนำข้อมูลไปเก็บที่ index นั้น เมื่อจะหาข้อมูล ก็ map key ของข้อมูลกับ index ของ array ด้วยวิธีเดิมเพื่อหาว่าเก็บข้อมูลไว้ที่ index ใด แล้วไปหาข้อมูลนั้นได้ทันที ไม่ต้องไล่หาดังวิธีอื่นที่เคยเรียนมาในเรื่อง searching ฟังก์ชั่นที่ใช้ map key ของข้อมูลเข้ากับ index ของ array เรียกว่า **hashing function** เช่นถ้า key คือ รหัสนักศึกษา ถ้าทุกคน id นำหน้าด้วย 54011 เราอาจใช้ hasing function เป็น

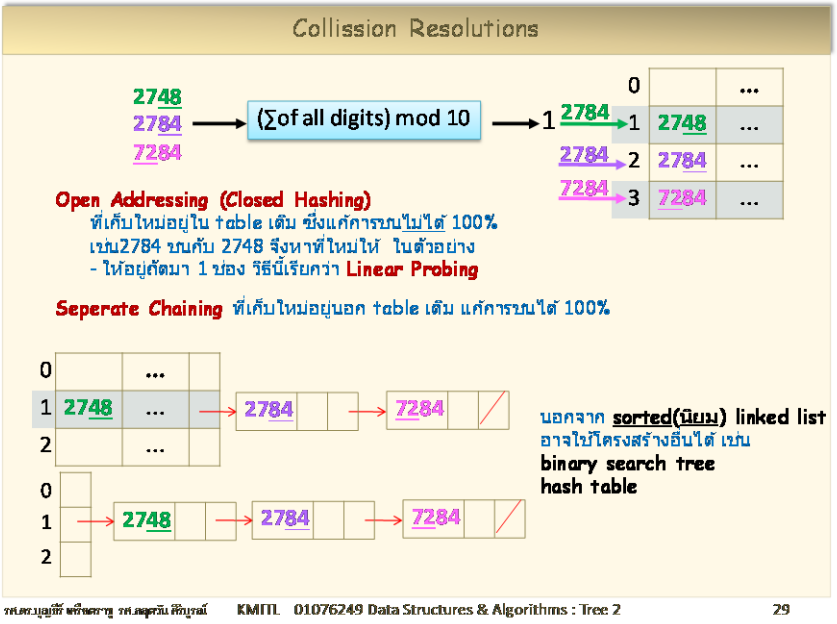
hf(key) = (key-54011000) mod ARRAY_SIZE

hf(54 01 1037) = 37

hf(54 01 1576) = 576

การ mod ด้วย ARRAY_SIZE ทำให้ได้ index ที่อยู่ใน range ของ array algorithm ที่ hash key ให้เป็น index เรียก hasing algorithm array (ตาราง) เรียก hashing table ในการ hash ข้อมูลอาจได้ index ที่ซ้ำกัน เรียกว่าเกิด การชนกัน collision หาก index นั้นมีข้อมูลอื่นเก็บอยู่แล้ว จึงต้องหา index ใหม่ให้ข้อมูลที่ไปชนกับเขา เรียกว่าการแก้ collision (collision resolution) ซึ่งแบ่งเป็น 2 วิธีใหญ่ๆ คือ

1. **Separate Chaining** : สร้าง linked list ณ index นั้น เพื่อเก็บ data ที่ hash แล้วได้ index นี้ทั้งหมด ซึ่งอาจเป็น linear list หรือ binary search tree ก็ได้ วิธีนี้แก้ collision ได้ 100%
2. **Open Addressing (close hashing)** : หา index ใหม่ในตาราง hash เดิม เรียกว่า **rehashing** คือ hash อีกครั้งให้ได้ index ใหม่ ซึ่งมีหลายวิธีเช่น **Linear Probing** (ลองช่องถัดไป) และ **Quadratic Probing** เป็นต้น วิธีนี้ไม่สามารถแก้ collision ได้ 100% อาจเกิดการชนขึ้นอีก



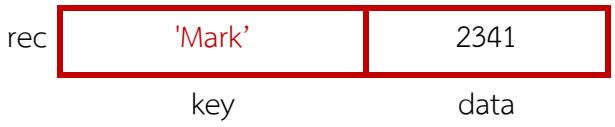
การชนกันจะเกิดขึ้นบ่อยถ้าตารางแน่น ความแน่นของตาราง **Load factor** λ

คือจำนวน element ใน hash table / table size ในวิธี Open Addressing แก้โดยขยายขนาดของตาราง แล้วนำข้อมูลมา hash อีกครั้งเพื่อใส่ตารางใหม่ ดังนั้นจึงเรียกว่า **rehash** เหมือนกัน การ resize ตารางอาจดูจากค่า λ ที่มากเกินต้องการ หรือ วัดจากจำนวน probe (จำนวนครั้งที่หยั่งลงไปในแต่ละช่องของตาราง) ที่มากเกินต้องการ แล้วแต่นโยบาย

การทดลอง

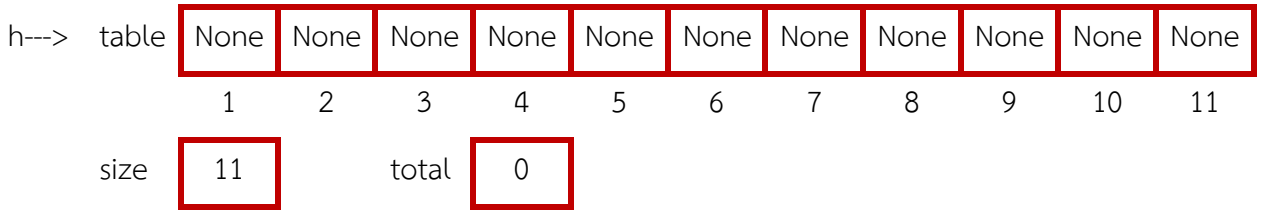
ต้องการเก็บข้อมูลพนักงาน มีชื่อ และ รหัส โดยใช้ key คือ ชื่อ

1. สร้าง class เพื่อเป็น record ของของที่ต้องการเก็บใน hash table มีค่า key และ data (เพื่อความง่ายสำหรับเตรียมข้อมูล แต่ความจริง อาจเก็บข้อมูลอื่นๆอีกได้ เช่น รายละเอียดสินค้า . . .)



ใน class เขียนฟังก์ชั่น `__str__(self)` เพื่อพิมพ์ (key, data) เช่นกรณีตัวอย่างข้างบนจะพิมพ์ ('mark', 2341)

2. สร้าง class HashTable เขียนฟังก์ชั่น __init__() ให้ใช้ python list สร้าง hash table และให้มี size ตั้งต้นเป็น prime 11 และให้ table ทั้ง 11 ช่องเป็น None เพราะเริ่มต้นยังไม่มี data h = hashTable() จะได้ รูปข้างล่าง (เก็บ table, size, total (จำนวนของใน hash table))



3. ใน class HashTable เขียนฟังก์ชั่น :

3.1. เขียนฟังก์ชั่น **`hash(str, tablesize)`** เพื่อ return ค่า hash ที่ได้จากการ hash string เข้า table ขนาด tablesize จะใช้วิธี sum ค่า ord(ch) ของ character ทุกตัวอักษรก็ได้ หรือ อาจใช้วิธี Horner's Rule คือเป็น polynomial ของ 32 ก็ได้ (ดู algorithm ในทฤษฎี) ให้ return ค่า index ที่ hash ได้

3.2. เขียนฟังก์ชั่น **`rehash()`** กำหนด parameter เองโดยให้ใช้จำนวนครั้งที่ rehash (ที่ชน) มาคำนวณด้วย โดยใช้ linear probing ให้ return ค่า index ที่ rehash ได้

ข้างล่างเป็น code ตัวอย่าง นศ อาจเขียนแบบอื่น หรือ ใช้ parameter แบบ อื่นก็ได้ ตัวอย่างข้างล่าง hash() และ rehash() ใช้เป็นฟังก์ชั่นภายใน

```
class rec:

    def __init__(self, key, data):
        self.key = key
        self.data = data

    def __str__(self):
        s = '(' + str(self.key) + ',' + str(self.data) + ')'
        return s

class HashTable:

    def __init__(self):
        self.size = 11
        self.table = [None]*self.size
        self.total = 0

    def hash(key, tablesize):
        """sumation of all ASCII"""
        sum = 0
        for pos in range(len(key)):
            sum = sum + ord(key[pos])
        return sum % tablesize

    def hash2(str, tablesize):
        """horner's rule polynomial of 32"""
        sum = 0
        for pos in range(len(str)):
            sum = (sum<<5) + ord(str[pos])
                            return sum%tablesize

    def rehash(j, firstHV, tablesize): #linear probing ชนครั้งที่ j
        return (firstHV + j) % tablesize
```

3.3. เขียนฟังก์ชั่น `put(self, key, data)` เพื่อใส่ rec ที่มี key และ data เข้าไปใน hash table โดยใช้ hash() และ rehash() ที่เขียนไว้แล้ว หากมี key อยู่แล้ว แสดงว่าต้องการเปลี่ยนค่า data ของ key ตัวนั้นเป็นค่าใหม่ ลองใช้ put() เพื่อเอา record ต่างๆ ใส่ใน hash table เช่น

```
h=HashTable()

h.put('Ann', 2431)
h.put('Tony', 7222)
h.put('Tony', 7221)
h.put('Jim',  1026)
```

```
*** puting Ann 2431 ***
----- table size = 3 , total = 1  -----
0 : (Ann,2431)

*** puting Tony 7222 ***
colission 1 at 0
----- table size = 3 , total = 2  -----
0 : (Ann,2431)
1 : (Tony,7222)

*** puting Tony 7221 ***
+++ already have this key, changing data
+++
----- table size = 3 , total = 2  -----
0 : (Ann,2431)
1 : (Tony,7221)

*** puting Jim 1026 ***
colission 1 at 0
colission 2 at 1
----- table size = 3 , total = 3  -----
0 : (Ann,2431)
1 : (Tony,7221)
2 : (Jim,1026)
```

3.4. เขียนฟังก์ชั่น `printTable(self)`

เพื่อพิมพ์ข้อมูลต่างๆใน table เพื่อตรวจสอบ

เช่น หากทุกครั้งที่ เรียก `h.printTable()` จะได้ output

3.5. เขียนฟังก์ชั่นพิเศษ เพื่อ overload subscript operator [ ] ให้สามารถ set item ณ index ที่กำหนด __setitem__(self, index, data) ดังนั้นสามารถ

เขียน `h['Tony'] = 7221` แทน `h.put('Tony', 7221)` ได้

__setitem__( ) เพื่อให้ทำ index ได้ และภายในเราเรียก put() อีกทีหนึ่งเพื่อใส่ record ลงใน hash table

```
def __setitem__(self, key, data):
        self.put(key,data)
```

3.6. เขียนฟังก์ชั่น `get(self, key)` เพื่อ return data ของ key จาก hash table การใช้เช่น

`print(h.get('Tony'))` ==> จะ print 7221

3.7. เขียนฟังก์ชั่นพิเศษ เพื่อ overload subscript operator [ ] ให้สามารถ get item ณ index ที่กำหนด __getitem__(self, index) ดังนั้นสามารถ

`print(h['Tony'])` แทน `print(h.get('Tony'))` ได้

__getitem__( ) เพื่อให้ทำ index ได้ และภายในเราจึงเรียก get() อีกทีหนึ่งเพื่อเอา record จาก hash table

```
def __getitem__(self,key):
    return self.get(key)
```

3.8. เขียนฟังก์ชั่น `resize(self)` เพื่อทำการย้ายข้อมูลทั้งหมดมายังตารางใหม่ที่มีค่า size เป็นค่า prime ตัวแรกที่มากกว่า 2 เท่าของขนาดเดิม และแก้ฟังก์ชั่น ที่ใส่ข้อมูลให้ตรวจสอบว่าหาก load factor มากกว่าหรือเท่ากับ 1.0 ให้ทำการ resize ก่อน แล้วจึงใส่ข้อมูลใหม่ได้ (กำหนดค่ามากที่สุดคือเต็มตารางเพื่อทดสอบการ rehash อาจกำหนดให้น้อยกว่านี้ได้เช่น 0.5) ทั้งนี้ นักศึกษาต้องเขียนฟังก์ชั่น `isPrime(n)` เพื่อทดสอบว่า n เป็น prime หรือไม่ก่อน isPrime() ไม่ต้องอยู่ใน class HashTable

4. นักศึกษาอาจลองใช้อ่านข้อมูลจาก file ใส่ลงใน table ในตัวอย่างข้างล่าง file ชื่อ `'d:input.txt'` มี format คือในแต่ละบรรทัดมี key และ data คั่นด้วย whitespace

```
with open('d:input.txt') as f:
   for line in f:
       list = line.split()
       key = list[0]
       data = list[1]
       h.put(key, data)
```

Mij	8127
Nan	5555
Nyto	2222
Ramk	4567
Jim	8127
Karm	5235
Tim	2244
Mit	4554

5. การเก็บสถิติ นักศึกษาอาจเพิ่มการเก็บสถิติ เช่น

   6.1 จำนวนครั้งที่ขยาย table 6.2 load factor 6.3 จำนวนครั้งที่เกิดการชนกันทั้งหมด

   6.4 ความยาวของ collision chain ที่ยาวที่สุด (ในกรณีใช้ linear probing การชนต่อเนื่องกันในการเก็บข้อมูลแต่ละตัว เอาตัวที่ชนต่อเนื่องยาวที่สุด)

```
#import sys

#orig_stdout = sys.stdout
#f = open('d:out.txt', 'w')
#sys.stdout = f

from math import sqrt
from itertools import count, islice
def isPrime(n):
    if n<2: return False
    for i in islice(count(2), int(sqrt(25))-2):
        #print(i)
        if not n % i:
            return False
    return True
#print(isPrime(5))
#print(isPrime(10))
#---------------------------------------
class rec:
    def __init__(self, key, data):
        self.key = key
        self.data = data
    def __str__(self):
        s = '(' + str(self.key) + ',' + str(self.data) + ')'
        return s
#---------------------------------------
class HashTable:
    def __init__(self):
        self.size = 3
        self.table = [None]*self.size
        self.total = 0

    def printTable(self):
        print('----- table size =', self.size, ', total =', self.total, ' -----')
        for i in range(self.size):
            if self.table[i] != None:
                print(i, ':' , self.table[i])

    def hash(key, tablesize):
        if type(key) is str:
            """sumation of all ASCII"""
            sum = 0
            for pos in range(len(key)):
                sum = sum + ord(key[pos])
            return sum % tablesize

    def hash2(str, tablesize):
        """horner's rule polynomial of 32"""
        sum = 0
        for pos in range(len(str)):
            sum = (sum<<5) + ord(str[pos])
        return sum%tablesize

    def rehash(j, firstHV, tablesize): #linear probling
        return (firstHV + j) % tablesize

    def rehash2(j, firstHV, tablesize): #quadratic probling
        return (firstHV + j*j) % tablesize
```

```
    def getIndex(self, key):
      index = None
      unsuccessful = False # หยุดหาเมื่อเป็น unsuccessful search ไม่มีของในตาราง
      found = False
      i = firstHV = HashTable.hash(key,self.size)
      j = 0
      while  not found and not unsuccessful:
         if self.table[i] != None and self.table[i].key == key:
           found = True
           index = i
         else:
           j += 1
           i = HashTable.rehash(j, firstHV, self.size)
           if i == firstHV: # หาจนหมดแล้ว only for linear probing
               unsuccessful = True
      return index

    def put(self, key, data):
        print('\n*** puting', key, data, '***')
        if self.total/self.size >= 1.0 :
            self.resize()
        i = self.getIndex(key)
        if i is not None:
            print('+++ already have this key, changing data +++')
            self.table[i] = rec(key, data) # replace with new rec
        else:   # this key is not in the table
            i = firstHV = HashTable.hash(key, self.size)
            if self.table[firstHV] is None:
                self.table[firstHV] = rec(key, data)
            else: # collision
                j = 1 # จำนวนการชน (rehash ครั้งที่ 1)
                print('colission', j, 'at', i)
                i = HashTable.rehash(j, firstHV, self.size)
                unsuccessful = False
                while self.table[i] != None and not unsuccessful:
                    j += 1
                    print('colission', j, 'at', i)
                    i = HashTable.rehash(j, firstHV, self.size) # rehash ครั้งที่ j
                    if i == firstHV: # หาจนหมดแล้ว only for linear probing
                        unsuccessful = True
                self.table[i] = rec(key, data) # put or replace with new rec
            self.total += 1

    def __contains__(self, key):
        """ return True for key in the class obj, False otherwise"""
        pass

    def get(self, key):
      data = None
      unsuccessful = False # หยุดหาเมื่อเป็น unsuccessful search ไม่มีของในตาราง
      found = False
      i = firstHV = HashTable.hash(key,self.size)
      j = 0
      while self.table[i] != None and not found and not unsuccessful:
         if self.table[i].key == key:
           found = True
           data = self.table[i].data
         else:
           j += 1
           i = HashTable.rehash(j, firstHV, self.size)
           if i == firstHV: # หาจนหมดแล้ว only for linear probing
               unsuccessful = True
      return data

    def __getitem__(self,key):
```

```
        return self.get(key)

    def __setitem__(self,key,data):
        self.put(key,data)


    def resize(self):
        oldSize =  self.size
        self.size = 2 * oldSize
        while not isPrime(self.size): # get next prime of double size
            self.size += 1

        print('===*** resize from', oldSize, 'to', self.size, '***===')

        oldTable = self.table

        self.table = [None]*self.size
        self.total = 0

        for i in range(oldSize):
            if oldTable[i] != None:
                self.put(oldTable[i].key, oldTable[i].data)

h=HashTable()

h['Ann'] = 2431
h.printTable()
h['Tony'] = 7222
h.printTable()
h['Tony'] =  7221
h.printTable()
h['Jim'] = 1026
h.printTable()



with open('d:input.txt') as f:
   for line in f:
       list = line.split()
       key = list[0]
       data = list[1]
       h.put(key, data)
       h.printTable()
```