

# บทที่ 10 searching and hashing

## บทเรียนย่อย

---

10.1 Searching and Hashing Introduction

10.2 Sequential Search Algorithm

10.3 Binary Search Algorithm

10.4 Hashing Algorithm

# วัตถุประสงค์

- นิสิตมีความรู้ และความเข้าใจเกี่ยวกับแนวคิดในการค้นหาข้อมูลรูปแบบต่าง ๆ
- นิสิตสามารถเขียนโปรแกรมเพื่อดำเนินการตามแนวคิดในการค้นหาข้อมูลรูปแบบต่าง ๆ
- นิสิตสามารถนำแนวคิดในการค้นหาข้อมูลมาประยุกต์ใช้งานในการพัฒนาโปรแกรม

# บทที่ 10 searching and hashing

## บทเรียนย่อย

---

10.1 Searching and Hashing Introduction

10.2 Sequential Search Algorithm

10.3 Binary Search Algorithm

10.4 Hashing Algorithm

## 10.1 Searching and Hashing Introduction

**การค้นหาข้อมูล** เป็นการค้นหาค่าใด ๆ ที่ต้องการจากโครงสร้างข้อมูลที่ได้จัดเก็บข้อมูลนั้นไว้ ซึ่งจะมีประสิทธิภาพมากน้อยเพียงใด ขึ้นอยู่กับขั้นตอนวิธีการค้นหาข้อมูลเป็นสำคัญ โดยการค้นหาข้อมูลมีหลายวิธี แต่ละวิธีก็มีข้อดีข้อเสียด้วยกันทั้งสิ้น และยังเหมาะสมเฉพาะงาน โดยวิธีการค้นหาข้อมูลหลัก ๆ มี 3 วิธี ดังนี้

- Sequential Search (การค้นหาแบบลำดับ)
- Binary Search (การค้นหาแบบทวิภาค)
- Hashing Search (การค้นหาแบบแฮช)

# บทที่ 10 searching and hashing

## บทเรียนย่อย

---

10.1 Searching and Hashing Introduction

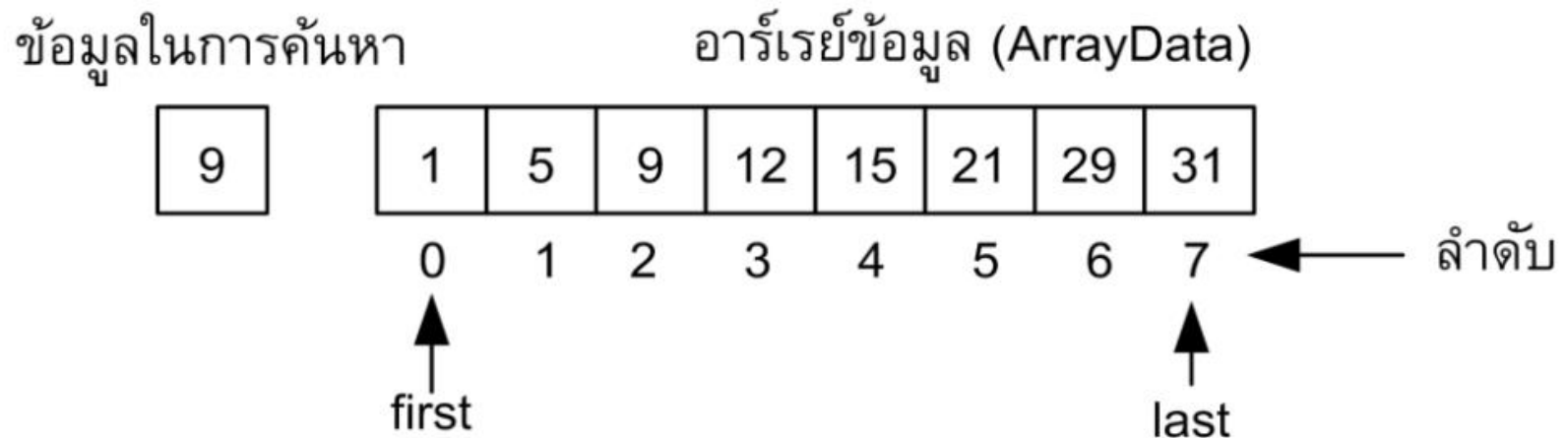
10.2 Sequential Search Algorithm

10.3 Binary Search Algorithm

10.4 Hashing Algorithm

## 10.2 Sequential Search Algorithm








**Sequential Search (การค้นหาแบบลำดับ)** เป็นวิธีการค้นหาข้อมูล ที่ง่ายที่สุด และสามารถกระทำบนโครงสร้างข้อมูลที่ถูกจัดเรียงหรือไม่ก็ได้ โดยหลักการของการค้นหาแบบลำดับ คือ การเอาค่าข้อมูลที่ต้องการหา ตำแหน่งในโครงสร้าง นำไปไล่เทียบกับข้อมูลในโครงสร้างทีละตัวตามลำดับ



# ตัวอย่างการทำงานของ Sequential Search

---

ข้อมูลในการค้นหา หรือ Key = 54 (มีข้อมูล)

|   |   |   |   |   |   |   |    |
|---|---|---|---|---|---|---|----|
| 3   | 7   | 12  | 25  | 32  | 48  | 54  | 78 |
| 0   | 1   | 2   | 3   | 4   | 5   | 6   | 7  |
|  |  |  |  |  |  |  |    |
| $i = 0$   | $i = i + 1$   | $i = i + 1$   | $i = i + 1$   | $i = i + 1$   | $i = i + 1$   | $i = i + 1$   |    |
| not found   | not found   | not found   | not found   | not found   | not found   | found   |    |
|   |   |   |   |   |   | $i = 6$   |    |

## ตัวอย่างการทำงานของ Sequential Search [2]

---

ข้อมูลในการค้นหา หรือ Key = 29 (ไม่มีข้อมูล)

|              |              |              |              |              |              |              |              |
|--------------|--------------|--------------|--------------|--------------|--------------|--------------|--------------|
| 3            | 7            | 12           | 25           | 32           | 48           | 54           | 78           |
| 0            | 1            | 2            | 3            | 4            | 5            | 6            | 7            |
| ↑            | ↑            | ↑            | ↑            | ↑            | ↑            | ↑            | ↑            |
| $i = 0$      | $i = i + 1$  | $i = i + 1$  | $i = i + 1$  | $i = i + 1$  | $i = i + 1$  | $i = i + 1$  | $i = i + 1$  |
| not<br>found | not<br>found | not<br>found | not<br>found | not<br>found | not<br>found | not<br>found | not<br>found |

**Not Found !!**



# การวิเคราะห์การทำงานของ Sequential Search

---

- ประสิทธิภาพในการค้นหาในกรณีที่ดีที่สุด (best-case) คือ เจอข้อมูลในตำแหน่งแรกของอาร์เรย์ ดังนั้น best-case ในการค้นหาคือ  $O(1)$
- กรณีของ worst-case คือการเจอข้อมูลในตำแหน่งสุดท้ายของอาร์เรย์ ดังนั้น worst-case ในการค้นหาคือ  $O(n)$
- กรณีของ average-case คือการที่คาดว่าจะหาข้อมูลเจอในตำแหน่งตรงกลางของอาร์เรย์ หรือ ตำแหน่งที่  $n/2$  ของอาร์เรย์ ดังนั้น average-case ในการค้นหาคือ  $O(n)$

## การวิเคราะห์การทำงานของ Sequential Search [2]

---

- ขั้นตอนวิธีข้างต้นของการค้นหาแบบเรียงลำดับจะเหมาะสมกับการค้นหาค่าในชุดข้อมูลที่ไม่ได้เรียง
- ถ้าข้อมูลมีการเรียงเรียบร้อยแล้ว จะมีข้อเสียบางประการ คือ กรณีที่ค้นหาไม่พบ เมื่อค้นหาค่าข้อมูลที่ต้องการในชุดข้อมูลจนถึงตัวที่มีค่ามากกว่าแล้ว การค้นหาจะยังไม่ยุติการค้นหา ยังคงวนรอบเพื่อเปรียบเทียบข้อมูลจนถึงตัวสุดท้ายในชุดข้อมูล ซึ่งทำให้เสียเวลา

# การปรับปรุงประสิทธิภาพของ Sequential Search

---

ปรับปรุงการค้นหาข้อมูลแบบลำดับด้วยเทคนิคการเรียงลำดับข้อมูล โดยเรียงลำดับข้อมูลจากน้อยไปหามาก

และเทคนิคเซล์ฟรีออร์เดอร์ริง (Self Reordering) เพื่อปรับปรุงขั้นตอนวิธีการค้นหาโดยการค้นหาข้อมูลจะหยุดทำการค้นหาเมื่อพบว่าคีย์ที่ใช้ในการค้นหามีค่าน้อยกว่าข้อมูลในชุดข้อมูล

# ตัวอย่างการทำงานของ Sequential Search ที่ทำการปรับปรุง

---

ข้อมูลในการค้นหา หรือ Key = 29 (ไม่มีข้อมูล)

|   |   |    |    |    |    |    |    |
|---|---|----|----|----|----|----|----|
| 3 | 7 | 12 | 25 | 32 | 48 | 54 | 78 |
|---|---|----|----|----|----|----|----|

0

1

2

3

4

5

6

7



$i = 0$

$i = i + 1$

$i = i + 1$

$i = i + 1$

$i = i + 1$

not  
found

not  
found

not  
found

not  
found

$K[i] > \text{key}$   
Stop..!!

Not Fund !!

# บทที่ 10 searching and hashing

## บทเรียนย่อย

---

10.1 Searching and Hashing Introduction

10.2 Sequential Search Algorithm

10.3 Binary Search Algorithm

10.4 Hashing Algorithm

## 10.3 Binary Search Algorithm

**Binary Search (การค้นหาแบบทวิภาค)** เป็นวิธีการค้นหาข้อมูลที่มีประสิทธิภาพมากกว่าการค้นหาแบบลำดับ ซึ่งหลักการการค้นหา คือ การนำขนาดของอาร์เรย์ทั้งหมดมาแบ่งครึ่ง และในการแบ่งครึ่งทุกครั้งจะทำการเปรียบเทียบเพื่อตรวจสอบว่าข้อมูลที่ต้องการค้นหาจะอยู่ในช่วงใดของอาร์เรย์ โดยจำนวนครั้งที่เปรียบเทียบจะเท่ากับจำนวนการแบ่งครึ่งอาร์เรย์ ซึ่งมีลำดับในการค้นหาข้อมูลดังนี้

1. ตรวจสอบครึ่งหนึ่งของอาร์เรย์ขนาด  $n$
2. ตรวจสอบครึ่งหนึ่งของอาร์เรย์ขนาด  $n/2$
3. ตรวจสอบครึ่งหนึ่งของอาร์เรย์ขนาด  $n/2^2$  และทำจนกระทั่งเจอข้อมูลหรือไม่ตรงตามเงื่อนไขของการค้นหาแบบไบนารี

# ตัวอย่างการค้นหาแบบ Binary Search

---

เริ่มต้นกำหนดให้ชุดข้อมูลเป็นดังนี้

ข้อมูลในการค้นหา

9

อาร์เรย์ข้อมูล (ArrayData)

|   |   |   |    |    |    |    |    |
|---|---|---|----|----|----|----|----|
| 1 | 5 | 9 | 12 | 15 | 21 | 29 | 31 |
|---|---|---|----|----|----|----|----|

0

1

2

3

4

5

6

7

first

last

← ลำดับ

# ตัวอย่างการค้นหาแบบ Binary Search

---

กำหนดข้อมูลในการค้นหา หรือ Key = 9 (มีข้อมูล)

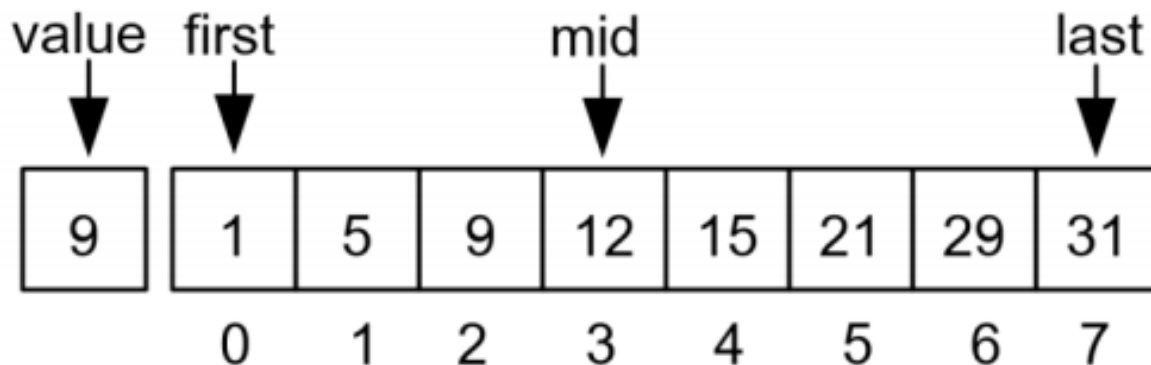
**รอบที่ 1** ค้นหา 9 (value = 9)

first = 0

last = 7

$mid = \frac{0+7}{2} = 3$

value < ArrayData[mid] (9 < 12)





## ตัวอย่างการค้นหาแบบ Binary Search [2]

---

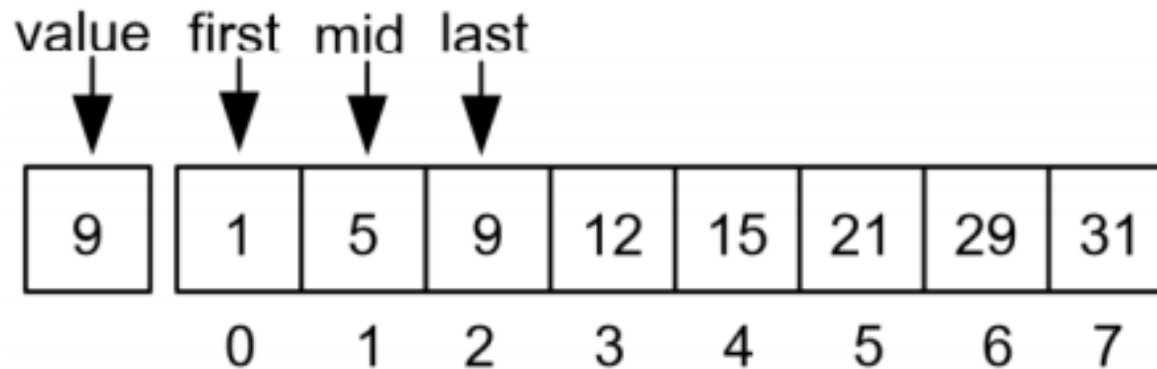
รอบที่ 2

first = 0

last = 2

$mid = \frac{0+2}{2} = 1$

value > ArrayData[mid] (9 > 5)



## ตัวอย่างการค้นหาแบบ Binary Search [3]

---

รอบที่ 3

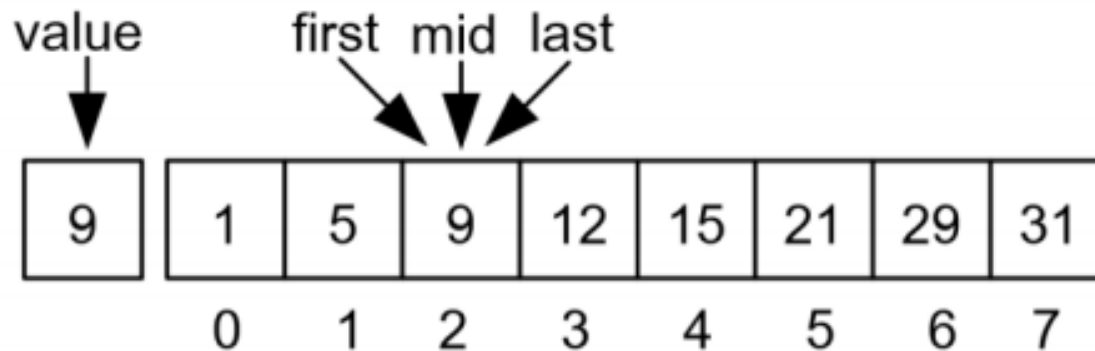
first = 2

last = 2

$mid = \frac{2+2}{2} = 2$

value = ArrayData[mid] (9 = 9)

return(mid)



## ตัวอย่างการค้นหาแบบ Binary Search [4]

---

กำหนดข้อมูลในการค้นหา หรือ Key = 6 (ไม่มีข้อมูล)

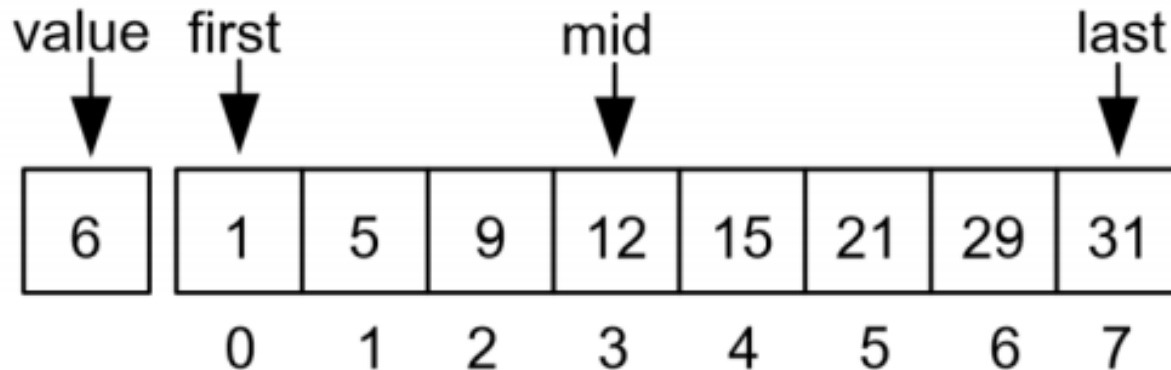
**รอบที่ 1** ค้นหา 6 (value = 6)

first = 0

last = 7

$mid = \frac{0+7}{2} = 3$

value < ArrayData[mid] (6 < 12)



## ตัวอย่างการค้นหาแบบ Binary Search [5]

---

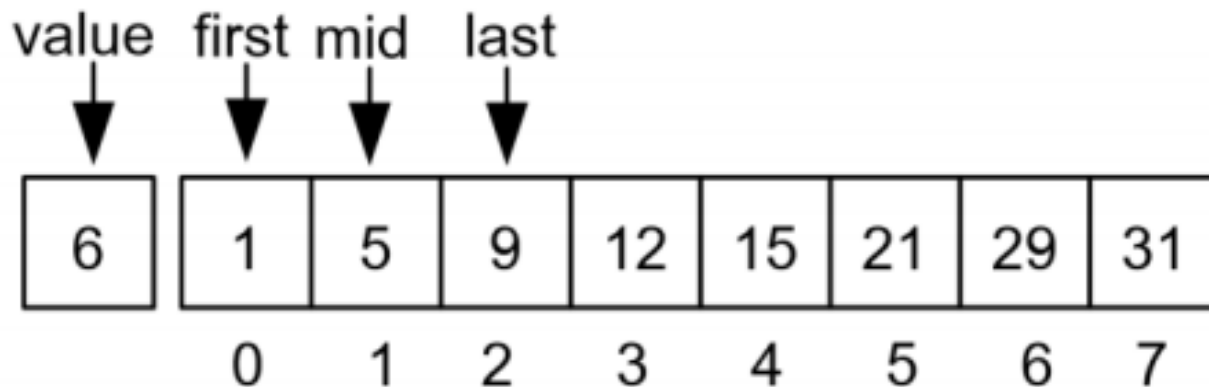
รอบที่ 2

first = 0

last = 2

$\text{mid} = \frac{0+2}{2} = 1$

value > ArrayData[mid] (6 > 5)



## ตัวอย่างการค้นหาแบบ Binary Search [6]

---

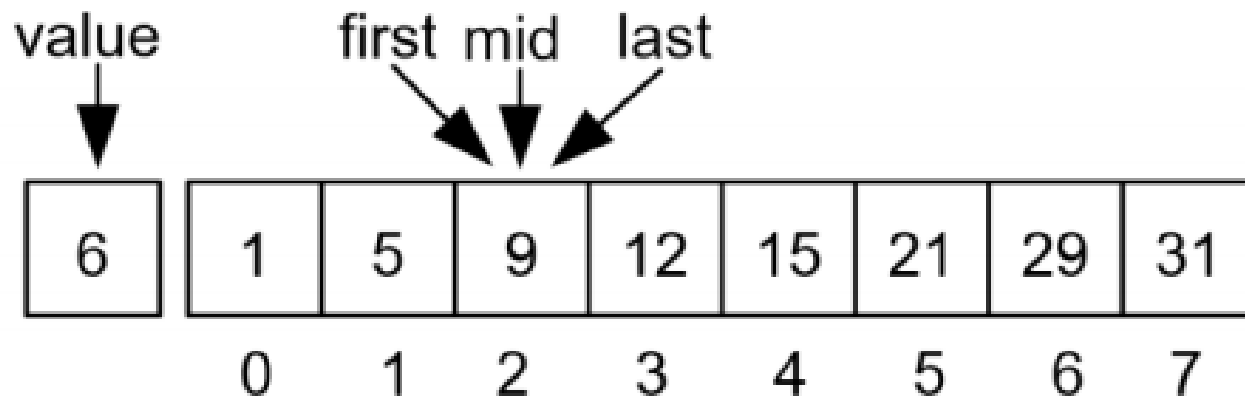
รอบที่ 3

first = 2

last = 2

$mid = \frac{2+2}{2} = 2$

value < ArrayData[mid] (6 < 9)



## ตัวอย่างการค้นหาแบบ Binary Search [7]

---

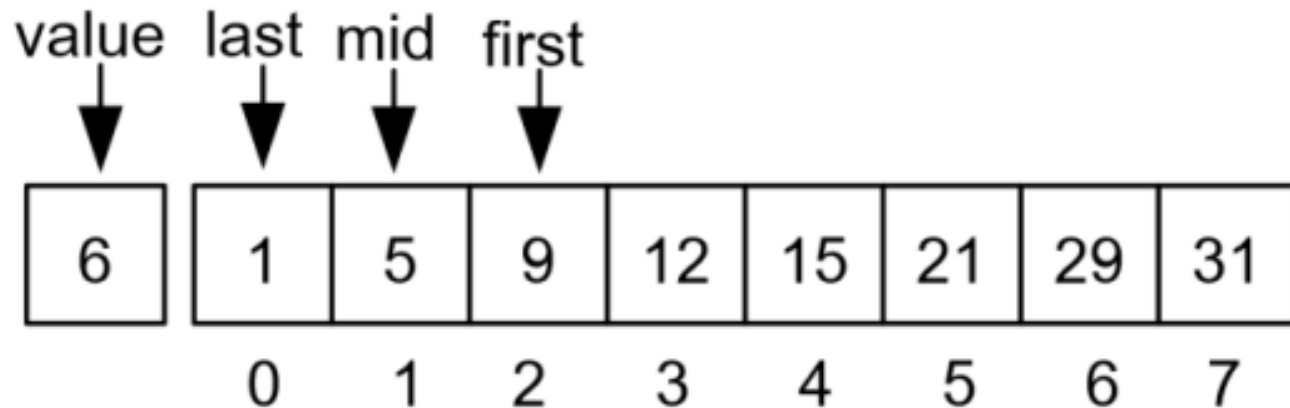
รอบที่ 4

first = 2

last = 1

First > last

return -1



# การวิเคราะห์การทำงานของ Binary Search

---

- กระบวนการทำงานแบบ Binary Search จะพบว่าเมื่อมีการเปรียบเทียบแต่ละครั้งจะมีการตัดข้อมูลในตารางออกไปได้ทีละครึ่งหนึ่งเสมอ ดังนั้นถ้าเริ่มต้นมีจำนวนข้อมูล  $n$  ตัว จำนวนข้อมูลที่นำมาเพื่อค้นหาค่าที่ต้องการหลังการเปรียบเทียบแต่ละครั้งจะเป็นดังนี้

$$\text{ครั้งที่ } 2 = n/2 \text{ (หรือ } n/2^1)$$

$$\text{ครั้งที่ } 3 = n/4 \text{ (หรือ } n/2^2)$$

$$\text{ครั้งที่ } 4 = n/8 \text{ (หรือ } n/2^3)$$

จึงสรุปได้ว่าประสิทธิภาพของการค้นหา คือ  $O(\log n)$

# บทที่ 10 searching and hashing

## บทเรียนย่อย

---

10.1 Searching and Hashing Introduction

10.2 Sequential Search Algorithm

10.3 Binary Search Algorithm

10.4 Hashing Algorithm



## 10.4 Hashing Algorithm

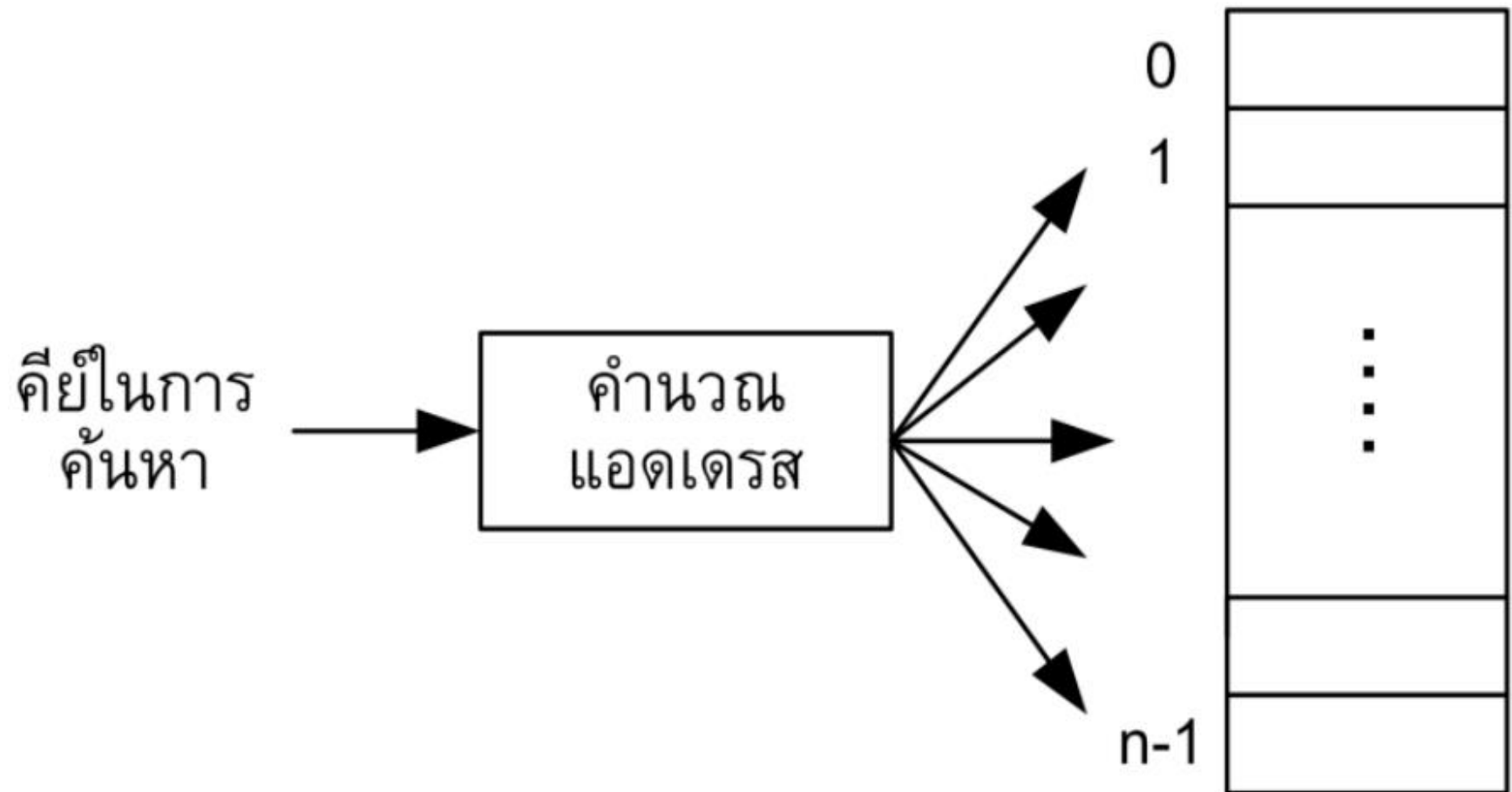
**Hashing Search (การค้นหาแบบแฮช)** เป็นวิธีการค้นหาข้อมูลที่ใช้การแปลงคีย์ (Key) ให้เป็นตำแหน่ง (Address) ที่อยู่ในพื้นที่เก็บข้อมูลโดยใช้เทคนิคการสร้างตารางมาเพื่อเก็บคีย์ดังกล่าวซึ่งมีชื่อเรียกว่า Hash Table

โดยการจะได้มาของ Hash Table จะต้องอาศัยแนวคิดต่าง ๆ ในการคำนวณเพื่อแปลงคีย์ที่เป็นเลขจำนวนเต็มให้เป็นตำแหน่ง เรียกว่าการทำ Hash Function ซึ่งมีหลากหลายวิธี ดังนี้

- Selection Digits (การเลือกหลัก)
- Folding (การบวกหลัก)
- Modulate arithmetic (การหารเอาเศษ)

# ลักษณะของการทำ Hash Table

---



# การทำ Hash Function แบบง่าย

---

Hash Function แบบการหารเอาเศษ

$$\text{hash(key)} = \text{key MOD TableSize}$$

โดยที่ TableSize คือ ขนาดของตารางที่ต้องการจัดเก็บ  
หรือค่าที่เป็นจำนวนเฉพาะ (Prime Number) เช่น 2, 3, 5, 7, 11,  
13, 17, 19 เป็นต้น

## การทำ Hash Function แบบง่าย [2]

---

### ตัวอย่าง

ชุดตัวเลข 9, 17, 23, 15 ถ้ากำหนด แฮชฟังก์ชันด้วย  
สมการ  $H(K) = K \text{ MOD } 5$  ค่าที่ได้จากแฮชฟังก์ชันแสดงได้  
ดังนี้

hash function

$$H(9) = 4$$

$$H(17) = 2$$

$$H(23) = 3$$

$$H(15) = 0$$



hash table

|   |    |
|---|----|
| 0 | 15 |
| 1 |    |
| 2 | 17 |
| 3 | 23 |
| 4 | 9  |

## การทำ Hash Function แบบง่าย [3]

---

### ตัวอย่าง

ชุดตัวเลข 9, 17, 24, 14, 13, 5 ถ้ากำหนด แฮชซึ่งฟังก์ชันด้วย  
สมการ  $H(K) = K \text{ MOD } 9$  ค่าที่ได้จากแฮชซึ่งฟังก์ชันแสดงได้ดังนี้

### hash function

$$H(9) = 0$$

$$H(17) = 8$$

$$H(24) = 6$$

$$H(14) = 5$$

$$H(13) = 4$$

$$H(5) = 5$$

## การทำ Hash Function แบบง่าย [3]

---

### ตัวอย่าง

ชุดตัวเลข 9, 17, 24, 14, 13, 5 ถ้ากำหนด แฮชซึ่งฟังก์ชันด้วย  
สมการ  $H(K) = K \text{ MOD } 9$  ค่าที่ได้จากแฮชซึ่งฟังก์ชันแสดงได้ดังนี้

#### hash function

$$H(9) = 0$$

$$H(17) = 8$$

$$H(24) = 6$$

$$H(14) = 5$$

$$H(13) = 4$$

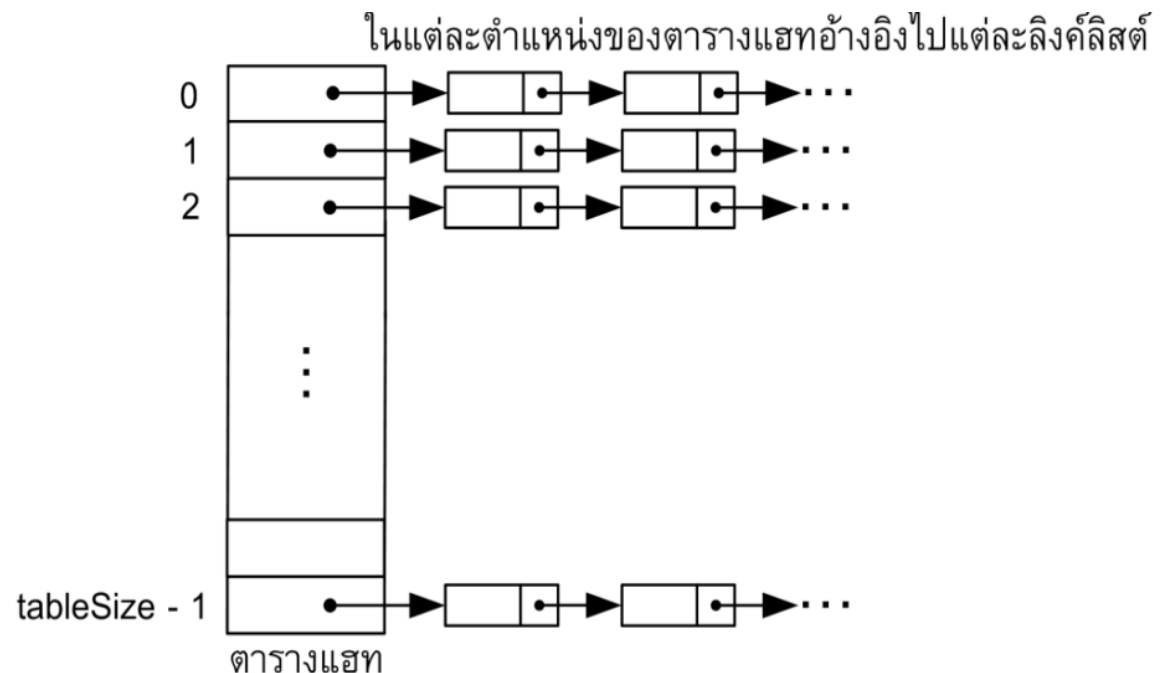
$$H(5) = 5$$

ลักษณะเช่นนี้เรียกว่า  
เกิดการชนกันของคีย์  
(Collision)

# การแก้ปัญหา Collision แบบ Chaining

---

วิธี Chaining เป็นวิธีการแก้ไขปัญหาของการชนกันที่ดีที่สุด มีการใช้ linked list เข้ามาช่วย โดยมีการแบ่งเนื้อที่ออกเป็น 2 ส่วน ส่วนแรกจะเก็บ address ส่วนที่สองจะเก็บค่าข้อมูลทั้ง list ที่มี address เดียวกัน



# การแก้ปัญหา Collision แบบ Chaining [2]

## ตัวอย่าง

ชุดตัวเลข 9, 17, 24, 14, 13, 5 ถ้ากำหนด แฮชฟังก์ชันด้วย  
สมการ  $H(K) = K \text{ MOD } 9$  ค่าที่ได้จากแฮชฟังก์ชันแสดงได้ดังนี้

hash function

$$H(9) = 0$$

$$H(17) = 8$$

$$H(24) = 6$$

$$H(14) = 5$$

$$H(13) = 4$$

$$H(5) = 5$$

