

แฮช

Hashing

แฮชทำไม?

- เพื่อเข้าถึงข้อมูลได้อย่างรวดเร็ว
- เราเคยใช้มาก่อนคือ **dictionary**
 - บอก **key** ได้ **value**

Key Value	John 0143	Jack 0001	Jim 0020	Jones 0989	Jane 1233	June 1234	July 9087	Joly 8765
--------------	--------------	--------------	-------------	---------------	--------------	--------------	--------------	--------------

ลองหาแบบ **List**

แฮชทำไม?

- เพื่อเข้าถึงข้อมูลได้อย่างรวดเร็ว
- เราเคยใช้มาก่อนคือ **dictionary**
 - บอก **key** ได้ **value**



Key	John	Jack	Jim	Jones	Jane	June	July	Joly
Value	0143	0001	0020	0989	1233	1234	9087	8765

July → Hash function → 6

ตารางเลข

- เก็บอยู่ในรูปของตารางแบ่งเป็น “ช่อง” (slot)

0	1	2	3	4	5	6	7	8	9	10
77	None	None	None	26	93	17	None	None	31	54

- ตารางเริ่มต้น

[illegible]

ตัวอย่าง

- สนใจเฉพาะตัวเลข
 - 54, 26, 93, 17, 77, 31
- ตารางจำนวน **11** ช่อง
- หากการจับคู่ (**map**) ระหว่างตัวเลขกับตำแหน่งช่อง
 - ง่ายสุดคือ **mod 11**
 - $h(item) = item \% 11$
 - ฟังก์ชันแฮช

Item	Hash Value
54	10
26	4
93	5
17	6
77	0
31	9

0	1	2	3	4	5	6	7	8	9	10
77	None	None	None	26	93	17	None	None	31	54

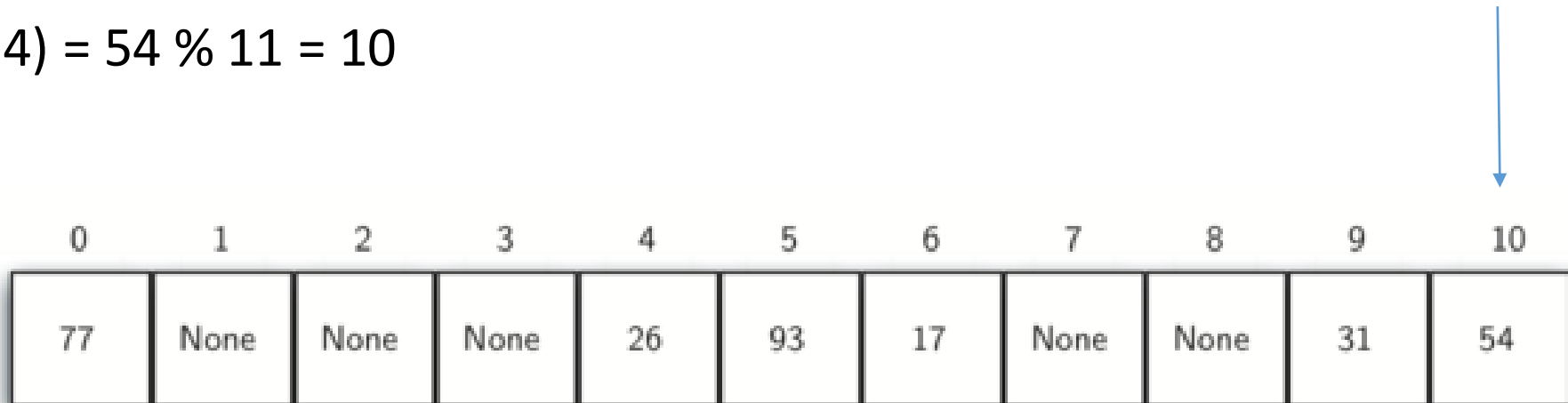
ค่า load factor

- $\lambda = \frac{\text{number of items}}{\text{table size}}$
- จากตัวอย่างคือ $\frac{6}{11}$

0	1	2	3	4	5	6	7	8	9	10
77	None	None	None	26	93	17	None	None	31	54

การเข้าถึงข้อมูล

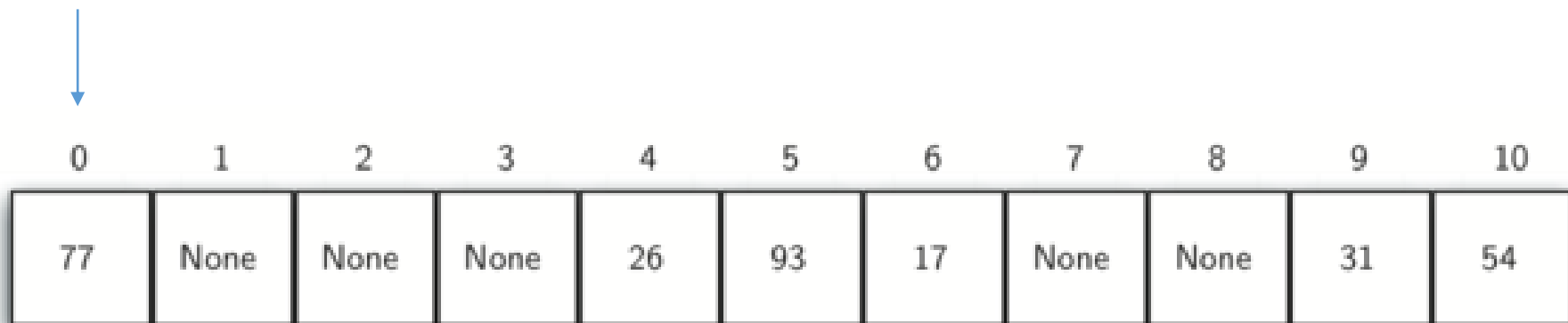
- ใช้ฟังก์ชันแฮช
- เช่น อยากถามว่ามี 54 หรือไม่
 - $h(\text{item}) = \text{item} \% 11$
 - $h(54) = 54 \% 11 = 10$
- $O(1)$



0	1	2	3	4	5	6	7	8	9	10
77	None	None	None	26	93	17	None	None	31	54

การเข้าถึงข้อมูล

- เหมือนจะดีถ้าข้อมูลทั้งหมดลงที่ช่องต่างกันหมด
- ถ้ามี 44 เพิ่มเข้ามา
 - $h(\text{item}) = \text{item} \% 11$
 - $h(44) = 44 \% 11 = 0$
- เกิดการชน (collision)



0	1	2	3	4	5	6	7	8	9	10
77	None	None	None	26	93	17	None	None	31	54

ฟังค์ชันแฮช

- เราอยากได้ฟังค์ชันแฮชที่ไม่มีการชน
 - 1 ช่อง 1 ข้อมูล
- ตัวอย่าง ID 3 หลัก
 - ความเป็นไปได้ทั้งหมดคือ 10^3
 - สร้างตารางขนาด 10^3 จะไม่มีการชนเกิดขึ้น
 - แต่ถ้ามีการใช้ ID จริงๆ แค่ 10 ?
 - Load factor ควรจะเยอะด้วย

ฟังก์ชันแฮชที่ดี

- ชนน้อย
- คำนวณง่าย
- กระจายตัว

วิธีการพับ (folding)

- แบ่งเลขเป็นช่วงๆ บวกกันแล้ว mod
- ตัวอย่างหมายเลขโทรศัพท์ 436-555-4601
- แบ่งช่วงละ 2 ได้ (43,65,55,46,01)
- บวกกัน $43+65+55+46+01 = 210$
- mod จำนวนช่อง $210 \% 11 = 1$
- จับใส่ช่อง 1

ตัวกลางยกกำลัง (mid square method)

- $44^2 = 1936$
- เอาสองตัวกลางมา **mod** จำนวนช่อง $93 \% 11 = 5$

Item	Remainder	Mid-Square
54	10	3
26	4	7
93	5	9
17	6	8
77	0	4
31	9	6

ถ้าเป็นตัวอักษร?

- แปลงตัวอักษรเป็นตัวเลข
- cat
- `ord('c') = 99`
- `ord('a') = 97`
- `ord('t') = 116`
- บวกกัน $99 + 97 + 116 = 312$
- mod ด้วยจำนวนช่อง $312 \% 11$

```
def hash(astring, tablesize):  
    sum = 0  
    for pos in range(len(astring)):  
        sum = sum + ord(astring[pos])  
  
    return sum%tablesize
```

Anagram

- ถ้าเป็น anagram จะได้ค่าแฮชเดียวกัน
- $\text{hash}(\text{'cat'}) = \text{hash}(\text{'act'})$ วน
- ใส่ค่าถ่วงน้ำหนักตามตำแหน่ง
- $\text{hash}(\text{'cat'}) = 99 * 1 + 97 * 2 + 116 * 3 = 641 \% 11 = 3$
- $\text{hash}(\text{'act'}) = 97 * 1 + 99 * 2 + 116 * 3 = 643 \% 11 = 5$

mod ด้วยจำนวนช่องเสมอ !!!!

แก้ปัญหาการชน

ไล่หาช่องว่างช่องแรกที่เราเจอ (linear probing)

- เพิ่ม 44
- $\text{hash}(44) = 44 \% 11 = 0$ ชน

0	1	2	3	4	5	6	7	8	9	10
77	None	None	None	26	93	17	None	None	31	54

- ช่องว่างถัดไปคือ 1

0	1	2	3	4	5	6	7	8	9	10
77	44	None	None	26	93	17	None	None	31	54

ไล่หาช่องว่างช่องแรกที่เราเจอ (linear probing)

- เพิ่ม 55
- $\text{hash}(55) = 55 \% 11 = 0$ ชน

0	1	2	3	4	5	6	7	8	9	10
77	44	None	None	26	93	17	None	None	31	54

- ช่องว่างถัดไปคือ 2

0	1	2	3	4	5	6	7	8	9	10
77	44	55	None	26	93	17	None	None	31	54

ไล่หาช่องว่างช่องแรกที่เราเจอ (linear probing)

- เพิ่ม 20
- $\text{hash}(20) = 20 \% 11 = 9$ ชน

0	1	2	3	4	5	6	7	8	9	10
77	44	55	None	26	93	17	None	None	31	54

- ช่องว่างถัดไปคือ **3** (หมดตารางวนกลับมาเริ่มที่ 0)

0	1	2	3	4	5	6	7	8	9	10
77	44	55	20	26	93	17	None	None	31	54

การเข้าถึงข้อมูลแบบ linear probing

- หาเจอตอบเลยว่า True
 - หา 93
 - $\text{hash}(93) = 93 \% 11 = 5$ เจอ
- หาไม่เจอยังตอบไม่ได้
 - หา 20
 - $\text{hash}(20) = 20 \% 11 = 9$ ชน
 - วิ่งหา 20 เจอตอบ True วิ่งเจอช่องว่าง (None) ถึงจะตอบ False

0	1	2	3	4	5	6	7	8	9	10
77	44	55	20	26	93	17	None	None	31	54

ปัญหาของ linear probing

- ข้อมูลที่ชนกันด้วยค่าแฮชเดียวกันจะถูกระงับตัว
- ตัวที่ $\text{mod } 11 = 0$ ถูกระงับตัวกัน
- เต็ม 20 ต้องไล่หลายช่องกว่าจะเจอช่องว่าง

0	1	2	3	4	5	6	7	8	9	10
77	44	55	None	26	93	17	None	None	31	54

0	1	2	3	4	5	6	7	8	9	10
77	44	55	20	26	93	17	None	None	31	54

กระโดดข้าม

- กระโดดหาช่องว่างทีละ 3 ช่อง
- เต็ม 44 คนที่ช่อง 0 กระโดดไป 3 เจอช่อง 3 ว่าง
- เต็ม 55 คนที่ช่อง 0 กระโดดไป 3 เจอช่อง 3 คนอีก กระโดดไป 3 ช่องเจอช่อง 6 คนอีก กระโดดไป 3 ช่องเจอช่อง 9 คนอีก กระโดดไป 3 เจอช่อง 1 ว่าง
- เต็ม 20 ?

0	1	2	3	4	5	6	7	8	9	10
77	None	None	None	26	93	17	None	None	31	54

0	1	2	3	4	5	6	7	8	9	10
77	55	None	44	26	93	17	20	None	31	54

การแฮช (rehash)

- การหาช่องว่างเมื่อเกิดการชน
- ฟังก์ชันการแฮช
- $\text{rehash}(\text{pos}) = (\text{pos} + 1) \% \text{sizeof table}$
- $\text{rehash}(\text{pos}) = (\text{pos} + 3) \% \text{sizeof table}$
- $\text{rehash}(\text{pos}) = (\text{pos} + \text{skip}) \% \text{sizeof table}$
- วิธีการกระโดดข้ามควรใช้กับขนาดตารางที่เป็นจำนวนเฉพาะ

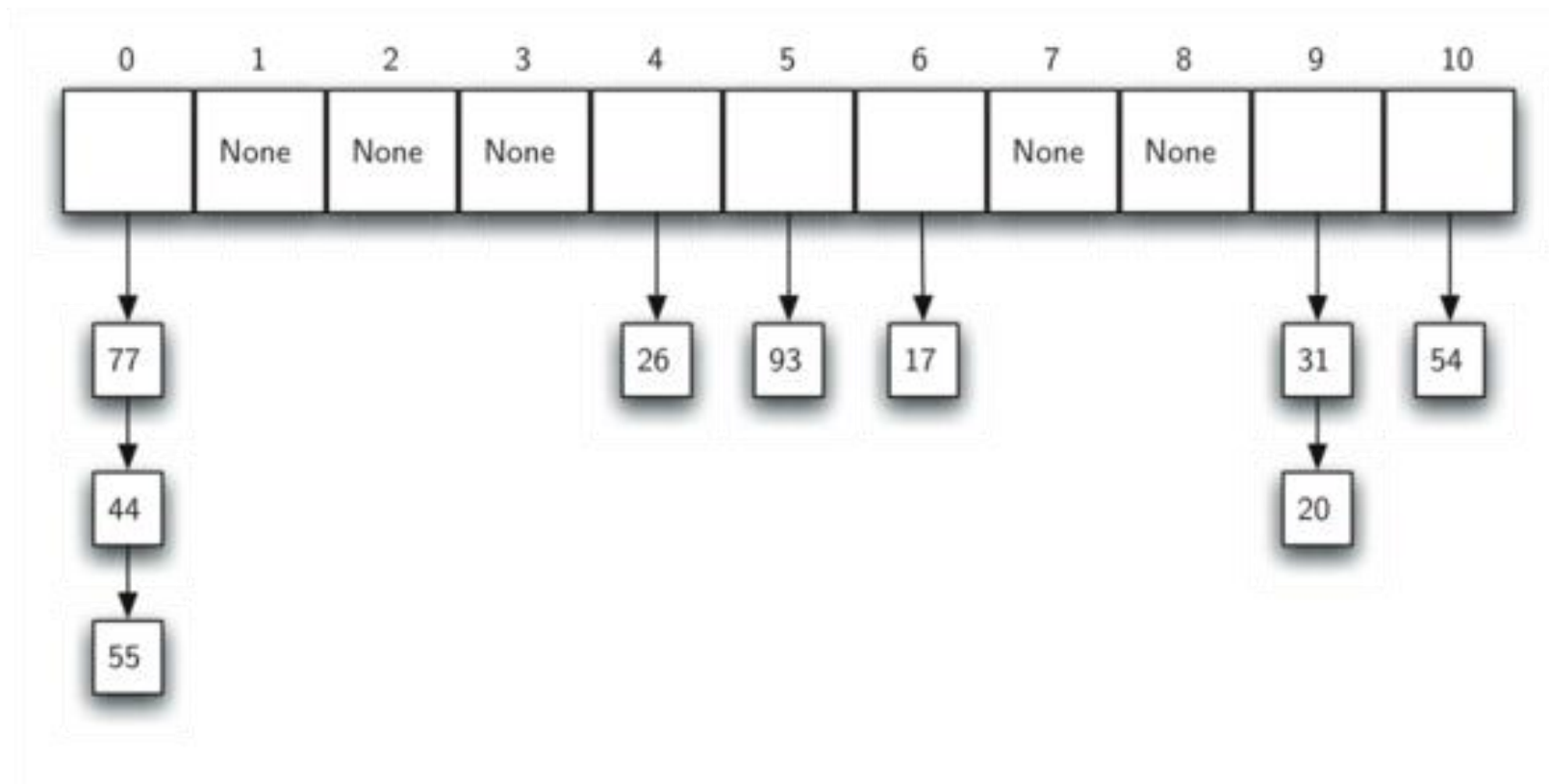
กระโดดแบบยกกำลังสอง

- ชนครั้งแรกกระโดดจากค่าแฮชเดิม $1^2 = 1$
- ชนครั้งที่สองกระโดดจากค่าแฮชเดิม $2^2 = 4$
- ชนครั้งที่สองกระโดดจากค่าแฮชเดิม $3^2 = 9$
- $\text{rehash}(h) = h + 1$
- $\text{rehash}(h) = h + 4$
- $\text{rehash}(h) = h + 9$

0	1	2	3	4	5	6	7	8	9	10
77	None	None	None	26	93	17	None	None	31	54

0	1	2	3	4	5	6	7	8	9	10
77	44	20	55	26	93	17	None	None	31	54

การเก็บเป็นลูกโซ่



Map ADT

- หลักการเดียวกับ **dictionary**
 - เรามองว่า **key** คือค่าที่จะนำไปคำนวณแฮชฟังก์ชันแล้วเก็บ **val** คู่ไปด้วย เรียกอีกอย่างว่า **Map**
- **Map()** สร้าง **map** ว่างแล้วคืนค่า **map** ว่างนั้น
- **put(key, val)** เพิ่มคู่ **key-val** เข้าใน **map** ถ้า **key** มีอยู่แล้วใน **map** ให้แทนที่ค่าเก่าด้วย **val** ใหม่
- **get(key)** คืนค่าที่ **key** นั้น หรือ **None** ถ้าไม่มี **key** นั้น
- **del map[key]** ลบค่าที่ **key**
- **len()** คืนค่าจำนวนที่เก็บอยู่ใน **map**.
- **key in map** คืนค่า **True** เมื่อมี **key** ใน **map** หรือ **False** เมื่อไม่มี **key** นั้น

สร้าง class HashTable

```
class HashTable:  
    def __init__(self):  
        self.size = 11  
        self.slots = [None] * self.size  
        self.data = [None] * self.size
```

```

def put(self, key, data):
    hashvalue = self.hashfunction(key, len(self.slots))

    if self.slots[hashvalue] == None:
        self.slots[hashvalue] = key
        self.data[hashvalue] = data
    else:
        if self.slots[hashvalue] == key:
            self.data[hashvalue] = data #replace
        else:
            nextslot = self.rehash(hashvalue, len(self.slots))
            while self.slots[nextslot] != None and \
                  self.slots[nextslot] != key:
                nextslot = self.rehash(nextslot, len(self.slots))

            if self.slots[nextslot] == None:
                self.slots[nextslot] = key
                self.data[nextslot] = data
            else:
                self.data[nextslot] = data #replace

def hashfunction(self, key, size):
    return key % size

def rehash(self, oldhash, size):
    return (oldhash + 1) % size

```

```

def get(self, key):
    startslot = self.hashfunction(key, len(self.slots))

    data = None
    stop = False
    found = False
    position = startslot
    while self.slots[position] != None and \
           not found and not stop:
        if self.slots[position] == key:
            found = True
            data = self.data[position]
        else:
            position = self.rehash(position, len(self.slots))
            if position == startslot:
                stop = True
    return data

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

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

```

ใช้งาน HashTable

```
>>> H=HashTable()
>>> H[54]="cat"
>>> H[26]="dog"
>>> H[93]="lion"
>>> H[17]="tiger"
>>> H[77]="bird"
>>> H[31]="cow"
>>> H[44]="goat"
>>> H[55]="pig"
>>> H[20]="chicken"
>>> H.slots
[77, 44, 55, 20, 26, 93, 17, None, None, 31, 54]
>>> H.data
['bird', 'goat', 'pig', 'chicken', 'dog', 'lion',
 'tiger', None, None, 'cow', 'cat']
```

```
>>> H[20]
'chicken'
>>> H[17]
'tiger'
>>> H[20]='duck'
>>> H[20]
'duck'
>>> H.data
['bird', 'goat', 'pig', 'duck', 'dog', 'lion',
 'tiger', None, None, 'cow', 'cat']
>> print(H[99])
None
```