Searching

Lecturers :

Boontee Kruatrachue   Room no. 913

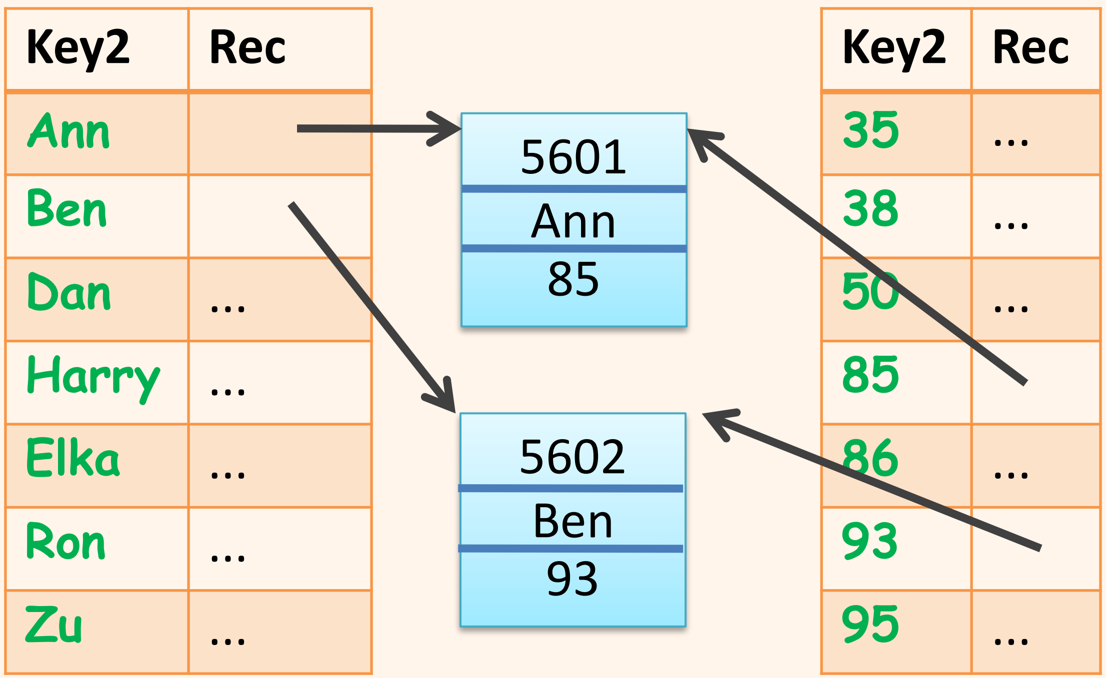
Kritawan Siriboon   Room no. 913

Searching Terminology

- **Search, Record, Table/File**
- **Key**
  - **Internal Key (Embedded Key)**
  - **External Key**

5601	5602	5603	...	...
Ann	Ben	Dan	...	...
85	93	50	...	...

record **table/file**



- **Successful Search/Unsuccessful Search**
- **Internal Search / External Search**

Searching

Search Unordered Table

- **Sentinel Search**
- **Move to Front**
- **Transposition**

Search Ordered Table

Array	List & Tree Search
- Sequential Search (Linear Search)	- Sequential Search (Linear Search)
- Binary Search	-Binary Search Tree

- Hashing	- Heap

Searching Unorder List

rec	19	56	2	7	25	18	...	40		
	1	2	3	4	5	6	...	n	pos	

```
found = false; //Typical version
i= 1;
loop  (i<=n) and (not found)
    if  (key == rec[i].key) {
       foundIndex = i;
       found = true;
    else
       i= i +1;
    end if
end loop
```

```
pos = n+1;      //More efficient version
i = 1;
loop (i<>pos)
   if (key == rec[i].key)
      pos = i;
   else
      i = i+1;
   end if
endloop

if (i<=n)
   search = i;
else
   search = 0;
end if
```

Sentinel Search

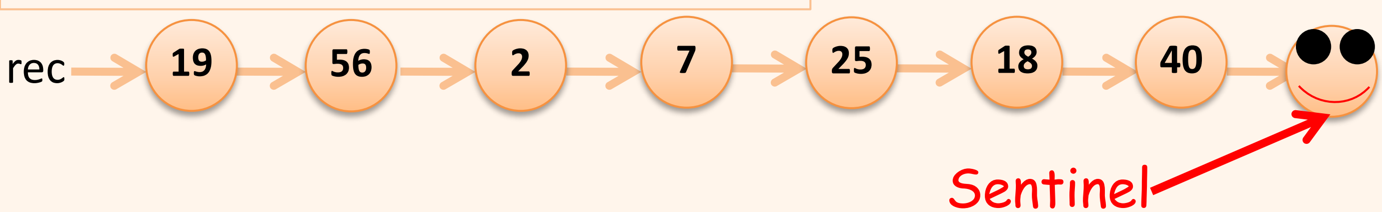
rec	19	56	2	7	25	18	...	40	Sentinel	
	1	2	3	4	5	6	...	n		

```
rec[n+1] = key;        //adding sentinel
i = 1;
loop (key <> rec[i].key)
    i = i+1;
end loop

if (i<n)
    search = i;
else
    search = 0;
end if
```

O(n)

Best Case	1
Worst Case	n
Avg Case	(n+1)/2



Move to Front Heuristic

rec → 19 → 56 → 2 → 7 → 25 → 18 → 40

rec → 7 → 56 → 2 → 19 → 25 → 18 → 40

Move to Front

- แต่ละครั้งที่ search พบ ให้เลื่อน record นั้นขึ้นไปอยู่หน้าสุดของ list (ต้องเป็น linked list version)
- แนวคิด : ของอะไรที่ใช้แล้วมีแนวโน้มที่น่าจะถูกใช้อีกจึงเอามาไว้ข้างหน้า

A **heuristic** is a mental shortcut that allows people to quickly make judgments and solve problems. These mental shortcuts are typically informed by our past experiences and allow us to act quickly. However, heuristics are really more of a rule-of-thumb; they don't always guarantee a correct solution.

Transposition Heuristic

rec										
rec	19	56	2	7	25	18	...	40		

rec										
rec	19	56	7	2	25	18	...	40		

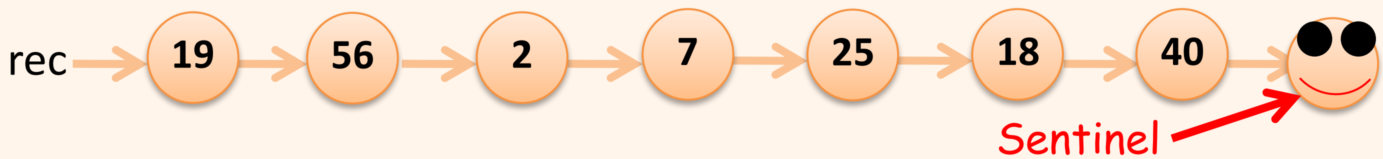
Transposition

- แต่ละครั้งที่ search พบ ให้สลับ record ที่ search พบขึ้นมาข้างหน้า 1 ตำแหน่ง
- แนวคิด : การใช้ครั้งเดียวไม่ได้แปลว่าจะใช้อีกครั้งหนึ่งเสมอไป แต่การสลับแบบนี้หากใช้มากๆ ก็จะเลื่อนขึ้นมาอยู่ข้างหน้าเอง

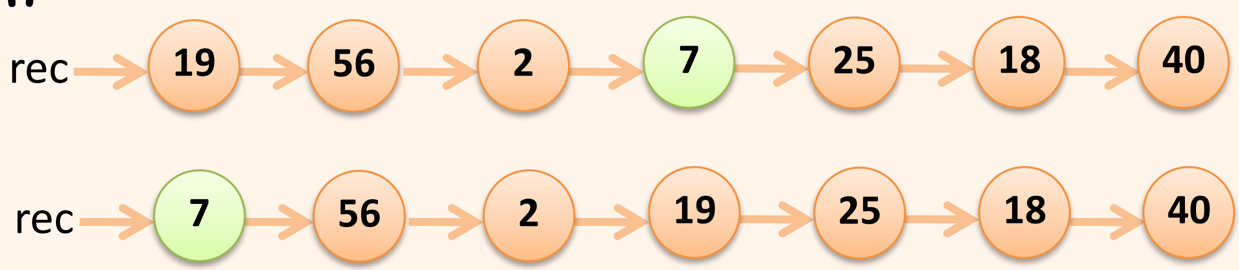
Searching Unordered Table

- **Sentinel Search**

rec									
19	56	2	7	25	18	...	40	Sentinel	



- **Move to Front**



- **Transposition**

rec

19	56	2	7	25	18	...	40		

rec

19	56	7	2	25	18	...	40		

Searching

Search Unordered Table

- Sentinel Search
- Move to Front
- Transposition

Search Ordered Table

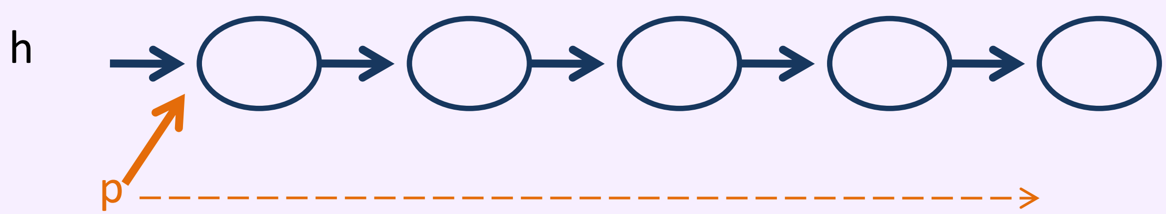
Array	List & Tree Search
- Sequential Search (Linear Search)	- Sequential Search (Linear Search)
- Binary Search	-Binary Search Tree

- Hashing	- Heap

Sequential Search (Linear Search)

2	5	7	8	10	12	15	18	20	22

i



O(n)
Best Case 1
Worst Case n
Avg Case (n+1)/2

Searching

Search Unordered Table

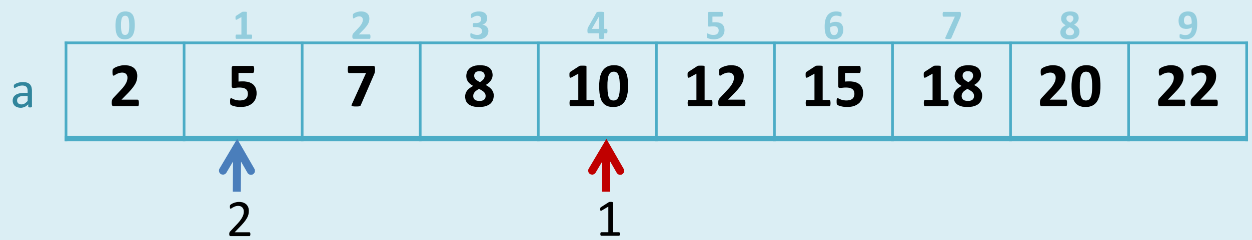
- Sentinel Search
- Move to Front
- Transposition

Search Ordered Table

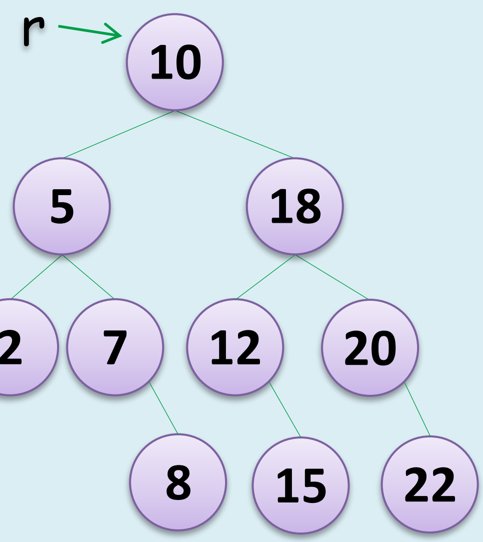
Array	List & Tree Search
- Sequential Search (Linear Search)	- Sequential Search (Linear Search)
- Binary Search	-Binary Tree Search

- Hashing	- Heap

Binary (Tree) Search




```
binarySearch
    low = 1, high = dataSize
    loop ( low <= high)
        mid = ( low + high ) / 2
      if( a[ mid ] < target )
            low = mid + 1
        else if( a[ mid ] > x )
            high = mid - 1
        else low = hight + 1
      end if
    end loop
    if (target == a[mid]
      return mid
    else  return -1
    end if
```



$O(\log_2 n)$

```
binarySearch
    p = q = root
    loop ( p is not null)
        if( target < p(data))
            p = p->left;
        else if ( target > p(data))
            p = p->right;
        else
            q = p
            p = null
        end if
    end loop
    if (target == q->data
      return q
    else  return null
    end if
```

Hashing concepts

1. searches ก่อนๆ นี้ต้องเช็คเทียบหลายครั้งจึงพบหรือพบว่าไม่มี การเช็คเทียบแต่ละครั้ง เรียก probe
2. ความต้องการของ hashing :
   - ทำอย่างไรเราจะหาของได้ใน 1 probe ?

เราต้องรู้ว่ามันอยู่ที่ไหน → เก็บอย่างมีหลักการณ์

Hasing Function

- ตัวตนของแต่ละ record อยู่ที่ไหน ?

  อยู่ที่ key ของมัน ดังนั้นต้อง

  โยง key เข้ากับ ที่เก็บ (array index)

key → Hashing function → index

hf(key) = index

- การโยง key ให้เป็นที่เก็บเรียก hashing algorithm
  ฟังก์ชั่นที่ใช้เรียกว่า hashing function
  เรา hash key ไปสู่ index
- Array ที่เก็บ records เรียก hash table.

Mapping Techniques

Subtraction

- ถ้าทุก id นำหน้าด้วย 54011

hf(key) = key-54 01 1000

hf(54 01 1004) = 4

hf(54 01 1037) = 37

hf(54 01 1576) = 576

Extraction

เอาแค่บางส่วนของ key

54 01 1576 -> 4156

Summation, Division, ...

– **Folding.**
  - แบ่ง key เป็นส่วนๆ
  - summation(/subtraction/...) ส่วนนั้นๆ

  54091576

  = 540 +915+76

  = 1531

  = 531 //array size = 1000

- **Modulo** เพื่อให้อยู่ใน range
- **Midsquare**

  key^2 หรือ $part\_of\_key^2$

  เอาตรงกลางมา

  54011576 -> 576*576

  = 331776 -> 177

Mapping String 1 - 2

Mapping String 1

string → sum all ASCII chars → int

HashString1

```
k = address of first char
HashVal = 0;
loop (*k != null)
   HashVal += *k++
endloop
return HashVal mod TableSize
```

- ปัญหา: table ใหญ่ 10,000 => กระจาย distribute ไม่ดี
- ASCII 0-127, 8 chars => 127*8 =>[0-1,016]

Mapping String 2

string → $(k[0] + 27\ k[1] + 27^2\ k[2])$ % Tablesize → int

HashString2

```
k = address of first char
hv = (k[0]+27* k[1]+729* k[2]) % Tablesize;
if (hv<0)                    // 27=26+blank
     return  hv+= Tablesize
else return  hv
end if
```

ถ้าคิด 26 ตัวไม่นับ blank คิดเฉพาะ 3 chars แรก ==> 17,576 combinations ==> แต่จริง ๆ แล้ว Eng. ไม่ random ดูตาม dic. ได้ 2,851 combinations = 28% (ของ 10,000) ดังนั้น table ใหญ่ ใช้ไม่เต็ม กระจายไม่ดี

Mapping String 3
Horner's rule : Polynomial of 32

string → $(32^4\,k[0]+32^3k[1]+32^2k[2]+32k[3]+k[4])$ % Tablesize → int

$$\sum_{i=0}^{keysize-1} k[keysize-i-1]+32^i$$

Horner's rule : Polynomial of 32

```
k = address of first char
loop (*k != null)
    hv += (hv<<5) + *k++
end loop
hv = hv % Tablesize
return  hv
```

// hv<<1 = hv*2,
// hv<<5=hv*2^5=hv*32

Mapping String 3
Horner's rule : Polynomial of 32

- if keysize = 5
- iteration #1 : 32*0 + K[0] = K[0]
- iteration #2 : 32*K[0] + K[1]
- iteration #3 : 32^2*K[0] + 32*K[1] + K[2]
- iteration #4 : 32^3*K[0] + 32^2*K[1] + 32*K[2] + K[3]
- iteration #4 : 32^4*K[0] + 32^3*K[1] + 32^2*K[2] + 32*K[3] + K[4]

Mapping String

- string ยาวๆ ใช้เวลามาก ==> **truncation**
- 438 Washington NY

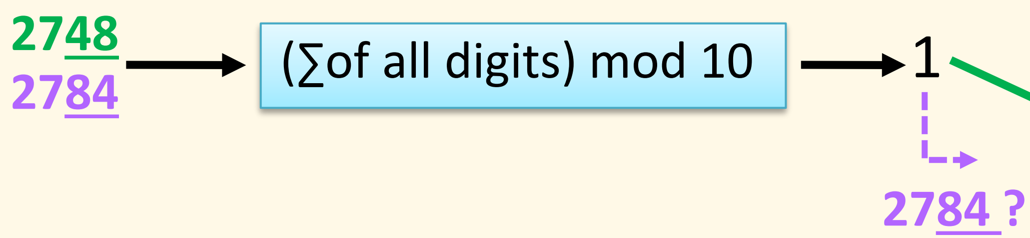
  = 438WaNY

  map -> array range

Collission, Synonym

Collission เกิดเมื่อ hash แล้วได้ index ที่มีของอยู่แล้ว เช่น

- insert 2748 hash(2748) = 1 ว่าง ใส่ใน slot 1
- ,, 2784 hash(2784) = 1 ไม่ว่าง collission ชน กับ 2748 ต้องหาที่เก็บใหม่ให้



	Key	Other Data
0		...
1	2748	...
2		...
3		...
....		
9		

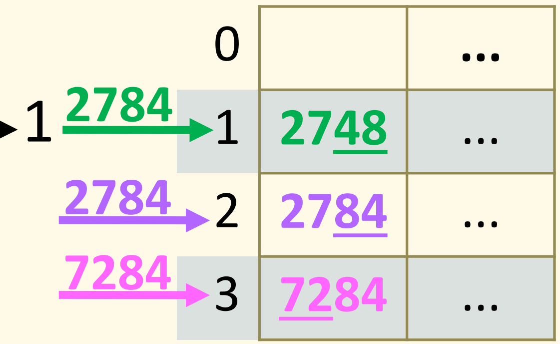
Synonyms : 2748 และ 2784 เป็น synonyms
set of keys ซึ่ง hash (map) ได้ค่าเดียวกัน

- ต้องหาที่เก็บ 2784 ใหม่ เพื่อแก้การชน **Collission Resolutions**
  - ***Open Addressing (Closed Hashing)***
    ที่เก็บใหม่อยู่ใน table เดิม ซึ่งแก้การชนไม่ได้ 100%
  - ***Seperate Chainning*** ที่เก็บใหม่อยู่นอก table เดิม แก้การชนได้ 100%

Collission Resolutions

2748
2784
7284

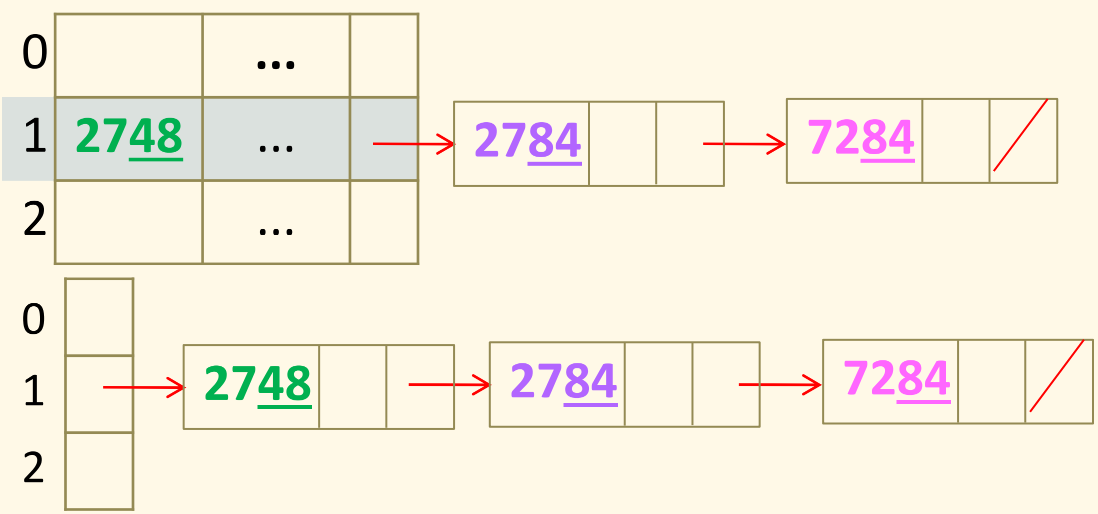
→ (∑of all digits) mod 10 → 1



Open Addressing (Closed Hashing)

ที่เก็บใหม่อยู่ใน table เดิม ซึ่งแก้การชนไม่ได้ 100%
เช่น2784 ชนกับ 2748 จึงหาที่ใหม่ให้ ในตัวอย่าง
- ให้อยู่ถัดมา 1 ช่อง วิธีนี้เรียกว่า **Linear Probing**

Seperate Chaining ที่เก็บใหม่อยู่นอก table เดิม แก้การชนได้ 100%



นอกจาก sorted(นิยม) linked list
อาจใช้โครงสร้างอื่นได้ เช่น
binary search tree
hash table

Good Hashing Function, Load Factor

Hashing Function ที่ดี

- อยู่ใน index range
  ==> mod tablesize
  ==> table size ควรเป็น prime.
- คำนวณ ง่าย รวดเร็ว
- กระจายดี กระจายทั่วถึง
- collision น้อย
- การคำนวณที่ใช้ทั้ง key แทนได้ดีกว่าที่ใช้บางส่วน

Collision การชนกัน

Collision (การชน) เกิดขึ้นเมื่อ hash ได้ index ที่มีของอยู่แล้ว จึงต้องหาที่เก็บใหม่ให้

มีทางแก้ 2 วิธี

1. **Seperate Chaining**

   ที่เก็บใหม่อยู่นอก table เดิม แก้การชนได้ 100%

2. **Open Addressing**

   ที่เก็บใหม่อยู่ใน table เดิม ซึ่งแก้การชนไม่ได้ 100%

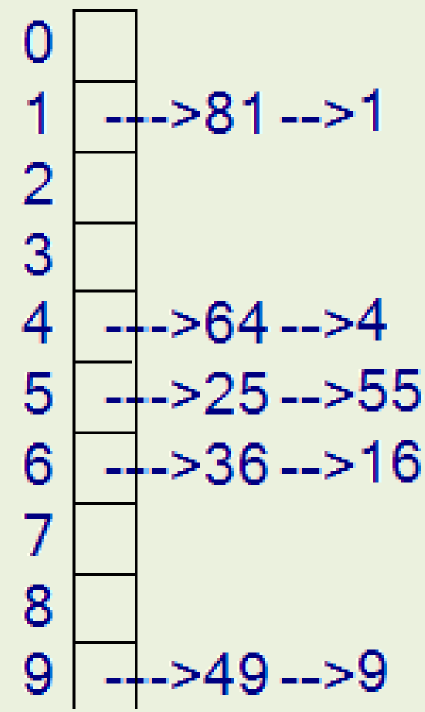
Seperate Chaining

ที่เก็บใหม่อยู่นอก table เดิม แก้การชนได้ 100%

Seperate Chaining (cont.)

- **Load factor λ** = จำนวน element ใน hash table / table size
  - ในรูปตัวอย่าง λ = 10/10 = 1.0
- จากการวิเคราะห์ : table size ไม่ค่อยสำคัญเท่าไหร่ ที่สำคัญคือ λ
- general rule สำหรับ separate chaining hash table
  - $\lambda \sim 1$
  - table size เป็น prime
- ข้อเสียของ separate chaining คือ ใช้ data structure ที่ 2 เช่น linked list ซึ่งทำให้ช้าลง (allocate new node, implement 2nd data structure)
- ข้อดี : 100% solved collission

- h(x) = x mod 10
- Table size = 10 (not prime for simplicity)



Open Addressing

ที่เก็บใหม่อยู่ใน table เดิม ซึ่งแก้การชน<u>ไม่ได้</u> 100%

Linear Probing

probe ชนครั้งที่ i	ลอง	h(k) + f(i)
Linear Probing	f(i) = i ลอง	: h(k), h(k)+1, h(k)+2, h(k)+3,...
Quadratic Probing	$f(i) = i^2$ ลอง	: h(k), h(k)+1, h(k)+4, h(k)+9,...

Rehash: hash อีกครั้งเพื่อให้ได้ที่ใหม่

hf(key) = key mod 10

h(1111) = 1 ชน

rehash ครั้งที่ 1 = 1+1 = 2 ชน

rehash ครั้งที่ 2 = 1+2 = 3 ชน

rehash ครั้งที่ 3 = 1+3 = 4 ว่าง ใส่ได้

รวมทั้งหมด 4 probes

0		...
1	2151	...
2	2872	...
3	4553	...
4	**1111**	
...	...	...
9		

array size = 10
(not prime for simplicity)

Linear Probing

- การคำนวณง่าย search ง่าย
- มีแนวโน้มให้เกิด
  การกระจุกตัว (**Clustering**) ของ data
  -> collission

Linear Probing

probe ชนครั้งที่ i	ลอง	h(k) + f(i)
Linear Probing f(i) = i	ลอง	: h(k), h(k)+1, h(k)+2, h(k)+3,...

insert 54
54%7 = 5

insert 70
70%7 = 0

insert 61
61%7 = 5

insert 83
83%7 = 6

insert 15
15%7 = 1

insert 34
34%7 = 6

0	
1	
2	
3	
4	
5	
6	

probes:

h(key) = key mod 7

Linear Probing

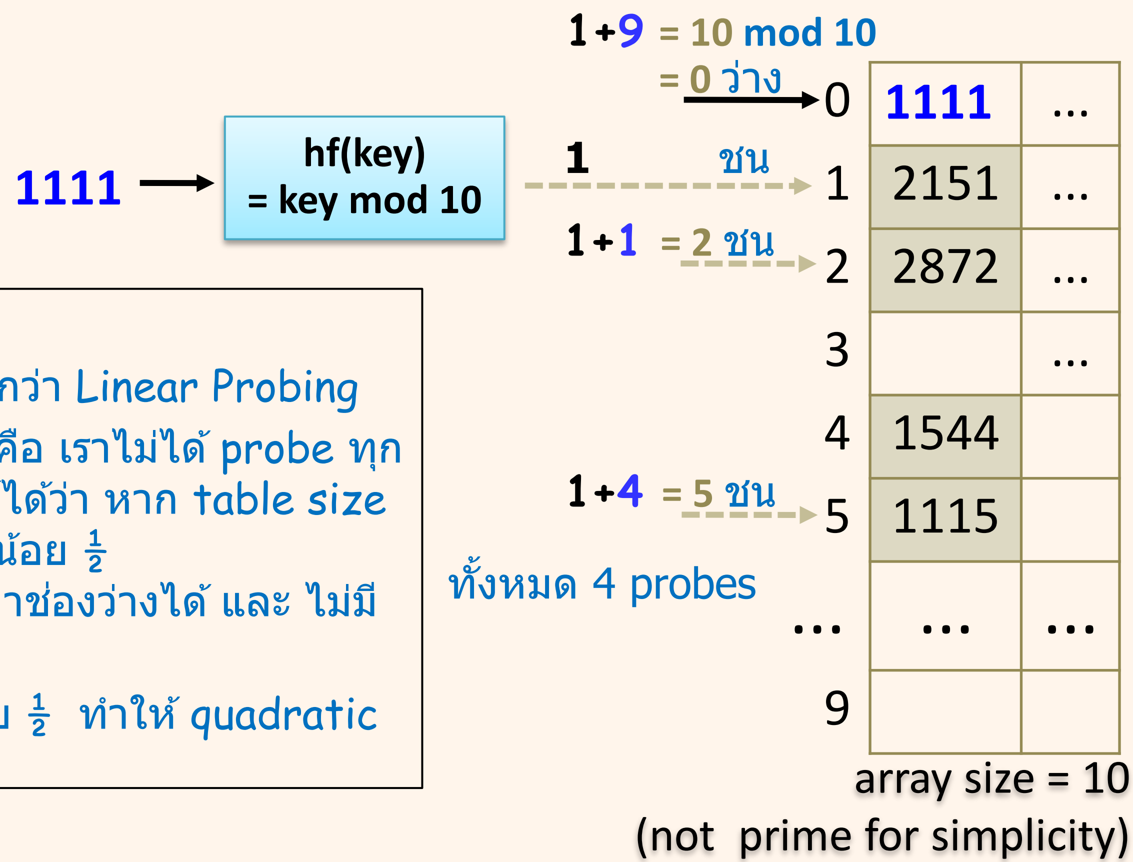
probe ชนครั้งที่ i	ลอง	h(k) + f(i)
Linear Probing f(i) = i	ลอง :	h(k), h(k)+1, h(k)+2, h(k)+3,...

	insert 54 54%7 = 5	insert 70 70%7 = 0	insert 61 61%7 = 5	insert 83 83%7 = 6	insert 15 15%7 = 1	insert 34 34%7 = 6
0		70	70	70	70	70
1				83	83	83
2					15	15
3						34
4						
5	54	54	54	54	54	54
6			61	61	61	61
probes:	1	1	2	3	2	5

h(key) = key mod 7

Quadratic Probing

probe ชนครั้งที่ i	ลอง	h(k) + f(i)
Quadratic Probing f(i) = i^2	ลอง :	h(k), h(k)+1, h(k)+4, h(k)+9,...
Linear Probing f(i) = i	ลอง :	h(k), h(k)+1, h(k)+2, h(k)+3,...



Quadratic Probing

- มีการกระจาย(distributes) มากกว่า Linear Probing
- ปัญหาของ quadratic probing คือ เราไม่ได้ probe ทุกช่องของ table แต่สามารถพิสูจน์ได้ว่า หาก table size เป็น prime และ table ว่างอย่างน้อย $\frac{1}{2}$ quadratic probing จะสามารถหาช่องว่างได้ และ ไม่มีการ probe ซ้ำช่องเดิม
- แต่การที่ table ต้องว่างอย่างน้อย $\frac{1}{2}$ ทำให้ quadratic probing : *space inefficient*

Quadratic Probing

probe ชนครั้งที่ i	ลอง	$h(k) + f(i)$
Quadratic Probing $f(i) = i^2$	ลอง	: $h(k)$, $h(k)+1$, $h(k)+4$, $h(k)+9$, . . .

insert 54	insert 72	insert 61	insert 1	insert 33	insert 70
54%7 = 5	72%7 = 2	61%7 = 5	1%7 = 1	33%7 = 5	70%7 = 0

index						
0						
1						
2						
3						
4						
5						
6						

probes:

h(key) = key mod 7

Quadratic Probing

probe ชนครั้งที่ i	**ลอง**	$h(k) + f(i)$
Quadratic Probing $f(i) = i^2$	**ลอง** :	$h(k)$, $h(k)+1$, $h(k)+4$, $h(k)+9$, ...

	insert 54 54%7 = 5	insert 72 72%7 = 2	insert 61 61%7 = 5	insert 1 1%7 = 1	insert 33 33%7 = 5	insert 70 70%7 = 0
0					33	33
1				1	1	1
2		72	72	72	72	72
3						
4						70
5	54	54	54	54	54	54
6			61	61	61	61
probes:	1	1	2	1	4	3

Try : 5,
5+1=6,
5+4=9%7=2,
5+9=14%7=0

Try : 0,
0+1=1,
0+4=4,

h(key) = key mod 7

Rehashing

Rehashing

- ถ้า table แน่นเกินไปจะเริ่มทำให้แต่ละ operation ใช้เวลานาน และอาจ insert ไม่ได้สำหรับ open addressing แบบ quadratic
- แก้ได้โดย สร้าง table ขึ้นใหม่ ใหญ่ขึ้น 2 เท่า และนำ data จาก table เดิมทั้งหมด มา hash ใส่ table ใหม่ เรียกว่า rehashing (rehash ทุก data เข้า table ใหม่) (rehash อีกความหมายหนึ่งคือ hash อีกครั้งเมื่อ collission)
- เมื่อใดบ้างที่ต้อง rehash
  - ทันทีที่ table เต็ม
  - เมื่อ insert ไม่ได้ (นโยบายเข้มงวด)
  - table เต็มถึงระดับที่กำหนดไว้ (ถึงค่า load factor ที่กำหนด) (นโยบายเข้มงวดปานกลาง)

Rehashing

0	6
1	15
2	
3	24
4	
5	
6	13

Insert :
13, 15, 6, 24

0	6
1	15
2	23
3	24
4	
5	
6	13

Insert : 23
Over 70 % full

h(x) = x mod 7, Linear Probing

Rehashing

- สร้าง table ขึ้นใหม่ ขนาด 17
  (ค่า prime ถัดไปที่ใหญ่ขึ้น 2 เท่า)
- insert data ใหม่ทั้งหมด ด้วย
  h(x) = x mod 17
- แพง = O(n)

Rehashing →

0	
1	
2	
3	
4	
5	
6	6
7	23
8	24
9	
10	
11	
12	
13	13
14	
15	15
16	

Done