

บทที่ 10
การค้นหาข้อมูล
(Searching)

สุนทรี คุ่มไพโรจน์

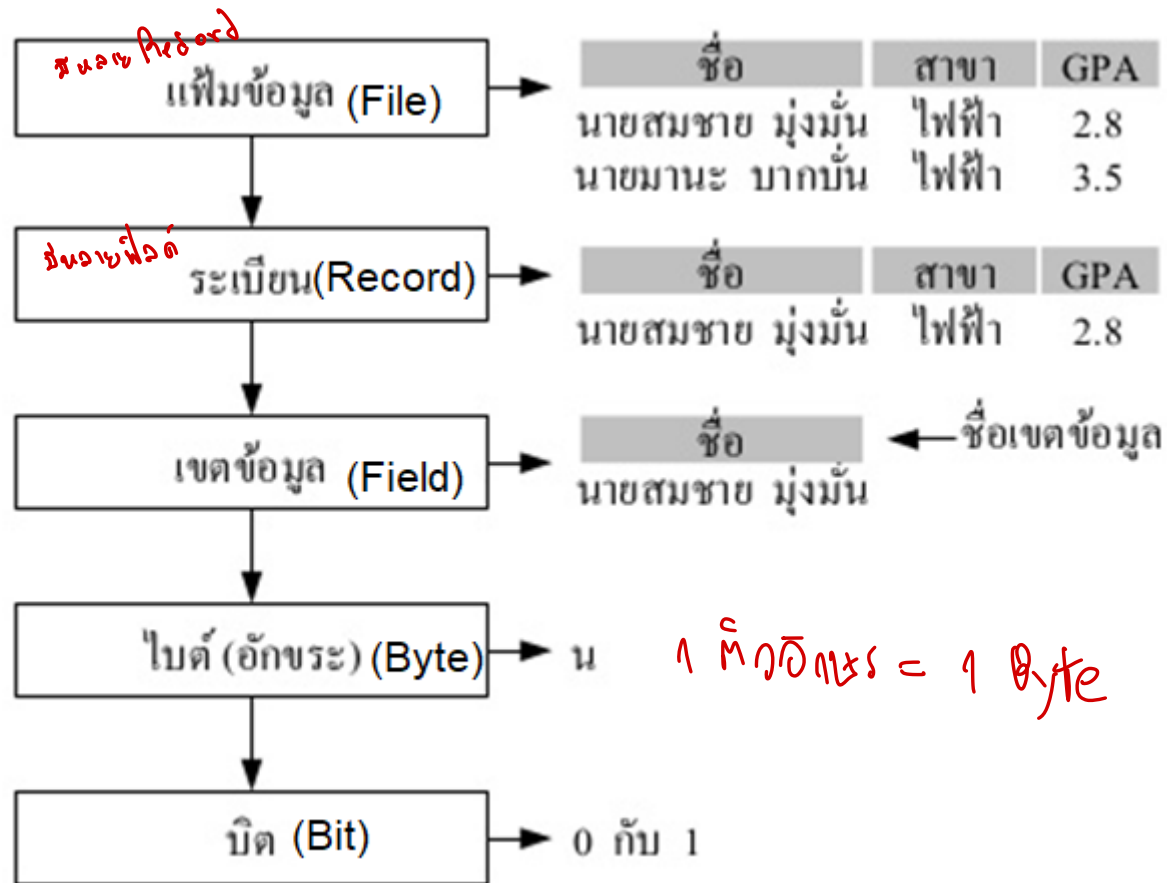
Searching

- การค้นหาข้อมูล จะให้ผลลัพธ์บอกได้ว่าค้นหาเจอหรือไม่
- กรณีที่ค้นหาเจอจะต้องบอกได้ว่าค้นหาเจอที่ตำแหน่งใด

รูปแบบการ search

- Sequential search หรือ คือ linear search
- Binary search แบ่งครึ่ง; ค้าง
- Search โดยใช้ Hashing function ใช้ func. แบ่งข้อมูล/หาตำแหน่ง

โครงสร้างของแฟ้มข้อมูล

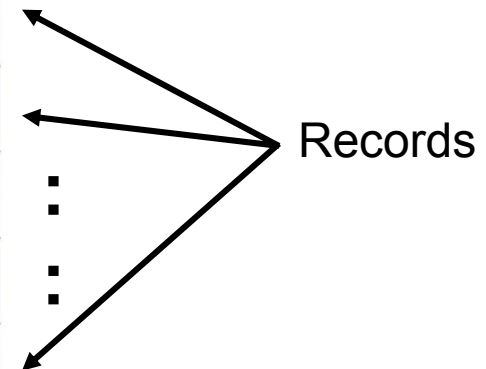


Primary keys

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



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



Sequential Search

- เป็นการค้นหาในลักษณะ**เชิงเส้น**
- เป็นวิธีที่ง่ายที่สุด แต่ก็ก็เป็นวิธีที่ด้อยประสิทธิภาพกว่าวิธีอื่น
- เหมาะสำหรับการประมวลผลข้อมูลกับบางประเภท เช่น
งานกลุ่ม (Batch Processing) งานสำรองข้อมูล (Backup)
งานปรับปรุงแก้ไขข้อมูลให้ทันสมัย (Update)
- **เปรียบเทียบ**คีย์ (key) ที่ต้องการค้นหา กับข้อมูลที่ละตำแหน่ง
ไปจนกว่าจะพบ หรือจนกว่าจะหมดข้อมูล

ตัวอย่าง

9 7 15 60 32 40 53 84 21 96

- การค้นหาข้อมูลจะต้องผ่านการตรวจสอบข้อมูลที่อยู่ข้างหน้า
ระเบียบข้อมูลที่ต้องการทุกตัวจนกว่าจะพบข้อมูล
- โดยมีจำนวนครั้งของการเปรียบเทียบ ดังนี้

- **Best Case**

น้อยที่สุด = 1 ครั้ง (ข้อมูลที่ต้องการอยู่เป็นระเบียบแรก)

- **Worst Case**

มากที่สุด = จำนวนข้อมูลทั้งหมดในแฟ้มข้อมูล (ระเบียบสุดท้าย)

- **Average Case**

โดยเฉลี่ย = $1/2$ ของจำนวนข้อมูลทั้งหมดในแฟ้มข้อมูล

Sequential Search

- ถ้าข้อมูลในแฟ้มข้อมูลเรียงตามลำดับ(Sort)

ขั้นตอนการค้นหาคำทำได้เร็วขึ้น คือ

ถ้าพบว่า **ค่าคีย์(Key)** ของระเบียบข้อมูลมีค่า **> ค่าของดัชนี**
แสดงว่าระเบียบข้อมูลที่ต้องการนั้น ไม่อยู่ในแฟ้มข้อมูลนี้
และไม่จำเป็นต้องทำการค้นหาอีกต่อไป

- ถ้าแฟ้มข้อมูลไม่ได้จัดเรียงลำดับ

จะต้องค้นหาข้อมูลในแฟ้มข้อมูลจนครบทุกตัว

จึงจะสรุปได้ว่าไม่มีระเบียบข้อมูลที่ต้องการนั้นอยู่ภายในแฟ้มข้อมูล

- พิจารณาประสิทธิภาพของการค้นหาได้เป็น $O(n)$ กำหนดข้อมูลต้น 0

2 3 4 6 8

อัลกอริทึม (search)

```
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**

จำนวนครั้งค้นหา
ทั้งหมด **5** ครั้ง
พบข้อมูล

1	1	← F
2	5	← F
3	7	← F
4	8	← F
5	11	← T
6	13	
7	14	
8	18	
9	20	

จากข้อมูล ถ้าต้องการค้นหา เลข **11** แบบ binary search
จำนวนครั้งในการค้นหาจะเป็นเท่าไร

ค้นหาข้อมูลที่เรียงลำดับแล้ว ค้นหา **6**

จำนวนครั้งค้นหา
ทั้งหมด **3** ครั้ง
ไม่พบข้อมูล

1	1	← F
2	5	← F
3	7	← F
4	8	
5	11	
6	13	
7	14	
8	18	
9	20	

เลข 6 ถ้าเรียงลำดับแล้ว
ต้องอยู่ระหว่าง
ตัวที่ 3 และตัวที่ 4
ดังนั้น จะหาตัวที่ 4 ได้เลย

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

ค้นหาข้อมูลที่ไม่เรียงลำดับ แล้วค้นหา **15**

จำนวนครั้งค้นหา
ทั้งหมด **5** ครั้ง
พบข้อมูล

1	11	← F
2	5	← F
3	18	← F
4	20	← F
5	15	← T
6	8	
7	14	
8	6	
9	3	

ค้นหาข้อมูลที่ ไม่เรียงลำดับ แล้วค้นหา **9**

จำนวนครั้งค้นหา
ทั้งหมด **9** ครั้ง
ไม่พบข้อมูล

1	11	← F
2	5	← F
3	18	← F
4	20	← F
5	15	← F
6	8	← F
7	14	← F
8	6	← F
9	3	← F

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 *ความน่าจะเป็นของแต่ละตัว*
- แต่ละตำแหน่งจะมีค่าความน่าจะเป็น $p = 1/n$ *คือ $\frac{1}{n}$*

$$\begin{aligned}C(n) &= 1*1/n + 2*1/n + \dots + n*1/n \\&= (1+2+\dots+n) * 1/n \\&= n(n+1)/2 * 1/n = (n+1)/2\end{aligned}$$

Rate of growth : Big O notation

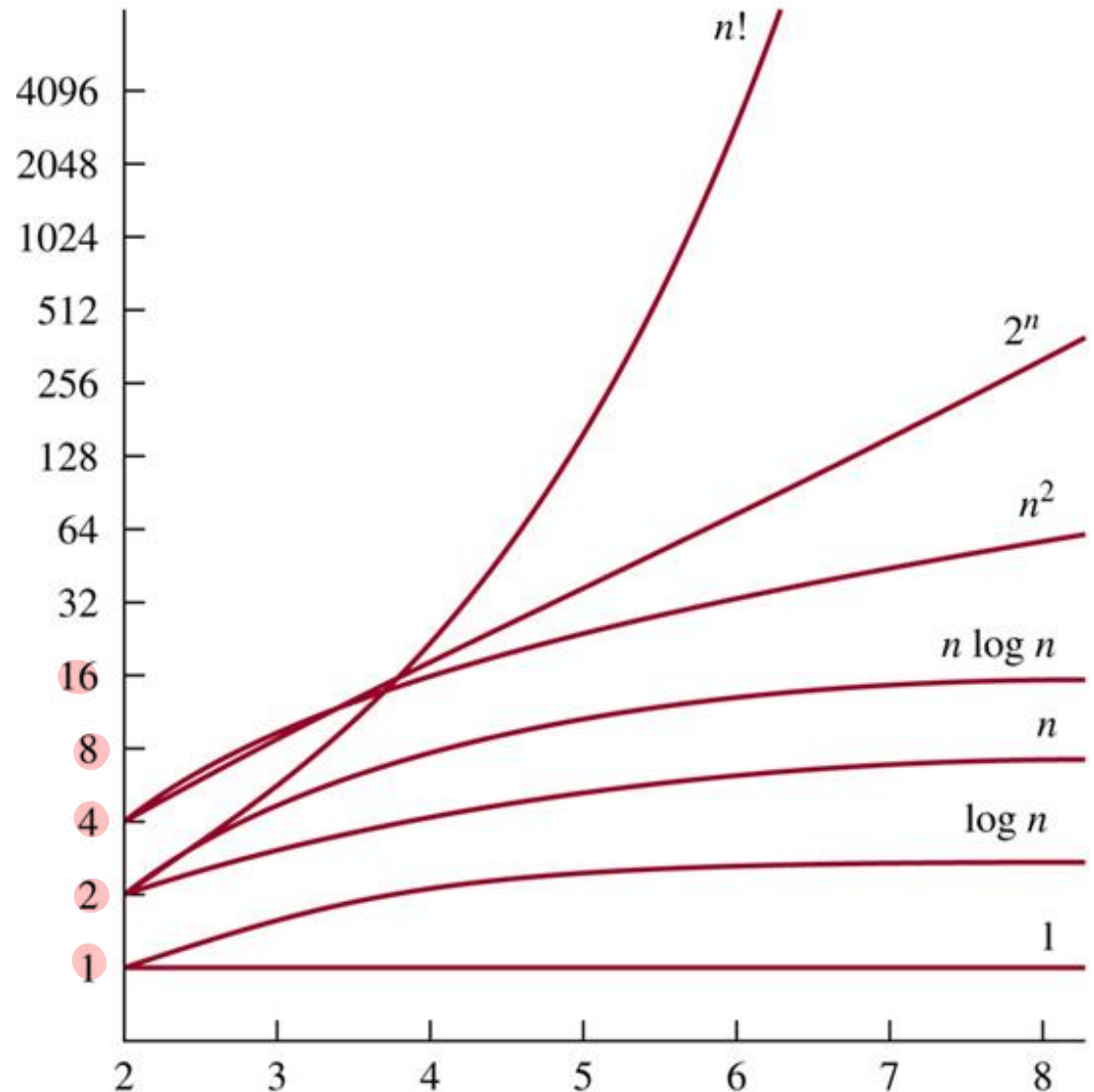
อัตราการเพิ่มของฟังก์ชันมาตรฐาน(ค่าที่เพิ่มขึ้นเมื่อ n เพิ่มขึ้น)

$n \backslash g(n)$	$\log n$	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

© The McGraw-Hill Companies, Inc. all rights reserved.

၁
၂
၃
၄
၅
၆
၇
၈
၉
၁၀



Big Oh Notation

time of Algorithm func คอมพิวเตอร์
สมมติ $f(n)$ และ $g(n)$ เป็นฟังก์ชันที่ให้ค่าจำนวนบวก
โดยที่ $f(n)$ ถูกจำกัดด้วย $g(n)$ สำหรับทุกค่าของ n และ $n > n_0$ *ค่าคงที่*

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

เขียนในรูป $f(n) = O(g(n))$ *f(n) ไม่เกิน 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
- การค้นหาตัวที่ต้องการจะต้องเลือกหาจากข้อมูลที่ละส่วน
โดยใช้วิธีเดิมคือแบ่งเป็นสองส่วนย่อย
หากพบในส่วนที่หนึ่ง ก็ไม่ต้องเสียเวลาหาในอีกส่วนหนึ่งอีกต่อไป
- เพื่อให้เกิดประโยชน์สูงสุดในการ **search**
Binary search จะมีการ **sort** ก่อน
เพื่อตัดข้อมูลส่วนที่ไม่ใช่ทิ้งไป

min

1 2 3 4 5 6 7 8 9 10

$$(L + R) \text{ div } 2$$

$$(1 + 10) \text{ div } 2$$

Binary Search

- 1. เปรียบเทียบค่าข้อมูลที่ต้องการค้นหา กับ Primary Key ของข้อมูล
 ระยะที่ 1
 - ถ้าใช่ ระยะที่ต้องการให้ไปทำขั้นตอนที่ 7
 - ถ้าไม่ ให้ทำขั้นตอนต่อไป
- 2. เปรียบเทียบค่าข้อมูลที่ต้องการค้นหา กับ Primary Key ของข้อมูล
 ระยะที่ n
 - ถ้าใช่ ระยะที่ต้องการให้ไปทำขั้นตอนที่ 7
 - ถ้าไม่ ให้ทำขั้นตอนต่อไป

Binary Search (ต่อ)

- 3. กำหนดจุดเริ่มต้นและจุดสุดท้ายของการค้นหา
- 4. เปรียบเทียบค่าข้อมูลที่ต้องการค้นหากับ Primary Key ของข้อมูล
ณ ตำแหน่งกึ่งกลางของชุดระเบียบข้อมูล
 - ถ้าใช่ ระเบียบที่ต้องการให้ข้ามไปทำขั้นตอนที่ 7
 - ถ้าไม่ ให้ทำขั้นตอนต่อไป
- 5. เปรียบเทียบข้อมูลที่ต้องการค้นหากับค่าของ Primary Key ของข้อมูล
ณ ตำแหน่งกึ่งกลางของชุดระเบียบข้อมูล
 - ถ้าค่าข้อมูลที่ต้องการค้นหามีค่ามากกว่า ให้เปลี่ยนตำแหน่งจุดเริ่มต้นเป็นตำแหน่งที่อยู่ทางขวาของตำแหน่งกึ่งกลางนั้น
 - ถ้าค่าข้อมูลที่ต้องการค้นหามีค่าน้อยกว่า ให้เปลี่ยนตำแหน่งจุดสุดท้าย เป็นตำแหน่งที่อยู่ก่อนหน้าตำแหน่งกึ่งกลางนั้น

Binary Search

- 6. ตรวจสอบจำนวนครั้งของการเปรียบเทียบเกิน $n/3$ ครั้งหรือไม่
 - ถ้าใช่ ให้แสดงข้อความแจ้งการค้นหาล้มเหลว และไปทำในขั้นตอนที่ 8
 - ถ้าไม่ ให้ย้อนกลับไปทำตั้งแต่ขั้นตอนที่ 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

- สมมติต้องการค้นหา 15 จากข้อมูลชุดนี้ (ก็คือ 15)

1. การหาตำแหน่งกึ่งกลางจะคำนวณโดยนำเลขตำแหน่งแรกของชุดข้อมูล
บวกกับเลขตำแหน่งสุดท้ายของชุด หารด้วยสอง

- แต่เนื่องจากการหารสอง อาจทำให้เกิดเลขทศนิยมได้ จึงใช้ฟังก์ชัน div
ช่วยในการหารแทน ดังนั้นจะได้ว่า $\text{Mid} = (\text{L} + \text{R}) \text{ div } 2$
- จากตัวอย่าง ตำแหน่งกึ่งกลางได้แก่ $\text{Mid} = (1 + 10) \text{ div } 2 = 5$

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. นำคีย์ที่ต้องการค้นหา(คีย์ 15) เปรียบเทียบกับข้อมูลในตำแหน่ง 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	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]
ข้อมูล	7	9	15	21	32	40	53	60	84	96
รอบที่ 1	L				Mid					R
รอบที่ 2	L		Mid		R					

- ตำแหน่ง Mid ของรอบที่ 2 คือ ตำแหน่งที่ 3 $\{(1+5)/2\}$
- นำ คีย์ คือ 15 มาเปรียบเทียบกับข้อมูลที่ตำแหน่ง Mid (ตำแหน่งที่ 3)
- สามารถตอบได้ว่าพบข้อมูลที่ต้องการที่ตำแหน่งที่ 3
- กรณีที่ขยับตำแหน่ง L R Mid จนกระทั่งขยับไม่ได้ anymore หมายถึงไม่พบข้อมูลที่ต้องการ

ประสิทธิภาพของการค้นหาข้อมูลด้วย Binary Search

- พิจารณากระบวนการทำงานแบบ Binary Search จะพบว่าเมื่อมีการเปรียบเทียบแต่ละครั้งจะมีการตัดข้อมูลในตารางออกไปได้ทีละครึ่งหนึ่งเสมอ ดังนั้นถ้าเริ่มต้นมีจำนวนข้อมูล n ตัว
- จำนวนข้อมูลที่นำมาเพื่อค้นหาค่าที่ต้องการหลังการเปรียบเทียบแต่ละครั้งจะเป็นดังนี้

- ครั้งที่ 1 = $n = \frac{n}{2^0} = n$
- ครั้งที่ 2 = $n/2 = \frac{n}{2^1}$
- ครั้งที่ 3 = $n/4 = \frac{n}{2^2}$
- ครั้งที่ 4 = $n/8 = \frac{n}{2^3}$

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

$$n/2^k = 1$$

k จำนวนครั้ง

หรือ

$$n = 2^k$$

จะได้

$$k = \log_2 n = O(\log n)$$

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

สรุปได้ว่าประสิทธิภาพของการค้นหาแบบ Binary คือ $O(\log n)$

การค้นหาด้วยวิธีแฮชซิง (Hashing Search)

- การค้นหาด้วยวิธีการแฮชซิง หรือแฮชซิงเซอร์ช (Hashing Searching) นี้ เป็นวิธีการแปลงคีย์ให้เป็นแอดเดรส (address)
- คำว่าคีย์(Key)ในที่นี้หมายถึง ข้อมูลที่ต้องการการค้นหา หรือข้อมูลในชุดข้อมูลที่ถูกค้นหา
- การแปลงคีย์ให้เป็นแอดเดรส คือการแปลงข้อมูลให้ไปอยู่ในตารางที่ เตรียมไว้ ซึ่งตารางนั้นเรียกว่า ตารางแฮช (Hash Table) โดยการแปลงนี้ ใช้ฟังก์ชันที่เรียกว่า แฮชซิงฟังก์ชัน (Hashing Function) ถ้าให้ H เป็นแฮชซิงฟังก์ชัน
K เป็นคีย์ และ
A เป็นแอดเดรสที่ได้หลังจากแปลงคีย์ไปแล้ว
อาจเขียนสัญลักษณ์การแปลงดังนี้

$$H(K) \rightarrow A$$

1. Mod
2. fold
3. ทวน

1. သိသိကွေ့ကွေ့ သော hash table

Address
ନିମ୍ନ Index

$$5 \bmod 3 = 2$$

1. Mod

- ฟังก์ชันแฮชที่นิยมกันมากก็คือ mod เขียนเป็นสัญลักษณ์ได้ดังนี้

$$H(K) = K \text{ MOD } M$$

- เมื่อกำหนดให้
 - **k** คือ ค่าข้อมูลที่ต้องการจัดเก็บ
 - **M** คือ ขนาดของเนื้อที่ที่กำหนดไว้
- แฮชซึ่งฟังก์ชันถูกใช้งานสองอย่าง คือ
 1. กำหนดแอดเดรสให้ชุดข้อมูล
 2. บอกแอดเดรสที่ต้องไปค้นหาเมื่อระบุข้อมูลที่ต้องการค้นซึ่งงานทั้งสองอย่างนี้ต่างกันก็คือ การแปลงคีย์ให้เป็นแอดเดรส

ตัวอย่าง 9 7 15 60 32 40 53 84 21 96

- 1. แปลงคีย์เป็นท็อยู่
- โดยใช้ฟังก์ชัน $\text{mod } 11$ เพื่อจัดเก็บข้อมูล
- $H(K) = K \text{ MOD } M$
- $H(9) = 9 \text{ mod } 11 = 9$
- $H(7) = 7 \text{ mod } 11 = 7$
- $H(15) = 15 \text{ mod } 11 = 4$

60777:

327.11:

0	53
1	84
2	21
3	96
4	15
5	60
6	
7	7
8	40
9	9
10	32

ตัวอย่าง

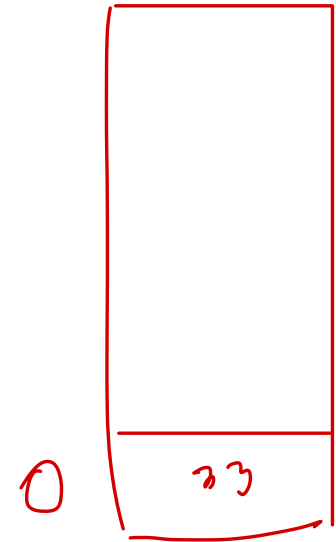
9 7 15 60 32 40 53 84 21 96

- 2. การค้นหาข้อมูล โดยการนำคีย์ที่ต้องการค้นหาผ่านฟังก์ชันแฮชเดียวกันกับการจัดเก็บ เช่น จากตัวอย่าง ต้องการค้นหาคีย์ 33

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

- ค้นหาที่ตำแหน่ง 0 ไม่พบข้อมูล 33
แสดงว่าไม่มีข้อมูล 33 อยู่ในชุดข้อมูลนี้

- จากกระบวนการในการค้นหาข้อมูลดังกล่าว ประสิทธิภาพของการค้นหาคือ $O(1)$
เนื่องจากไม่ว่าจะมีข้อมูลมากหรือน้อยเท่าใด การเปรียบเทียบก็จะกระทำเพียง 1 ครั้งเท่านั้น นับว่าเป็นวิธีที่มีประสิทธิภาพสูง



ตัวอย่างของวิธีการหาร **Mod**

0	1	2	3	4	5	6	7	8	9
100	251		123						

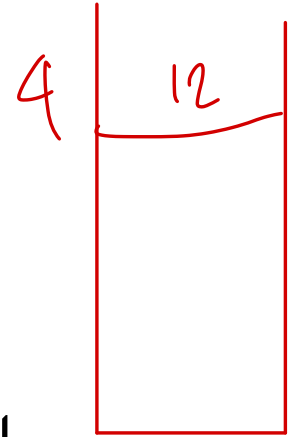
$$H(100) = 100 \bmod 10 = 0$$

$$H(251) = 251 \bmod 10 = 1$$

$$H(123) = 123 \bmod 10 = 3$$

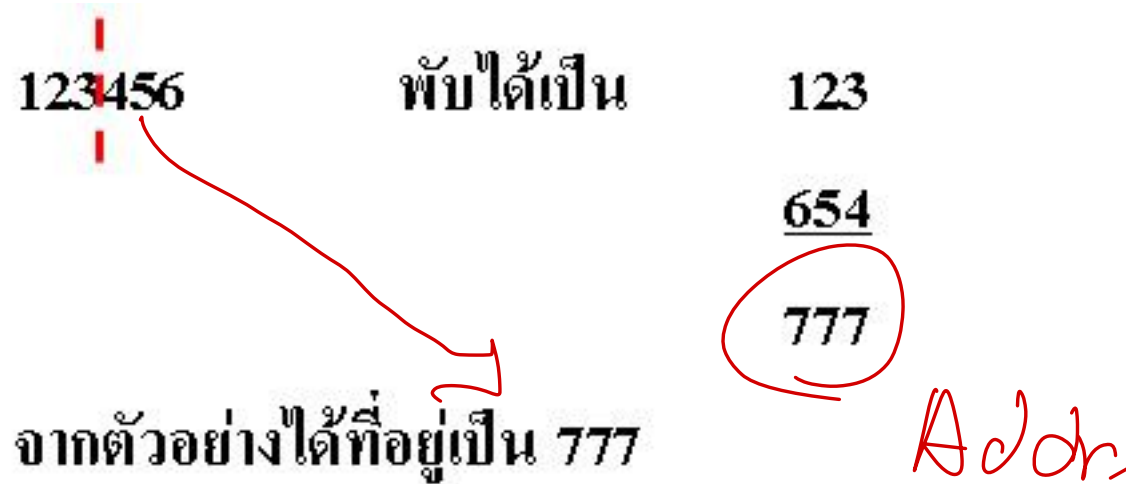
2.Mid-Square

- คือการนำคิย์มายกกำลังสอง
แล้วเลือกเฉพาะค่ากลางของข้อมูล
จำนวนหลักขึ้นอยู่กับการต้องการในการใช้งาน
- ตัวอย่างเช่น
- คิย์คือ 12 ผ่านฟังก์ชัน Mid-Square คือ $12^2 = 144$ เลือกเฉพาะ
ตำแหน่งกลางได้ค่าที่อยู่เป็น 4



3.Folding วิธีการพับตัวเลข

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

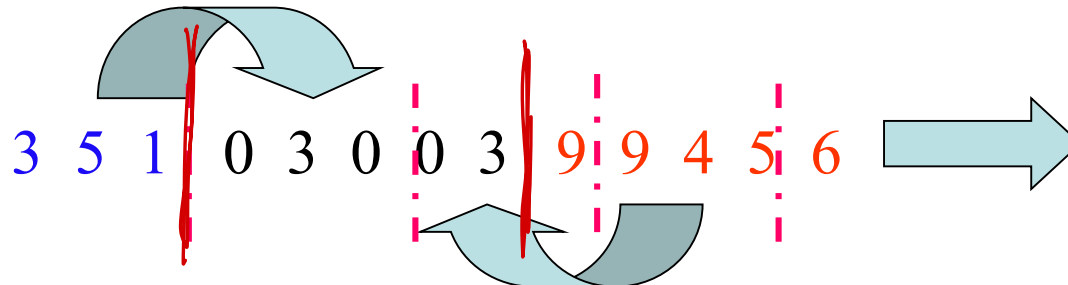


3. วิธีการพับตัวเลข (Folding Method)

- 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 เกิดจากการที่นำคีย์มาผ่านฟังก์ชันแล้วได้ที่อยู่เป็นตำแหน่งเดียวกัน ตัวอย่างเช่น
 - ต้องการจัดเก็บ 22 และ 33 ในตารางแฮช
 - $H(22) = 22 \bmod 11 = 0$
 - $H(33) = 33 \bmod 11 = 0$
 - วิธีการในการแก้ปัญหการชนกันแบ่งเป็น 2 วิธีใหญ่ ๆ ดังนี้



1. Open Addressing

- เป็นวิธีที่ใช้ตำแหน่งที่เหลือในตารางในการเก็บข้อมูลที่ชน มีวิธีย่อย ๆ คือ
- **1.1 Linear Probing** คือการมองหาช่องว่างถัดไปในตาราง แล้วจัดเก็บข้อมูลที่ชนในช่องว่างแรกที่พบ หากใช้วิธีนี้ในการจัดเก็บ ในขั้นตอนของการค้นหา ก็จะต้องดำเนินการด้วยวิธีเดียวกัน
- แต่ Linear Probing อาจทำให้เกิดปัญหาตามมาคือ การแทนที่ที่ไม่ถูกต้อง นั่นคือ
- เมื่อพบช่องว่างสำหรับคีย์ที่ชนกันแล้ว หากหลังจากนี้มีคีย์ที่เข้ามาใหม่ ที่เมื่อผ่านฟังก์ชันแล้วได้ที่อยู่ที่แท้จริงเป็นตำแหน่งเดียวกันกับตัวที่ชน ก็จะทำให้เกิดการชนกันเช่นนี้เรื่อยไป

ตัวอย่าง Linear

0	1	2	3	4	5	6	7	8	9
100	251	250	123	543					

$$H(100) = 100 \bmod 10 = 0$$

$$H(251) = 251 \bmod 10 = 1$$

$$H(123) = 123 \bmod 10 = 3$$

$$H(250) = 250 \bmod 10 = 0$$

$$H(543) = 543 \bmod 10 = 3$$

1.2 Double Hashing

- เป็นความพยายามแก้ปัญหาของ **linear probing** ที่เสาะหาในตำแหน่งที่ซ้ำ ๆ อันเป็นสาเหตุของการชนครั้งแล้วครั้งเล่า
- โดยอาศัย **randomness** ของ **hash function** ในการหาตำแหน่งใหม่ หลังจากเกิดการชนกันขึ้น **collision resolution function** ก็สร้างขึ้นในรูปแบบตรงไปตรงมา
- ที่นิยมกันมาก คือ $F(i) = i * h_2(x)$ ซึ่ง $h_2(x)$ เป็นฟังก์ชันใหม่ที่ไม่ซ้ำกับ **hash function** เดิม

Double Hashing

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

Create a second hash function $h_2(x)$ to be the increment (generalize linear probing).

Example:

$M=11$, $h(x) = x \bmod 11$, $h_2(x) = x \bmod 7 + 1$

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

14	$\bmod 11 = 3$
17	$\bmod 11 = 6$
25	$\bmod 11 = 3$
37	$\bmod 11 = 4$
34	$\bmod 11 = 1$
16	$\bmod 11 = 5$
26	$\bmod 11 = 4$

จำนวนจริง

Probes

1

1

2

1

1

1

2

$25 \bmod 7 + 1 = 5$

$26 \bmod 7 + 1 = 6$

9 probes or $9/7 \sim 1$ probe per key

จำนวน

/ บ่อยครั้ง มีพื้นที่

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

Find 47:

$h(47) = 47 \bmod 11 = 3$ not there

$h_2(47) = 47 \bmod 7 + 1 = 6$ (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)

แต่ละตำแหน่งหลักจึงมี list ของตนเองที่เก็บข้อมูลที่ซ้ำกันอันเป็นผลพวงมาจากการ mapped หรือ hashed เข้าสู่ตำแหน่งหลักเดียวกัน

ตัวอย่างเช่น ต้องการจัดเก็บ 22 และ 33 ในตารางแฮช

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

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

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



	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		

ตัวอย่างเช่น

ต้องการจัดเก็บ 22 และ 33 ในตารางแฮช

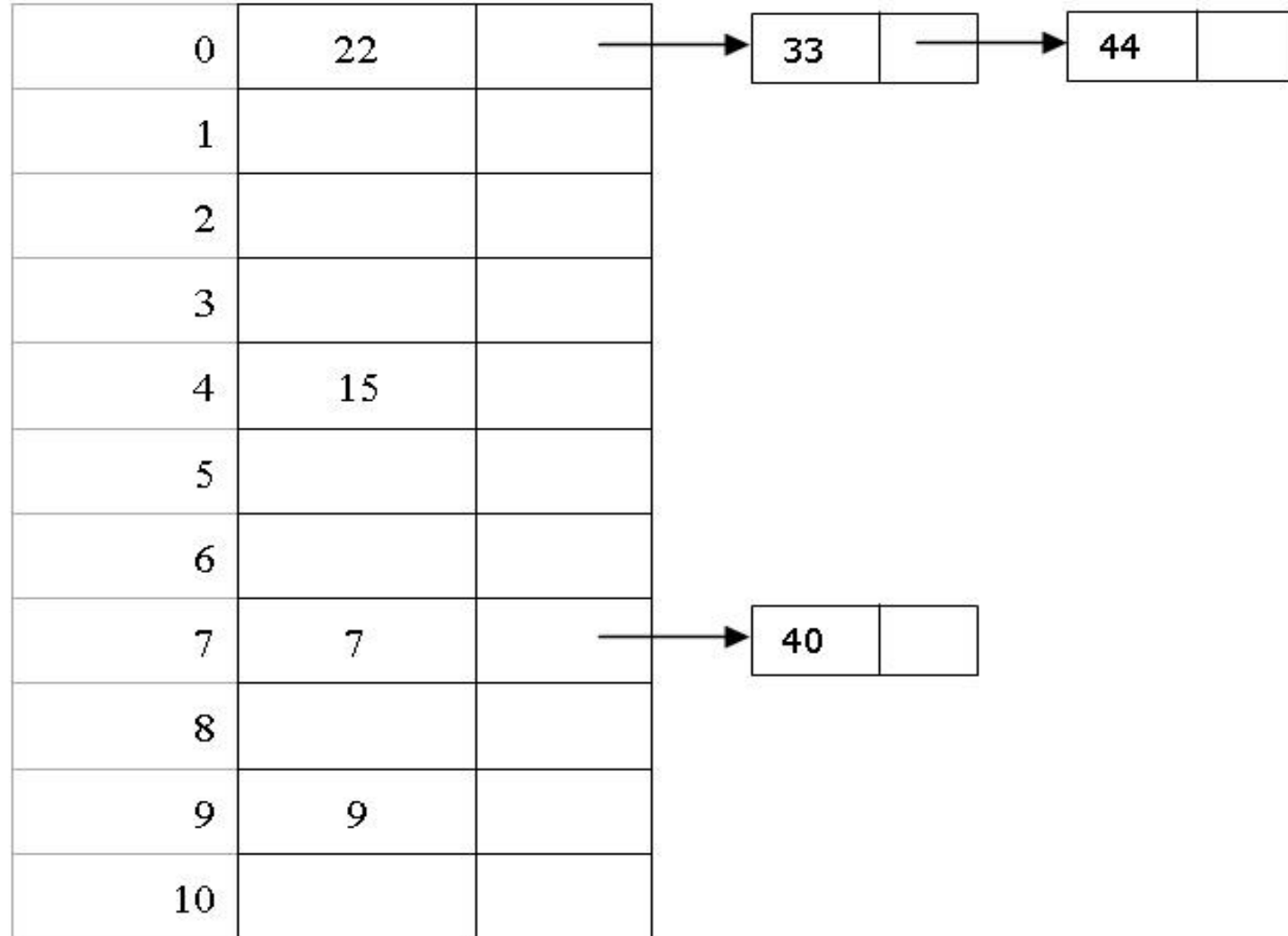
$$H(22) = 22 \bmod 11 = 0$$

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

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

Overflow

จากตัวอย่างเป็นการใช้พื้นที่ในตารางเหมือนกับ **Open Addressing** แต่ใช้หลักการของการเชื่อมโยง และพื้นที่ที่ใช้เป็นส่วนที่เรียกว่าพื้นที่ส่วนเกิน คือพื้นที่ที่กั้นไว้สำหรับคีย์ที่ชน 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)

12 99 58 32 10 8 19 70

$$H(k) = k \bmod m$$

$$H(12) = 12 \bmod 11 = 1$$

$$H(99) = 99 \bmod 11 = 0$$

$$H(58) = 58 \bmod 11 = 3$$

$$H(32) = 32 \bmod 11 = 10$$

$$H(10) = 10 \bmod 11 = 10$$

$$H(8) = 8 \bmod 11 = 8$$

$$H(19) = 19 \bmod 11 = 8$$

$$H(70) = 70 \bmod 11 = 4$$

10	32
9	19
8	8
7	
6	
5	
4	70
3	58
2	10
1	12
0	99

[A (B ())]

Pay =

sum
if not Empty

Pop number no ()
sum = Empty

Pay += sum * 0.05