บทที่ 10
การค้นหาข้อมูล
(Searching)

สุนทรี คุ้มไพโรจน์

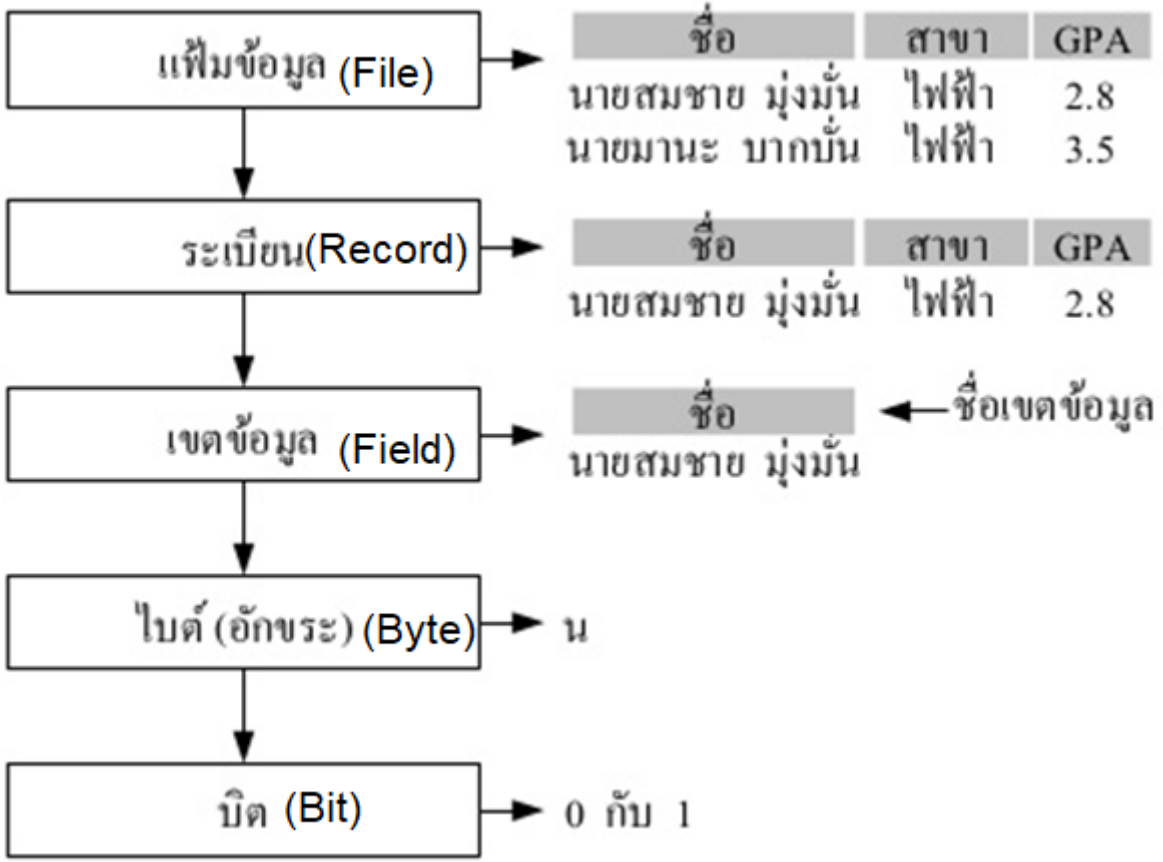
Searching

- การค้นหาข้อมูล จะให้ผลลัพธ์บอกได้ว่าค้นหาเจอหรือไม่
- กรณีที่ค้นหาเจอจะต้องบอกได้ว่าค้นหาเจอที่ตำแหน่งใด

รูปแบบการ search

- Sequential search
- Binary search
- Search โดยใช้ Hashing function

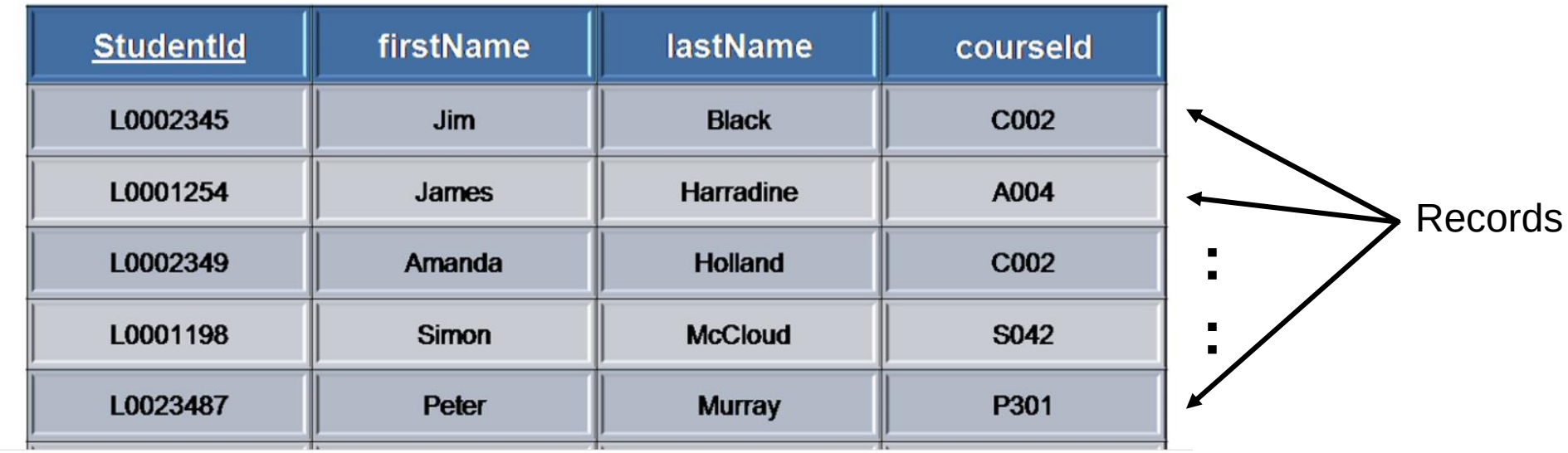
โครงสร้างของแฟ้มข้อมูล



ที่มาภาพ https://sites.google.com/site/dgtryrg/home/naewkhid-kar-rwbrwm-kar-cad-rabeiyb-than-khxmul

Primary keys

Primary Keys เป็น คีย์หลักที่กำหนดจากฟิลด์ที่ไม่มีข้อมูลซ้ำซ้อน (unique) ใน Table เดียวกัน โดยเด็ดขาด และจะต้องมีค่าเสมอ จะเป็นค่าว่าง (Null) ไม่ได้



StudentId	firstName	lastName	courseId
L0002345	Jim	Black	C002
L0001254	James	Harradine	A004
L0002349	Amanda	Holland	C002
L0001198	Simon	McCloud	S042
L0023487	Peter	Murray	P301

ที่มา http://rdbms.opengrass.net/2_Database%20Design/2.1_TermsOfReference/2.1.2_Keys.html

Sequential Search

- **เป็นการค้นหาในลักษณะเชิงเส้น**
- **เป็นวิธีที่ง่ายที่สุด** แต่**ก็เป็นวิธีที่ด้อยประสิทธิภาพกว่าวิธีอื่น**
- เหมาะสมสำหรับการประมวลผลข้อมูลกับบางประเภท เช่น งานกลุ่ม (Batch Processing) งานสำรองข้อมูล (Backup) งานปรับปรุงแก้ไขข้อมูลให้ทันสมัย (Update)
- **เปรียบเทียบคีย์ (key) ที่ต้องการค้นหา กับข้อมูลทีละตำแหน่ง ไปจนกว่าจะพบ หรือจนกว่าจะหมดข้อมูล**

ตัวอย่าง

9 7 15 60 32 40 53 84 21 96

- การค้นหาข้อมูลจะต้องผ่านการตรวจสอบข้อมูลที่อยู่ข้างหน้าระเบียนข้อมูลที่ต้องการทุกตัวจนกว่าจะพบข้อมูล
- โดยมีจำนวนครั้งของการเปรียบเทียบ ดังนี้
- **Best Case**

  น้อยที่สุด = 1 ครั้ง (ข้อมูลที่ต้องการอยู่เป็นระเบียนแรก)
- **Worst Case**

  มากที่สุด = จำนวนข้อมูลทั้งหมดในแฟ้มข้อมูล (ระเบียนสุดท้าย)
- **Average Case**

  โดยเฉลี่ย = 1/2 ของจำนวนข้อมูลทั้งหมดในแฟ้มข้อมูล

Sequential Search

- **ถ้าข้อมูลในแฟ้มข้อมูลเรียงตามลำดับ(Sort)**
  ขั้นตอนการค้นหาจะทำได้เร็วขึ้น คือ
  ถ้าพบว่าค่าคีย์(Key)ของระเบียนข้อมูลมีค่า > ค่าของดัชนี
  แสดงว่าระเบียนข้อมูลที่ต้องการนั้นไม่อยู่ในแฟ้มข้อมูลนี้
  และไม่จำเป็นต้องทำการค้นหาอีกต่อไป
- **ถ้าแฟ้มข้อมูลไม่ได้จัดเรียงลำดับ**
  จะต้องค้นหาข้อมูลในแฟ้มข้อมูลจนครบทุกตัว
  จึงจะสรุปได้ว่าไม่มีระเบียนข้อมูลที่ต้องการนั้นอยู่ภายในแฟ้มข้อมูล
- **พิจารณาประสิทธิภาพของการค้นหาได้เป็น O(n)**

2 3 4 6 8

อัลกอริทึม

```
int  sequentialSearch (int key[],int size, int value){
    int i=0;
    for(; i<size && key[i] !=value; i++);
        if(i<size)
            return i;
    return -1;
}
```

ตัวอย่าง

ค้นหาข้อมูลที่เรียงลำดับแล้ว ค้นหาเลข 11

จากข้อมูล ถ้าต้องการค้นหา เลข 11 แบบ binary search จำนวนครั้งในการค้นหาจะเป็นเท่าไร

ค้นหาข้อมูลที่เรียงลำดับแล้ว ค้นหา 6

จากข้อมูล ถ้าต้องการค้นหา **เลข 6** แบบ binary search
จำนวนครั้งในการค้นหาจะเป็นเท่าไร

ค้นหาข้อมูลที่ไม่เรียงลำดับ แล้วค้นหา 15

ค้นหาข้อมูลที่ **ไม่เรียงลำดับ** แล้ว**ค้นหา** **9**

Complexity of Algorithms

- ความซับซ้อนของอัลกอริทึม
- เป็นการวิเคราะห์ประสิทธิภาพของอัลกอริทึม
  - ขนาดของข้อมูล (n)
  - เวลา วัดจากจำนวนการทำงาน จำนวนการเปรียบเทียบ
  - พื้นที่ วัดจากจำนวนพื้นที่หน่วยความจำที่ใช้ในอัลกอริทึม
- อยู่ในรูปของฟังก์ชัน f(n)

การหาค่าความซับซ้อน

- Best case ค่าต่ำสุดของ f(n)
- Average case ค่าเฉลี่ยของ f(n)
- Worst case ค่าสูงสุดของ f(n)

Average case

- วิเคราะห์โดยใช้หลักการทางสถิติเกี่ยวกับความน่าจะเป็นของข้อมูล
- ข้อมูลอาจจะอยู่ที่ตำแหน่งใด ๆ ในระหว่าง 1 2 3 ...n
- แต่ละตำแหน่งจะมีค่าความน่าจะเป็น p = 1/n

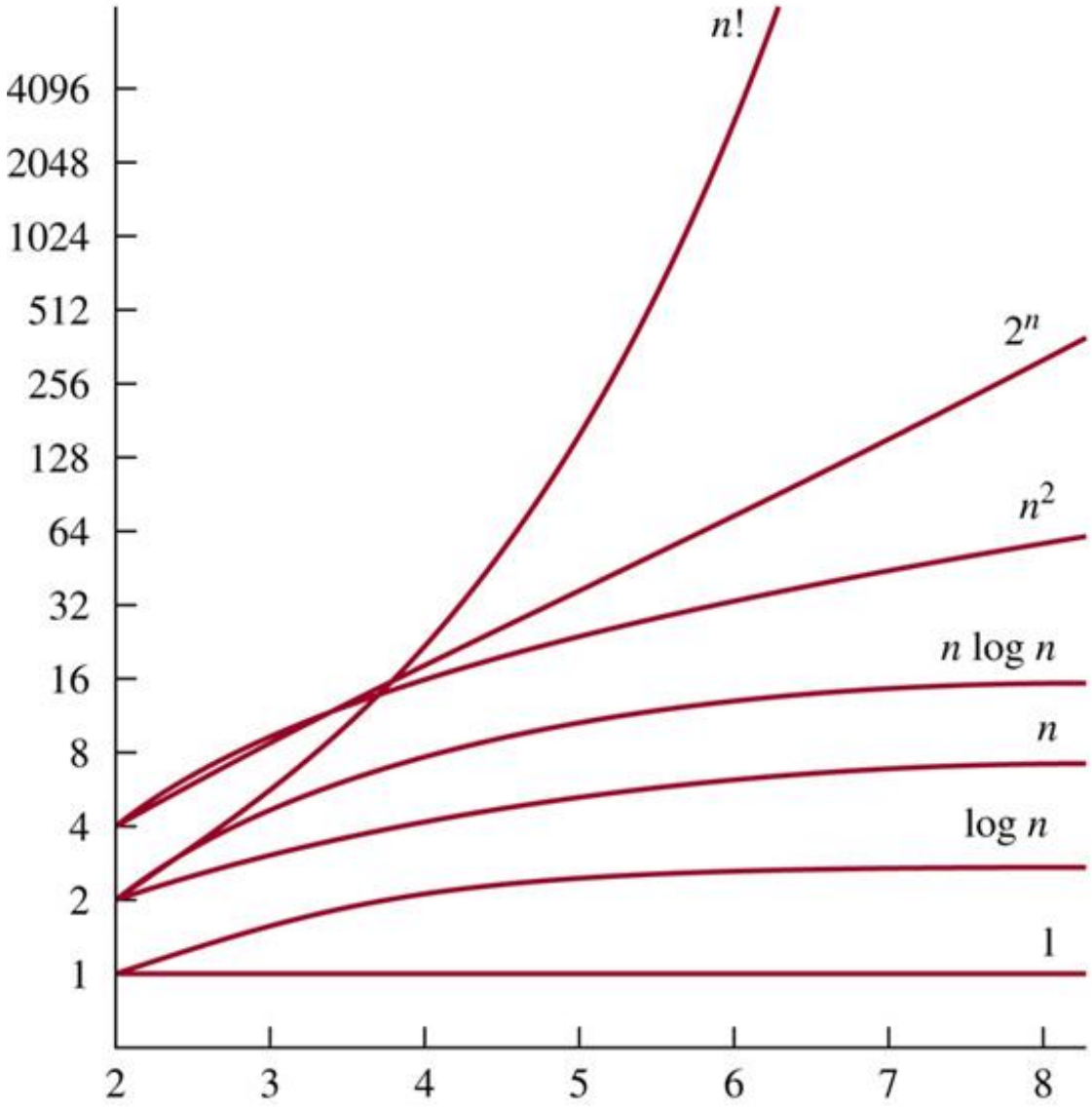
$$C(n) = 1*1/n+2*1/n+\ldots+n*1/n$$
$$= (1+2+\ldots+n) * 1/n$$
$$= n(n+1)/2 * 1/n = (n+1)/2$$

Rate of growth : Big O notation

อัตราการเพิ่มของฟังก์ชันมาตรฐาน(ค่าที่เพิ่มขึ้นเมื่อ n เพิ่มขึ้น)

n \ g(n)	log n	n	n log n	n^2	n^3	2^n
5	3	5	15	25	125	32
10	4	10	40	100	1000	1000
100	7	100	700	10000	10^6	10^{30}
1000	10	1000	10000	10^6	10^9	10^{300}

Rate of growth



Big Oh Notation

สมมติ $f(n)$ และ $g(n)$ เป็นฟังก์ชันที่ให้ค่าจำนวนบวก

โดยที่ $f(n)$ ถูกจำกัดด้วย $g(n)$ สำหรับทุกค่าของ n และ $n > n_0$

$$f(n) \leq g(n)$$

เขียนในรูป $f(n) = O(g(n))$

อ่านว่า $f(n)$ is of order $g(n)$

ตัวอย่างเช่น ให้ $f(n) = 8n^3 – 576n^2+8n-24 = O(n^3)$

Binary Search

- การค้นหาแบบทวิภาค
- การค้นหาโดยใช้วิธีแบ่งครึ่งข้อมูลทั้งหมดออกเป็นสองส่วน
- การค้นหาตัวที่ต้องการจะต้องเลือกหาจากข้อมูลทีละส่วน

  โดยใช้วิธีเดิม**คือ**แบ่งเป็นสองส่วนย่อย

  หากพบในส่วนที่หนึ่ง ก็ไม่ต้องเสียเวลาหาในอีกส่วนหนึ่งอีกต่อไป
- เพื่อให้เกิดประโยชน์สูงสุดในการ search

  Binary search จะมีการ sort ก่อน

  เพื่อตัดข้อมูลส่วนที่ไม่ใช่ทิ้งไป

Binary Search

- 1. เปรียบเทียบค่าข้อมูลที่ต้องการค้นหากับ Primary Key ของข้อมูล **ระเบียนที่ 1**

  - ถ้าใช่ ระเบียนที่ต้องการให้ไปทำขั้นตอนที่ 7
  - ถ้าไม่ ให้ทำขั้นตอนต่อไป

- 2. เปรียบเทียบค่าข้อมูลที่ต้องการค้นหากับ Primary Key ของข้อมูล **ระเบียนที่ n**

  - ถ้าใช่ ระเบียนที่ต้องการให้ไปทำขั้นตอนที่ 7
  - ถ้าไม่ ให้ทำขั้นตอนต่อไป

Binary Search (ต่อ)

- 3. **กำหนดจุดเริ่มต้นและจุดสุดท้ายของการค้นหา**
- 4. เปรียบเทียบค่าข้อมูลที่ต้องการค้นหากับ Primary Key ของข้อมูล ณ **ตำแหน่งกึ่งกลางของชุดระเบียนข้อมูล**
  - **ถ้าใช่ ระเบียนที่ต้องการให้ข้ามไปทำขั้นตอนที่ 7**
  - ถ้าไม่ ให้ทำขั้นตอนต่อไป
- 5. เปรียบเทียบข้อมูลที่ต้องการค้นหากับค่าของPrimary Key **ของข้อมูล ณ ตำแหน่งกึ่งกลางของชุดระเบียนข้อมูล**
  - ถ้าค่าข้อมูลที่ต้องการค้นหามี**ค่ามากกว่า** ให้เปลี่ยนตำแหน่งจุดเริ่มต้นเป็นตำแหน่งที่อยู่**ทางขวา**ของตำแหน่งกึ่งกลางนั้น
  - ถ้าค่าข้อมูลที่ต้องการค้นหามีค่า**น้อยกว่า** ให้เปลี่ยนตำแหน่งจุดสุดท้าย เป็นตำแหน่งที่อยู่**ก่อนหน้า**ตำแหน่งกึ่งกลางนั้น

Binary Search

- 6. ตรวจสอบจำนวนครั้งของการเปรียบเทียบเกิน n/3 ครั้งหรือไม่
  - ถ้าใช่ ให้แสดงข้อความ**แจ้งการค้นหาไม่พบ** และไปทำใน**ขั้นตอนที่ 8**
  - ถ้าไม่ ให้***ย้อนกลับไปทำตั้งแต่ขั้นตอนที่ 4 อีกครั้ง***
  7. แสดงข้อความแจ้งการค้นพบตำแหน่งที่เก็บข้อมูลที่ต้องการ
- 8. จบการค้นหา

ตัวอย่าง

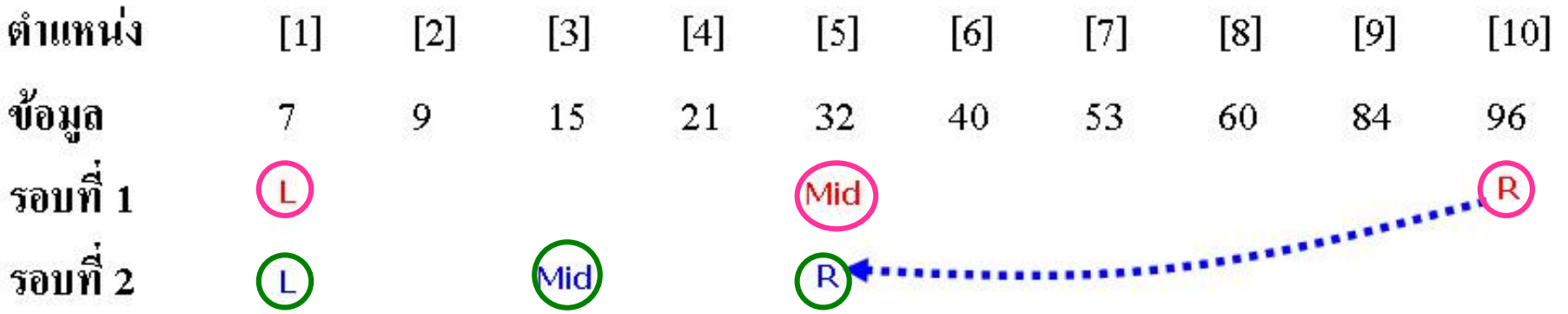
ตำแหน่ง	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]
ข้อมูล	7	9	15	21	32	40	53	60	84	96
	L									R

- สมมติต้องการ**ค้นหา 15** จากข้อมูลชุดนี้ (**คือ 15**)

1. การหาตำแหน่งกึ่งกลางจะคำนวณโดยนำเลข**ตำแหน่งแรก**ของชุดข้อมูล **บวกกับเลขตำแหน่งสุดท้าย**ของชุด **หารด้วยสอง**
   - แต่เนื่องจากการหารสอง อาจทำให้เกิดเลขทศนิยมได้ จึงใช้ฟังก์ชั่น div ช่วยในการหารแทน ดังนั้นจะได้ว่า **Mid = (L+R) div 2**
   - จากตัวอย่าง ตำแหน่งกึ่งกลางได้แก่ **Mid = (1+10) div 2 = 5**

ตำแหน่ง	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]
ข้อมูล	7	9	15	21	32	40	53	60	84	96
	L				Mid					R

- **2.** นำ**คีย์ที่ต้องการค้นหา(คีย์ 15)** เปรียบเทียบกับข้อมูลใน**ตำแหน่ง Mid**
  - **หากคีย์มีค่า > จะตัดข้อมูลด้านซ้ายของ Mid ทิ้งไป** (คือไม่พิจารณาอีกแล้ว เนื่องจากเป็นข้อมูลที่มีค่าน้อยกว่าคีย์ทั้งสิ้น)
  - **หากคีย์มีค่า < ค่าข้อมูลในตำแหน่ง Mid ก็จะตัดข้อมูลด้านขวาออก**
- **จากตัวอย่างพบว่า**
  - **คีย์ คือ 15** มีค่า **< ค่าข้อมูลที่ตำแหน่ง Mid คือ 32**
  - **จึงทำการย้ายค่า R มาไว้ที่ตำแหน่ง Mid (คือการตัดข้อมูลด้านขวาทิ้ง)**



- **ตำแหน่ง Mid ของรอบที่ 2** คือ **ตำแหน่งที่ 3** **{(1+5)/2}**
- นำ คีย์ **คือ 15** มาเปรียบเทียบกับข้อมูลที่ตำแหน่ง Mid **(ตำแหน่งที่ 3)**
- สามารถตอบได้ว่าพบข้อมูลที่ต้องการที่**ตำแหน่งที่ 3**

- กรณีที่ขยับตำแหน่ง L R Mid จนกระทั่งขยับไม่ได้อีกแล้ว หมายถึง**ไม่พบข้อมูลที่ต้องการ**

ประสิทธิภาพของการค้นหาข้อมูลด้วย Binary Search

- พิจารณากระบวนการทำงานแบบ Binary Search จะพบว่าเมื่อมีการ**เปรียบเทียบแต่ละครั้งจะมีการตัดข้อมูลในตารางออกไปได้ทีละครึ่งหนึ่ง**เสมอ ดังนั้นถ้าเริ่มต้นมีจำนวน**ข้อมูล n ตัว**
- จำนวนข้อมูลที่นำมาเพื่อค้นหาค่าที่ต้องการหลังการเปรียบเทียบแต่ละครั้งจะเป็นดังนี้
- ครั้งที่ 1 = n
- ครั้งที่ 2 = n/2
- ครั้งที่ 3 = n/4
- ครั้งที่ 4 = n/8

- หรืออาจเขียนได้เป็น $n \rightarrow n/2^1 \rightarrow n/2^2 \rightarrow n/2^3 \ldots \rightarrow n/2^k$
- ดังนั้นถ้าให้ k เป็นจำนวนครั้งที่มากที่สุดที่ใช้ในการเปรียบเทียบ จะได้ว่า

ทำไป k รอบ แล้ว ผลลัพธ์เท่ากับ 1 นั่นคือ $n/2^k = 1$

$$n/2^k = 1$$

หรือ $$n = 2^k$$

จะได้ $$k = \log_2 n = O(\log n)$$

สรุปได้ว่าประสิทธิภาพของการค้นหาแบบ Binary คือ $O(\log n)$

การค้นหาด้วยวิธีแฮชชิ่ง (Hashing Search)

- การค้นหาด้วยวิธีการแฮชชิ่ง หรือแฮชชิ่งเซอร์ช (Hashing Searching) นี้เป็นวิธีการแปลงคีย์ให้เป็นแอดเดรส (address)
- คำว่า**คีย์(Key)**ในที่นี้หมายถึง **ข้อมูลที่ต้องการการค้นหา** หรือข้อมูลในชุดข้อมูลที่ถูกค้นหา
- การแปลงคีย์ให้เป็นแอดเดรส คือการแปลงข้อมูลให้ไปอยู่ในตารางที่เตรียมไว้ ซึ่งตารางนั้นเรียกว่า *ตารางแฮช* (Hash Table) โดยการแปลงนี้ใช้ฟังก์ชันที่เรียกว่า ***แฮชชิ่งฟังก์ชัน*** (Hashing Function)
  ถ้าให้ H เป็นแฮชชิ่งฟังก์ชัน
  K เป็นคีย์ และ
  A เป็นแอดเดรสที่ได้หลังจากแปลงคีย์ไปแล้ว
  อาจเขียนสัญลักษณ์การแปลงดังนี้

H(K) → A

Hashing Function

- มีหลายฟังก์ชั่น การเลือกใช้ขึ้นอยู่กับความเหมาะสมของข้อมูล
  ตัวอย่างของฟังก์ชั่นแฮชมีดังนี้

- **1. Mod** คือการนำค่าคีย์มา mod ด้วยค่า n ใด ๆ ฟังก์ชั่น mod จะให้ผลลัพธ์เป็นเศษที่ได้จากการหาร เช่น 10 mod 3 = 1

  5 mod 3 = 2

- ดังนั้น ค่า n จึงมีผลต่อขนาดของตารางแฮช
  ข้อแนะนำ ค่าที่นำมาใช้ควรเป็น**<u>ค่าจำนวนเฉพาะ</u>**ที่ใกล้เคียงกับขนาดของตารางที่ต้องการ
  เช่น หากต้องการสร้างตารางแฮชที่มีขนาด 100 ช่อง
  ค่า n ที่แนะนำคือ 101

1. Mod

- ฟังก์ชั่นแฮชที่นิยมกันมากก็คือ mod เขียนเป็นสัญลักษณ์ได้ดังนี้

  H(K) = K MOD M

- เมื่อกำหนดให้
  - **k คือ ค่าข้อมูลที่ต้องการจัดเก็บ**
  - **M คือ ขนาดของเนื้อที่ที่กำหนดไว้**
- แฮชชิ่งฟังก์ชันถูกใช้งานสองอย่าง คือ

  1. กำหนดแอดเดรสให้ชุดข้อมูล
  2. บอกแอดเดรสที่ต้องไปค้นหาเมื่อระบุข้อมูลที่ต้องการค้น

  ซึ่งงานทั้งสองอย่างนี้ต่างกันก็คือ การแปลงคีย์ให้เป็นแอดเดรส

ตัวอย่าง 9 7 15 60 32 40 53 84 21 96

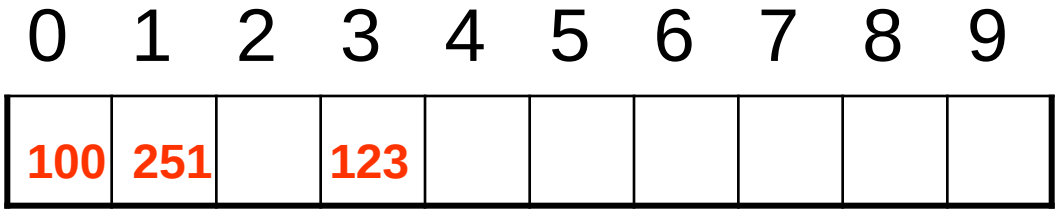
- 1. แปลงคีย์เป็นที่อยู่
- โดยใช้ฟังก์ชั่น mod 11 เพื่อจัดเก็บข้อมูล
- **H(K) = K MOD M**
- **H(9) = 9 mod 11 = 9**
- **H(7) = 7 mod 11 = 7**
- **H(15) = 15 mod 11 = 4**

0	
1	
2	
3	
4	15
5	
6	
7	7
8	
9	9
10	

ตัวอย่าง 9 7 15 60 32 40 53 84 21 96

- **2. การค้นหาข้อมูล** โดยการนำคีย์ที่ต้องการค้นหามาผ่านฟังก์ชั่นแฮชเดียวกันกับการจัดเก็บ เช่น **จากตัวอย่าง ต้องการค้นหาคีย์ 33**
- **H(33) = 33 mod 11 = 0**
- **ค้นหาที่ตำแหน่ง 0 ไม่พบข้อมูล 33**
  แสดงว่าไม่มีข้อมูล 33 อยู่ในชุดข้อมูลนี้
- จากกระบวนการในการค้นหาข้อมูลดังกล่าว
  ประสิทธิภาพของการค้นหาคือ **O(1)**
  เนื่องจากไม่ว่าจะมีข้อมูลมากหรือน้อยเท่าใด
  การเปรียบเทียบก็จะ**กระทำเพียง 1 ครั้งเท่านั้น**
  นับว่าเป็นวิธีที่มีประสิทธิภาพสูง

ตัวอย่างของวิธีการหาร **Mod**



H(100) = 100 mod 10 = 0

H(251) = 251 mod 10 = 1

H(123) = 123 mod 10 = 3

2.Mid-Square

- คือ**การนำคีย์มายกกำลังสอง แล้วเลือกเฉพาะค่ากลางของข้อมูล** จำนวนหลักขึ้นอยู่กับความต้องการในการใช้งาน
- **ตัวอย่างเช่น**
- คีย์คือ 12 ผ่านฟังก์ชั่น Mid-Square คือ $12^2 = 1\underline{4}4$ เลือกเฉพาะตำแหน่งกลางได้ค่าที่อยู่เป็น 4

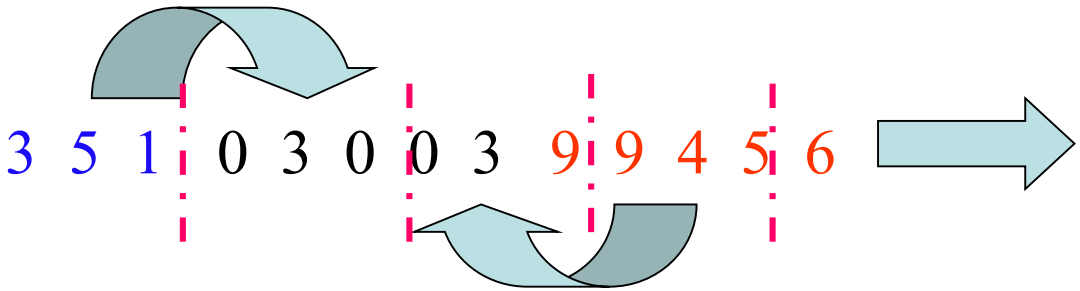
3.Folding วิธีการพับตัวเลข

- คือการนำคีย์มาแบ่งเป็นส่วน ๆ พับทบเข้าหากันแล้วหาผลรวม
  เช่น **กำหนดตำแหน่งที่อยู่ไว้ 3 หลัก**

3. วิธีการพับตัวเลข (Folding Method)

- จะต้องทราบว่ามีการกำหนดตำแหน่งสำหรับเก็บข้อมูลไว้สูงสุดเป็นกี่หลัก
- หลังจากนั้นจะนำค่าที่ต้องการไปจัดเก็บลงบนตำแหน่งต่าง ๆ นั้น
- เช่น 3510300399456 กำหนดตำแหน่งกำหนดไว้ 5 หลัก

ตัวอย่าง วิธีการพับตัวเลข



0 3 0 0 3

1 5 3

6 5 4 9 9

8 3 8 0 2

กำหนดตำแหน่งที่อยู่ไว้ 5 หลัก

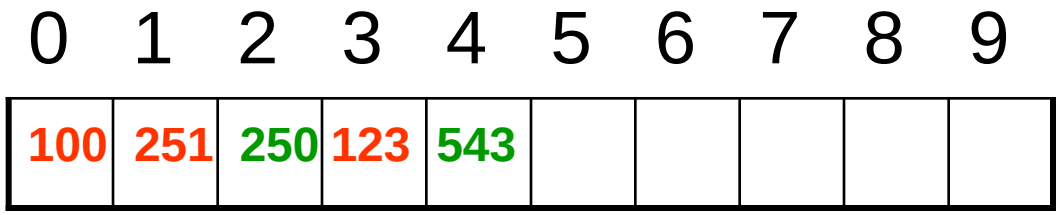
การชนกันของข้อมูล

- การจัดเก็บข้อมูลลงในตารางแฮช อาจพบปัญหาที่เรียกว่าการชนกัน (Collision) ของข้อมูล เกิดจากการที่นำคีย์มาผ่านฟังก์ชั่นแล้วได้ที่อยู่เป็นตำแหน่งเดียวกัน **ตัวอย่างเช่น**
- ต้องการจัดเก็บ **22 และ 33** ในตารางแฮช
- **H(22) = 22 mod 11 = 0**
- **H(33) = 33 mod 11 = 0**
- วิธีการในการแก้ปัญหาการชนกันแบ่งเป็น 2 วิธีใหญ่ ๆ ดังนี้

1. Open Addressing

- เป็นวิธีที่ใช้ตำแหน่งที่เหลือในตารางในการเก็บข้อมูลที่ชน มีวิธีย่อย ๆ คือ
- 1.1 **Linear Probing** คือการ**มองหาช่องว่างถัดไปในตาราง แล้วจัดเก็บข้อมูลที่ชนในช่องว่างแรกที่พบ** หากใช้วิธีนี้ในการจัดเก็บ ในขั้นตอนของการค้นหา ก็จะต้องดำเนินการด้วยวิธีเดียวกัน
- แต่ Linear Probing อาจทำให้เกิดปัญหาตามมาคือ การแทนที่ที่ไม่ถูกต้อง นั่นคือ
- เมื่อพบช่องว่างสำหรับคีย์ที่ชนกันแล้ว หากหลังจากนี้มีคีย์ที่เข้ามาใหม่ที่เมื่อผ่านฟังก์ชั่นแล้วได้ที่อยู่ที่แท้จริงเป็นตำแหน่งเดียวกันกับตัวที่ชน ก็จะทำให้เกิดการชนกันเช่นนี้เรื่อยไป

ตัวอย่าง Linear



H(100) = 100 mod 10 = 0

H(251) = 251 mod 10 = 1

H(123) = 123 mod 10 = 3

H(250) = 250 mod 10 = 0

H(543) = 543 mod 10 = 3

1.2 Double Hashing

- เป็นความพยายาม**แก้ปัญหาของ linear probing** ที่เสาะหาใน**ตำแหน่งที่ซ้ำ ๆ อันเป็นสาเหตุของการชนครั้งแล้วครั้งเล่า**
- โดยอาศัย **randomness ของ hash function ในการหาตำแหน่งใหม่** หลังจากเกิดการชนกันขึ้น collision resolution function ก็สร้างในรูปแบบตรงไปตรงมา
- **ที่นิยมกันมาก คือ F(i) = i * h2(x) ซึ่ง h2(x)** เป็นฟังชันใหม่**ที่ไม่ซ้ำกับ hash function เดิม**

Double Hashing people.cs.pitt.edu/~kirk/cs1501/notes/double%20hashing.txt

Create a second hash function h2(x) to be the increment (generalize linear probing).

Example:

M=11, h(x) = x mod 11, h2(x) = x mod 7 + 1

Hash: 14, 17, 25, 37, 34, 16, 26

	Probes	
14 mod 11 = 3	1	
17 mod 11 = 6	1	
25 mod 11 = 3	2	25 mod 7 + 1 = 5
37 mod 11 = 4	1	
34 mod 11 = 1	1	
16 mod 11 = 5	1	
26 mod 11 = 4	2	26 mod 7 + 1 = 6
	9 probes	or 9/7 ~ 1 probe per key

```
 0:
 1: 34
 2:
 3: 14
 4: 37
 5: 16
 6: 17
 7:
 8: 25
 9:
10: 26
```

Find 47:

h(47) = 47 mod 11 = 3 not there
h2(47) = 47 mod 7 + 1 = 6 (cell is empty)

Probes	
1	
1	
2 probes	vs. 6 for linear probing (see previous example)

2. Chaining

วิธีนี้เป็นการสร้าง **singly linked list**
ที่มีตำแหน่งหลักใน **hash table เป็นฐาน (หัว) ของ chain**
(หนังสือบางเล่มเรียกว่า bucket)
แต่ละตำ แหน่งหลักจึงมี list ของตนเองที่เก็บข้อมูลที่ซ้ำกันอันเป็นผลพวง
มาจากการ mapped หรือ hashed เข้าสู่ตำ แหน่งหลักเดียวกัน

ตัวอย่างเช่น ต้องการจัดเก็บ 22 และ 33 ในตารางแฮช

H(22) = 22 mod 11 = 0

H(33) = 33 mod 11 = 0

H(44) = 44 mod 11 = 0

	Data	Link
0	22	11
1		
2		
3		
4	15	
5	60	
6		
7	7	12
8		
9	9	
10	32	
11	33	13
12	40	
13	44	
14		
15		
16		
17		

Overflow

ตัวอย่างเช่น

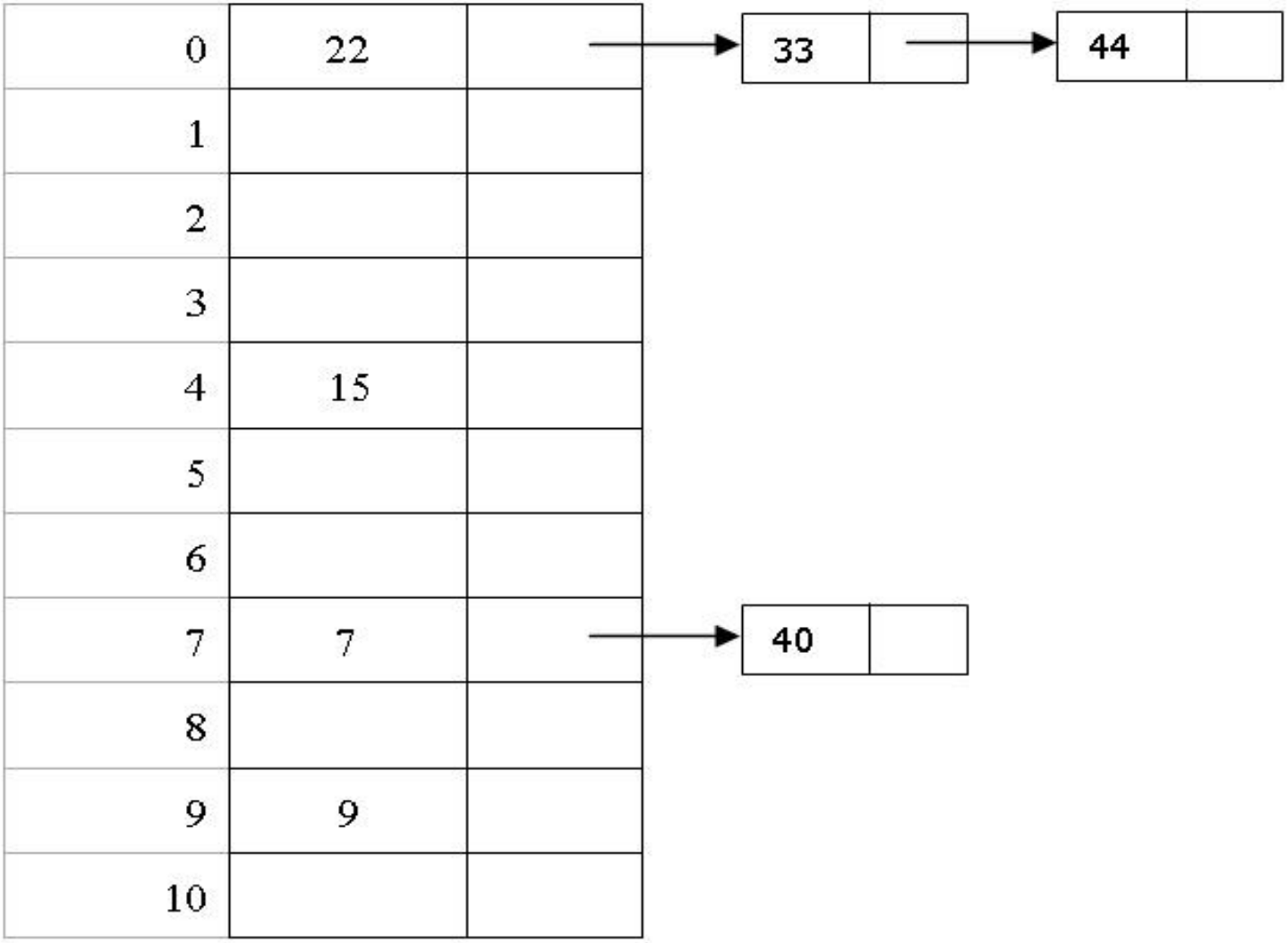
ต้องการจัดเก็บ 22 และ 33 ในตารางแฮช

$$H(22) = 22 \bmod 11 = 0$$

$$H(33) = 33 \bmod 11 = 0$$

$$H(44) = 44 \bmod 11 = 0$$

จากตัวอย่างเป็นการใช้พื้นที่ในตารางเหมือนกับ **Open Addressing** แต่ใช้หลักการของการเชื่อมโยง และ**พื้นที่ที่ใช้เป็นส่วนที่เรียกว่าพื้นที่ส่วนเกิน คือพื้นที่ที่กันไว้สำหรับคีย์ที่ชน เราอาจใช้การเชื่อมโยงโดยใช้ singly link list** ได้ดังรูป



แบบฝึกหัด

- 1. กำหนดให้มีการเก็บข้อมูลในแถวลำดับต่อไปนี้

  12 99 58 32 10 8 19 70

  จงหาจำนวนครั้งในการค้นหาข้อมูลที่มีค่า 19 ในแถวลำดับด้วยวิธีเรียงลำดับ(Sequential Search)

- 2. กำหนดให้มีการเก็บข้อมูลในแถวลำดับต่อไปนี้

  1 5 7 9 10 45 56 88 99

  จงแสดงขั้นตอนการค้นหา 9 ด้วยวิธีการค้นหาแบบทวิภาค(Binary Search)