แฮชชิงและมอดูโลแฮชชิงฟังก์ชัน (Hashing and Modulo hashing function)

เสนอ

อาจารย์อาจารี นาโค

โดย

นางสาวกิ่งพลอย แซ่ยาง เลขประจำตัวนิสิต 652021042

นายจิรศักดิ์ แว่นนาค เลขประจำตัวนิสิต 652021044

นายธราดล ทองสุทธิ์ เลขประจำตัวนิสิต 652021054

นางสาวรินรดา หวานดี เลขประจำตัวนิสิต 652021067

นายวีรวัฒน์ อินปลอด เลขประจำตัวนิสิต 652021069

รายงานฉบับนี้เป็นส่วนหนึ่งของวิชา 0214212 โครงสร้างข้อมูลและอัลกอริทึม

หลักสูตรวิทยาศาสตรบัณฑิต สาขาวิชาวิทยาการคอมพิวเตอร์

คณะวิทยาศาสตร์และนวัตกรรมดิจิทัล มหาวิทยาลัยทักษิณ

ภาคเรียนที่ 1 ปีการศึกษา 2566

คำนำ

รายงานฉบับนี้จัดทำขึ้นเพื่อเป็นส่วนหนึ่งของวิชา 0214241 โครงสร้างข้อมูลและขั้นตอนวิธี เพื่อให้ได้ศึกษาหาความรู้ในเรื่องราวของแฮชชิงและมอดูโลแฮชชิงฟังก์ชัน (Hashing and Modulo hashing function)โดยได้ศึกษาผ่านแหล่งเรียนรู้จากเว็บไซต์ต่างๆ โดยรายงานฉบับนี้ต้องมีเนื้อหาเกี่ยวกับ แนวคิดของเนื้อหา อัลกอริทึมที่ใช้ ประสิทธิภาพของอัลกอริทึม โจทย์ตัวอย่างและวิธีทำ ของการเรียงข้อมูลแบบรวม

คณะผู้จัดทำหวังเป็นอย่างยิ่งว่าการจัดทำเอกสารฉบับนี้จะมีประโยชน์ต่อผู้ที่สนใจศึกษา เรื่องแฮชชิงและมอดูโลแฮชชิงฟังก์ชัน (Hashing and Modulo hashing function)เป็นอย่างดี

คณะผู้จัดทำ

สารบัญ

เรื่อง	**หน้าที่**
Hashing	1-2
หลักการพื้นฐานของแฮชชิ่ง	2-4
ฟังก์ชันแฮช	4-7
การแก้ปัญหาการชนกันของคีย์	7-11
Source Code	12-21
อ้างอิง	22

Hashing

แฮชชิง (Hashing) คือกระบวนการที่ใช้สำหรับแปลงข้อมูล (เช่น ข้อความ รหัส) ใด ๆ เป็นข้อมูลชุดหมายเลขความยาวคงที่ที่เรียกว่า "แฮช" (hash) โดยทั่วไปแล้วแฮชนั้นมีความยาวคงที่ หรือจำนวนบิตที่คงที่ ที่จะไม่เปลี่ยนแปลงตามขนาดข้อมูลของข้อมูลต้นฉบับ การแฮชชิงมักถูกใช้ในการเก็บข้อมูลแบบเรียงลำดับเพื่อให้เราสามารถค้นหาข้อมูลอย่างมีประสิทธิภาพ โดยเฉพาะในโครงสร้างข้อมูลที่มีการใช้แฮชแบบเรียงลำดับ เช่น ตารางแฮช (hash table) หรือแฮชแมพ (hash map) ซึ่งช่วยในการค้นหาข้อมูลได้อย่างรวดเร็ว.

มอดูโลแฮชชิงฟังก์ชัน (Modulo Hashing Function) เป็นหนึ่งในวิธีการแฮชชิงที่ง่ายที่สุด แม่แบบของมอดูโลแฮชชิงฟังก์ชันคือการใช้การหารเอาเศษ (modulo) ของค่าแฮชด้วยขนาดของตารางแฮช เพื่อให้ได้หมายเลขแฮชที่อยู่ในช่วงของตารางแฮช ยกตัวอย่างเช่น ถ้าเรามีตารางแฮชขนาด 10 และเราต้องการแฮชข้อความ "Hello" เราสามารถใช้มอดูโลแฮชชิงฟังก์ชันดังนี้:

1. นับจำนวนอักขระในข้อความ "Hello" และนับค่ารวมของรหัส ASCII ของแต่ละอักขระ (ค่ารวม ASCII ของ "Hello" = 72 + 101 + 108 + 108 + 111 = 500).

2. ใช้การหารเอาเศษ (modulo) 500 ด้วยขนาดของตารางแฮช (ในที่นี้คือ 10) ดังนั้น 500 % 10 = 0.

3. หมายเลขแฮชที่ได้คือ 0 ซึ่งหมายถึงข้อมูล "Hello" จะถูกจัดเก็บในช่องที่ 0 ของตารางแฮช.

มอดูโลแฮชชิงฟังก์ชันนี้มีข้อจำกัด เช่น การแบ่งแฮชไม่สามารถแยกแยะข้อมูลที่แฮชเป็นค่าเท่ากันได้ และมีโอกาสเกิดการชนแฮช (collision) ซึ่งหมายความว่าสองข้อมูลอาจจะมีค่าแฮชเท่ากัน และต้องใช้การจัดการข้อมูลที่ชนแฮชอย่างเหมาะสมในกรณีนี้ เช่น ใช้รายการเชื่อมต่อ (linked list) หรือเทคนิคการแก้ปัญหาการชนแฮชอื่น ๆ แบบ open addressing หรือ double hashing.

การค้นหาแบบแฮช เป็นวิธีการค้นหาข้อมูลโดยที่ใช้การแปลงคีย์ (Key) ให้เป็นตำแหน่ง (Address) ที่อยู่ในพื้นที่เก็บข้อมูล โดยใช้เทคนิคการสร้างตารางมาเพื่อเก็บคีย์ดังกล่าว ในการแปลงคีย์ให้เป็นแอดเดรส คือการแปลงข้อมูลให้อยู่ในตารางแอดเดรสที่เตรียมไว้ซึ่งตารางนี้เรียกว่า ตารางแฮช (Hash Table)

หลักการของ Hashing search

เป็นวิธีการค้นหาข้อมูลที่ใช้การแปลงคีย์ (Key) ให้เป็นตำแหน่ง (Address) ที่อยู่ในพื้นที่เก็บข้อมูล โดยใช้เทคนิคการสร้างตารางมาเพื่อเก็บคีย์ดังกล่าว

1.การแปลงคีย์ให้เป็นแอดเดรส คือการแปลงข้อมูลให้ไปอยู่ในตารางแอดเดรสที่เตรียมไว้ซึ่งเรียตารางนี้ว่า ตารางแฮช (Hash Table)

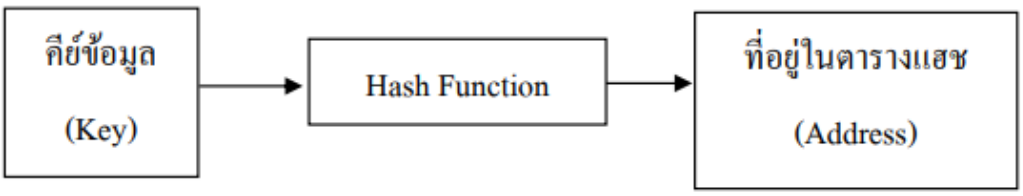
2.การแปลงค่านี้ต้องอาศัยฟังก์ชัน H(k) เป็นตัวช่วยในการหาแอดเรดสของค่าคีย์ k (ค่า H(k) คือแอดเรดสของคีย์ k นั้นเอง)

3.การค้นหาด้วยวิธีนี้จะไม่ขึ้นอยู่กับจำนวนข้อมูล

4.อาศัยหลักการคำนวณตำแหน่งที่เก็บข้อมูลจากคีย์ที่กำหนด นั้นคือจะต้องหา Hashing Function ที่เหมาะสม

หลักการพื้นฐานของแฮชชิ่ง

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



Hash(key) = key MOD TableSize

TableSize คือ ขนาดของตารางที่จัดเก็บ ควรเลือกค่าที่เป็นจำนวนเฉพาะ (Prime Numbers)

ตัวอย่าง

ชุดตัวเลข 9,17,22,14,13,5 กำหนดแฮชชิ่งฟังก์ชันด้วยสมการ H(K) = K MOD 9

ค่าที่ได้จากแฮชชิ่งฟังก์ชันแสดงได้ดังนี้

H(9) = 8 H(17) = 8 H(22) = 4

H(14) = 5 H(13) = 4 H(8) = 8

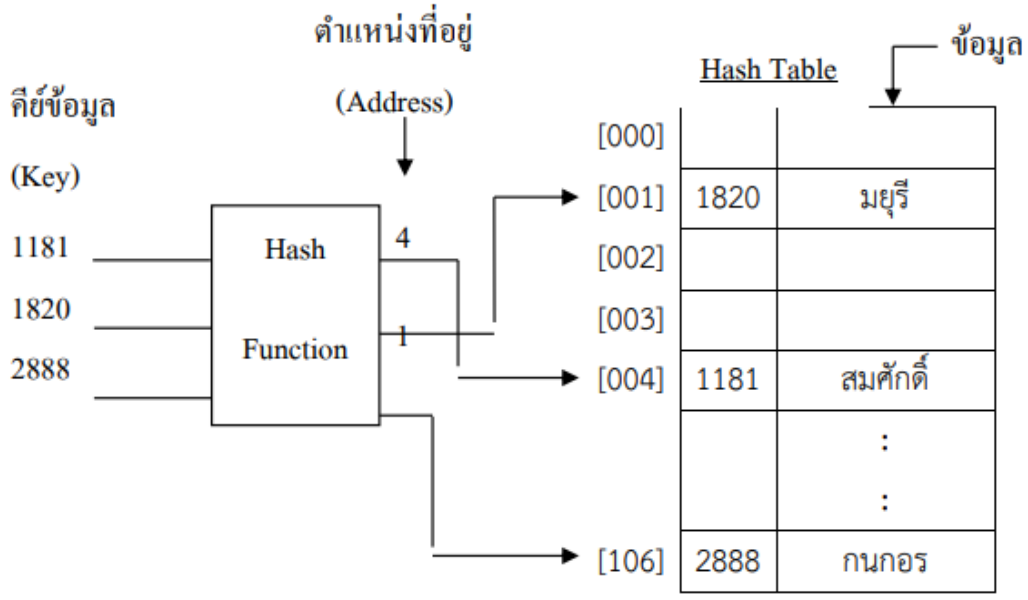
คีย์ 17 ได้ตำแหน่ง แอดเดรสตรงกับคีย์ 8

คีย์ 22 ได้ตำแหน่ง แอดเดรสตรงกับคีย์ 13

ส่วนประสิทธิภาพของการแฮชชิ่ง จะดีหรือไม่นั้นขึ้นอยู่กับฟังก์ชั่นแฮชที่ใช้เป็นหลัก ซึ่งลักษณะของฟังก์ชั่นแฮชที่ดีคือ ต้องสามารถกระจายคีย์ข้อมูลที่ป้อนเข้ามาให้ไปอยู่ในตำแหน่งที่อยู่ที่ไม่เกิดการชนกันเลยในตารางแฮช หรือถ้าจำเป็นต้องชนกันก็ให้เกิดการชนกันน้อยครั้งที่สุด

ตัวอย่างการทำแฮชชิ่ง (Hasting) ลงในตารางแฮช (hash Table)

จากรูปเมื่อรับค่าคีย์ข้อมูล 1181 ซึ่งเป็นรหัสพนักงานเข้ามา เมื่อผ่านฟังก์ชันแฮช จะได้ค่าแอดเดรส 4



นั่นคือชี้ไปที่ตารางแฮช 004 ซึ่งเป็นตำแหน่งที่อยู่ที่จะนำข้อมูลไปเก็บ ในที่นี้ทำ การจัดเก็บชื่อ สมศักดิ์ ลงในตาราง ในบางครั้งอาจเกิดปัญหาขึ้นในด้านการแปลงคีย์คืออาจได้ ตำแหน่งแอดเดรสเดียวกัน ซึ่งจะเรียกว่า เกิดการชนกัน (Collision) ดังนั้นเราจะต้องดำเนินการ แก้ปัญหานี้เพื่อให้มีการจัดตำแหน่งที่ถูกต้อง

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 ที่เราสามารถ

เก็บข้อมูลนั้นๆ ลงไปได้

5. การชนกัน (Collision) คือ การนำคีย์ข้อมูล 2 ค่ามาผ่านฟังก์ชั่นแฮช แล้วเกิดได้ค่า

ตำแหน่งที่อยู่เดียวกัน (ซ้ำกัน) หรือมีข้อมูลอื่นเก็บไว้ก่อนอยู่แล้ว ซึ่ง เราจะเรียกเหตุการณ์เช่นนี้ว่า

เกิดการชนกัน ซึ่งเมื่อเกิดเหตุการณ์นี้ขึ้นจะต้องหาวิธีในการจัดการแก้ไขปัญหานี้ เพื่อที่จะหา

ตำแหน่งที่อยู่ใหม่ให้กับข้อมูลตัวที่มาชน

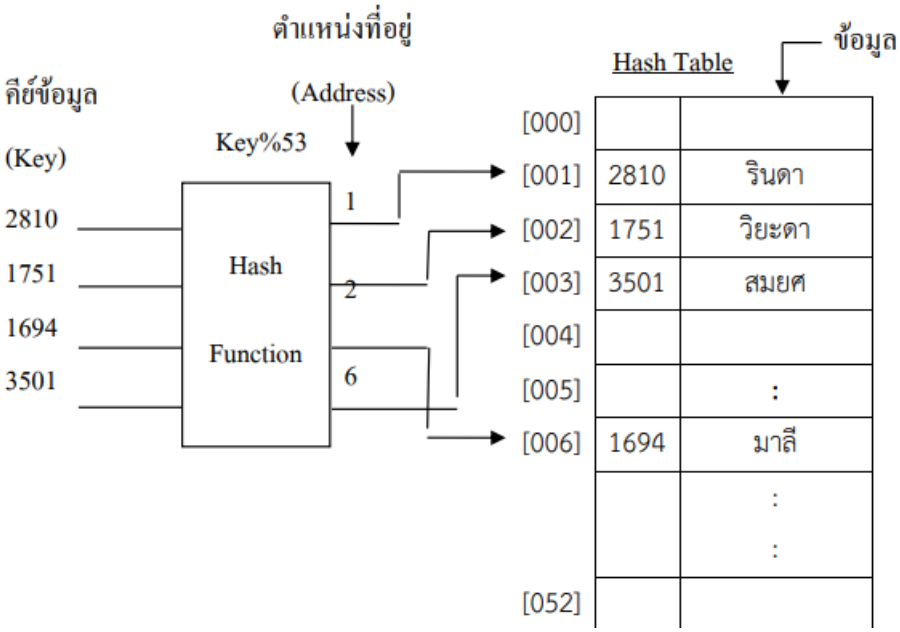
ฟังก์ชันแฮช

ฟังก์ชั่นแฮช (Hash Function) มีด้วยกันหลายรูปแบบ แต่ในที่นี้จะขอกล่าวถึงเพียงรูปแบบเดียว คือ วิธีการหาร (Division Hashing) เนื่องจากเป็นวิธีที่เข้าใจง่ายและเป็นที่นิยมใช้กันอย่างแพร่หลาย วิธีการหาร คือ การหาตำแหน่งที่อยู่ในตารางแฮช โดยการนำค่าคีย์ข้อมูลมาหารแบบ Modolo ด้วยขนาดของตาราง (การหารแบบ Modolo หรือ Mod คือ การหารโดยผลลัพธ์ที่ได้ คือค่าเศษที่เหลือจากการหาร เช่น 5 Mod 2 = 1) โดยมีสูตรการคำนวณ ดังนี้

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

เมื่อ K คือ ค่าคีย์ข้อมูล N คือ ขนาดของตารางแฮชที่ใช้เก็บค่าข้อมูลตามปกติแล้วขนาดของตารางแฮชจะขึ้นอยู่กับความต้องการของผู้ใช้งาน แต่โดยทั่วไปมักมีการกำหนดขนาดตารางแฮชให้มีขนาดใหญ่กว่าจำนวนคีย์ที่มีอยู่ และกำหนดให้มีค่าเป็นจำนวนเฉพาะ (Prime Number) เพื่อไม่ให้เกิดการชนกันของคีย์หรือให้เกิดการชนกันน้อยที่สุดเท่าที่จะทำได้ เช่น ถ้าข้อมูลมี 50 ค่า ก็จะทำการคัดเลือกตัวเลขที่มีค่ามากกว่า 50 คือ 53 มาเป็นตัวหาร ดังนั้นจึงได้ตารางแฮชขนาด 53 ช่องเพื่อใช้รองรับแอดเดรสตั้งแต่ 0 ถึง 52 จากรายละเอียดข้อมูลดังกล่าว สามารถนำเสนอให้เห็นภาพได้ดังรูป

ตัวอย่างการทำแฮชชิ่งด้วยวิธีการหาร (Division Hashing)



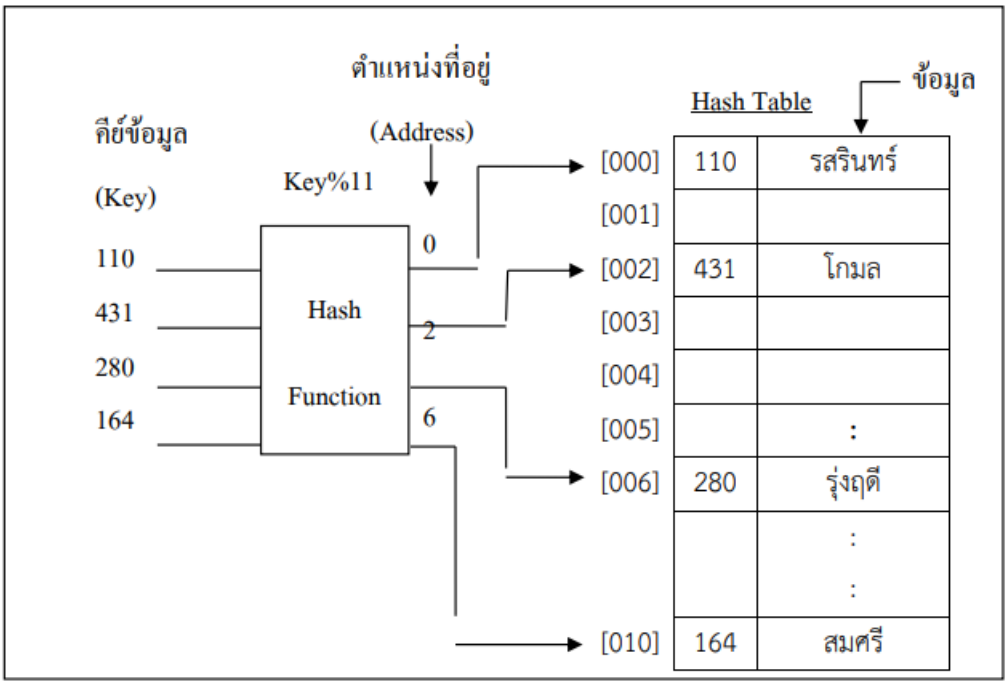
<u>ตัวอย่าง</u> การใช้ฟังก์ชั่นแฮชด้วยวิธีการหาร คำนวณหาตำแหน่งที่อยู่ของข้อมูลในตารางแฮช

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



ตัวอย่าง จงใช้ฟังก์ชั่นแฮชด้วยวิธีการหาร คำนวณหาตำแหน่งที่อยู่ของข้อมูลในตารางแฮช โดยกำหนดขนาดของตารางแฮช (N) เท่ากับ 13 พร้อมทั้งนับจำนวนครั้งของการเกิดการชนกันของคีย์ด้วย

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

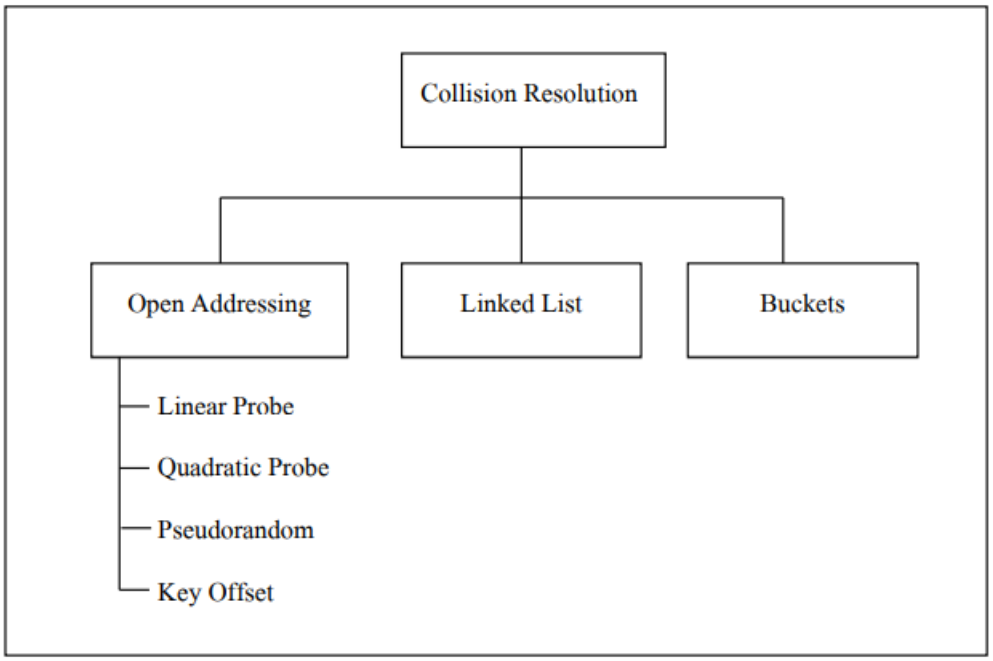
ข้อมูล	Division Hashing	Address
32	32 Mod 13	6
29	43 Mod 13	3
52	52 Mod 13	0
14	14 Mod 13	1
42	12 Mod 13	3
9	3 Mod 13	9
7	7 Mod 13	7
19	19 Mod 13	6

ดังนั้นจากค่าตำแหน่งที่อยู่ชุดนี้จะมีการชนกัน 2 ครั้ง

การแก้ปัญหาการชนกันของคีย์

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

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



จากรูป จะเห็นได้ว่าวิธี Open Addressing จะประกอบด้วยหลายวิธีด้วยกัน แต่ในที่นี้จะขอกล่าวเพียงวิธี Linear Probe เท่านั้น

การแก้ปัญหาการชนกันของคีย์แบบ Open Addressingด้วยวิธีลิเนียร์โพรบ (Linear Probe)

ข้อดี

-เป็นวิธีที่ง่าย

-ข้อมูลที่แทรกเข้าไปใหม่ (กรณีคีย์ชนกัน) จะอยู่ใกล้ตำแหน่งแอดเดรสจริงของตัวมันเอง

ข้อเสีย

-ฟังก์ชันแฮชที่ดี จะต้องออกแบบให้คีย์มีการกระจายสม่ำเสมอ เพื่อลดการชนกันของคีย์ให้มากที่สุด

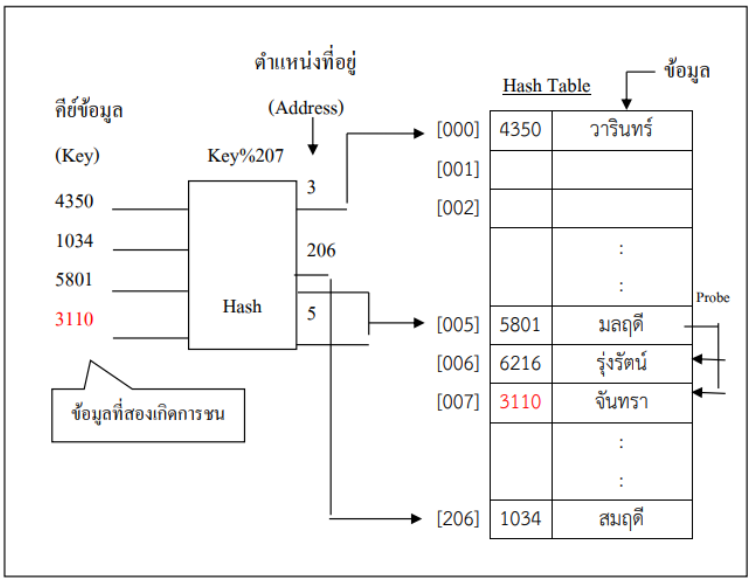
-เมื่อเกิดการชนกันของคีย์ ก็จะมีความพยายามแทรกข้อมูลในตำแหน่งว่างถัดไปจากแอดเดรสจริง ส่งผลให้เกิดการรวมกลุ่มของข้อมูลมากขึ้น

-ส่งผลต่อการกระจายคีย์ในตารางแฮชสะดุดลง (คีย์ชนกันมากขึ้น)

-ประสิทธิภาพการค้นหาข้อมูลลดลงไป

การแก้ไขปัญหาการชนกันของคีย์ด้วยวิธีLinear Probe

การแก้ไขปัญหาการชนกันของคีย์ด้วยวิธีLinear Probe เป็นวิธีที่มีรูปแบบเรียบง่าย โดยเมื่อข้อมูลไม่สามารถจัดเก็บในตำแหน่งแอดเดรสของตนได้ (มีข้อมูลจัดเก็บอยู่แล้ว) ก็จะดำเนินการแก้ไขโดยการนำแอดเดรสปัจจุบันมาบวกเพิ่มอีกหนึ่งตำแหน่ง ให้พิจารณารูป



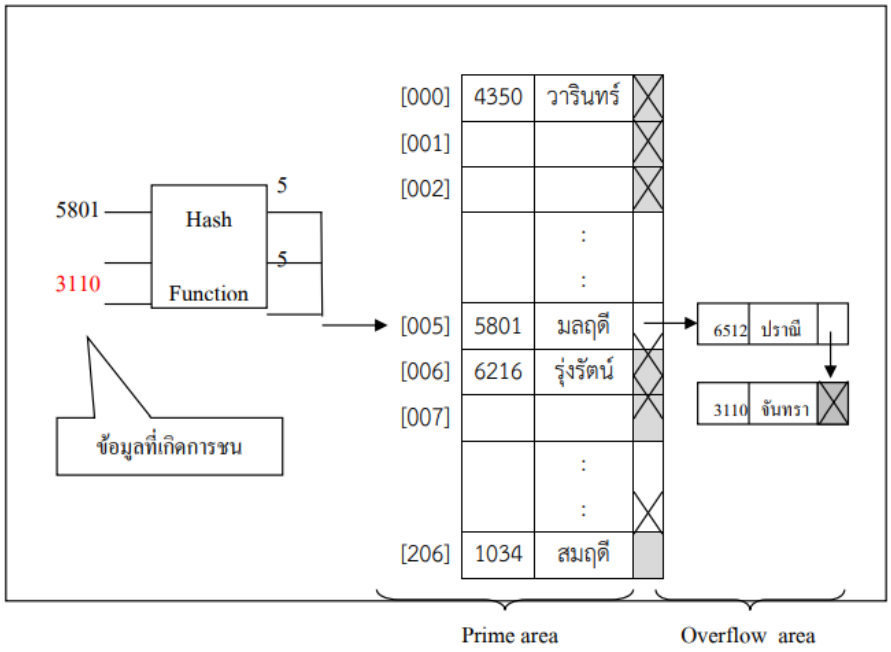
จากรูป เมื่อมีการนำค่าคีย์ข้อมูล 5801 มาผ่านฟังก์ชั่นแฮชจะได้แอดเดรสที่ 5 ซึ่งแอดเดรสนี้ยังว่างอยู่สามารถจัดเก็บคีย์ข้อมูลลงไปในตารางแฮชได้ แต่เมื่อรับค่าคีย์ข้อมูล 3110 เข้ามาผ่านฟังก์ชั่นแฮชจะได้แอดเดรสที่ 5 เช่นกัน นั่นหมายความว่าได้เกิดเหตุการณ์การชนกันของคีย์ขึ้นแล้ว จึงต้องทำการแก้ไข คือ บวกตำแหน่งแอดเดรสเพิ่มอีกหนึ่งตำแหน่ง (Probe) เพื่อหาตำแหน่งแอดเดรสใหม่ หลังจากบวกแล้วได้แอดเดรสที่ 6 แต่แอดเดรสที่ 6 ก็ไม่ว่างเนื่องจากมีข้อมูลอยู่แล้ว ดังนั้นจึงต้องทำการ Probe ในรอบที่สองด้วยการบวกเพิ่มอีกหนึ่งตำแหน่งเพื่อหาแอดเดรสที่ว่างถัดไป ผลปรากฏว่าแอดเดรสที่ 7 ว่าง ดังนั้นจึงทำการจัดเก็บค่าคีย์3110 ไว้ในตำแหน่งนี้

สำหรับการแก้ไขปัญหาการชนกันของคีย์ด้วยวิธี Linear Probe มีข้อดีอยู่ 2 ประการ คือ ประการแรกเป็นวิธีที่เรียบง่าย ทำให้ง่ายต่อการพัฒนาเพื่อใช้งาน และประการที่สองคือ ข้อมูลที่แทรกเข้าไปใหม่ (กรณีคีย์ชนกัน) จะอยู่ใกล้ตำแหน่งแอดเดรสจริงของตัวมันเอง แต่สำหรับข้อเสียคือเริ่มแรกฟังก์ชั่นแฮชจะทำให้การกระจายตัวของข้อมูลในตารางแฮชสม่ำเสมอ แต่เมื่อมีการชนกันและแทรกข้อมูลลงในที่ว่างถัดมาสักระยะหนึ่งจะทำให้การกระจายของข้อมูลเกิดความไม่สม่ำเสมอเนื่องจากเกิดการกระจุกตัวของข้อมูลเป็นกลุ่มก้อน (Clustering) หากเกิดกรณีนี้มากขึ้นๆประสิทธิภาพของการค้นหาจะลดลง

การแก้ไขปัญหาการชนกันของคีย์ด้วยวิธีLinked List

การแก้ไขปัญหาการชนกันของคีย์ด้วยวิธี Linked List หรืออาจเรียกว่า วิธี Chaining เป็นวิธีการเก็บข้อมูลที่มีตำแหน่งที่อยู่เดียวกันไว้ในลิงค์ลิสต์เรียงต่อกันไปเรื่อยๆ โดยเมื่อคีย์ได้ผ่านฟังก์ชั่นแฮชแล้วเกิดการชน

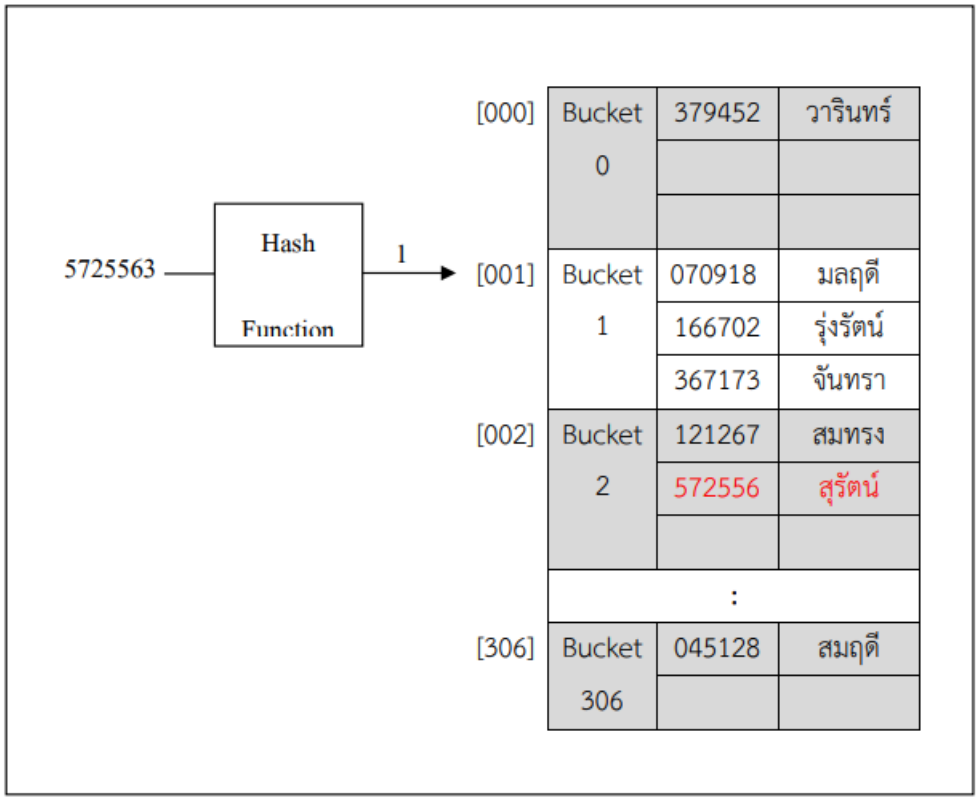
กันของตำแหน่งแอดเดรสในตารางแฮช ก็จะสร้างลิงค์ลิสต์ขึ้นมาใหม่เพื่อเก็บข้อมูลที่มาชนไว้ ณ ตำแหน่งที่อยู่เดียวกันนั้น รูปแบบการเก็บข้อมูลสามารถแสดงได้ดังรูป



จากรูป จะเห็นว่าคีย์ข้อมูล 5801 3110 และ 6512 ถูกจัดเก็บในแอดเดรสเดียวกันคือ 005 และวิธีแก้ไขปัญหาการชนกันของคีย์ด้วยวิธี Linked List นั้น จะมีการใช้พื้นที่เพื่อจัดเก็บข้อมูลอยู่สองส่วน คือ ส่วนแรกเป็น พื้นที่หลัก (Prime area) ซึ่งเป็นพื้นที่ปกติที่ต้องมีอยู่แล้ว ใช้สำหรับเก็บตำแหน่งข้อมูลทั้งหมด และพื้นที่อีกส่วนหนึ่งที่เพิ่มเข้ามา คือ พื้นที่โอเวอร์โฟลว์(Overflow) สำหรับเก็บข้อมูลที่มาชน ดังนั้นพื้นที่หลักจึงมีการจัดเก็บฟิลด์เพิ่มอีกหนึ่งฟิลด์เรียกว่า เฮดพอยน์เตอร์ สำหรับเชื่อมโยงไปยังข้อมูลในส่วนโอเวอร์โฟลว์ในกรณีที่เกิดการชนกันของคีย์

การแก้ไขปัญหาการชนกันของคีย์ด้วยวิธีBuckets

การแก้ไขปัญหาการชนกันของคีย์ด้วยวิธี Buckets นั้นเมื่อคีย์ได้ถูกจัดเก็บลงใน Bucketที่เปรียบเสมือนตะกร้าในตารางแฮชแล้ว คีย์ที่ชนกันยังสามารถจัดเก็บลงในตารางแฮชร่วมกันภายในตะกร้าเหล่านั้นได้อีก เนื่องจากมีการจัดสรรตำแหน่งที่เก็บข้อมูลในรูปแบบของตารางหลายช่องและหากมีการชนกันของคีย์อีกก็จะจัดเก็บลงในตารางตำแหน่งถัดไปจนกระทั่งเต็ม และเมื่อแต่ละ Bucket เต็มแต่ข้อมูลยังเหลือก็จะย้ายลงไปยัง Bucket ต่อไปที่ยังว่างอยู่ ดังรูป



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

Source Code 1

```c
#include <stdio.h>
  #include <string.h>
  #include <stdlib.h>

  struct hash *hashTable = NULL;
  int eleCount = 0;

  struct node {
        int key, age;
        char name[100];
        struct node *next;
  };

  struct hash {
        struct node *head;
        int count;
  };

  struct node * createNode(int key, char *name, int age) {
        struct node *newnode;
        newnode = (struct node *)malloc(sizeof(struct node));
        newnode->key = key;
        newnode->age = age;
        strcpy(newnode->name, name);
        newnode->next = NULL;
        return newnode;
  }


  void insertToHash(int key, char *name, int age) {
        int hashIndex = key % eleCount;
        struct node *newnode =  createNode(key, name, age);
        /* head of list for the bucket with index "hashIndex" */
        if (!hashTable[hashIndex].head) {
              hashTable[hashIndex].head = newnode;
              hashTable[hashIndex].count = 1;
              return;
        }
        /* adding new node to the list */
        newnode->next = (hashTable[hashIndex].head);
        /*
         * update the head of the list and no of
         * nodes in the current bucket
         */
        hashTable[hashIndex].head = newnode;
        hashTable[hashIndex].count++;
        return;
  }


```

บรรทัดที่ 5-27 คือฟังก์ชันการทำงานตามเงื่องไข้ของ Hashinh และจัดตำแหน่งของค่า kay หารด้วยจำนวน Table

บรรทัดที่ 5-27 คือฟังก์ชันการทำงานในการรับค่า kay

```
51  void deleteFromHash(int key) {
52      /* find the bucket using hash index */
53      int hashIndex = key % eleCount, flag = 0;
54      struct node *temp, *myNode;
55      /* get the list head from current bucket */
56      myNode = hashTable[hashIndex].head;
57      if (!myNode) {
58          printf("Given data is not present in hash Table!!\n");
59          return;
60      }
61      temp = myNode;
62      while (myNode != NULL) {
63          /* delete the node with given key */
64          if (myNode->key == key) {
65              flag = 1;
66              if (myNode == hashTable[hashIndex].head)
67                  hashTable[hashIndex].head = myNode->next;
68              else
69                  temp->next = myNode->next;
70
71              hashTable[hashIndex].count--;
72              free(myNode);
73              break;
74          }
75          temp = myNode;
76          myNode = myNode->next;
77      }
78      if (flag)
79          printf("Data deleted successfully from Hash Table\n");
80      else
81          printf("Given data is not present in hash Table!!!!\n");
82      return;
83  }
84
85  void searchInHash(int key) {
86      int hashIndex = key % eleCount, flag = 0;
87      struct node *myNode;
88      myNode = hashTable[hashIndex].head;
89      if (!myNode) {
90          printf("Search element unavailable in hash table\n");
91          return;
92      }
93      while (myNode != NULL) {
```

บรรทัดที่ 51-83 คือฟังก์ชันในการลบ Key ออกจาก Table

```
92          }
93          while (myNode != NULL) {
94                  if (myNode->key == key) {
95                          printf("VoterID  : %d\n", myNode->key);
96                          printf("Name     : %s\n", myNode->name);
97                          printf("Age      : %d\n", myNode->age);
98                          flag = 1;
99                          break;
00                  }
01                  myNode = myNode->next;
02          }
03          if (!flag)
04                  printf("Search element unavailable in hash table\n");
05          return;
06  }
07
08  void display() {
09          struct node *myNode;
10          int i;
11          for (i = 0; i < eleCount; i++) {
12                  if (hashTable[i].count == 0)
13                          continue;
14                  myNode = hashTable[i].head;
15                  if (!myNode)
16                          continue;
17                  printf("\nData at index %d in Hash Table:\n", i);
18                  printf("VoterID     Name           Age   \n");
19                  printf("--------------------------------\n");
20                  while (myNode != NULL) {
21                          printf("%-12d", myNode->key);
22                          printf("%-15s", myNode->name);
23                          printf("%d\n", myNode->age);
24                          myNode = myNode->next;
25                  }
26          }
27          return;
28  }
29
30  int main() {
31          int n, ch, key, age;
32          char name[100];
33          printf("Enter the number of elements:");
34          scanf("%d", &n);
35          eleCount = n;
36          /* create hash table with "n" no of buckets */
37          hashTable = (struct hash *)calloc(n, sizeof (struct hash));
38          while (1) {
```

บรรทัดที่ 85-106 คือฟังก์ชันในการค้นหา Key ใน Table มาแสดงหน้าจอ

บรรทัดที่ 85-106 คือฟังก์ชันในการแสดงข้อมูลใน Table ทั้งหมด

```
132         char name[100];
133         printf("Enter the number of elements:");
134         scanf("%d", &n);
135         eleCount = n;
136         /* create hash table with "n" no of buckets */
137         hashTable = (struct hash *)calloc(n, sizeof (struct hash));
138         while (1) {
139                 printf("\n1. Insertion\t2. Deletion\n");
140                 printf("3. Searching\t4. Display\n5. Exit\n");
141                 printf("Enter your choice:");
142                 scanf("%d", &ch);
143                 switch (ch) {
144                         case 1:
145                                 printf("Enter the key value:");
146                                 scanf("%d", &key);
147                                 getchar();
148                                 printf("Name:");
149                                 fgets(name, 100, stdin);
150                                 name[strlen(name) - 1] = '\0';
151                                 printf("Age:");
152                                 scanf("%d", &age);
153                                 /*inserting new node to hash table */
154                                 insertToHash(key, name, age);
155                                 break;
156
157                         case 2:
158                                 printf("Enter the key to perform deletion:");
159                                 scanf("%d", &key);
160                                 /* delete node with "key" from hash table */
161                                 deleteFromHash(key);
162                                 break;
163
164                         case 3:
165                                 printf("Enter the key to search:");
166                                 scanf("%d", &key);
167                                 searchInHash(key);
168                                 break;
169                         case 4:
170                                 display();
171                                 break;
172                         case 5:
173                                 exit(0);
174                         default:
175                                 printf("U have entered wrong option!!\n");
176                                 break;
177                 }
178         }
179         return 0;
180 }
```

บรรทัดที่ 132-135 คือฟังก์ชันในการรับข้อมูลจำนวน table

บรรทัดที่ 137-180 คือฟังก์ชันในการเรียกใช้งานฟังก์ชั่นต่างๆโดยใช้ while กับ switch case

ผลการรันของโปรแกรม

```
Data at index 0 in Hash Table:
VoterID     Name          Age
--------------------------------
9           lary          30
14          hun           26
3           kill          24

1. Insertion    2. Deletion
3. Searching    4. Display
5. Exit
Enter your choice:2
Enter the key to perform deletion:9
Data deleted successfully from Hash Table

1. Insertion    2. Deletion
3. Searching    4. Display
5. Exit
Enter your choice:4

Data at index 0 in Hash Table:
VoterID     Name          Age
--------------------------------
14          hun           26
3           kill          24

1. Insertion    2. Deletion
3. Searching    4. Display
5. Exit
Enter your choice:3
Enter the key to search:3
VoterID  : 3
Name     : kill
Age      : 24

1. Insertion    2. Deletion
3. Searching    4. Display
5. Exit
Enter your choice:
```

```
Enter the number of elements:1

1. Insertion    2. Deletion
3. Searching    4. Display
5. Exit
Enter your choice:1
Enter the key value:3
Name:kill
Age:24

1. Insertion    2. Deletion
3. Searching    4. Display
5. Exit
Enter your choice:1
Enter the key value:14
Name:hun
Age:26

1. Insertion    2. Deletion
3. Searching    4. Display
5. Exit
Enter your choice:1
Enter the key value:9
Name:lary
Age:30

1. Insertion    2. Deletion
3. Searching    4. Display
5. Exit
Enter your choice:4

Data at index 0 in Hash Table:
VoterID     Name          Age
--------------------------------
9           lary          30
14          hun           26
3           kill          24
```

Source Code 2

```c
#include <stdio.h>
#include <stdlib.h>

struct set
{
  int key;
  int data;
};
struct set *array;
int capacity = 10;
int size = 0;

int hashFunction(int key)
{
  return (key % capacity);
}
int checkPrime(int n)
{
  int i;
  if (n == 1 || n == 0)
  {
  return 0;
  }
  for (i = 2; i < n / 2; i++)
  {
  if (n % i == 0)
  {
    return 0;
  }
  }
  return 1;
}
int getPrime(int n)
{
  if (n % 2 == 0)
  {
  n++;
  }
  while (!checkPrime(n))
  {
  n += 2;
  }
  return n;
}
void init_array()
{
  capacity = getPrime(capacity);
  array = (struct set *)malloc(capacity * sizeof(struct set));
  for (int i = 0; i < capacity; i++)
  {
```

บรรทัดที่ 4-16 คือฟังก์ชันในการเก็บข้อมูลของ hashing

บรรทัดที่ 18-44 คือฟังก์ชันในการจัดตำแหน่งของค่า kay หารด้วยจำนวน Table

```
50    {
51    array[i].key = 0;
52    array[i].data = 0;
53    }
54  }
55
56  void insert(int key, int data)
57  {
58    int index = hashFunction(key);
59    if (array[index].data == 0)
60    {
61    array[index].key = key;
62    array[index].data = data;
63    size++;
64    printf("\n Key (%d) has been inserted \n", key);
65    }
66    else if (array[index].key == key)
67    {
68    array[index].data = data;
69    }
70    else
71    {
72    printf("\n Collision occured  \n");
73    }
74  }
75
76  void remove_element(int key)
77  {
78    int index = hashFunction(key);
79    if (array[index].data == 0)
80    {
81    printf("\n This key does not exist \n");
82    }
83    else
84    {
85    array[index].key = 0;
86    array[index].data = 0;
87    size--;
88    printf("\n Key (%d) has been removed \n", key);
89    }
90  }
91  void display()
92  {
93    int i;
94    for (i = 0; i < capacity; i++)
95    {
96    if (array[i].data == 0)
97    {
```

บรรทัดที่ 45-54 คือฟังก์ชัน array ในจักเก็บและกำหนดจำนวน Table

บรรทัดที่ 56-74 คือฟังก์ชันในการรับของค่า kay มาเก็บใน Table

บรรทัดที่ 76-90 คือฟังก์ชันในการลบค่า kay ออกจาก Table

```
97      {
98        printf("\n array[%d]: / ", i);
99      }
100     else
101     {
102       printf("\n key: %d array[%d]: %d \t", array[i].key, i, array[i].data);
103     }
104     }
105   }
106
107   int size_of_hashtable()
108   {
109     return size;
110   }
111
112   int main()
113   {
114     int choice, key, data, n;
115     int c = 0;
116     init_array();
117
118     do
119     {
120     printf("1.Insert item in the Hash Table"
121        "\n2.Remove item from the Hash Table"
122        "\n3.Check the size of Hash Table"
123        "\n4.Display a Hash Table"
124        "\n\n Please enter your choice: ");
125
126     scanf("%d", &choice);
127     switch (choice)
128     {
129     case 1:
130
131       printf("Enter key -:\t");
132       scanf("%d", &key);
133       printf("Enter data -:\t");
134       scanf("%d", &data);
135       insert(key, data);
136
137       break;
138
139     case 2:
140
141       printf("Enter the key to delete-:");
142       scanf("%d", &key);
143       remove_element(key);
144
145       break;
```

บรรทัดที่ 91-150 คือฟังก์ชันในการแสดงค่า kay ทั้งหมดใน Table

```
128 {
129 case 1:
130
131   printf("Enter key -:\t");
132   scanf("%d", &key);
133   printf("Enter data -:\t");
134   scanf("%d", &data);
135   insert(key, data);
136
137   break;
138
139 case 2:
140
141   printf("Enter the key to delete-:");
142   scanf("%d", &key);
143   remove_element(key);
144
145   break;
146
147 case 3:
148
149   n = size_of_hashtable();
150   printf("Size of Hash Table is-:%d\n", n);
151
152   break;
153
154 case 4:
155
156   display();
157
158   break;
159
160 default:
161
162   printf("Invalid Input\n");
163 }
164
165 printf("\nDo you want to continue (press 1 for yes): ");
166 scanf("%d", &c);
167
168 } while (c == 1);
169 }
```

บรรทัดที่ 112-169 คือฟังก์ชันในการเรียกใช้งานฟังก์ชั่นต่างๆโดยใช้ do while กับ switch case

ผลการรันของโปรแกรม

```
1.Insert item in the Hash Table
2.Remove item from the Hash Table
3.Check the size of Hash Table
4.Display a Hash Table

 Please enter your choice: 1
Enter key -:    1
Enter data -:   23

 Key (1) has been inserted

Do you want to continue (press 1 for yes): 1
1.Insert item in the Hash Table
2.Remove item from the Hash Table
3.Check the size of Hash Table
4.Display a Hash Table

 Please enter your choice: 1
Enter key -:    2
Enter data -:   22

 Key (2) has been inserted

Do you want to continue (press 1 for yes): 1
1.Insert item in the Hash Table
2.Remove item from the Hash Table
3.Check the size of Hash Table
4.Display a Hash Table

 Please enter your choice: 4

 array[0]: /
 key: 1 array[1]: 23
 key: 2 array[2]: 22
 array[3]: /
 array[4]: /
 array[5]: /
 array[6]: /
 array[7]: /
 array[8]: /
 array[9]: /
 array[10]: /
Do you want to continue (press 1 for yes): 1
1.Insert item in the Hash Table
2.Remove item from the Hash Table
3.Check the size of Hash Table
4.Display a Hash Table
```

```
Do you want to continue (press 1 for yes): 1
1.Insert item in the Hash Table
2.Remove item from the Hash Table
3.Check the size of Hash Table
4.Display a Hash Table

 Please enter your choice: 3
Size of Hash Table is-:2

Do you want to continue (press 1 for yes): 1
1.Insert item in the Hash Table
2.Remove item from the Hash Table
3.Check the size of Hash Table
4.Display a Hash Table

 Please enter your choice: 2
Enter the key to delete-:2

 Key (2) has been removed

Do you want to continue (press 1 for yes): 1
1.Insert item in the Hash Table
2.Remove item from the Hash Table
3.Check the size of Hash Table
4.Display a Hash Table

 Please enter your choice: 4

 array[0]: /
 key: 1 array[1]: 23
 array[2]: /
 array[3]: /
 array[4]: /
 array[5]: /
 array[6]: /
 array[7]: /
 array[8]: /
 array[9]: /
 array[10]: /
Do you want to continue (press 1 for yes):
```

อ้างอิง

1.การค้นหาข้อมูล (Searching)

ลิงค์ : https://ajpra.files.wordpress.com/2011/11/e0b89ae0b897e0b897e0b8b5e0b988-9.pdf

2.การค้นหาข้อมูล

ลิงค์ : http://ctc.chontech.ac.th/files/1909071111525751_22070513133234.pdf

3. Hashing

ลิงค์ : http://pioneer.netserv.chula.ac.th/~sperapho/files/class/263/ch5.pdf

4.การค้นหาแบบแฮช (Hashing search)

ลิงค์ : https://www.mindphp.com/%E0%B8%9A%E0%B8%97%E0%B8%84%E0%B8%A7%E0%B8%B2%E0%B8%A1/31-%E0%B8%84%E0%B8%A7%E0%B8%B2%E0%B8%A1%E0%B8%A3%E0%B8%B9%E0%B9%89%E0%B8%97%E0%B8%B1%E0%B9%88%E0%B8%A7%E0%B9%84%E0%B8%9B/7042-hashing-search.html

5. Source code

ลิงค์ที่1 : https://see-programming.blogspot.com/2013/05/chain-hashing-separate-chaining-with.html?m=1

ลิงค์ที่2 : https://www.programiz.com/dsa/hash-table