บทที่ 3

แนวคิดและทฤษฎีสำคัญ

พื้นฐาน 2 อย่างที่ผู้ศึกษาต้องทำการเรียนรู้ในการทำโปรแกรมคอมพิวเตอร์คือ

- โครงสร้างข้อมูล (Data Structure)
- อัลกอริทึม (Algorithm)

โครงสร้างข้อมูล

โครงสร้างข้อมูล (Data Structure) หมายถึง รูปแบบของการจัดระเบียบของข้อมูล ซึ่งอาจจะจัดให้อยู่ได้ในหลายรูปแบบที่สำคัญได้แก่ โครงสร้างแบบแถวลำดับ แบบกองซ้อน แบบแถวคอย (Queue) แบบบัญชีรายการเชื่อมโยง (Link List) และโครงสร้างแบบต้นไม้ (Tree)

อัลกอริทึม

อัลกอริทึม (Algorithm) เป็นขั้นตอนหรือวิธีการในการแก้ไขปัญหาหนึ่งๆ ซึ่งมีหนึ่งหรือหลายแบบของอัลกอริทึมที่จะนำมาใช้ เช่น ถ้าต้องการเรียงลำดับข้อมูลก็จะมีอัลกอริทึมเรียงลำดับข้อมูลให้เลือกใช้ได้หลายวิธีเช่น อัลกอริทึมเรียงลำดับข้อมูลแบบเลือก (Selection Sort) แบบแทรก (Insertion Sort) แบบฟอง (Bubble Sort) หรือแบบเร็ว (Quick Sort) เป็นต้น ในส่วนโครงการวิจัยนี้ได้ทำการศึกษาและพัฒนาระบบจินตทัศน์อัลกอริทึมค้นหาข้อมูล และการหาที่อยู่แบบแฮชเพื่อให้เกิดความรู้ความเข้าใจในอัลกอริทึมทั้งสองจะขอกล่าวถึงหลักการทำงานของทั้งอัลกอริทึมค้นหาข้อมูล และการหาที่อยู่แบบแฮชดังนี้

อัลกอริทึมค้นหาข้อมูล

ค้นหาข้อมูลมีความสำคัญมากอย่างหนึ่งในการใช้คอมพิวเตอร์ เป็นการนำส่วนหนึ่งของสารสนเทศที่สนใจจากสารสนเทศที่จัดเก็บอยู่ภายในคอมพิวเตอร์ซึ่งมีอยู่เป็นจำนวนมาก (Sedgewick, 1992) โดยปรกติสารสนเทศจะถูกแบ่งออกแฟ้มข้อมูล (File) ในแต่ละแฟ้มข้อมูลจะจัดเก็บเป็นระเบียน (Record) ค้นหาข้อมูลจะต้องมีการกำหนดเขตข้อมูล (Field) เป็นส่วนที่ใช้ในค้นหาข้อมูลซึ่งจะเรียกเขตข้อมูลนี้ว่าเขตข้อมูลหลัก (Key Field) โดยอาจเป็นกลุ่มตัวเลขหรือตัวอักษรก็ได้ และการค้นหาก็คือการหาว่าเขตข้อมูลหลักที่กำหนดขึ้นนี้มีตรงกับเขตข้อมูลหลักของระเบียนต่างๆที่อยู่ในแฟ้มข้อมูลหรือไม่ หากมีอยู่ที่ใด ระบบที่ใช้ค้นหาข้อมูลมีอยู่เป็นจำนวนมากและมีการใช้งานที่แตกต่างกันไป เช่นการธนาคารมีการจัดเก็บรายการเคลื่อนไหวของลูกค้า

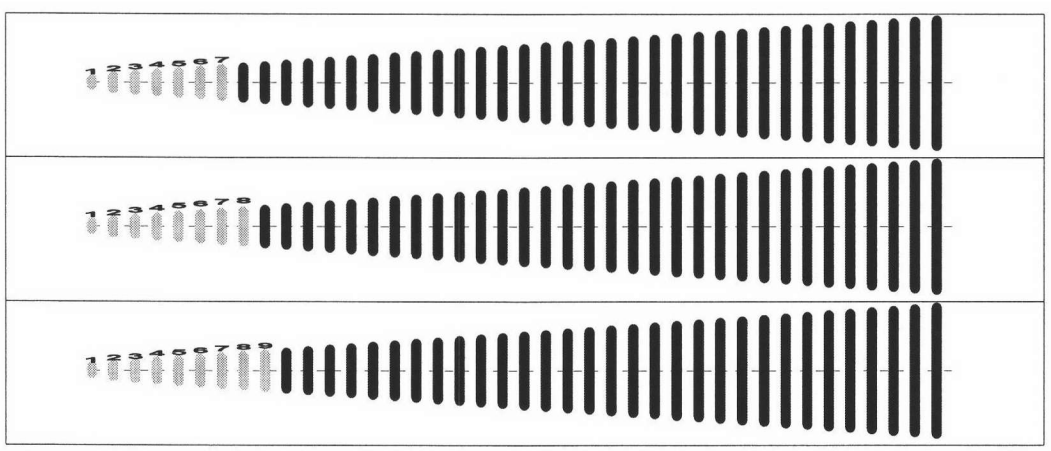
และใช้รหัสลูกค้าในค้นหาข้อมูล บริษัทต่างๆมีการจัดเก็บประวัติพนักงานโดยมีรหัสพนังงานเป็นเขตข้อมูลหลัก หรือสมุดโทรศัพท์จะมีชื่อเป็นเขตข้อมูลหลักในการหาหมายเลขโทรศัพท์ ในค้นหาข้อมูลให้มีประสิทธิภาพจะมีปัจจัยที่ต้องคำนึงถึง 2 ประการคือ

- ข้อมูลมีการจัดเรียงในแบบใด
- อัลกอริทึมที่ถูกเลือกให้ใช้ในค้นหาข้อมูล

เทคนิคที่ใช้ในค้นหาข้อมูลที่ต้องการมีหลายวิธี โครงการวิจัยนี้จะนำเสนอ 3 แบบคือ

1. ค้นหาข้อมูลแบบลำดับ (Sequential Search), (Sedgewick, 1992)

เป็นวิธีที่ง่ายในการค้นหาโดยจะทำการกำหนดเขตข้อมูลหลักที่ต้องการค้นหา นำเขตข้อมูลนี้มาเปรียบเทียบกับเขตข้อมูลหลักของแฟ้มข้อมูลตั้งแต่ระเบียนแรกไปทีละระเบียนเรียงลำดับกันไปจนพบระเบียนที่มีเขตข้อมูลหลักเท่ากับเขตข้อมูลหลักที่ต้องการค้นหา ก็แสดงว่าพบระเบียนที่ต้องการ แต่ถ้าเปรียบเทียบไปจนหมดแฟ้มข้อมูลแล้วแสดงว่าไม่พบ ซึ่งในส่วนค้นหาที่ไม่พบเขตข้อมูลหลักที่ต้องการจะมีได้ใน 2 ลักษณะขึ้นอยู่กับลักษณะข้อมูลคือเมื่อข้อมูลมีการจัดเรียงแบบไม่เรียงลำดับต้องทำค้นหาข้อมูลไปจนหมดข้อมูล แต่ถ้าข้อมูลมีการจัดเรียงลำดับก็จะค้นหาถึงตำแหน่งที่เขตข้อมูลหลักในแฟ้มข้อมูลมีค่ามากกว่าเขตข้อมูลหลักที่ต้องการเท่านั้น รูปที่ 3.1 แต่ละแท่งแสดงข้อมูลแต่ละตัวโดยใช้ความสูงของแท่งแทนค่าข้อมูล เมื่อเริ่มต้นทุกแท่งจะมีสีดำเมื่อการค้นหาเริ่มดำเนินการ จะตรวจสอบไปทีละตำแหน่งเริ่มที่ตำแหน่งที่ 1, 2, 3,...,n (n=จำนวนข้อมูลทั้งหมด) ขณะที่ตรวจสอบแล้วพบว่าไม่ใช่ค่าข้อมูลที่ต้องการก็จะเปลี่ยนสีของแท่งข้อมูลเป็นสีเทา ตรวจสอบไปจนกว่าจะพบเขตข้อมูลที่ต้องการหรือจนกว่าแท่งข้อมูลมีค่ามากกว่าเขตข้อมูลที่ต้องการแสดงว่าไม่มีเขตข้อมูลที่ต้องการหา โดยสามารถแสดงเป็นโปรแกรมคอมพิวเตอร์ได้ดังรูปที่ 3.2



รูปที่ 3.1 แสดงภาพการค้นหาแบบลำดับ

```
1)    Sub Alg_Sequential_Search ()
2)        Dim i  As Integer
3)        For i = 1 To NumRnd
4)            nCompare = nCompare + 1
5)            If Data_Ord(i) = KeySearch Then
6)                Found_Pos = i
7)                Exit For
8)            End If
9)        Next i
10)   End Sub
```

รูปที่ 3.2 แสดงโปรแกรมการค้นหาแบบลำดับ

เมื่อกำหนดให้ NumRnd = จำนวนข้อมูลทั้งหมด
nCompare = จำนวนการเปรียบเทียบทั้งหมด
Found_Pos = ตำแหน่งที่พบเขตข้อมูลที่ต้องการ

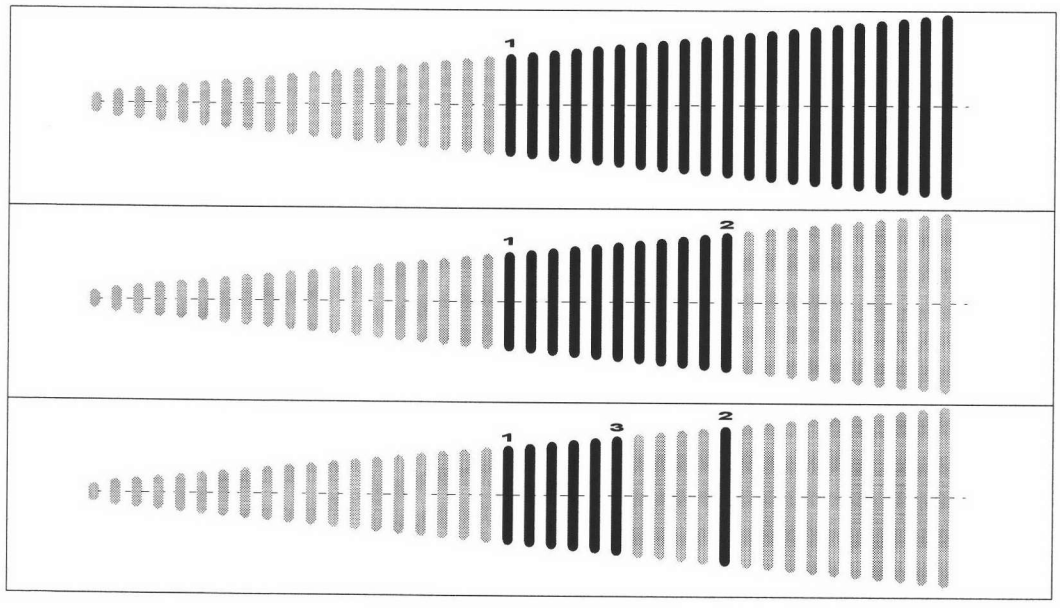
ในการทำงานของอัลกอริทึมค้นหาข้อมูลแบบลำดับเมื่อข้อมูลไม่มีการจัดเรียงลำดับไม่อาจคาดเดาได้ว่าจะต้องการทำการเปรียบเทียบกี่ครั้งจึงจะพบเขตข้อมูลหลักที่ต้องการที่ต้องการ เพราะถ้าข้อมูลมี n ตัว อาจมีการเปรียบเทียบเพียงครั้งเดียวถ้าข้อมูลที่ต้องการอยู่ที่ตำแหน่งแรก หรือ n ครั้งคือข้อมูลอยู่ตัวสุดท้าย หรือ n+1 เมื่อไม่พบเขตข้อมูลหลักที่ต้องการ ดังนั้นจำนวนการเปรียบเทียบโดยเฉลี่ยของการพบเขตข้อมูลหลักที่ต้องการได้เท่ากับ (n+1)/2 ส่วนไม่พบเขตข้อมูลหลักที่ต้องการจะได้เท่ากับ n (Tenenbaum, Langsam, Augenstein, 1990)

แต่ถ้าข้อมูลมีการจัดเรียงลำดับเมื่อทำการตรวจสอบมาถึงตำแหน่งที่เขตข้อมูลหลักมีค่ามากกว่าเขตข้อมูลหลักที่ต้องการเมื่อเรียงจากน้อยไปมาก หรือถึงตำแหน่งที่เขตข้อมูลหลักมีค่าน้อยกว่าเขตข้อมูลหลักที่ต้องการเมื่อเรียงจากมากไปน้อย ก็สามารถจบการทำงานของอัลกอริทึมได้เพราะไม่พบเขตข้อมูลหลักที่ต้องการที่ต้องการแล้ว ดังนั้นจำนวนการเปรียบเทียบโดยเฉลี่ยของการพบและไม่พบจะได้เท่ากับ n/2 (Sedgewick, 1992)

2. ค้นหาข้อมูลแบบทวิภาค (Binary Search), (Sedgewick, 1992)

ค้นหาข้อมูลแบบลำดับนั้นมีวิธีการโปรแกรมที่ง่ายและเหมาะสำหรับข้อมูลที่มีจำนวนไม่มาก แต่ถ้าข้อมูลมีจำนวนมากวิธีดังกล่าวจะไม่เหมาะสมเพราะจะมีประสิทธิภาพที่ต่ำมาก เช่น ถ้าต้องการค้นหาชื่อนายสมชายในสมุดโทรศัพท์ก็จะต้องค้นหาเรียงตั้งแต่ตัว ก, ข, ... จนถึง ส แล้วค้นหาชื่อสมชาย วิธีหนึ่งที่ลดจำนวนครั้งของการค้นหาได้ก็โดยการหาตำแหน่งตรงกลางของข้อมูลทั้งหมดแล้วตรวจสอบดูว่าค่าเขตข้อมูลหลักที่ต้องการมีค่ามากกว่าหรือน้อยกว่าค่า ณ

ตำแหน่งกลาง ซึ่งจะเป็นการลดจำนวนข้อมูลที่ทดสอบลงครึ่งหนึ่งในทุกๆรอบการทำงานตามรูปที่ 3.3 รอบที่หนึ่งแบ่งข้อมูลที่ตำแหน่งกึ่งกลาง นำค่าที่ตำแหน่งนั้นเปรียบเทียบกับค่าเขตข้อมูลหลักที่ต้องการได้ว่าค่าเขตข้อมูลหลักมีค่ามากกว่าค่าที่ตำแหน่งแบ่งครึ่ง ก็จะใช้ข้อมูลถัดจากตำแหน่งแบ่งครึ่งไปถึงตำแหน่งสุดท้ายเป็นข้อมูลที่จะทำการแบ่งครึ่งเพื่อตรวจในรอบที่สอง แล้วทำการแบ่งเช่นนี้ไปจนกว่าจะพบค่าเขตข้อมูลหลักที่ต้องการ ข้อกำหนดอย่างหนึ่งของวิธีนี้คือค่าของข้อมูลต้องจัดอยู่ในลักษณะเรียงลำดับ แสดงเป็นโปรแกรมคอมพิวเตอร์ได้ตามรูปที่ 3.4



รูปที่ 3.3 แสดงภาพการค้นหาแบบทวิภาค

```
1)    Sub Alg_Binary_Search ()
2)        Dim Pos_top, Pos_bottom, Pos_mid, i As Integer
3)        Pos_bottom = 1
4)        Pos_top = NumRnd
5)        Do
6)            Pos_mid = Int((Pos_top + Pos_bottom) / 2)
7)            nCompare = nCompare + 1
8)            If KeySearch = Data_Ord(Pos_mid) Then
9)                Found_Pos = Pos_mid
10)               Exit Do
11)           ElseIf KeySearch < Data_Ord(Pos_mid) Then
12)               Pos_top = Pos_mid - 1
```

```
13)             Else
14)                 Pos_bottom = Pos_mid + 1
15)             End If
16)         Loop Until (KeySearch = Data_Ord(Pos_mid))
                    Or (Pos_top < Pos_bottom)
17)     End Sub
```

รูปที่ 3.4 แสดงโปรแกรมการค้นหาแบบทวิภาค

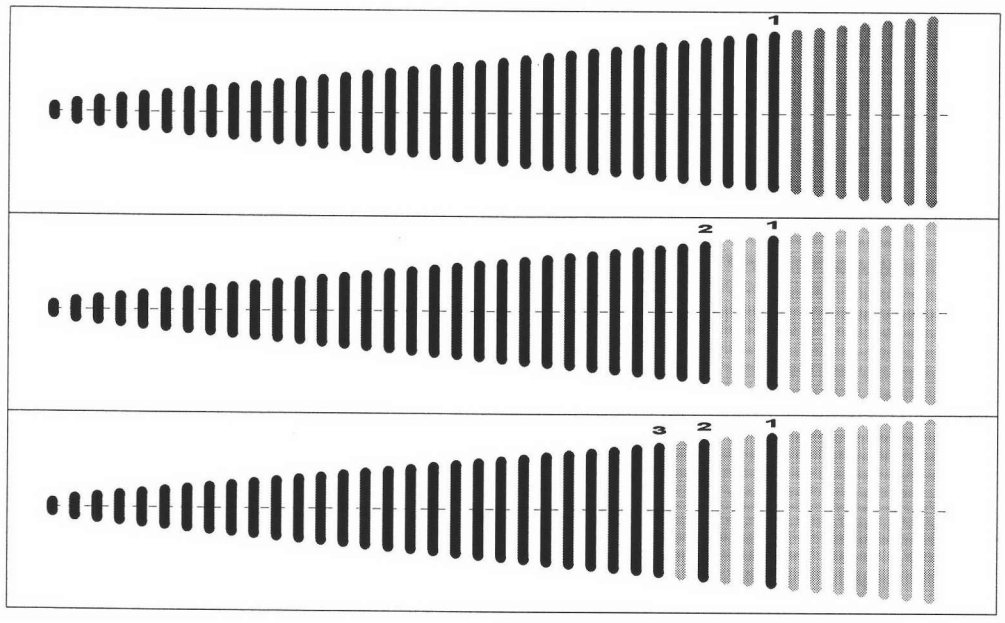
เมื่อกำหนดให้ Pos_top = ตำแหน่งสุดท้ายของข้อมูลที่ใช้ในการค้นหา
Pos_bottom = ตำแหน่งแรกของข้อมูลที่ใช้ในการค้นหา
Pos_mid = ตำแหน่งกลางของข้อมูลที่ใช้เปรียบเทียบกับเขตข้อมูลที่ต้องการ

จากโปรแกรมคอมพิวเตอร์สรุปได้ดังนี้

- กำหนดเขตข้อมูลหลักที่ต้องการค้นหา
- แบ่งครึ่งแฟ้มข้อมูล
- นำเขตข้อมูลหลักที่ต้องการเปรียบเทียบกับเขตข้อมูลหลักในแฟ้มข้อมูล ณ ตำแหน่งแบ่งครึ่ง
- ถ้าเขตข้อมูลในแฟ้มข้อมูลมีค่าน้อยกว่า ทำการค้นหาโดยแบ่งครึ่งข้อมูลใหม่แต่ใช้เขตข้อมูลครึ่งค่ามาก ทำไปเรื่อยๆจนพบเขตข้อมูลที่ต้องการหรือไม่พบ จะได้จำนวนครั้งการเปรียบเทียบเฉลี่ยของการพบเขตข้อมูลที่ต้องการหรือไม่พบจะมีค่าเท่ากับ $\lg n + 1$ (Sedgewick, 1992)

3. ค้นหาข้อมูลแบบประมาณค่า (Interpolation Search), (Sedgewick, 1992)

ถ้าข้อมูลมีการจัดเรียงลำดับและมีการกระจายของค่าข้อมูล แทนที่จะค้นหาข้อมูลในแบบทวิภาคที่แบ่งครึ่งจำนวนข้อมูลไปเรื่อยๆจนกว่าจะพบ หรืออยู่นอกเหนือข้อกำหนด ก็ใช้การค้นหาในแบบประมาณค่าแทน ซึ่งป็นการกะประมาณว่าค่าเขตข้อมูลหลักที่ต้องการควรอยู่ช่วงตำแหน่งใดของข้อมูลทั้งหมด เช่นการหาชื่อในสมุดโทรศัพท์ถ้าชื่อที่ต้องการคือ "กัญญา" ควรอยู่ในช่วงต้นๆ แต่ถ้าเป็นชื่อ "อรอนงค์" ก็ควรอยู่ตอนท้ายๆ แสดงให้เห็นวิธีการทำได้จากรูปที่ 3.5 และโปรแกรมคอมพิวเตอร์ในรูปที่ 3.6



รูปที่ 3.5 แสดงภาพการค้นหาแบบประมาณค่า

```
01)  Sub Alg_Interpolation_Search ()
02)      Dim Pos_bottom, Pos_top, Pos_mid As Integer
03)      Pos_bottom = 1
04)      Pos_top = NumRnd
05)      Do
06)          If   KeySearch < Data_Ord(Pos_bottom) Then
07)              Pos_mid = Pos_bottom + 1 * (Pos_top - Pos_bottom) /
                 (Data_Ord(Pos_top) - Data_Ord(Pos_bottom))
08)          Else
09)              Pos_mid = Pos_bottom + (KeySearch - Data_Ord(Pos_bottom))
                           * (Pos_top - Pos_bottom) / (Data_Ord(Pos_top) -
                                                      Data_Ord(Pos_bottom))
10)      End If
11)      nCompare = nCompare + 1
12)      If   Pos_mid < Pos_bottom Then
13)          Exit Do
14)      End If
15)      If   KeySearch = Data_Ord(Pos_mid) Then
16)          Found_Pos = Pos_mid
```

```
17)             Exit Do
18)         ElseIf KeySearch < Data_Ord(Pos_mid) Then
19)             Pos_top = Pos_mid - 1
20)         ElseIf KeySearch > Data_Ord(Pos_mid) Then
21)             Pos_bottom = Pos_mid + 1
22)         End If
23)     Loop Until (Pos_bottom >= Pos_top) Or (KeySearch = Data_Ord(Pos_mid))
24)     End Sub
```

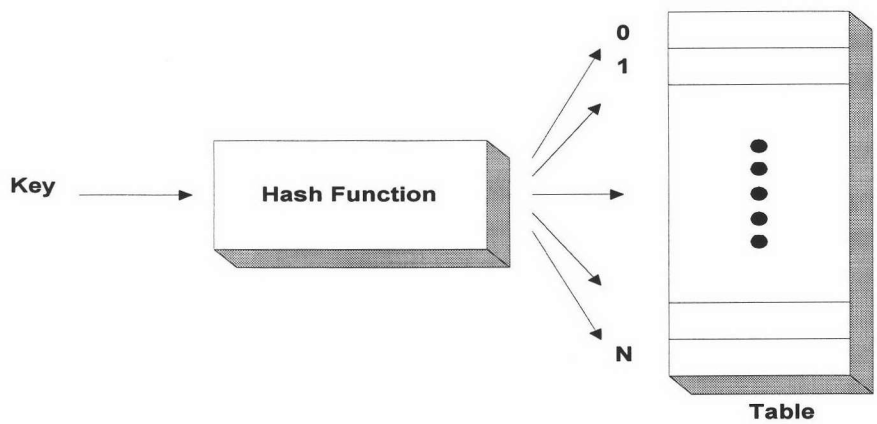
รูปที่ 3.6 แสดงโปรแกรมการค้นหาแบบประมาณค่า

จากรูปที่ 3.5 จะเห็นว่าตำแหน่งที่ใช้ในการตรวจสอบแต่ละรอบจะมีตำแหน่งไม่แน่นอนขึ้นอยู่กับตำแหน่งข้อมูลตัวแรก สุดท้าย และค่าของข้อมูลซึ่งจะนำมาคำนวนด้วยกันเพื่อประมาณตำแหน่งที่จะตรวจสอบ โดยดูได้จากโปรแกรมในบรรทัดที่ 7 และบรรทัดที่ 9 ได้ตำแหน่ง Pos_mid แล้วทำการตรวจสอบกับค่าเขตข้อมูลที่ต้องการถ้าเท่ากันจะได้ตำแหน่งนั้นเป็นตำแหน่งของเขตข้อมูลที่ต้องการ ถ้าค่าเขตข้อมูลที่ต้องการน้อยกว่าค่า ณ ตำแหน่งที่ได้จะกำหนดตำแหน่งข้อมูลตัวสุดท้ายใหม่ตามโปรแกรมคอมพิวเตอร์ในบรรทัดที่ 19 และถ้ามากกว่าจะกำหนดตำแหน่งข้อมูลตัวแรกใหม่ตามบรรทัดที่ 21 ซึ่งในการค้นหาแบบประมาณค่าจะมีจำนวนครั้งการเปรียบเทียบโดยเฉลี่ยของการพบและไม่พบเขตข้อมูลเท่ากับ log(log n) (Tenenbaum, Langsam, Augenstein, 1990)

ในค้นหาข้อมูลในแบบประมาณค่าจะให้ผลที่ดีและมีประสิทธิภาพเมื่อข้อมูลมีลักษณะค่าข้อมูลกระจายอย่างสม่ำเสมอ แต่ถ้าข้อมูลไม่กระจายมีการเกาะกลุ่มของข้อมูลและถ้าการเกาะกลุ่มมีจำนวนมากจะทำให้การค้นหาในแบบประมาณค่าได้ผลลัพธ์ที่ไม่ดีเช่นจากที่จะค้นหาชื่อ “อรอนงค์” ในสมุดโทรศัพท์ ถ้ามีชื่อที่ขึ้นต้นด้วย “อ” และ ชื่อ “อรอนงค์” อยู่เป็นจำนวนมาก เมื่อค้นหาแบบประมาณค่าจะได้ค่าของการประมาณผิดบ่อยครั้งโดยเฉพาะเมื่อพบชื่ออรอนงค์อาจจะประมาณตำแหน่งเรียงเป็นลำดับทีละหนึ่งตำแหน่งเลยทีเดียว และถึงแม้จะมีการกระจายของข้อมูลซึ่งการค้นหาในแบบประมาณค่าจะดีกว่าการค้นหาในแบบทวิภาค แต่ถ้ามองในลักษณะการคำนวนด้วยคอมพิวเตอร์แล้ว การคำนวนตำแหน่งในแบบประมาณค่าจะคำนวนช้า เพราะมีการคำนวนทั้งการบวก การคูณ และการหาร และยังนำค่าข้อมูลมาเกี่ยวข้องด้วยอีก ในขณะที่การคำนวนในแบบทวิภาคจะใช้ตัวเลขจำนวนเต็มมาทำการบวก และ หารด้วย 2 เท่านั้น

อัลกอริทึมการหาตำแหน่งที่อยู่แบบแฮช

ในการจัดเก็บข้อมูลในคอมพิวเตอร์จะใช้เขตข้อมูลหลักในการบ่งชี้ว่าระเบียนอยู่ที่ใด ถ้าเราทราบว่าเขตข้อมูลหลักมีค่าจาก 1 ถึง N เรียงต่อกันไปการหาตำแหน่งให้แต่ละระเบียนก็สามารถกำหนดให้ตรงกับค่าของเขตข้อมูลหลักได้เลย (Sedgewick, 1992) แต่ในความเป็นจริงเราไม่สามารถทราบเขตข้อมูลหลักทุกตัวได้ เช่นรหัสของสินค้า เลขประจำตัว หรือชื่อในสมุดโทรศัพท์ เมื่อกำหนดเป็นตารางเพื่อใช้เก็บตำแหน่งอาจต้องใช้พื้นที่ขนาดใหญ่มาก ซึ่งอาจมีข้อมูลอยู่ในตารางเพียงเล็กน้อยทำให้สูญเสียพื้นที่ไปโดยเปล่าประโยชน์ วิธีการหนึ่งในระบบคอมพิวเตอร์ คือนำเขตข้อมูลหลักมาเข้าสูตรทางคณิตศาสตร์ผลที่ได้จะเป็นตำแหน่งที่อยู่ในตารางที่กำหนด สูตรที่ใช้คำนวนนั้นจะเรียกว่าฟังก์ชันแบบแฮช (Hash Function) ตามรูปที่ 3.7



รูปที่ 3.7 แสดงการนำเขตข้อมูลเข้าสูตรคำนวนหาตำแหน่งที่อยู่แบบแฮช

จากภาพจะเป็นหลักการของการหาตำแหน่งโดยจะนำเขตข้อมูลเข้าคำนวนในฟังก์ชันแบบแฮช ผลที่ได้จะเป็นตำแหน่งในตารางที่จัดเตรียมไว้ แต่มีปัญหาสองข้อที่ต้องคำนึงถึงเสมอถ้าจะใช้ฟังก์ชันแบบแฮชในการหาตำแหน่งคือ หนึ่งจะหาฟังก์ชันแบบแฮชที่ดีได้อย่างไร และสองจะแก้ไขการซ้ำตำแหน่งที่อาจเกิดขึ้นด้วยวิธีใด (Kruse, Leung, Tondo, 1991) ในเรื่องของการเลือกใช้ฟังก์ชันแบบแฮชจะมีฟังก์ชันที่มีลักษณะการทำงานที่แตกต่างกันไปหลายวิธีดังนี้

1. ใช้บางส่วนของเขตข้อมูล (Truncation or Extraction), (Drozdek, L.Simon, 1995)

จากเขตข้อมูลที่มีขนาดยาวจะเอาเพียงบางส่วนของเขตข้อมูล เช่นเขตข้อมูลเป็นตัวเลขขนาด 8 ตัว มีค่า “34567891” ถ้ากำหนดขนาดตารางแฮชเท่ากับ 1000 จะต้องเอาตัวเลขสามหลักจากเขตข้อมูลหลักอาจกำหนดให้เอาจากตำแหน่งที่ 2, 4, 6 ได้เท่ากับ 468 หรืออาจจะเอาสามหลักแรก หรือสามหลักสุดท้าย ในการเลือกตำแหน่งต้องระมัดระวังไม่เลือกตำแหน่งที่เมื่อ

นำมารวมกันแล้วเกิดการซ้ำกันมากๆ วิธีนี้เป็นวิธีที่ใช้ในการคำนวนหาตำแหน่งที่รวดเร็ว เพราะไม่ต้องใช้การคำนวนที่ยุ่งยากซับซ้อนแต่อย่างใด แต่มักเกิดเหตุการณ์ที่ตำแหน่งที่ได้ไม่ค่อยมีการกระจายในตารางที่ดี (Kruse, Leung, Tondo, 1991)

2. แบ่งเป็นส่วนๆ (Folding), (Drozdek, L.Simon, 1995)

แบ่งข้อมูลออกเป็นกลุ่มแล้วนำมาบวกกันซึ่งมีด้วยกัน 2 วิธี

2.1 ใช้การขยับของข้อมูล (Shift Folding)

ขยับข้อมูลที่ตำแหน่งที่ระบุทำให้แยกข้อมูลออกจากกันให้เห็นเป็นกลุ่ม เช่นจากข้อมูล "34567891" ขยับตำแหน่งที่ 4 และที่ 7 ออกจะได้ 345 678 91 ซึ่งจะได้ 3 กลุ่ม แล้วนำแต่ละกลุ่มบวกเข้าด้วยกันผลลัพธ์คือ 345 + 678 + 91 = 1114

2.2 ใช้การพับเหมือนการพับกระดาษ (Bound Folding) ลองนึกภาพจากตัวเลขเดิมพับกระดาษที่ตำแหน่งที่ 4 และ 7 เหมือนการขยับข้อมูล ให้เลข 345 อยู่ด้านหน้าพับลงที่ตำแหน่งที่ 4 แล้วพับขึ้นตำแหน่งที่ 7 ถ้ากระดาษสามารถมองเห็นตัวเลขทั้งสามชุดจะเห็นว่าตัวเลขชุดที่สองจะกับด้านอยู่เมื่อนำมาเรียงจะได้ 345 876 91 แล้วนำมาบวกกันได้ 1312
วิธีแบ่งเป็นส่วนๆจะให้ผลการกระจายของตำแหน่งที่ดีกว่าแบบใช้บางส่วนของเขตข้อมูล (Kruse, Leung, Tondo, 1991)

3. ค่ากลางของเลขยกกำลังสอง (Mid-Square), (Drozdek, L.Simon, 1995)

จากวิธีแรกถ้ามีการเลือกตำแหน่งกลางเป็นผลลัพธ์ แต่ ณ ตำแหน่งกลางที่เลือกมีการซ้ำกันของข้อมูลอยู่มาก วิธีขจัดการซ้ำของข้อมูลก็โดยนำข้อมูลดังกล่าวมาทำการยกกำลังสองเพื่อให้ได้ตัวเลขใหม่แล้วจึงนำค่ากลางมาใช้ เช่นเขตข้อมูลหลัก 3121 เมื่อนำมาหาตำแหน่งโดยวิธีนี้จะได้ 3121^2 = 97**406**41 ตำแหน่งที่ได้คือ 406

4. เปลี่ยนเลขฐาน (Radix Transformation), (Drozdek, L.Simon, 1995)

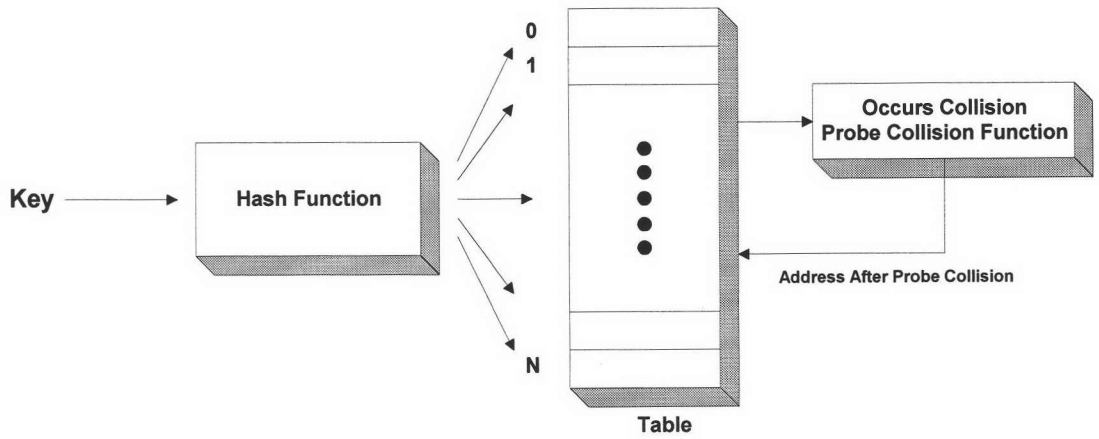
นำเขตข้อมูลที่จะทำการคำนวนด้วยฟังก์ชันแฮช มาเปลี่ยนฐานของตัวเลขเสียใหม่เช่นจากเดิมเป็นเลขฐาน 10 ก็เปลี่ยนเป็นเลขฐาน 9 เช่น 345 ฐาน 10 = 423 ฐาน 9 แล้วนำผลที่ได้ไปคำนวนหาตำแหน่งอีกทีหนึ่ง

5. เศษจากการหาร (Modulo), (Drozdek, L.Simon, 1995)

นำเขตข้อมูลหารด้วยขนาดของตารางแฮช ผลลัพธ์คือเศษจากการหาร วิธีนี้ถือว่าเป็นวิธีที่ให้ผลลัพธ์ที่กระจายข้อมูลได้ดี แต่ต้องพยายามกำหนดให้ขนาดของตารางแฮชเป็นเลขเฉพาะ (Prime Number) เช่นแทนที่จะกำหนดขนาดตารางเท่ากับ 1000 ก็กำหนดเป็น 997 หรือ

1009 แทน (Kruse, Leung, Tondo, 1991) และในโครงการวิจัยนี้ก็จะใช้เศษจากการหารนี้เป็นฟังก์ชันในการหาตำแหน่งในตารางแฮช

เมื่อทำการเปลี่ยนเขตข้อมูลหลักให้เป็นตำแหน่งในตารางแฮช และเมื่อต้องการค้นหาระเบียนของเขตข้อมูลหลักที่ต้องการ ก็จะนำเขตข้อมูลหลักนั้นมาคำนวนด้วยฟังก์ชันแฮชในทำนองเดียวกัน ถ้าสูตรในการคำนวนสามารถหาตำแหน่งที่อยู่ให้กับเขตข้อมูลหลักที่ต่างกันได้โดยไม่มีการซ้ำกันเลยถือว่าเป็นสูตรที่สมบูรณ์ที่สุดเพราะจะทำให้ค้นหาข้อมูลเป็นด้วยความรวดเร็วเพราะจะเป็นการหาตำแหน่งที่อยู่เพียงครั้งเดียวก็จะพบระเบียนที่ต้องการได้เลย แต่มักจะไม่สามารถหาสูตรที่สมบูรณ์แบบได้ การนำเขตข้อมูลหลักเข้าคำนวนหาตำแหน่ง ตำแหน่งที่ได้อาจมีข้อมูลตัวอื่นอยู่แล้วเรียกลักษณะเช่นนี้ว่าเกิดการชนกัน (Collision) ดังนั้นก็จะต้องมีวิธีในการหาตำแหน่งใหม่ให้กับระเบียนนั้นๆเรียกการแก้ปัญหาการชนกัน (Collision Resolution) ตามรูปที่ 3.8



รูปที่ 3.8 แสดงการนำเขตข้อมูลที่ซ้ำตำแหน่งคำนวนหาตำแหน่งใหม่

โครงการวิจัยนี้จะนำเสนอการหาเลขที่อยู่แบบแฮชใน 2 ลักษณะ 8 แบบดังนี้

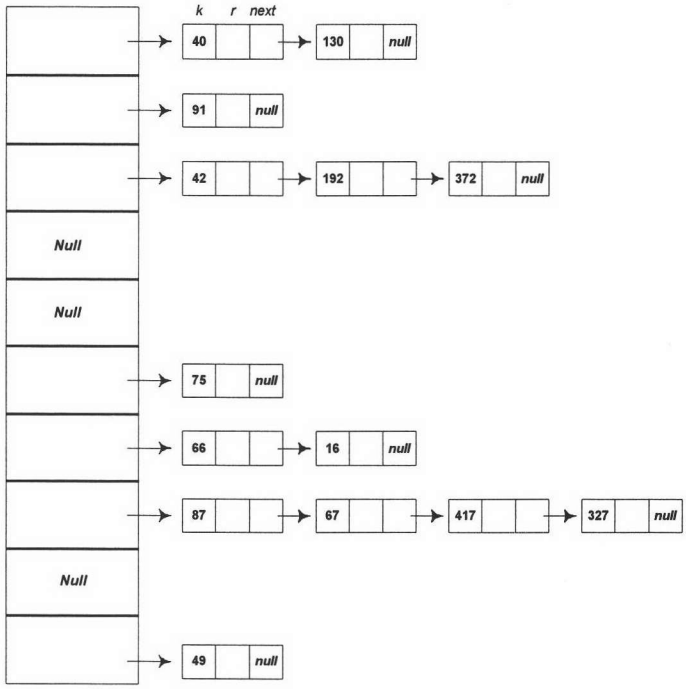
1. การหาเลขที่อยู่แบบแฮชเมื่อเกิดการซ้ำตำแหน่งไม่หาตำแหน่งที่อยู่ใหม่ในตาราง

1.1 ไม่มีการบันทึกระเบียน (Non Resolution)

เมื่อนำค่าเขตข้อมูลหลักเข้าคำนวนในฟังก์ชันแบบแฮชแล้วได้ตำแหน่งในตาราง ถ้าตำแหน่งในตารางไม่ว่างจะไม่ทำรายการนั้น วิธีมีไว้เพื่อเพียงตรวจสอบว่าในการกำหนดจำนวนข้อมูลแต่ละชุดและกำหนดขนาดตารางขนาดต่างๆ เมื่อนำข้อมูลผ่านฟังก์ชันแบบแฮชแล้วมีกี่จำนวนของข้อมูลที่ไม่สามารถหาตำแหน่งที่อยู่ได้ และเพื่อดูว่าข้อมูลที่หาตำแหน่งที่อยู่แบบแฮชมีการรวมกลุ่มหรือกระจายกันมากน้อยเพียงใด

1.2 ใช้รายการเชื่อมโยง (Separate Chainning)

กำหนดให้ตารางแฮชเก็บตำแหน่งของเขตข้อมูลหลัก เมื่อนำเขตข้อมูลหลักคำนวนในฟังก์ชันแบบแฮชแล้วตำแหน่งที่ได้ซ้ำกันจะสร้างตำแหน่งใหม่แล้วนำตำแหน่งใหม่ไปบันทึกเข้าส่วนหน้ามีส่วนชี้ไปยังตำแหน่งในตารางแฮชแล้วนำค่าที่มีอยู่แล้วต่อท้าย และถ้ามีการซ้ำเพิ่มในตำแหน่งนี้อีกก็จะทำในลักษณะต่อๆกันไปตามรูปที่ 3.9 ในการทำเช่นนี้จะมีข้อดีตรงที่ไม่ต้องมีการคำนวนเพื่อหาตำแหน่งใหม่ให้กับเขตข้อมูลที่มีการซ้ำตำแหน่งกัน คือไม่ต้องมีสูตรมาทำการหาตำแหน่งใหม่ จะทำเพียงนำแต่ละตำแหน่งมาเชื่อมต่อกัน



k = key
r = pointer to record
next = pointer to next link

รูปที่ 3.9 แสดงลักษณะตารางแบบแฮชแบบรายการโยง

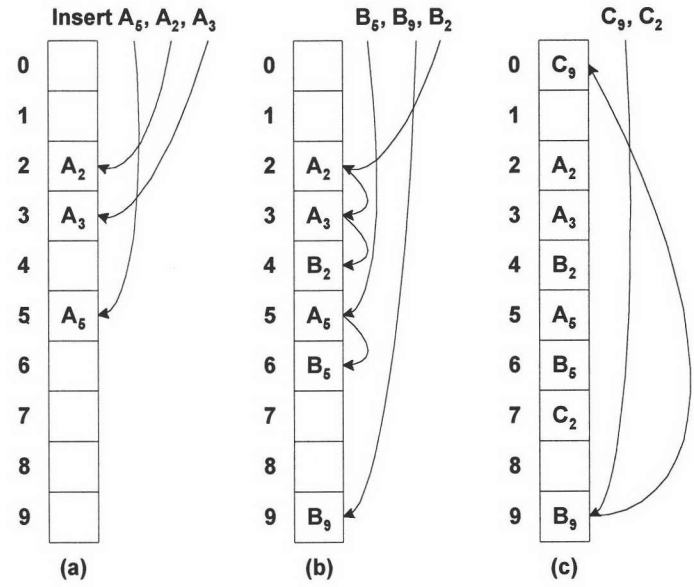
จากรูปตารางมีขนาดเท่ากับ 10 แต่ละช่องใช้เก็บตำแหน่งในการเชื่อมโยงในครั้งแรกจะถูกกำหนดให้เป็นตำแหน่งว่าง (Null) เมื่อค่าข้อมูล 130 เข้าคำนวนเพื่อหาตำแหน่ง ได้ตำแหน่งในตารางเท่ากับ k(130) = 130 mod 10 = 0 พร้อมสร้างจุดเชื่อม (Node) ที่มีข้อมูลในสามส่วน ส่วนแรกเก็บค่าเขตข้อมูล (key) ส่วนที่สองเก็บตำแหน่งที่ชี้ไปยังระเบียน ส่วนที่สามเก็บตำแหน่งของจุดเชื่อมตัวต่อไป แต่ถ้ายังไม่มีจุดเชื่อมที่ชี้ไปจะกำหนดให้ค่าในส่วนนี้เป็นตำแหน่งว่าง (null) โดยตำแหน่งตารางที่ 0 จะนำตำแหน่งของจุดเชื่อมของเขตข้อมูลที่มีค่า 130 มาเก็บไว้ เมื่อเขตข้อมูลที่มีค่า 40 เข้าคำนวนหาตำแหน่งได้เท่ากับ 0 เช่นเดียวกัน จะนำตำแหน่งที่

เก็บอยู่ที่ตารางตำแหน่งที่ 0 ไปเก็บอยู่ในส่วนที่สามของของจุดเชื่อมค่าข้อมูล 40 แล้วนำตำแหน่งของจุดเชื่อมค่าข้อมูล 40 ไปเก็บในตารางตำแหน่งที่ 0 แทน ทำเช่นนี้กับทุกค่าข้อมูล การเชื่อมโยงแบบนี้จะเป็นการเชื่อมโยงเข้าส่วนหน้า แต่ก็สามารถใช้วิธีเชื่อมโยงเข้าตอนท้าย หรือทำการเชื่อมโยงโดยตรวจสอบก่อนเพื่อให้มีการเรียงลำดับด้วย โดยเฉพาะการเชื่อมโยงที่ให้มีการเรียงลำดับจะมีประโยชน์อย่างมากเมื่อต้องค้นหาข้อมูล เมื่อข้อมูลที่ต้องการค้นหาไม่มีอยู่ในรายการโยงจะตรวจสอบรายการโยงถึงตำแหน่งที่มีค่ามากกว่าหรือน้อยกว่าแล้วแต่การจัดเรียงของรายการโยง ก็จะทราบได้ว่าไม่พบข้อมูลที่ต้องการโดยไม่ต้องตรวจสอบไปจนหมดรายการโยงเช่นเดียวกับการทำงานในค้นหาข้อมูลแบบลำดับ เมื่อข้อมูลมีการจัดเรียงลำดับ

2. การหาเลขที่อยู่แบบแฮชเมื่อเกิดการซ้ำตำแหน่งหาตำแหน่งที่อยู่ใหม่ในตารางแฮช

2.1 หาตำแหน่งที่ว่างถัดไปแบบลำดับ (Linear)

เมื่อเกิดการซ้ำตำแหน่งจะหาตำแหน่งว่างโดยตรวจสอบตำแหน่งถัดไปเรื่อยๆทีละตำแหน่ง โดยเริ่มจากตำแหน่งที่เกิดการซ้ำเหมือนลักษณะการค้นหาแบบลำดับ จนพบตำแหน่งว่าง ก็จะบันทึกลง ณ ตำแหน่งนั้น ถ้าหาตำแหน่งว่างไปจนถึงตำแหน่งสุดท้ายของตารางแล้วยังไม่พบจะเริ่มหาตำแหน่งว่างที่ตำแหน่งแรกของตารางไปจนถึงตำแหน่งก่อนหน้าตำแหน่งซ้ำหนึ่งตำแหน่ง ตามรูปที่ 3.10



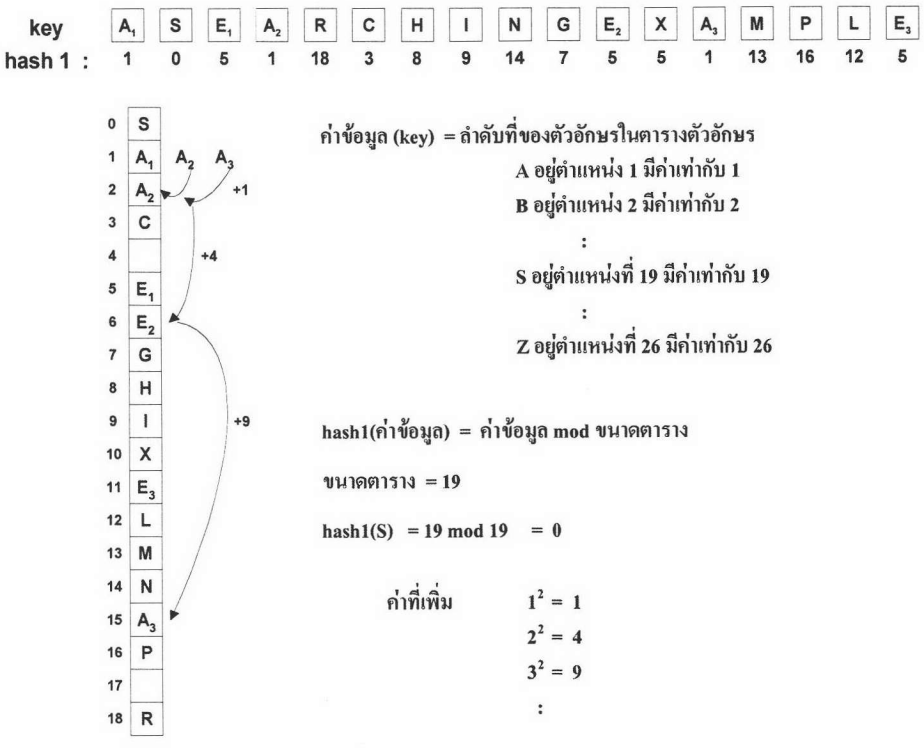
รูปที่ 3.10 แสดงการหาตำแหน่งว่างเมื่อเกิดการชนด้วยวิธีเชิงเส้น

จากรูปเมื่อนำข้อมูลในชุด (a) ซึ่งประกอบด้วย A_5 คำนวนด้วยฟังก์ชันแบบแฮชได้ตำแหน่งในตารางตามดรรชีล่างคือ 5, A_2 ในตำแหน่ง 2, A_3 ในตำแหน่ง 3 ต่อจากนั้นก็นำข้อมูลชุด (b) เข้าคำนวนตำแหน่งในตารางได้ตำแหน่งตามดรรชีล่างเช่นกัน โดย B_5 ซ้ำตำแหน่งกับ A_5 B_5 หา

ตำแหน่งใหม่แบบลำดับทีละหนึ่งได้ตำแหน่งว่างในตำแหน่งที่ 6 B_9 ลงตำแหน่งที่ 9 B_2 ซ้ำตำแหน่งกับ A_2 หาตำแหน่งถัดไปคือ 3 มี A_3 อยู่แล้วหาตำแหน่งถัดไปได้ตำแหน่งที่ 4 นำข้อมูลชุด (c) เข้าคำนวนในลักษณะเดียวกันจะได้ C_9 จะมาอยู่ที่ตำแหน่ง 0 และ C_2 เลื่อนไปถึงตำแหน่งที่ 7 หลังจากที่ข้อมูลชุด (b) ทำฟังก์ชันแฮชหมดแล้วจะเห็นว่าตารางมีลักษณะของข้อมูลจับกันเป็นกลุ่มเรียกลักษณะเช่นนี้ว่าเกิดกลุ่ม Cluster เมื่อเกิดกลุ่มจะมีผลเสียทั้งในตอนเพิ่มข้อมูลเข้าตาราง หรือตอนค้นหาข้อมูลในตารางจะกระทำได้ช้า เพราะต้องทำการตรวจสอบหลายครั้ง อย่างกรณีเพิ่ม C2 ในข้อมูลชุด (c) จะต้องหาตำแหน่งว่างถึง 5 ครั้งจึงจะพบตำแหน่งว่างได้

2.2 หาตำแหน่งที่ว่างถัดไปแบบกำลังสอง (Quadratic)

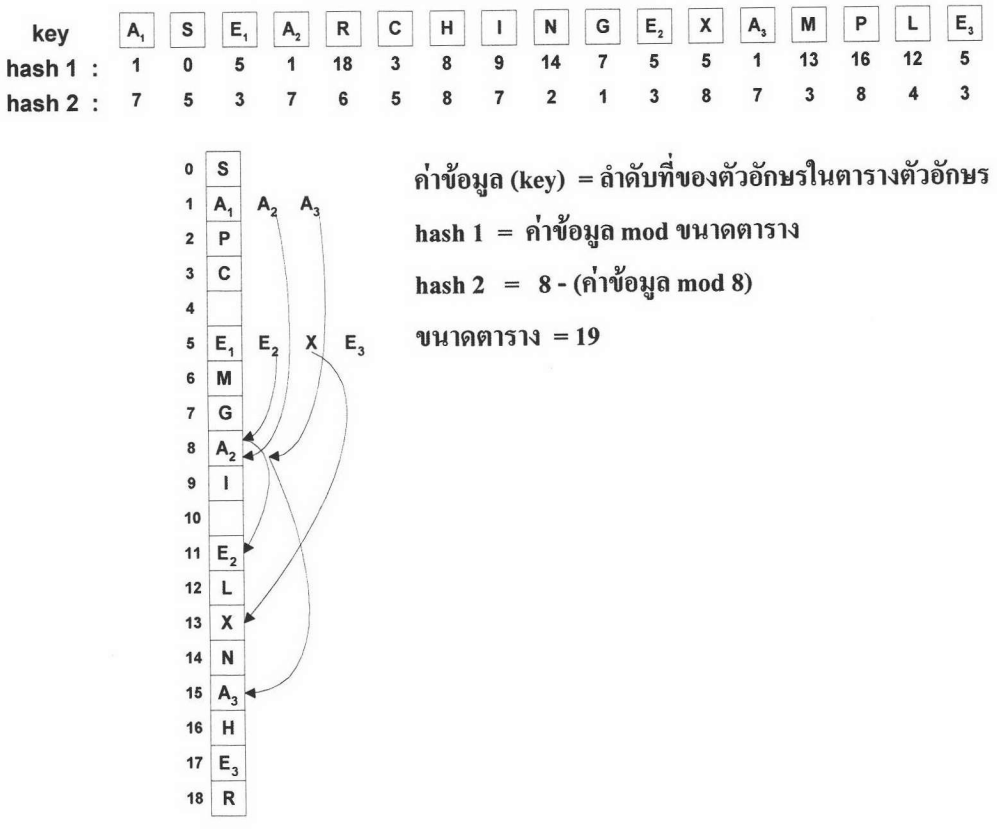
ลักษณะของการหาตำแหน่งว่างถัดไปจะคล้ายกับแบบลำดับ แต่แทนที่จะเพิ่มตำแหน่งในการหาที่ว่างถัดไปทีละหนึ่งตำแหน่ง ก็จะใช้สูตรในการเพิ่มตำแหน่งเพื่อหาตำแหน่งว่างถัดไปเพื่อไม่ให้เกิดการเกาะกลุ่มของการหาตำแหน่ง ตัวเลขที่ใช้ในการเพิ่มได้จากการนำเลขลำดับที่จาก 1,2,3...n โดยที่ n จะเท่ากับขนาดของตารางลบหนึ่ง (TableSize - 1) มายกกำลังสองซึ่งจะได้ $1^2 = 1$, $2^2 = 4$, $3^2 = 9$, ... i^2 ผลที่ได้จะนำไปรวมกับตำแหน่งที่ซ้ำกันได้ตำแหน่งในการหาที่ว่างถัดไปคือ h+1, h+4, h+9, $h+i^2$



รูปที่ 3.11 แสดงการหาตำแหน่งว่างเมื่อเกิดการชนด้วยวิธีกำลังสอง

2.3 หาตำแหน่งที่ว่างถัดไปแบบใช้ฟังก์ชันแฮชสองครั้ง (Double Hash)

เป็นอีกวิธีหนึ่งที่จะหลีกเลี่ยงการเกาะกลุ่มกันในตารางแฮช โดยนำเขตข้อมูลที่ใช้ในฟังก์ชันแบบแฮชเพื่อให้ได้ตำแหน่ง มาทำในฟังก์ชันแบบแฮชอีกครั้งหนึ่ง เพียงแต่ครั้งที่สองจะไม่ใช้ขนาดของตารางเป็นตัวหาร แต่จะหาค่าๆหนึ่งที่เหมาะสมหรืออาจเป็นเลขเฉพาะที่น้อยกว่าขนาดตารางก็ได้เป็นตัวหาร เศษจากการหารคือผลลัพธ์ซึ่งจะเป็นค่าที่จะนำไปเพิ่มในแต่ละรอบของการหาตำแหน่งว่างถัดไป ลักษณะการทำจะคล้ายกับแบบกำลังสอง แต่แบบใช้ฟังก์ชันแฮชสองครั้งจะไม่ได้เพิ่มตำแหน่งการหาโดยใช้กำลังสองของเลขลำดับแต่จะนำค่าข้อมูลมาทำการหาตำแหน่งเพิ่มแทน แสดงได้ดังรูปที่ 3.12



รูปที่ 3.12 แสดงการหาตำแหน่งว่างเมื่อเกิดการชนด้วยฟังก์ชันแฮชสองครั้ง

จากรูปเมื่อกำหนดฟังก์ชันแบบแฮชสองครั้งเท่ากับ 8 - (ค่าข้อมูล mod 8)
เมื่อนำ A_1 คำนวนด้วยฟังก์ชันแฮชได้ตำแหน่งที่ 1, S ได้ตำแหน่ง 0, E_1 ได้ตำแหน่ง 5
A_2 “ “ 1 เกิดซ้ำตำแหน่งกับ A_1 A_2 ต้องทำการหาตำแหน่งใหม่ นำค่า A_2 เข้าคำนวนในฟังก์ชันแฮชที่สองได้ 8 - (1 mod 8) = 7

นำค่าที่ได้เพิ่มเข้าในตำแหน่งที่มีการชนกันได้ 1 + 7 = 8 ทำการตรวจสอบว่าตำแหน่งที่ 8 ในตารางว่างหรือไม่ ถ้าว่างก็นำค่า A_2 เข้าในตาราง แต่ถ้าไม่ว่างก็จะทำการเพิ่มตำแหน่งการหาที่ว่างอีกทีละ 7 ไปเรื่อยๆจนกว่าจะพบตำแหน่งว่าง และจะทำเช่นนี้กับทุกค่าเขตข้อมูลจนหมดค่าเขตข้อมูล

2.4 หาตำแหน่งที่ว่างถัดไปแบบจัดใหม่แบบเรียง (Ordering)

ใช้ฟังก์ชันแฮชสองครั้ง (Double Hash) ในการหาตำแหน่งใหม่ ถ้าเกิดการซ้ำตำแหน่งกันจะตรวจสอบก่อนว่า ที่ตำแหน่งซ้ำกันนั้นค่าข้อมูลตัวใดมีค่ามากกว่าจะอยู่ที่ตำแหน่งนั้น แล้วเอาตัวที่มีค่าข้อมูลน้อยกว่าไปหาตำแหน่งใหม่ ซึ่งจะได้ว่าตำแหน่งที่เกิดการชนกันทั้งหมดจะมีการเรียงลำดับจากมากไปน้อยทันที จะทำให้ค้นหาข้อมูลมีประสิทธิภาพมากขึ้นเพราะไม่จำเป็นต้องตรวจสอบในทุกตำแหน่งที่เกิดการซ้ำตำแหน่งเพื่อหาเขตข้อมูลที่ต้องการเหมือนค้นหาข้อมูลแบบลำดับซึ่งมีข้อมูลที่มีการจัดเรียงลำดับอยู่แล้ว แสดงเป็นโปรแกรมคอมพิวเตอร์ได้ดังรูปที่ 3.13

```
        *** Brent's reoganization hashing : Insertion ***
Sub Alg_Hash_Open_Ordered ()
   Dim hf, n As Integer
   Dim key As Integer

   For n = 1 To NumRnd
      key = Data_Rnd(n)
      hf = hashfunction(key)
      HashArray(hf).Row = HashArray(hf).Row + 1
      While (HashArray(hf).Row <> 1) And (HashArray(hf).Value <> key)
         If (HashArray(hf).Value > key) Then
            temp = key
            key = HashArray(hf).Value
            HashArray(hf).Value = temp
         End If
         hf = (hf + increment(key)) Mod TableSize
         HashArray(hf).Row = HashArray(hf).Row + 1
      Wend
   Next n
```

```
End Sub
```

รูปที่ 3.13 แสดงโปรแกรมการหาที่อยู่แบบแฮชด้วยวิธีจัดลำดับ

2.5 หาตำแหน่งที่ว่างถัดไปแบบจัดใหม่แบบเบรน (Brent Reoganization)

เป็นอีกอัลกอริทึมหนึ่งที่ปรับปรุงประสิทธภาพของการใช้ฟังก์ชั่นแฮชสองครั้งในการหาตำแหน่งใหม่ โดยที่ตำแหน่งที่เกิดการชนกันจะนำข้อมูลที่หาตำแหน่งและข้อมูลที่มีอยู่แล้วในตาราง ณ ตำแหน่งที่ซ้ำกันไปทำการหาตำแหน่งใหม่ทั้งสองค่าเพื่อดูว่าข้อมูลตัวใดสามารถหาตำแหน่งใหม่ได้ก่อนก็จะนำข้อมูลนั้นลงที่ตำแหน่งใหม่เลย ดังนั้นจะทำให้จำนวนการซ้ำตำแหน่ง (Collision) เกิดน้อยลง อันจะมีผลต่อประสิทธิภาพเมื่อทำค้นหาข้อมูล แสดงเป็นโปรแกรมคอมพิวเตอร์ได้ดังนี้รูปที่ 3.14

```
        *** Brent's reoganization hashing : Insertion ***
Sub Alg_Hash_Open_Brent ()
   Dim i, ii, inc, hf, j, jj As Integer
   Dim n, key As Integer
   For n = 1 To NumRnd
      key = Data_Rnd(n)       '*** key = ค่าเขตข้อมูล ***
      hf = hashfunction(key)  '*** hf  = ค่าที่ได้จากฟังก์ชันแฮช ***
      inc = increment(key)    '*** inc =ค่าที่จะนำไปเพิ่มในการหาตำแหน่ง ***
      For i = 0 To n
         For j = i To 0 Step -1
            jj = (hf + (inc * j)) Mod TableSize
            ii = (jj + increment(HashArray(jj).Value) * (i - j)) Mod TableSize
            HashArray(ii).Row = HashArray(ii).Row + 1
            If HashArray(ii).Row = 1 Then
               HashArray(ii) = HashArray(jj)
               HashArray(jj).Value = key
               GoTo 999
            End If
         Next j
      Next i
999 :
   Next n
```

```
End Sub
```

รูปที่ 3.14 แสดงโปรแกรมการหาที่อยู่แบบแฮชด้วยวิธีเบรน

2.6 หาตำแหน่งที่ว่างถัดไปแบบจัดใหม่แบบทวิภาค (Binary Reorgainization)

ใช้ฟังก์ชันแฮชสองครั้งในการเพิ่มเพื่อหาตำแหน่งใหม่ แล้วนำค่าที่ต้องการหาตำแหน่งกับค่าที่อยู่ ณ ตำแหน่งที่ซ้ำไปหาตำแหน่งใหม่ โดยใช้รูปแบบการหาตำแหน่งใหม่แบบต้นไม้ทวิภาค (Binary tree pattern) แสดงเป็นโปรแกรมคอมพิวเตอร์ได้รูปที่ 3.15 ดังนี้

```
*** Binary tree reoganization hashing : Insertion ***
Sub Alg_Hash_Open_Binary ()
    Dim i, j, n, inc, hf As Integer
    Dim key As Integer
    For n = 1 To NumRnd
        key = Data_Rnd(n)
        hf = hashfunction(key)
        inc = increment(key)
        i = 0
        j = -1
        While (i <= n) And (j < 0) 'And (n < TableSize)
            j = SearchMove(hf, inc, i)
            i = i + 1
        Wend
        If j > -1 Then
            HashArray(j).Value = key
        End If
    Next n
End Sub

Function SearchMove (ByVal init As Integer, ByVal inc As Integer, ByVal level As
                        Integer) As Integer
    Dim i, ii, incl, j, k As Integer
    i = (init + (inc * level)) Mod Tablesize
    HashArray(i).Row = HashArray(i).Row + 1
```

```
    If HashArray(i).Row = 1 Then
        SearchMove = i
    Else
        For j = level - 1 To 0 Step -1
            i = (init + (inc * j)) Mod Tablesize  'Fomula is the same Fomula Above
            incl = increment(HashArray(i).Value)
            k = SearchMove((i + incl) Mod Tablesize, incl, level - j - 1)
            If k > -1 Then
                HashArray(k) = HashArray(i)
                SearchMove = i
                GoTo End_Search_Move
            End If
        Next j
        SearchMove = -1   'Could not found hold
    End If
End Function
```

รูปที่ 3.15 แสดงโปรแกรมการหาที่อยู่แบบแฮชด้วยวิธีจัดเรียงแบบต้นไม้ทวิภาค