หน่วยที่ 9

การค้นหาข้อมูล

หัวข้อเรื่อง

1. การค้นหาข้อมูล
2. การค้นหาแบบลำดับ
3. การค้นหาแบบไบนารี
4. การค้นหาแบบแฮชชิ่ง

สาระสำคัญ

ในคอมพิวเตอร์การค้นหาข้อมูล (Searching) ถือเป็นการดำเนินการที่มีความสำคัญมากอีกเรื่องหนึ่ง เนื่องจากเป็นการเข้าไปค้นหาว่าข้อมูลที่จัดเก็บในโครงสร้างข้อมูลนั้น มีข้อมูลตัวที่เราต้องการอยู่หรือไม่ ซึ่งรูปแบบของการค้นหาข้อมูลนั้นมีหลายรูปแบบ และในการเลือกวิธีการค้นหาข้อมูลแบบใดนั้นจะต้องทราบถึงกระบวนการจัดเก็บ ตลอดจนลักษณะของข้อมูล เพื่อที่จะทำการค้นหาได้อย่างรวดเร็ว เช่น การค้นหาแบบลำดับ (Sequential Search) เหมาะกับข้อมูลที่มีปริมาณน้อยและยังไม่ได้จัดเรียง การค้นหาแบบไบนารี (Binary Search) เหมาะกับข้อมูลที่มีปริมาณมากและมีการจัดเรียงข้อมูล เพราะสามารถลดการเปรียบเทียบได้ทีละครึ่ง ส่วนการค้นหาแบบแฮชสามารถค้นหาข้อมูลได้ตามรูปแบบของการแปลงคีย์

จุดประสงค์เชิงพฤติกรรม

1. อธิบายหลักการค้นหาข้อมูลได้
2. อธิบายหลักการค้นหาแบบลำดับ แบบไบนารี และแบบแฮชได้
3. แสดงขั้นตอนการค้นหาข้อมูลแบบลำดับ แบบไบนารี และแบบแฮชได้
4. อธิบายวิธีการแก้ไขปัญหาการชนกันของคีย์ได้
5. อธิบายศัพท์เกี่ยวกับการค้นหาข้อมูลได้

การค้นหาข้อมูล

ในคอมพิวเตอร์การค้นหาข้อมูล (Searching) ถือเป็นอีกหนึ่งการดำเนินการที่มีความสำคัญมาก เนื่องจากเป็นการเข้าไปค้นหาว่าข้อมูลที่จัดเก็บในโครงสร้างข้อมูลนั้น มีข้อมูลตัวที่เราต้องการ (Target) อยู่หรือไม่ ซึ่งคำตอบที่ได้จะมีอยู่ด้วยกัน 2 แบบคือ

1. **สำเร็จ (Successful)** หมายถึง เมื่อค้นหาแล้วพบข้อมูลตัวที่ต้องการในโครงสร้างข้อมูล
2. **ไม่สำเร็จ (Unsuccessful)** หมายถึง เมื่อค้นหาข้อมูลแล้วไม่พบข้อมูลตัวที่ต้องการในโครงสร้างข้อมูล

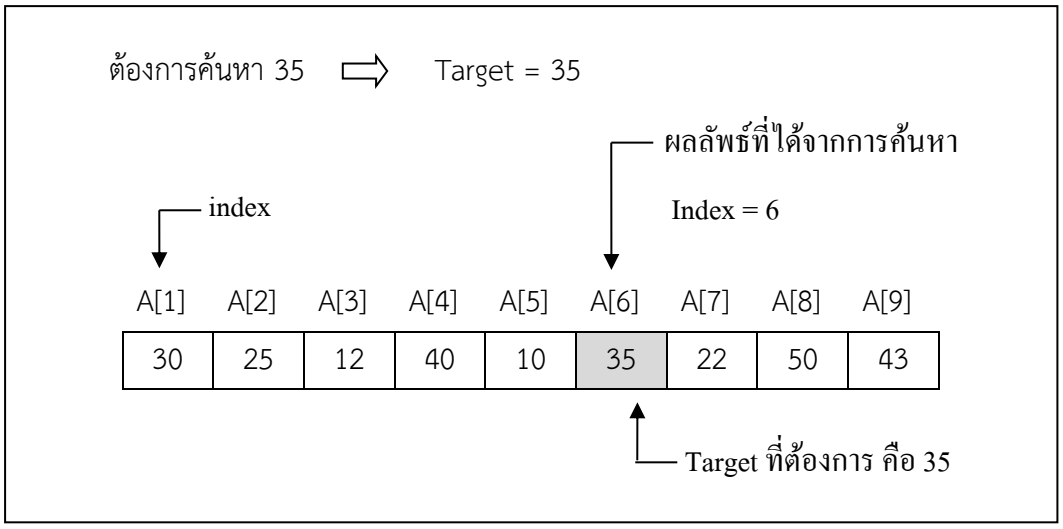
โดยหลักการค้นหาข้อมูลนั้น จะต้องกำหนดคีย์หลักเพื่อใช้ในการเปรียบเทียบกับข้อมูลที่จัดเก็บในโครงสร้างข้อมูลด้วยอัลกอริทึมที่กำหนดเพื่อใช้ในการเปรียบเทียบ ซึ่งความเร็วและประสิทธิภาพของการค้นหาข้อมูลนั้น ขึ้นอยู่กับอัลกอริทึมที่เลือกใช้ว่าเหมาะสมกับลักษณะของข้อมูลที่จัดเก็บในโครงสร้างต่างๆ หรือไม่ และการจะเลือกใช้อัลกอริทึมใดนั้นมีหลายสิ่งมาประกอบเพื่อการตัดสินใจ เช่น

1. ลักษณะโครงสร้างข้อมูลเป็นแบบใด เป็นแบบอาร์เรย์ แบบลิงค์ลิสต์ หรือแบบต้นไม้
2. ลักษณะของการจัดเก็บข้อมูลในโครงสร้างข้อมูล ข้อมูลมีการเรียงลำดับหรือไม่เรียงลำดับ
3. ปริมาณของข้อมูลในโครงสร้างข้อมูล ข้อมูลมีปริมาณมากหรือมีปริมาณน้อย

สำหรับวิธีการในการค้นหาข้อมูลนั้นมีหลายแบบ เช่น การค้นหาแบบลำดับ (Sequential Search) การค้นหาแบบไบนารี (Binary Search) และการค้นหาแบบแฮชชิ่ง (Hashing)

การค้นหาแบบลำดับ

การค้นหาแบบลำดับ (Sequential Search) หรือ การค้นหาแบบเชิงเส้น (Linear Search) เป็นวิธีการค้นหาที่ง่ายที่สุดและไม่ซับซ้อนเมื่อเทียบกับวิธีอื่น โดยการดำเนินการค้นหาข้อมูลจะทำในลักษณะเรียงลำดับ มีหลักการในการค้นหาคือ จะนำค่าที่ต้องการค้นหา (Target)ไปเปรียบเทียบกับข้อมูลภายในกลุ่มข้อมูลหรือลิสต์ทีละตัวว่ามีค่าตรงกันหรือไม่ ถ้าไม่ตรงก็จะดำเนินการค้นหาตัวถัดไปโดยจะดำเนินการเปรียบเทียบเช่นนี้ไปเรื่อยๆ จนถึงข้อมูลสุดท้าย หากไม่ตรงกับค่าใดก็แสดงว่าข้อมูลที่ต้องการค้นหาไม่ได้อยู่ในกลุ่มข้อมูลที่ค้น หลักการของการค้นหาสามารถแสดงได้ดังรูปที่ 9.1



รูปที่ 9.1 หลักการค้นหาข้อมูล

สำหรับหลักการค้นหาแบบลำดับนั้น มีส่วนประกอบในการค้นหาข้อมูลอยู่ 2 สิ่งคือ

1. **ค่าข้อมูลที่ต้องการค้นหา (Target given)** ซึ่งค่าข้อมูลนี้เป็นข้อมูลที่เราต้องการโดยจะถูกจัดเก็บอยู่ในอาร์เรย์นั่นเอง

2. **ตำแหน่งที่ต้องการ (Location Wanted)** ซึ่งหมายถึงตำแหน่งอ้างอิงที่ใช้ในการจัดเก็บข้อมูล โดยอาร์เรย์จะทำการกำหนดช่องในการจัดเก็บ ซึ่งมีตัวชี้ (Index) เป็นตัวระบุถึงตำแหน่งในหน่วยความจำ

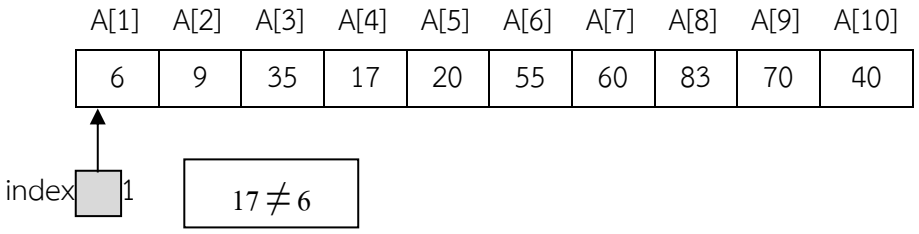
เพื่อช่วยให้เข้าใจหลักการค้นหาข้อมูลแบบลำดับได้ง่ายขึ้น ให้ศึกษาวิธีการทำงานจากตัวอย่างต่อไปนี้

ตัวอย่างที่ 9.1 แสดงการค้นหาข้อมูลค่า 17 จากชุดข้อมูลที่จัดเก็บในอาร์เรย์ดังนี้

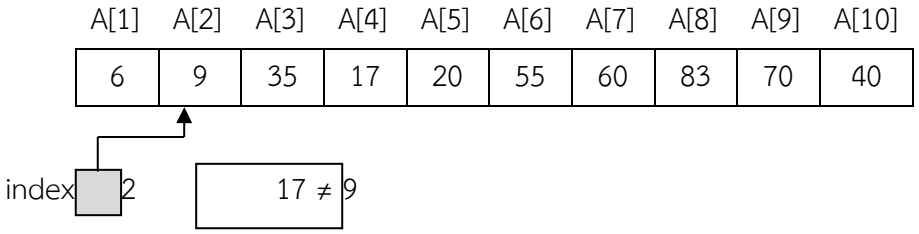
A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]	A[9]	A[10]
6	9	35	17	20	55	60	83	70	40

ค่าที่ต้องการค้นหา คือ 17 => Target = 17

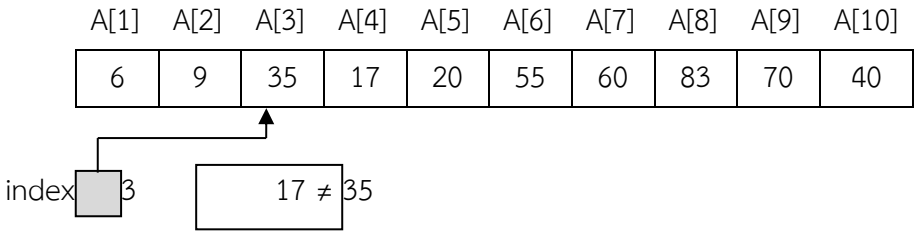
ครั้งที่ 1



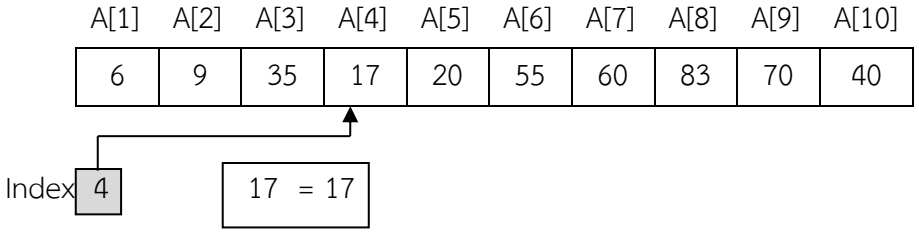
ครั้งที่ 2



ครั้งที่ 3



ครั้งที่ 4



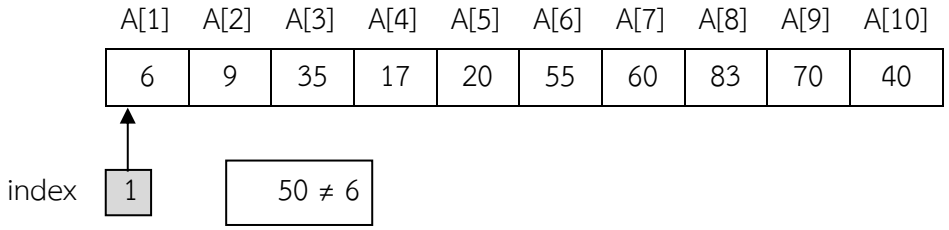
จากขั้นตอนการค้นหาข้างต้น กำหนดค่าที่ต้องการค้นหา คือ 17 เริ่มทำการค้นหาตั้งแต่ตำแหน่งแรก (Index = 1) และทำการค้นหาไปเรื่อยๆ จนกระทั่งพบค่าที่ต้องการในตำแหน่งที่ 4 (Index = 4) นั่นหมายถึง การค้นหาสำเร็จ (Successful)

ตัวอย่างที่ 9.2 แสดงการค้นหาข้อมูลค่า 50 จากชุดข้อมูลที่จัดเก็บในอาร์เรย์ดังนี้

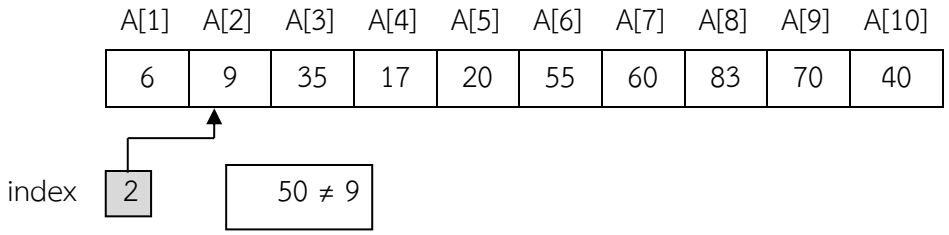
A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]	A[9]	A[10]
6	9	35	17	20	55	60	83	70	40

ค่าที่ต้องการค้นหา คือ 50 => Target = 50

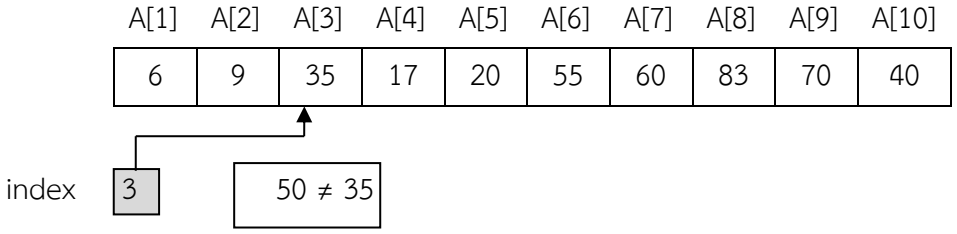
<u>ครั้งที่ 1</u>



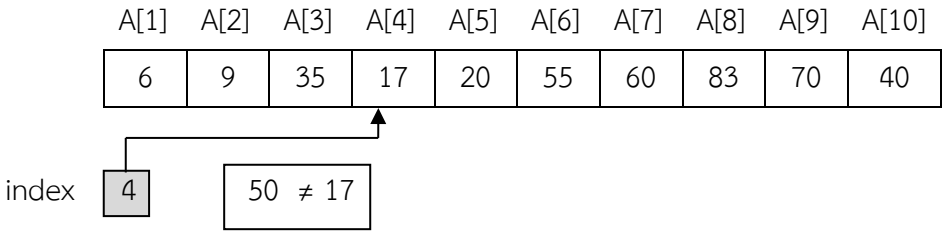
<u>ครั้งที่ 2</u>



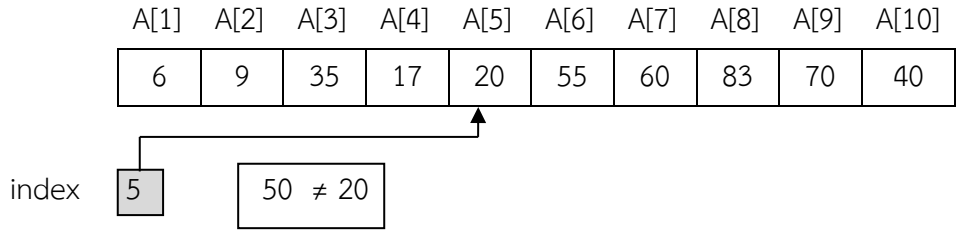
<u>ครั้งที่ 3</u>



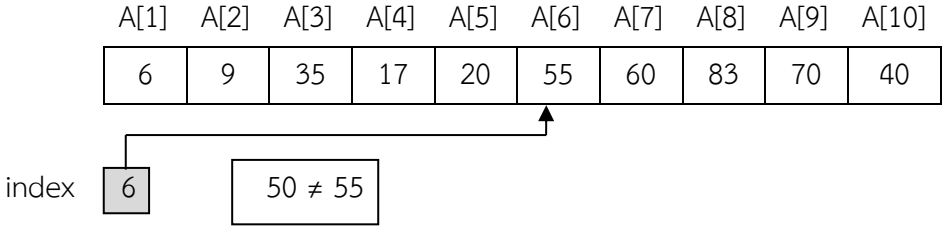
ครั้งที่ 4



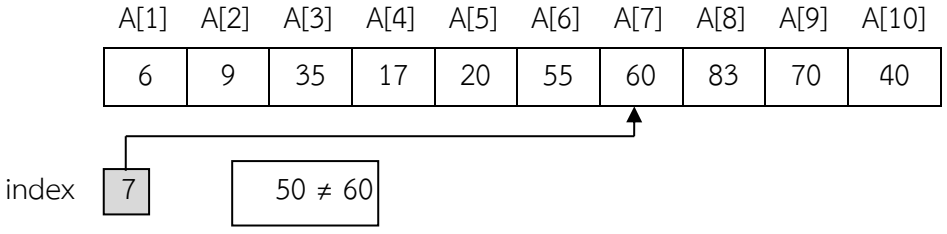
ครั้งที่ 5



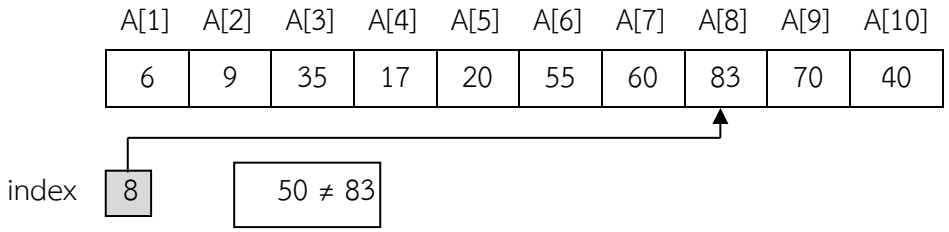
ครั้งที่ 6



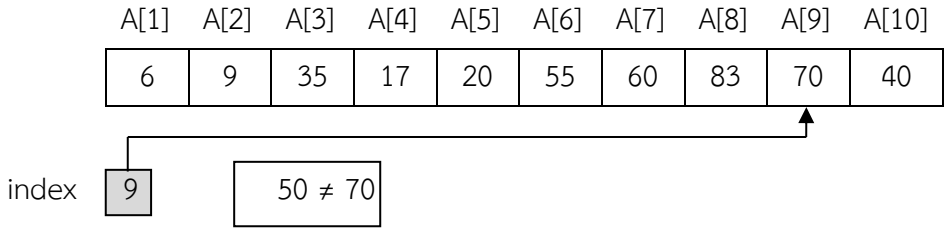
ครั้งที่ 7



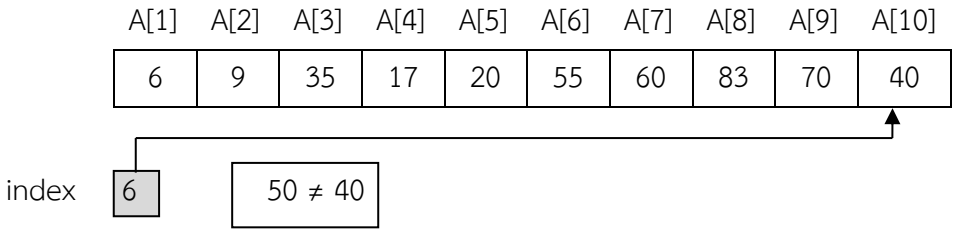
ครั้งที่ 8



ครั้งที่ 9



ครั้งที่ 10



จากตัวอย่างเป็นกรณีค้นหาข้อมูลแล้วไม่พบ โดยได้กำหนดค่าที่ต้องการค้นหา คือ 50 เริ่มทำการค้นหาตั้งแต่ตำแหน่งแรก (Index = 1) และทำการค้นหาไปเรื่อยๆ จนกระทั่งถึงตำแหน่งสุดท้าย ปรากฏว่าค้นหาข้อมูลที่ต้องการไม่พบ นั่นหมายถึง การค้นหาไม่สำเร็จ (Unsuccessful)

การค้นหาข้อมูลแบบลำดับนั้น จะมีประสิทธิภาพดีเมื่อโครงสร้างข้อมูลมีปริมาณข้อมูลน้อย เพราะอัลกอริทึมไม่ได้มีขั้นตอนการประมวลผลอะไรมากนัก เพียงแต่ไล่เปรียบเทียบไปเรื่อยๆ เท่านั้น แต่ถ้านำไปใช้งานในโครงสร้างข้อมูลที่มีข้อมูลปริมาณมากๆ แล้วการค้นหาแบบนี้จะไม่มีประสิทธิภาพ เนื่องจากจะเสียเวลามาก

การค้นหาแบบไบนารี

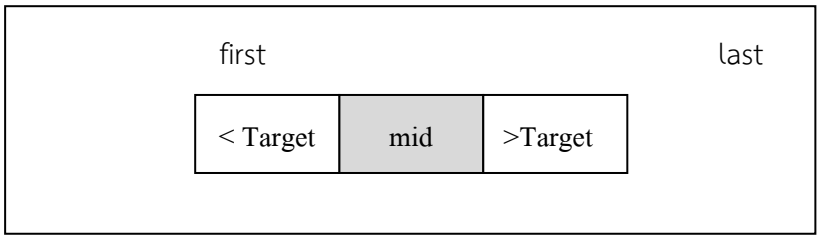
การค้นหาแบบไบนารี (Binary Search) จัดเป็นเทคนิคที่มีประสิทธิภาพสูงและสามารถใช้งานได้กับโครงสร้างข้อมูลหลายแบบ คือ โครงสร้างข้อมูลแบบอาร์เรย์ โครงสร้างข้อมูลแบบลิงค์ลิสต์ และโครงสร้างข้อมูลแบบต้นไม้ โดยการค้นหาแบบไบนารีเหมาะกับการนำไปใช้ค้นหาข้อมูลที่มีปริมาณมาก และข้อมูลต้องถูกจัดเรียงลำดับแล้วเท่านั้น

หลักการทำงานของการค้นหาแบบไบนารี คือ จะแบ่งชุดข้อมูลหรือลิสต์ออกเป็น 2 กลุ่ม จากนั้นทำการเปรียบเทียบข้อมูลที่ต้องการค้นหากับข้อมูลที่อยู่ตรงกึ่งกลางของลิสต์ ถ้าไม่เท่ากันก็แสดงว่ายังไม่พบข้อมูลจะต้องทำการค้นหาต่อไป โดยหากค่าที่ต้องการค้นหาน้อยกว่าข้อมูลที่อยู่ตรงกึ่งกลางลิสต์ ก็จะพิจารณาครึ่งแรกแล้วตัดครึ่งหลังทิ้งไป ทำนองเดียวกันหากค่าที่ต้องการค้นหามากกว่าข้อมูลที่อยู่ตรงกึ่งกลางลิสต์ ก็จะพิจารณาครึ่งหลังและตัดครึ่งแรกทิ้งไป จากนั้นก็จะหาตัวกึ่งกลางใหม่จากข้อมูลครึ่งที่เก็บไว้เพื่อเปรียบเทียบต่อไป ทำเช่นนี้ไปเรื่อยๆ จนกว่าจะค้นพบข้อมูลหรือจนกว่าจะแบ่งครึ่งไม่ได้อีกต่อไป นั่นแสดงว่าค่าที่ต้องการค้นหาไม่ได้อยู่ในลิสต์ชุดนี้

การค้นหาตำแหน่งกึ่งกลางของลิสต์ ใช้สูตรดังนี้

```
mid = (first + last)/2
```

เมื่อ mid คือ ตำแหน่งกึ่งกลางของชุดข้อมูลหรือลิสต์
first คือ ตำแหน่งข้อมูลแรก
last คือ ตำแหน่งข้อมูลสุดท้าย



รูปที่ 9.2 ตำแหน่งของค่า first, mid และ last

เพื่อให้เข้าใจหลักการค้นหาข้อมูลแบบไบนารีได้ง่ายขึ้น ให้ศึกษาวิธีการทำงานจากตัวอย่างต่อไปนี้

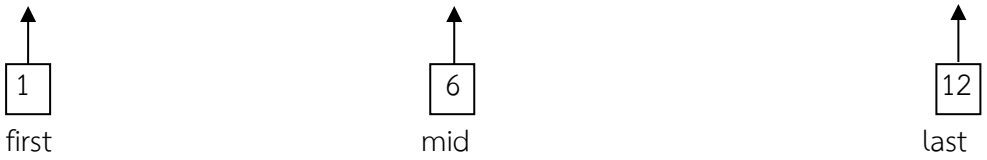
ตัวอย่างที่ 9.3 แสดงการค้นหาข้อมูลแบบไบนารี โดยให้ค้นหา 80 จากอาร์เรย์ที่เรียงลำดับแล้ว ดังนี้

A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]	A[9]	A[10]	A[11]	A[12]
4	7	19	22	35	50	67	71	80	85	92	98

Target = 80

<u>ครั้งที่ 1</u> คำนวณหาตำแหน่งกึ่งกลาง จะได้ mid = (1+12)/2 = 6.5 = 6 (ตัดเศษทิ้ง)

A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]	A[9]	A[10]	A[11]	A[12]
4	7	19	22	35	50	67	71	80	85	92	98

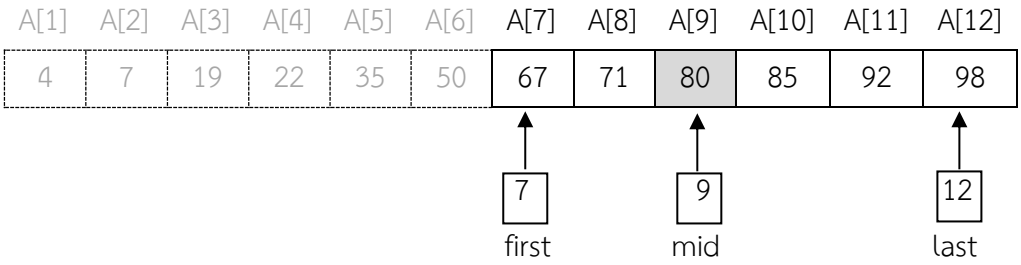


ให้นำค่า Target ไปเปรียบเทียบกับค่ากึ่งกลาง (80>50) ผลปรากฏว่า Target มีค่ามากกว่า ดังนั้นให้ทิ้งข้อมูลตั้งแต่ตำแหน่งที่ 1 ถึงตำแหน่งที่ 6 เนื่องจากข้อมูลส่วนนี้มีค่าน้อยกว่า Target จึงไม่ต้องนำมาใช้เพื่อการค้นหาอีก

<u>ครั้งที่ 2</u> คำนวณหาตำแหน่งกึ่งกลางใหม่จากลิสต์ครึ่งหลัง เนื่องจาก Target มากกว่า ค่า mid ดังนั้นจะหาค่า first ใหม่ ดังนี้

first = mid + 1 = 6 + 1 = 7

mid = (7+12)/2 = 9.5 = 9 (ตัดเศษทิ้ง)



นำค่า Target ไปเปรียบเทียบกับค่ากึ่งกลาง (80=80) ผลปรากฏว่า Target มีค่าเท่ากับค่ากลาง แสดงว่าค้นพบ Target หรือค่าที่ต้องการแล้ว ซึ่งอยู่ในตำแหน่งที่ 9 ดังนั้นการค้นหาจึงสิ้นสุดลงในรอบนี้

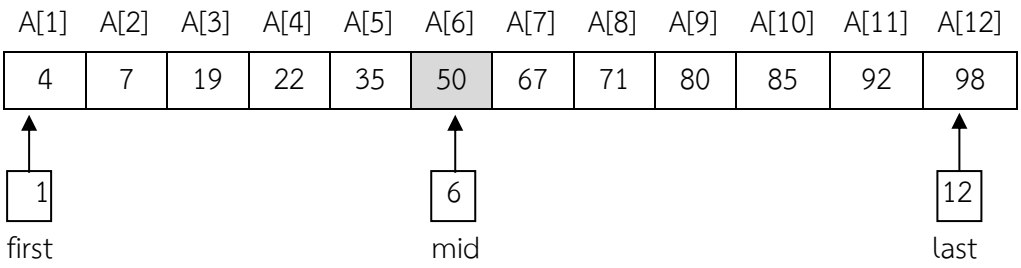
จากขั้นตอนการทำงานข้างต้น จะเป็นลักษณะของการแบ่งข้อมูลและเปรียบเทียบค่าข้อมูล ซึ่งถือว่าให้ประสิทธิภาพการค้นหาที่รวดเร็ว โดยเฉพาะหากมีจำนวนข้อมูลที่มากก็จะลดจำนวนการเปรียบเทียบข้อมูลไปได้ครึ่งหนึ่ง

ตัวอย่างที่ 9.4 แสดงการค้นหาข้อมูลแบบไบนารี โดยให้ค้นหา 22 จากอาร์เรย์ที่เรียงลำดับแล้วดังนี้

A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]	A[9]	A[10]	A[11]	A[12]
4	7	19	22	35	50	67	71	80	85	92	98

Target = 22

<u>ครั้งที่ 1</u> คำนวณหาตำแหน่งกึ่งกลาง จะได้ mid = (1+12)/2 = 6.5 = 6 (ตัดเศษทิ้ง)

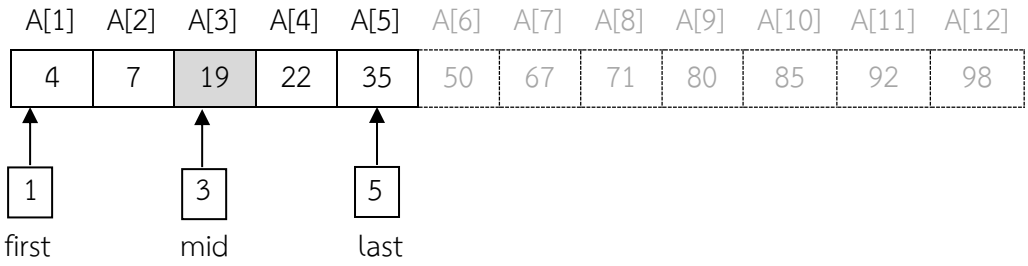


Target < A[mid]

22 < 50 => $last_{ใหม่}$ = mid – 1

= 6 – 1 = 5

<u>ครั้งที่ 2</u> คำนวณหาตำแหน่งกึ่งกลางใหม่ จะได้ mid = (1+5)/2 = 3

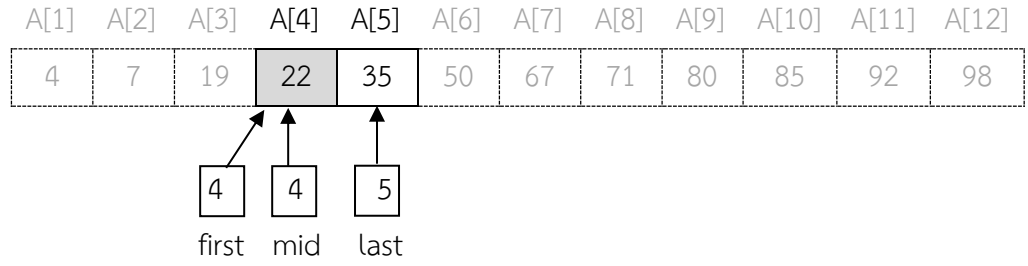


Target > A[mid]

22 > 19 => $first_{ใหม่}$ = mid + 1

= 3 + 1 = 4

<u>ครั้งที่ 3</u> คำนวณหาตำแหน่งกึ่งกลางใหม่ จะได้ mid = (4+5)/2 = 4.5 = 4 (ตัดเศษทิ้ง)



Target = A[mid]

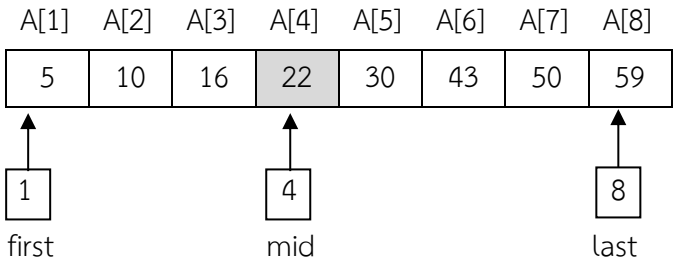
2 = 22 => Successful ที่ตำแหน่ง index = 4

ตัวอย่างที่ 9.5 แสดงการค้นหาข้อมูลแบบไบนารี แล้วไม่พบข้อมูล โดยให้ค้นหา 15 จากอาร์เรย์ที่เรียงลำดับแล้วดังนี้

A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]
5	10	16	22	30	43	50	59

Target = 15

<u>ครั้งที่ 1</u> คำนวณหาตำแหน่งกึ่งกลาง จะได้ mid = (1+8)/2 = 4.5 = 4 (ตัดเศษทิ้ง)

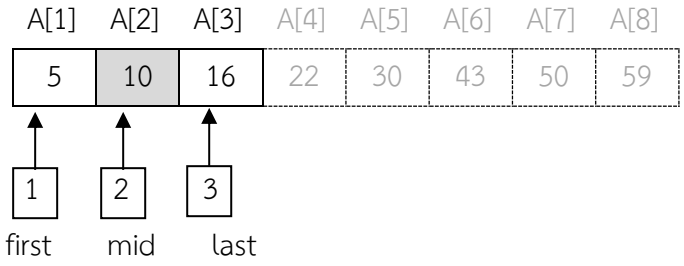


Target < A[mid]

15 < 22 => $last_{ใหม่}$ = mid – 1

= 4 – 1 = 3

<u>ครั้งที่ 2</u> คำนวณหาตำแหน่งกึ่งกลางใหม่ จะได้ mid = (1+3)/2 = 2

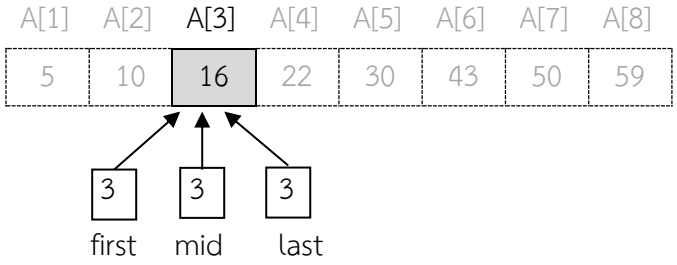


Target > A[mid]

15 > 10 => $first_{ใหม่}$ = mid + 1

= 2 + 1 = 3

<u>ครั้งที่ 3</u> คำนวณหาตำแหน่งกึ่งกลางใหม่ จะได้ mid = (3+3)/2 = 3



Target < A[mid]

15 < 16

เนื่องจากในรอบนี้ first = mid = last และผลการเปรียบเทียบคีย์ไม่เท่ากัน จึงถือว่า Unsuccessful นั่นคือ ไม่พบข้อมูลที่ต้องการค้นหา จึงสิ้นสุดการทำงานในรอบนี้

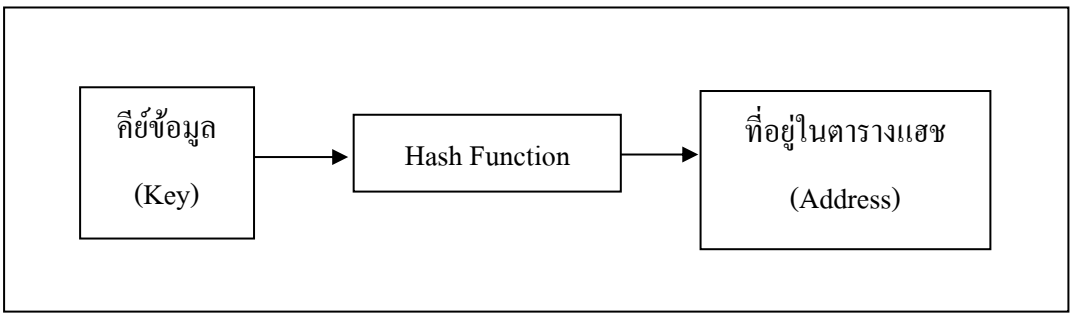
การค้นหาแบบแฮชชิ่ง

แฮชชิ่ง (Hashing) เป็นวิธีการจัดเก็บข้อมูลที่มีปริมาณมากๆ ไว้ในตาราง โดยตารางที่ใช้จัดเก็บข้อมูลนั้นเรียกว่า **ตารางแฮช** (Hash Table) ส่วนเทคนิคที่ใช้หาตำแหน่งที่เก็บข้อมูลในตารางแฮช จะเรียกว่า **ฟังก์ชั่นแฮช** (Hash Function) ซึ่งเป็นฟังก์ชั่นการคำนวณทางคณิตศาสตร์

การค้นหาแบบแฮชชิ่ง (Hashing Search) เป็นวิธีที่สามารถเข้าถึงข้อมูลได้โดยตรง นั่นคือ สามารถค้นหาข้อมูลที่ต้องการได้ภายในครั้งเดียว ซึ่งการค้นหาแบบแฮชชิ่งจะมีคีย์เพื่อใช้ในการค้นหา โดยจะนำคีย์ไปผ่านฟังก์ชั่นแฮชเพื่อแปลงค่าคีย์เป็นแอดเดรส (Address) หรือตำแหน่งที่อยู่ของคีย์นั้น จึงทำให้การค้นหาแบบแฮชชิ่งสามารถเข้าถึงตำแหน่งข้อมูลด้วยคีย์ได้ทันที

หลักการพื้นฐานของแฮชชิ่ง

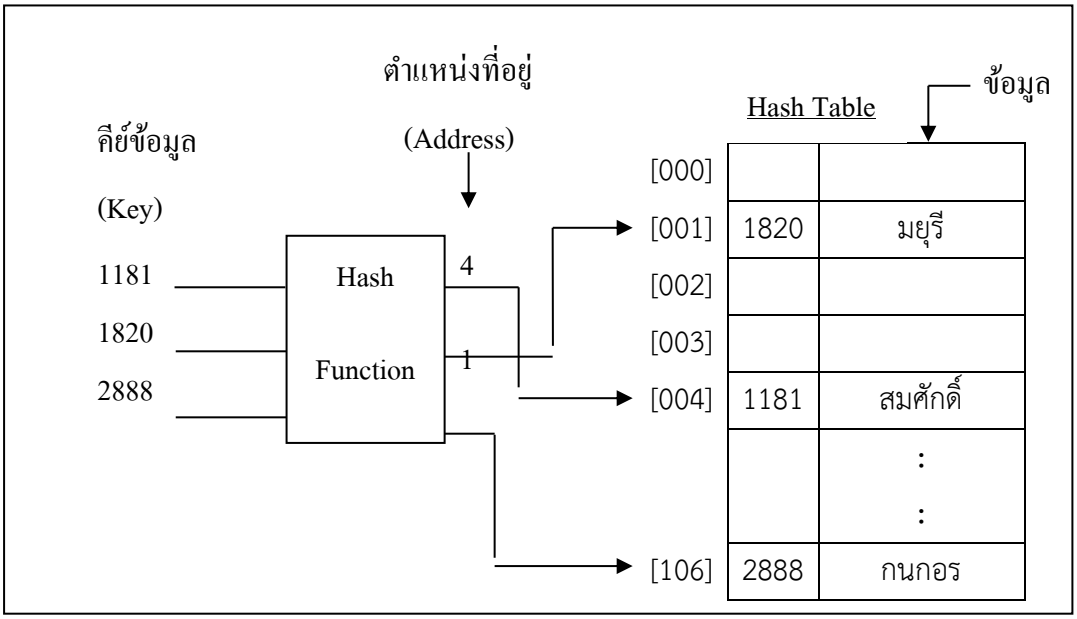
การแฮชชิ่ง คือ การแปลงค่าคีย์ข้อมูลให้กลายเป็นตำแหน่งที่อยู่ (Address) เพื่อที่จะสามารถเก็บค่าคีย์ข้อมูลนั้นๆ ลงในตารางแฮช (Hash Table) ได้ และสามารถค้นหาค่าคีย์ข้อมูลนั้นได้ในภายหลัง โดยเทคนิคที่ใช้ในการแปลงค่าคีย์ คือ ฟังก์ชั่นแฮช (Hash Function) สามารถเขียนสัญลักษณ์การแปลงได้ดังนี้



รูปที่ 9.3 หลักการพื้นฐานของแฮชชิ่ง

จากรูปที่ 9.3 หมายความว่าคีย์ข้อมูล (key) ที่รับเข้ามาจะถูกแปลงค่าด้วยฟังก์ชั่นแฮช แล้วได้ผลลัพธ์เป็นตำแหน่งที่อยู่ (Address) ในตารางแฮชเพื่อจัดเก็บคีย์ข้อมูล และเมื่อต้องการค้นหาข้อมูลก็ใช้กระบวนการเดียวกัน เพื่อค้นหาตำแหน่งของข้อมูลที่ต้องการ

ส่วนประสิทธิภาพของการแฮชชิ่ง จะดีหรือไม่นั้นขึ้นอยู่กับฟังก์ชั่นแฮชที่ใช้เป็นหลัก ซึ่งลักษณะของฟังก์ชั่นแฮชที่ดีคือ ต้องสามารถกระจายคีย์ข้อมูลที่ป้อนเข้ามาให้ไปอยู่ในตำแหน่งที่อยู่ที่ไม่เกิดการชนกันเลยในตารางแฮช หรือถ้าจำเป็นต้องชนกันก็ให้เกิดการชนกันน้อยครั้งที่สุด



รูปที่ 9.4 ตัวอย่างการทำแฮชชิ่ง (Hasting) ลงในตารางแฮช (hash Table)

จากรูปที่ 9.4 เมื่อรับค่าคีย์ข้อมูล 1181 ซึ่งเป็นรหัสพนักงานเข้ามา เมื่อผ่านฟังก์ชันแฮช จะได้ค่าแอดเดรส 4 นั่นคือชี้ไปที่ตารางแฮช 004 ซึ่งเป็นตำแหน่งที่อยู่ที่จะนำข้อมูลไปเก็บ ในที่นี้ทำการจัดเก็บชื่อ สมศักดิ์ ลงในตาราง ในบางครั้งอาจเกิดปัญหาขึ้นในด้านการแปลงคีย์ คืออาจได้ตำแหน่งแอดเดรสเดียวกัน ซึ่งจะเรียกว่า **เกิดการชนกัน** (Collision) ดังนั้นเราจะต้องดำเนินการแก้ปัญหานี้เพื่อให้มีการจัดตำแหน่งที่ถูกต้อง

อย่างไรก็ตาม ในการศึกษาเกี่ยวกับแฮชชิ่ง มีศัพท์พื้นฐานที่ต้องทำความเข้าใจเสียก่อน ดังนี้

1. **คีย์ข้อมูล (Key)** คือ ข้อมูลที่ต้องการนำไปค้นหา ซึ่งจะไม่มีการซ้ำกัน

2. **ฟังก์ชั่นแฮช (Hash Function)** คือ สูตรหรือฟังก์ชั่นที่ใช้สำหรับแปลงคีย์ให้เป็นตำแหน่งแอดเดรส (Address) เมื่อได้แอดเดรสแล้วสามารถนำแอดเดรสนี้เข้าถึงตำแหน่งของข้อมูลที่ต้องการในตารางแฮชได้

3. **ตารางแฮช (Hash Table)** คือ ตารางที่ใช้สำหรับเก็บข้อมูล

4. **ตำแหน่งที่อยู่ (Address)** คือ ตำแหน่งของช่องข้อมูลในตารางแฮช ที่เราสามารถเก็บข้อมูลนั้นๆ ลงไปได้

5. **การชนกัน (Collision)** คือ การนำคีย์ข้อมูล 2 ค่ามาผ่านฟังก์ชั่นแฮช แล้วเกิดได้ค่าตำแหน่งที่อยู่เดียวกัน (ซ้ำกัน) หรือมีข้อมูลอื่นเก็บไว้ก่อนอยู่แล้ว ซึ่ง เราจะเรียกเหตุการณ์เช่นนี้ว่าเกิดการชนกัน ซึ่งเมื่อเกิดเหตุการณ์นี้ขึ้นจะต้องหาวิธีในการจัดการแก้ไขปัญหานี้ เพื่อที่จะหาตำแหน่งที่อยู่ใหม่ให้กับข้อมูลตัวที่มาชน

ฟังก์ชันแฮช

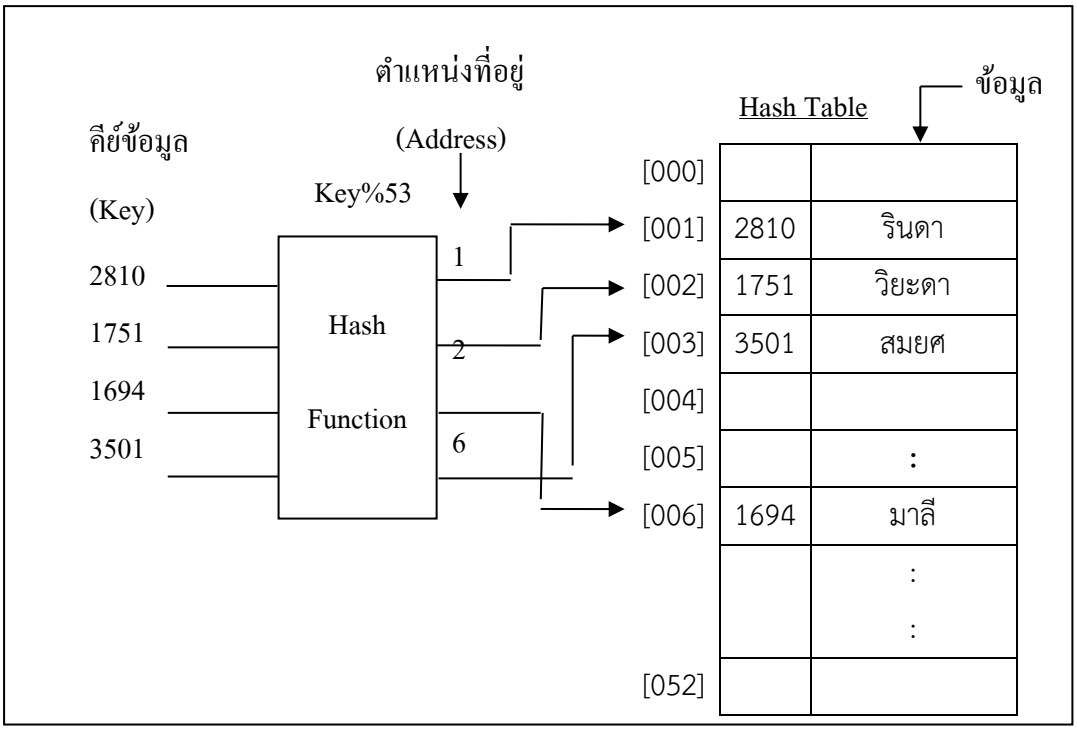
ฟังก์ชั่นแฮช (Hash Function) มีด้วยกันหลายรูปแบบ แต่ในที่นี้จะขอกล่าวถึงเพียงรูปแบบเดียว คือ วิธีการหาร (Division Hashing) เนื่องจากเป็นวิธีที่เข้าใจง่ายและเป็นที่นิยมใช้กันอย่างแพร่หลาย

วิธีการหาร คือ การหาตำแหน่งที่อยู่ในตารางแฮช โดยการนำค่าคีย์ข้อมูลมาหารแบบ Modolo ด้วยขนาดของตาราง (การหารแบบ Modolo หรือ Mod คือ การหารโดยผลลัพธ์ที่ได้ คือ ค่าเศษที่เหลือจากการหาร เช่น 5 Mod 2 = 1) โดยมีสูตรการคำนวณ ดังนี้

$$\text{Address} = K \ \text{Mod} \ N$$

เมื่อ K คือ ค่าคีย์ข้อมูล N คือ ขนาดของตารางแฮชที่ใช้เก็บค่าข้อมูล

ตามปกติแล้วขนาดของตารางแฮชจะขึ้นอยู่กับความต้องการของผู้ใช้งาน แต่โดยทั่วไปมักมีการกำหนดขนาดตารางแฮชให้มีขนาดใหญ่กว่าจำนวนคีย์ที่มีอยู่ และกำหนดให้มีค่าเป็นจำนวนเฉพาะ (Prime Number) เพื่อไม่ให้เกิดการชนกันของคีย์หรือให้เกิดการชนกันน้อยที่สุดเท่าที่จะทำได้ เช่น ถ้าข้อมูลมี 50 ค่า ก็จะทำการคัดเลือกตัวเลขที่มีค่ามากกว่า 50 คือ 53 มาเป็นตัวหาร ดังนั้นจึงได้ตารางแฮชขนาด 53 ช่องเพื่อใช้รองรับแอดเดรสตั้งแต่ 0 ถึง 52 จากรายละเอียดข้อมูลดังกล่าวสามารถนำเสนอให้เห็นภาพได้ดังรูปที่ 9.5



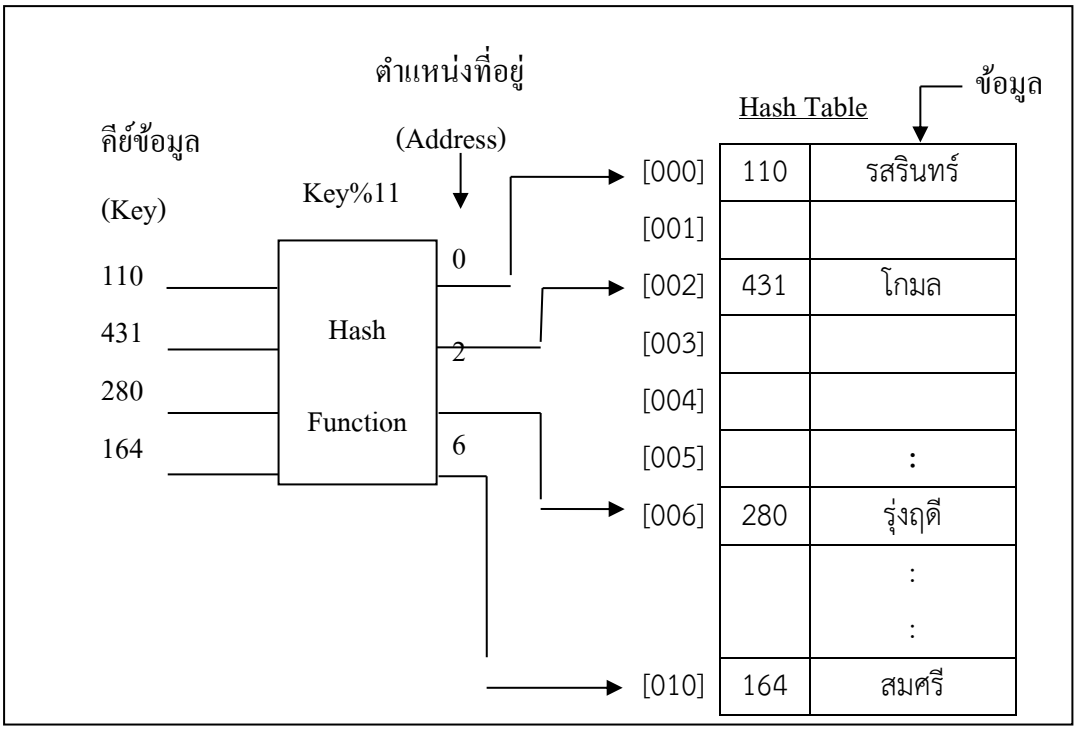
รูปที่ 9.5 ตัวอย่างการทำแฮชชิ่งด้วยวิธีการหาร (Division Hashing)

ตัวอย่างที่ 9.6 การใช้ฟังก์ชั่นแฮชด้วยวิธีการหาร คำนวณหาตำแหน่งที่อยู่ของข้อมูลในตารางแฮช โดยกำหนดให้ N = 11

สมมติว่ากลุ่มข้อมูลที่รับเข้ามา (key) ได้แก่ 110 431 280 164 เมื่อใช้ฟังก์ชั่นแฮช จะได้ผลลัพธ์ของตำแหน่งที่อยู่ของข้อมูลในตารางแฮชดังนี้

ข้อมูล	Division Hashing	Address
110	110 Mod 11	0
431	431 Mod 11	2
280	280 Mod 11	6
164	164 Mod 11	10

สามารถแสดงภาพการจัดเก็บข้อมูลในตารางแฮชได้ดังนี้



รูปที่ 9.6 การจัดเก็บข้อมูลในตารางแฮช

ตัวอย่างที่ 9.7 จงใช้ฟังก์ชั่นแฮชด้วยวิธีการหาร คำนวณหาตำแหน่งที่อยู่ของข้อมูลในตารางแฮช โดยกำหนดขนาดของตารางแฮช (N) เท่ากับ 13 พร้อมทั้งนับจำนวนครั้งของการเกิดการชนกันของคีย์ด้วย

สมมติกลุ่มข้อมูลที่รับเข้ามา (key) มีดังนี้ 32 43 52 14 12 3 7 19 เมื่อใช้ฟังก์ชั่นแฮชจะได้ผลลัพธ์ของตำแหน่งที่อยู่ของข้อมูลในตารางแฮชดังนี้

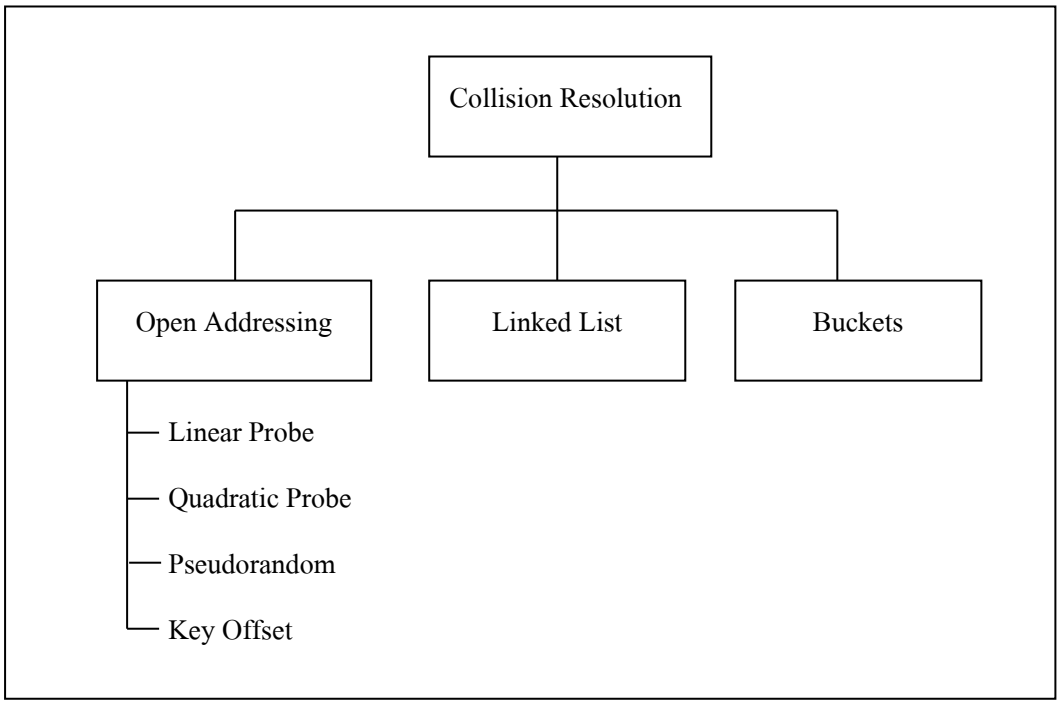
ข้อมูล	Division Hashing	Address
32	32 Mod 13	6
29	43 Mod 13	3
52	52 Mod 13	0
14	14 Mod 13	1
42	12 Mod 13	3
9	3 Mod 13	9
7	7 Mod 13	7
19	19 Mod 13	6

ดังนั้นจากค่าตำแหน่งที่อยู่ชุดนี้จะมีการชนกัน 2 ครั้ง

การแก้ปัญหาการชนกันของคีย์

ข้อเสียประการหนึ่งของการใช้วิธีแฮชชิ่ง คือ เมื่อทำการแปลงคีย์ด้วยฟังก์ชั่นแฮช อาจได้ค่าแอดเดรสเดียวกัน เรียกว่า เกิดการชนกันของคีย์ (Collision) ซึ่งสามารถเกิดขึ้นได้เสมอ ทำให้ไม่สามารถทำการจัดเก็บค่าคีย์นั้นในตำแหน่งแอดเดรสเดียวกันได้ ดังนั้นเพื่อลดปัญหาการชนกันของคีย์จะต้องออกแบบฟังก์ชั่นแฮชที่มีการกระจายตัวของคีย์สม่ำเสมอที่สุด แต่ก็ไม่ได้หมายความว่าจะไม่มีการชนกันเกิดขึ้นอีก ยังคงมีแต่ต้องหาทางหลีกเลี่ยงเพื่อให้เกิดการชนกันน้อยที่สุด นั่นคือเมื่อเกิดการชนกันของคีย์ก็มีวิธีการแก้ปัญหา ซึ่งประกอบด้วยวิธีหลักๆ 3 วิธีด้วยกัน คือ

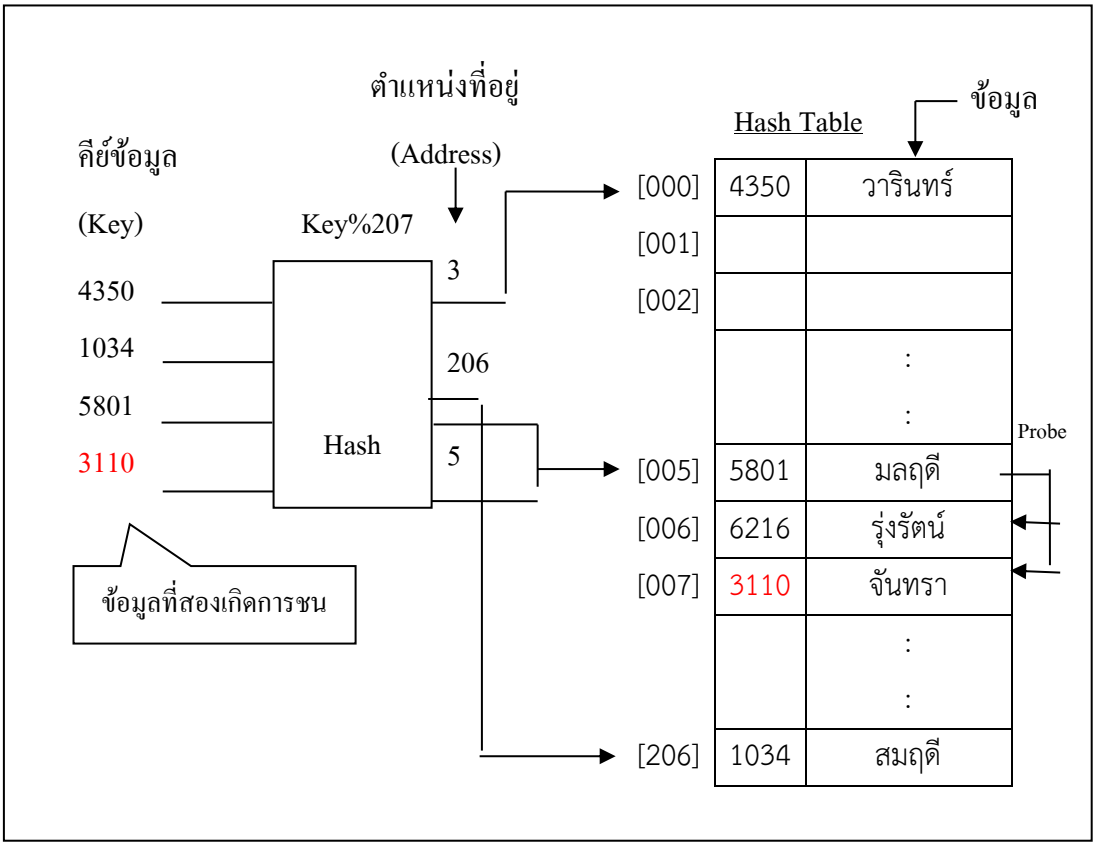
1. วิธี Open Addressing
2. วิธี Linked List
3. วิธี Buckets



จากรูปที่ 9.7 จะเห็นได้ว่าวิธี Open Addressing จะประกอบด้วยหลายวิธีด้วยกัน แต่ในที่นี้จะขอกล่าวเพียงวิธี Linear Probe เท่านั้น

การแก้ไขปัญหาการชนกันของคีย์ด้วยวิธี Linear Probe

การแก้ไขปัญหาการชนกันของคีย์ด้วยวิธี Linear Probe เป็นวิธีที่มีรูปแบบเรียบง่าย โดยเมื่อข้อมูลไม่สามารถจัดเก็บในตำแหน่งแอดเดรสของตนได้ (มีข้อมูลจัดเก็บอยู่แล้ว) ก็จะดำเนินการแก้ไขโดยการนำแอดเดรสปัจจุบันมาบวกเพิ่มอีกหนึ่งตำแหน่ง ให้พิจารณารูปที่ 9.8



รูปที่ 9.8 การแก้ปัญหาการชนกันของคีย์ด้วยวิธี Linear Probe

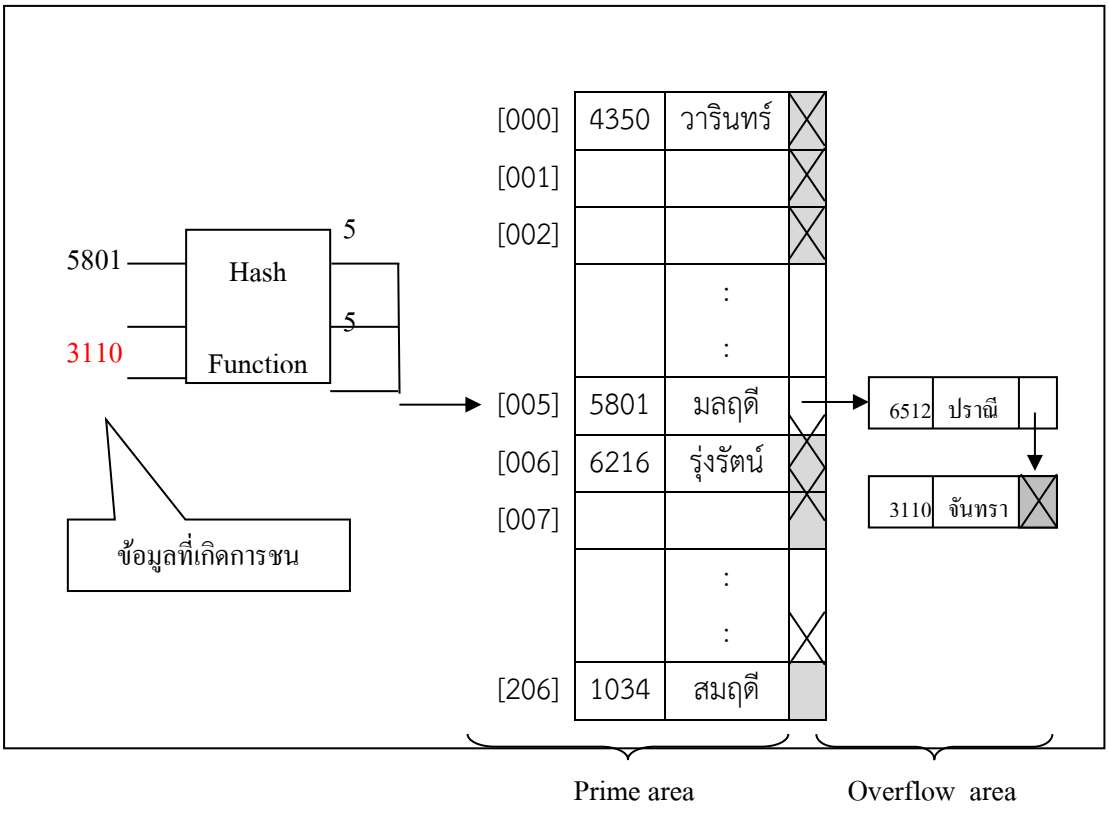
จากรูปที่ 9.8 เมื่อมีการนำค่าคีย์ข้อมูล 5801 มาผ่านฟังก์ชั่นแฮชจะได้แอดเดรสที่ 5 ซึ่งแอดเดรสนี้ยังว่างอยู่สามารถจัดเก็บคีย์ข้อมูลลงไปในตารางแฮชได้ แต่เมื่อรับค่าคีย์ข้อมูล 3110 เข้ามาผ่านฟังก์ชั่นแฮชจะได้แอดเดรสที่ 5 เช่นกัน นั่นหมายความว่าได้เกิดเหตุการณ์การชนกันของคีย์ขึ้นแล้ว จึงต้องทำการแก้ไข คือ บวกตำแหน่งแอดเดรสเพิ่มอีกหนึ่งตำแหน่ง (Probe) เพื่อหาตำแหน่งแอดเดรสใหม่ หลังจากบวกแล้วได้แอดเดรสที่ 6 แต่แอดเดรสที่ 6 ก็ไม่ว่างเนื่องจากมีข้อมูลอยู่แล้ว ดังนั้นจึงต้องทำการ Probe ในรอบที่สองด้วยการบวกเพิ่มอีกหนึ่งตำแหน่งเพื่อหาแอดเดรสที่ว่างถัดไป ผลปรากฏว่าแอดเดรสที่ 7 ว่าง ดังนั้นจึงทำการจัดเก็บค่าคีย์ 3110 ไว้ในตำแหน่งนี้

สำหรับการแก้ไขปัญหาการชนกันของคีย์ด้วยวิธี Linear Probe มีข้อดีอยู่ 2 ประการ คือ ประการแรกเป็นวิธีที่เรียบง่าย ทำให้ง่ายต่อการพัฒนาเพื่อใช้งาน และประการที่สองคือ ข้อมูลที่

แทรกเข้าไปใหม่ (กรณีคีย์ชนกัน) จะอยู่ใกล้ตำแหน่งแอดเดรสจริงของตัวมันเอง แต่สำหรับข้อเสียคือ เริ่มแรกฟังก์ชั่นแฮชจะทำให้การกระจายตัวของข้อมูลในตารางแฮชสม่ำเสมอ แต่เมื่อมีการชนกันและแทรกข้อมูลลงในที่ว่างถัดมาสักระยะหนึ่ง จะทำให้การกระจายของข้อมูลเกิดความไม่สม่ำเสมอ เนื่องจากเกิดการกระจุกตัวของข้อมูลเป็นกลุ่มก้อน (Clustering) หากเกิดกรณีนี้มากขึ้นๆ ประสิทธิภาพของการค้นหาจะลดลง

การแก้ไขปัญหาการชนกันของคีย์ด้วยวิธี Linked List

การแก้ไขปัญหาการชนกันของคีย์ด้วยวิธี Linked List หรืออาจเรียกว่า วิธี Chaining เป็นวิธีการเก็บข้อมูลที่มีตำแหน่งที่อยู่เดียวกันไว้ในลิงค์ลิสต์เรียงต่อกันไปเรื่อยๆ โดยเมื่อคีย์ได้ผ่านฟังก์ชั่นแฮชแล้วเกิดการชนกันของตำแหน่งแอดเดรสในตารางแฮช ก็จะสร้างลิงค์ลิสต์ขึ้นมาใหม่เพื่อเก็บข้อมูลที่มาชนไว้ ณ ตำแหน่งที่อยู่เดียวกันนั้น รูปแบบการเก็บข้อมูลสามารถแสดงได้ดังรูปที่ 9.9



รูปที่ 9.9 การแก้ปัญหาการชนกันของคีย์ด้วยวิธี Linked List

จากรูปที่ 9.9 จะเห็นว่าคีย์ข้อมูล 5801 3110 และ 6512 ถูกจัดเก็บในแอดเดรสเดียวกันคือ 005 และวิธีแก้ไขปัญหาการชนกันของคีย์ด้วยวิธี Linked List นั้น จะมีการใช้พื้นที่เพื่อจัดเก็บข้อมูลอยู่สองส่วน คือ ส่วนแรกเป็น **พื้นที่หลัก** (Prime area) ซึ่งเป็นพื้นที่ปกติที่ต้องมีอยู่แล้ว

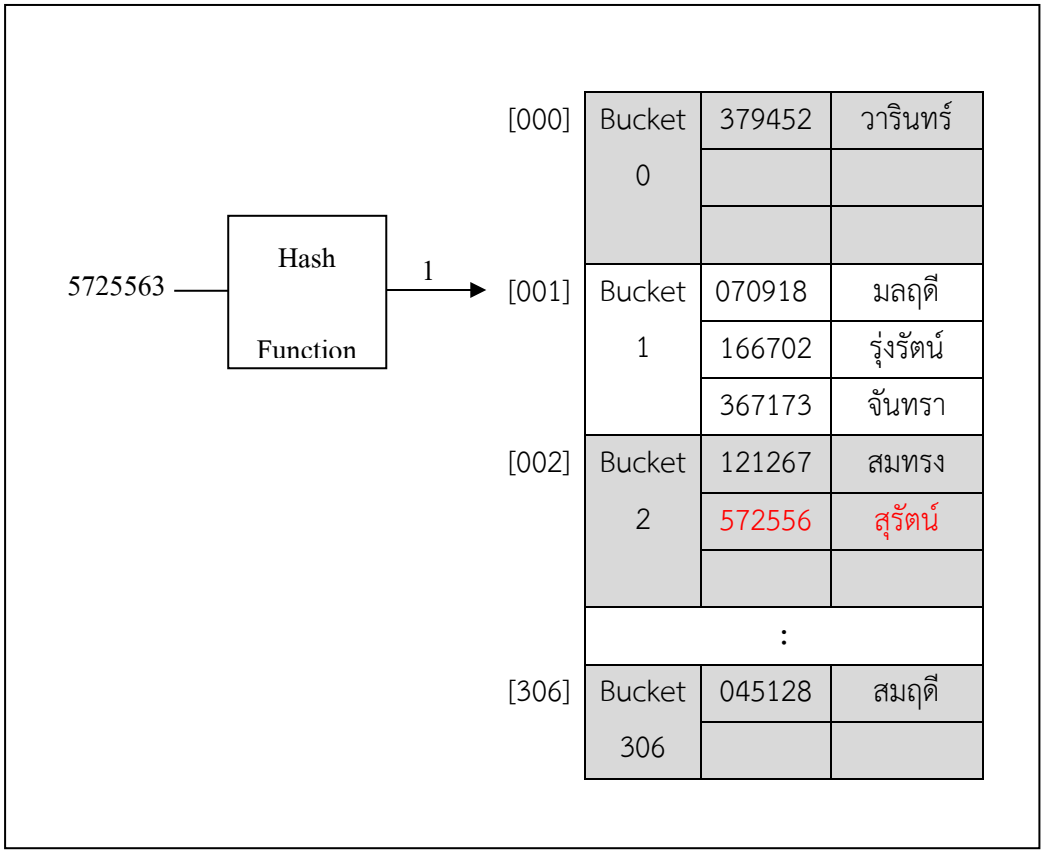
ใช้สำหรับเก็บตำแหน่งข้อมูลทั้งหมด และพื้นที่อีกส่วนหนึ่งที่เพิ่มเข้ามา คือ **พื้นที่โอเวอร์โฟลว์** (Overflow) สำหรับเก็บข้อมูลที่มาชน ดังนั้นพื้นที่หลักจึงมีการจัดเก็บฟิลด์เพิ่มอีกหนึ่งฟิลด์ เรียกว่า

เฮดพอยน์เตอร์ สำหรับเชื่อมโยงไปยังข้อมูลในส่วนโอเวอร์โฟลว์ในกรณีที่เกิดการชนกันของคีย์

การเรียงลำดับข้อมูลตรงส่วนพื้นที่โอเวอร์โฟล์ว จะเรียงลำดับตามลำดับการเพิ่มข้อมูลเข้ามาก่อน-หลัง โดยจะเพิ่มข้อมูลที่เข้ามาชนใหม่ทางด้านหน้าของลิงค์ลิสต์ ซึ่งเรียกว่าเป็นลำดับแบบ LIFO (Last-In, First-Out) คือ เข้าทีหลังออกก่อน เนื่องจากสามารถทำให้การทำงานเร็วและง่าย นั่นคือ ถ้ามีข้อมูลใหม่มาชนก็เพิ่มใส่ด้านหน้าของลิงค์ลิสต์ได้เลย ไม่ต้องเสียเวลาในการท่องเข้าไปในลิงค์ลิสต์แล้วค่อยเพิ่มที่ท้ายลิงค์ลิสต์ เพราะในการค้นหาข้อมูลก็ต้องค้นหาในลิงค์ลิสต์จนกว่าจะพบข้อมูลอยู่แล้ว

การแก้ไขปัญหาการชนกันของคีย์ด้วยวิธี Buckets

การแก้ไขปัญหาการชนกันของคีย์ด้วยวิธี Buckets นั้นเมื่อคีย์ได้ถูกจัดเก็บลงใน Bucket ที่เปรียบเสมือนตะกร้าในตารางแฮชแล้ว คีย์ที่ชนกันยังสามารถจัดเก็บลงในตารางแฮชร่วมกันภายในตะกร้าเหล่านั้นได้อีก เนื่องจากมีการจัดสรรตำแหน่งที่เก็บข้อมูลในรูปแบบของตารางหลายช่องและหากมีการชนกันของคีย์อีกก็จะจัดเก็บลงในตารางตำแหน่งถัดไปจนกระทั่งเต็ม และเมื่อแต่ละ Bucket เต็มแต่ข้อมูลยังเหลือก็จะย้ายลงไปยัง Bucket ต่อไปที่ยังว่างอยู่ ดังรูปที่ 9.10



[000]	Bucket 0	379452	วารินทร์
[001]	Bucket 1	070918	มลฤดี
		166702	รุ่งรัตน์
		367173	จันทรา
[002]	Bucket 2	121267	สมทรง
		572556	สุรัตน์
	:		
[306]	Bucket 306	045128	สมฤดี

รูปที่ 9.10 การแก้ปัญหาการชนกันของคีย์ด้วยวิธี Buckets

จากรูปที่ 9.10 คีย์ 572556 เมื่อผ่านฟังก์ชั่นแฮชจากสูตร 572556 % 307 ผลลัพธ์ที่ได้คือแอดเดรสที่ 1 และเนื่องจากแอดเดรสที่ 1 มีการบรรจุข้อมูลจนเต็มตะกร้าแล้ว การแก้ไขปัญหาก็จะใช้วิธี Linear Probe เข้ามาช่วยด้วยการค้นหาตำแหน่งถัดไปใน Bucket 2 ผลลัพธ์ที่ได้คือสามารถแทนที่คีย์ดังกล่าวลงในแอดเดรสลำดับที่ 2 ของ Bucket 2 ได้

ต่อไปนี้เป็นตารางเปรียบเทียบประสิทธิภาพของการค้นหาทั้ง 3 รูปแบบ

รูปแบบการค้นหา	กรณีดีที่สุด	กรณีเฉลี่ย	กรณีแย่ที่สุด
การค้นหาแบบลำดับ	O(1)	O(N)	O(N)
การค้นหาแบบไบนารี	O(1)	$O(\log_2 N)$	$O(\log_2 N)$
การค้นหาแบบแฮช	O(1)	O(1)	O(N)

แบบฝึกหัดหน่วยที่ 9

ตอนที่ 1 จงตอบคำถามต่อไปนี้

1. จงอธิบายหลักการค้นหาข้อมูลมาให้เข้าใจ
2. จงอธิบายหลักการและแสดงขั้นตอนการค้นหาข้อมูลแบบลำดับ
3. จงอธิบายหลักการและแสดงขั้นตอนการค้นหาข้อมูลแบบไบนารี
4. จงอธิบายหลักการทำงานของการค้นหาแบบแฮช
5. ฟังก์ชั่นแฮช (Hash Function) คืออะไร
6. ฟังก์ชั่นแฮชที่ดีควรมีลักษณะอย่างไร
6. การชนกันของคีย์ (Collision) คืออะไร
7. จงอธิบายการแก้ไขปัญหาการชนกันของคีย์ด้วยวิธี Linear Probe Linked List และ Buckets มา
   ให้เข้าใจ
8. มีข้อมูลจัดเก็บในอาร์เรย์ดังนี้

5	8	14	29	40	59	66	78	85	93

หากค่า Target ที่ต้องการหาคือ 59 ให้แสดงขั้นตอนการค้นหาแบบไบนารี

9. จากชุดข้อมูลที่กำหนดให้ต่อไปนี้ 25 98 30 66 49 14 78
   9.1 จงแสดงการค้นหาคีย์ 49 แบบลำดับ
   9.2 จงแสดงการค้นหาคีย์ 30 แบบไบนารี
   9.3 จงสร้างตารางแฮชที่เก็บค่าได้ 7 ค่าโดยใช้วิธีการหาร และแก้ไขปัญหาการชนกันของคีย์ด้วยวิธี Linear Probe

ตอนที่ 2 จงอธิบายศัพท์ที่เกี่ยวข้องกับการค้นหาข้อมูล ดังต่อไปนี้

1. Successful.....
   .....
2. Unsuccessful.....
   .....
3. Sequential Search.....
   .....
4. Target.....
   .....
5. Location Wanted.....
   .....
6. Hash Table.....
   .....
7. Hashing.....
   .....
8. Hash Function.....
   .....
9. Collision.....
   .....
10. Linear Probe.....
    .....

กิจกรรมฝึกทักษะหน่วยที่ 9

จุดประสงค์

1. เพื่อให้มีความรู้ความเข้าใจหลักการค้นหาข้อมูลแบบต่างๆ
2. ปฏิบัติการค้นหาข้อมูลแบบลำดับ แบบไบนารี และแบบแฮชได้
3. ปฏิบัติการแก้ไขปัญหาการชนกันของคีย์ได้
4. ทำงานร่วมกันตามกลุ่มที่ได้รับมอบหมายด้วยตนเอง และตรงต่อเวลา

กิจกรรม

1. ให้นักศึกษาแบ่งกลุ่มๆ ละ 5 คน
2. ให้แต่ละกลุ่มสนทนาเชิงปฏิบัติการในเรื่องต่อไปนี้
   2.1 การค้นหาแบบลำดับ
   2.2 การค้นหาแบบไบนารี
   2.3 การค้นหาแบบแฮช
   2.4 การแก้ไขปัญหาการชนกันของคีย์
3. สรุปผลการสนทนาเชิงปฏิบัติการลงในกระดาษ A4 และนำเสนอด้วยโปรแกรมเพาเวอร์พอยน์ บริหารเวลากลุ่มละ 10 นาที