บทที่ 6

การคำนวณหาตำแหน่งของแฟ้มข้อมูลแบบสุ่ม

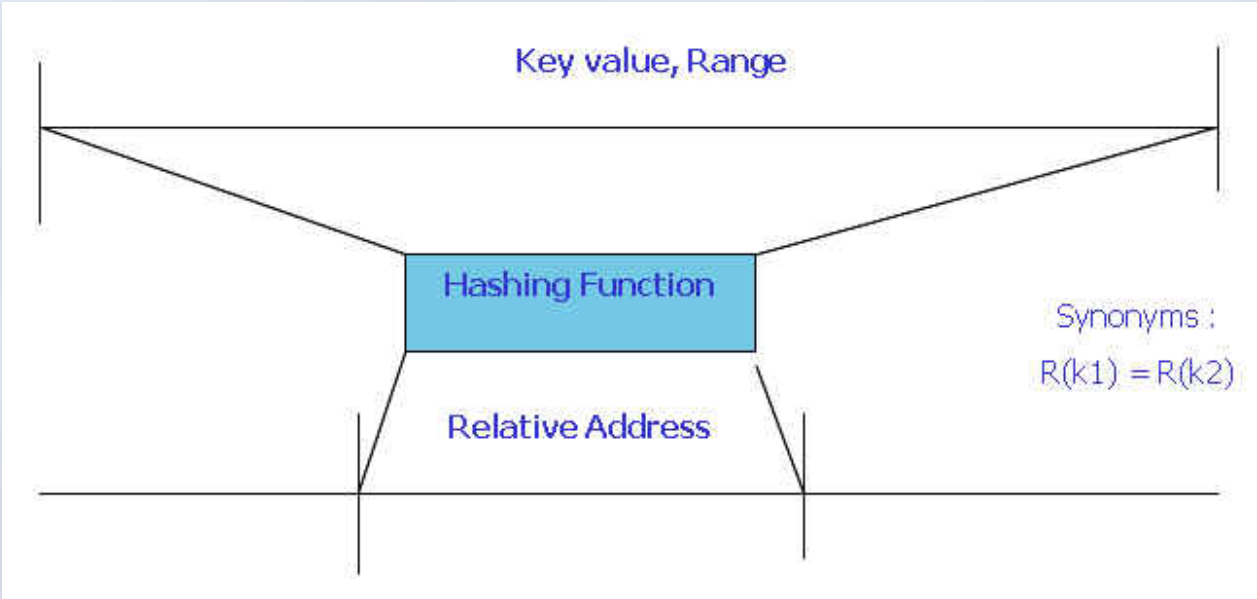
ภาควิชาวิทยาการคอมพิวเตอร์ คณะวิทยาศาสตร์ มหาวิทยาลัยสงขลานครินทร์

การคำนวณตำแหน่ง : Address Calculation

- **Primary key** มีข้อมูลเป็นตัวเลขขนาดกว้างทำให้ เมื่อนำมาใช้เป็น Relative Address ทำให้เกิดเนื้อที่ว่างมาก จะต้องปรับให้ให้เหมาะสมกับจำนวนระเบียนจริงๆ ที่เก็บ โดยใช้ฟังก์ชันช่วย
- โปรแกรมเมอร์เป็นผู้สร้างหรือกำหนดฟังก์ชันขึ้นมาใช้เอง เพื่อเปลี่ยน Primary Key --> Relative Address
- ใน 2 วิธีแรกจะให้ตำแหน่งที่ไม่ซ้ำกัน แต่วิธีนี้มีโอกาสทำให้ Relative Address ที่คำนวณได้อาจจะซ้ำกันก็ได้

Collision, Synonyms, Overflow Record

- **Collision** (ชนกัน) คือ การที่ใช้ Primary key(K) ของ 2 records คำนวณแล้วได้ Relative Address ตำแหน่งเดียวกันโดย ค่าของ K1 not= K2
- **Synonyms** คือ คำที่ใช้เรียก K1 และ K2 ที่ทำให้เกิดการชนกันของคีย์
- **Overflow Record** (ระเบียนส่วนล้น) คือ คำที่ใช้เรียกระเบียนที่ไม่สามารถเก็บอยู่ในตำแหน่งที่คำนวณได้ (Home Address) ว่า



แนวคิดการคำนวณตำแหน่ง

- แนวคิดการคำนวณตำแหน่ง แต่เดิมใช้กับการเข้าถึงข้อมูลในตารางสัญลักษณ์ (Symbol table) ในหน่วยความจำ ต่อมาได้ใช้การคำนวณตำแหน่งสำหรับระเบียนของ DASD เทคนิคการคำนวณตำแหน่งต่างๆ จึงได้พัฒนาขึ้นมากมาย
  - Randomization technique
  - Key to address transform
  - **Hashing function**
  - Scatter storage technique

ปัจจัยที่มีอิทธิพลต่อประสิทธิภาพของแฟ้มสุ่ม ที่ใช้วิธีการคำนวณตำแหน่ง Hashing Function

- ขนาดของ Bucket
- Loading factor
- ชุดของ Primary Key
- Hashing Function ที่ใช้
- วิธีการจัดการเกี่ยวกับระเบียนส่วนล้น (Overflow Record)

BUCKET

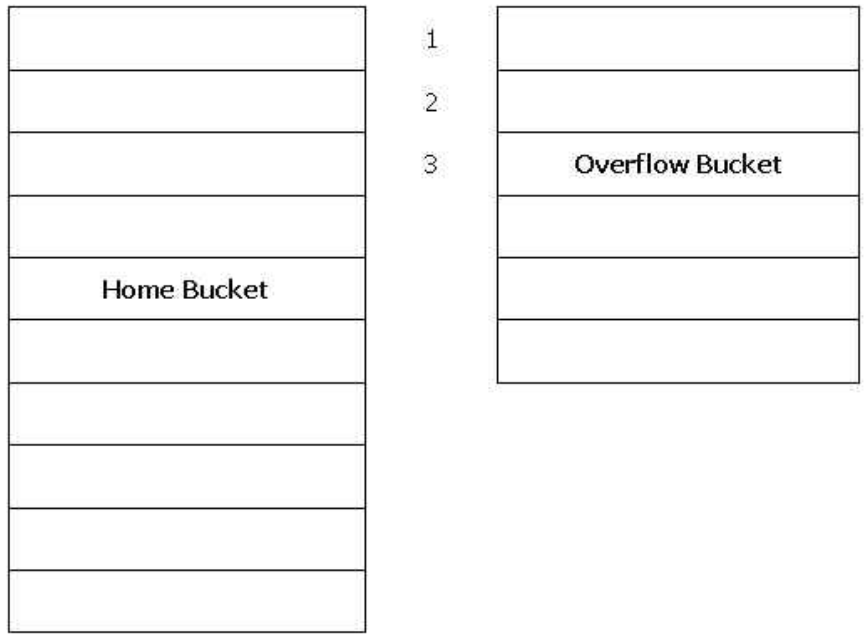
- เป็นที่เก็บข้อมูลของแฟ้มสุ่ม (Random File) สำหรับการคำนวณตำแหน่ง ถูกแบ่งออกเป็นส่วนๆ เรียกว่า bucket
- แต่ละ bucket สามารถเก็บระเบียนได้ 1 หรือมากกว่า 1 ระเบียน
- ความยาวของระเบียน และความจุของ bucket ขึ้นอยู่กับผู้ออกแบบ
- ทุกๆ ระเบียนจะจับคู่กับ Home Bucket อันใดอันหนึ่งโดยกรรมวิธีของ Hashing Function

ขนาดของ Bucket

- ถ้า Bucket มีขนาดเล็ก จะมีอัตราส่วนในการเกิด Overflow ค่อนข้างสูง ทำให้ต้องเสียเวลาอ่าน Bucket
- การใช้ Bucket ขนาดใหญ่จะช่วยลด Overflow record
- ถ้า Bucket จุมากกว่า 1 ระเบียนจะต้องอ่านเข้าหน่วยความจำก่อน แล้วจึงค้นหาระเบียนที่ต้องการได้
- ปกติในการปฏิบัติ จะกำหนดขนาดความจุ Bucket กำหนดให้เท่ากับ ขนาด Block เพราะการอ่านข้อมูลแต่ละครั้งจะอ่านครั้งละ 1 Block

Home Bucket , Overflow Bucket

- 1 แฟ้มข้อมูลอาจประกอบด้วย bucket 2 ประเภท คือ
  - Home Bucket
  - Overflow Bucket



ตำแหน่งที่ผ่านการคำนวณ ได้ Home Address Bucket ที่ได้จากการคำนวณจะเป็น Home Bucket ส่วนที่ไม่สามารถเก็บใน Home Bucket จะต้องเก็บใน Overflow Bucket

Load Factor ของแฟ้มข้อมูล

- คืออัตราส่วนของจำนวน ระเบียนที่มีอยู่ในแฟ้มข้อมูลที่เก็บ ต่อ จำนวนเนื้อที่ที่มีให้ใช้ในแฟ้ม

$$\text{Loading Factor} = \frac{\text{\# Record in file}}{\text{Max. \# of Record file can contain}}$$

- Loading factor มี่ค่าน้อยโอกาส Record ชนกันมีน้อย

หากมีค่ามาก โอกาส Record ชนกันมีมาก

ค่าของ loading factor ประมาณ 0.7-0.8 กำลังดีสูงกว่านี้จะต้องทำ Reorganization ใหม่

Hashing Function

- เป็น Function ที่ใช้เปลี่ยน Primary Key ให้เป็น Relative Address
- Hashing function ที่ดีจะต้องให้จำนวน Synonym น้อยที่สุด หรือไม่มีเลย และควรเป็นฟังก์ที่ไม่ซับซ้อน
- ถ้านำระเบียนต่างๆ ผ่านฟังก์ชันแล้วได้ตำแหน่งกระจายอย่างสม่ำเสมอ ถือว่าชุดของ Primary key เข้ากันได้ดีกับ Hashing Function นั้น

วิธีการ Hashing Function

- วิธีการคำนวณ Hashing Function เพื่อการเก็บระเบียนในแฟ้มข้อมูล ค่า Relative Address ที่คำนวณได้ มีหลายวิธี เช่น
  - The Division Remain Method
  - Folding
  - Radix Conversion
  - Mid - Square

วิธีการใช้เศษเหลือจากการหาร (The Division Remain Method)

- เป็นวิธีการแปลงค่าของ Primary Key ของระเบียน ให้เป็นเลขที่ตำแหน่งอยู่ในช่วง Home Bucket ของแฟ้มข้อมูลโดยใช้จำนวนเฉพาะ (Prime Number) ที่มีค่าใกล้เคียง จำนวน Bucket ทั้งหมด แต่ไม่เกินจำนวน Bucket ในแฟ้มข้อมูล มาเป็นตัวหาร primary key ของระเบียน เศษที่เหลือ จะได้เป็น Relative Address ของระเบียนนั้น

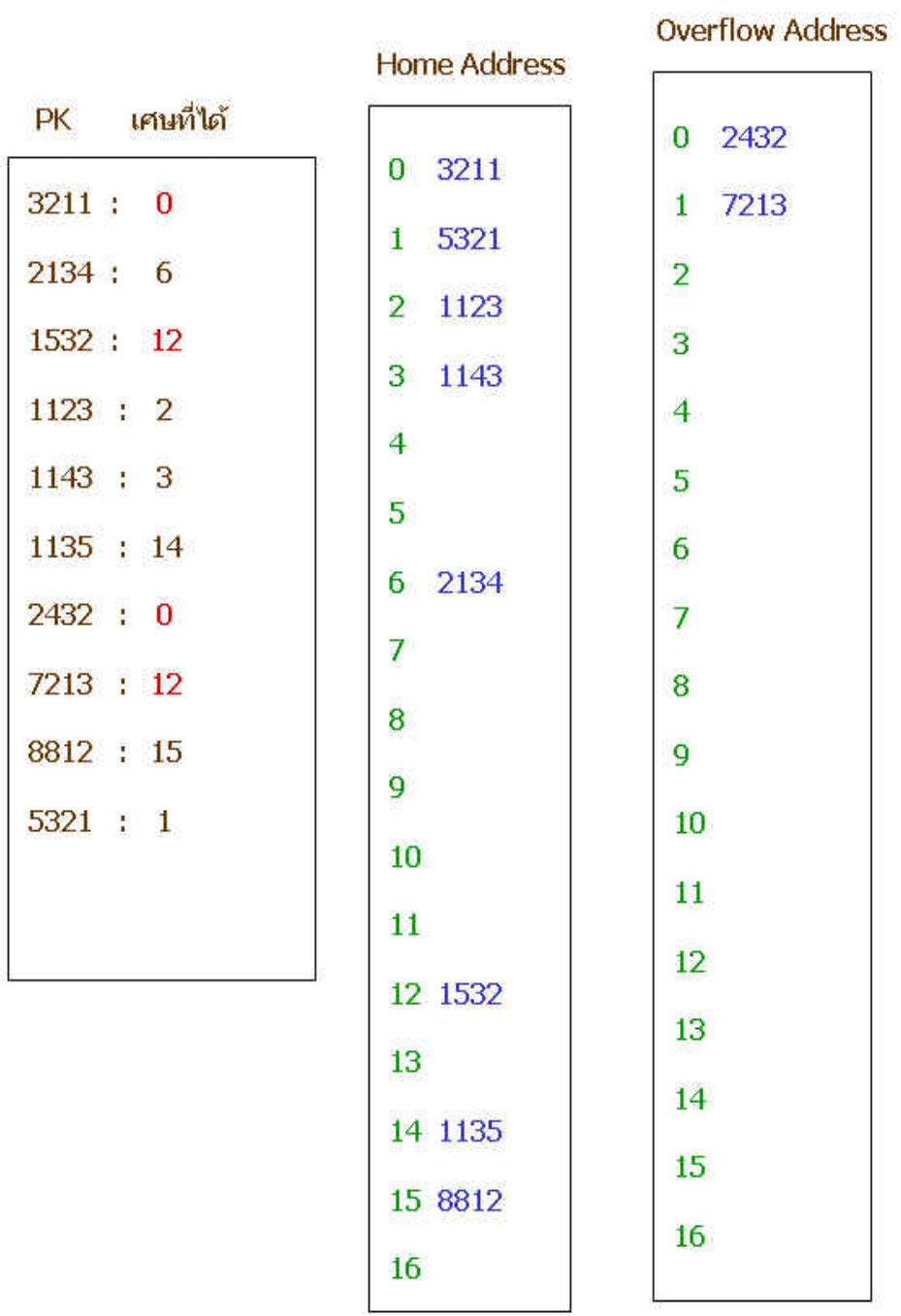
$$F(kv) = mod(kv,pn)$$

mod : ฟังก์ชันที่ใช้หาเศษที่เหลือ, kv : Key Value (เป็นคีย์ของระเบียน)

pn : Largest Prime Number มีค่าไม่เกินจำนวน Home Bucket

ตัวอย่าง

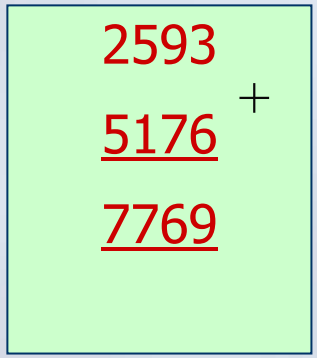
- กำหนดให้ Bucket Capacity = 1 Record และมีข้อมูลที่เก็บมีได้ ไม่เกิน 20 Record
- จากข้อกำหนดเลือก Prime Number (pn) = 19
- 3211 2134 1532 1123 1143 1135 2432 7213 8812 5321



วิธีพับคีย์ (Key Floding)

- เป็นวิธีการคำนวณ Hashing Function โดยการพับค่าของ key ออกเป็นส่วนๆ แล้วรวมค่านั้นเข้าด้วยกัน เพื่อหาค่าสรุปเป็น Address ของระเบียนในแฟ้มข้อมูล
- ***ตัวอย่าง*** ให้ค่า key 8 หลัก ต้องการจัดค่า Address อยู่ในช่วง 0-9999 (10,000 ระเบียน) สามารถใช้
- วิธีพับครึ่ง

  key = 25936715 --> **2593 6715**



วิธีพับครึ่ง (ต่อ)

- key = 55936745 --> **5593 6745**

5593
\+ **5476**
11069

- **เนื่องจากต้องการเพียง 10,000 ระเบียน จึงต้องการเนื้อที่สำหรับ Relativre address แค่ 4 ตำแหน่งแต่ค่าที่คำนวณได้เกิน จำนวณระเบียนสูงสุด ให้ตัดค่าหลักแรกทางซ้ายสุดออก**
- **ดังนั้น ค่า Relativre address คือ 1069**

วิธีพับแบ่งค่าเป็น 3 ส่วน

key = 24936258 --> 24 936 258

key = 95936715 --> 95 936 715

59
936
+ 517
2043

42
936
+ 852
2208

ใช้หลักเลขคู่เลขคี่

- เราอาจจะใช้วิธีการแบ่งเป็นหลักเลขคู่เลขคี่ก็ได้

key = 25936715 --> **5375** **2961**

เลขคี่ **เลขคู่**

5375

\+ **2961**

8336

ข้อดี-ข้อเสีย

- วิธีการพับแบ่งค่านี้เหมะสำหรับค่า key ที่เป็นตัวเลขขนาดใหญ่
- วิธีการพับแบ่งเป็นวิธีการที่ง่าย แต่เกิด collision มาก

25936715 ----> 8759 <---- 39106684

65392715 ----> 1711 <---- 53756336

25936715 ---->

23956715 ---->

75936215 ---->

<---- <---- <----

วิธีการ Mid-Square

- เป็นวิธีการดึงเอาเลขจำนวนหนึ่งออกมาจากค่าของ key แล้วนำมายกกำลังสอง ได้เป็นค่าของ Address เลขที่จะดึงออกมาจากค่ากลางของ key จะเป็นจำนวนเท่าไร (กี่หลัก) ขึ้นอยู่กับขนาดของ Address ที่ต้องการ
- **ตัวอย่าง** ในกรณีที่ค่า key เป็น 8 หลัก 29615834 อาจจะดึงเอาเลขมา 3 หลัก ตรงกลางของ key 158 และนำมายกกำลังสองจะได้ 24964 แต่สมมติเราต้องการค่า Address 4 หลัก (0-9999) สามารถดึง 4964 ได้

วิธีการ Mid-Square (ต่อ)

- วิธีการของ Mid-Square ในบาง Algorithm อาจจะนำค่า key มายกกำลังสองก่อน แล้วค่อยดึงค่ากลางออกมาใช้ เช่น

  123456789 ยกกำลัง 2 ---> **15241578750195052 1**

  987654321 ยกกำลัง 2 ---> **97546105789971041**

- วิธีการ Mid-Square เหมาะสำหรับกรณีที่ค่าของ key ไม่มีเลข 0 อยู่มากนักและในกรณีที่ Loading Factor ต่ำ

  000000472 ยกกำลัง 2 ---> **0000000000222784**

  117400000 ยกกำลัง 2 ---> **1378276000000000 0**

วิธีการ Radix Conversion / Different Radix

- เป็นวิธีการคำนวณ Hashing Function โดยการเปลี่ยน key ของระเบียน ซึ่งเป็นตัวเลขฐานสิบ ไปเป็นฐานอื่น เช่น
- ค่า key = 123456 สมมติเปลี่ยนเป็นเลขฐาน 11 (ให้คิดว่า key ที่ได้มานั้นเป็นฐาน 11 เมื่อนำมาใช้ให้แปลงเป็นฐาน 10) เพื่อหาค่าตำแหน่งเพียง 4 หลัก สามารถทำได้ดังนี้

$1x11^5 + 2x11^4 + 3x11^3 + 4x11^2 + 5x11^1 + 6 = 194871$

เลือกได้ ==> 4871

วิธีการ Perfect Hashing

- เป็นวิธีการที่ถูกเรียกว่าเป็นวิธีการคำนวณ Hashing Function ที่สมบูรณ์แบบ เพราะเมื่อคำนวณแล้วจะได้ค่าที่ไม่ทำให้เกิดการชนกันเลย ***ทำได้โดยการนำค่า key เดิมมาเรียงกันก่อน แล้วจัดตำแหน่งให้*** เช่น สมมติให้
- ค่า key ของระเบียนเป็น 3, 17, 23, 8 , 1, 14
- นำมาเรียงจะได้ 1, 3, 8, 14, 17, 23
- กำหนดค่า Relative Address เป็น 0, 1, 2, 3, 4, 5 ตามลำดับ

วิธีการ Perfect Hashing (ต่อ)

- วิธีการแบบ Perfect Hashing ต้องอาศัยโปรแกรมที่ซับซ้อน และ***ส่วนใหญ่ใช้ได้กับข้อมูลที่คงที่เท่านั้น*** ซึ่งไม่เหมาะสมกับการใช้งานเมื่อนำไปประยุกต์ใช้กับงานที่มีการ เพิ่ม ลบ ข้อมูล

ข้อดี-ข้อเสีย ของการใช้วิธี Hashing Function

ข้อดี

- Primary Key ที่ใช้เป็นไปตามความต้องการของผู้ใช้
- ถ้ามีการทำ Reorganization แฟ้มสุ่ม อาจจะเปลี่ยน Hashing Function แต่ค่าของ key ไม่ต้องเปลี่ยนตาม

ข้อเสีย

- เสียเวลากับการคำนวณ Hashing Function
- เสียเวลากับการแก้ปัญหาการชนกันของระเบียนที่มี Key เป็น Synonym กัน

การจัดการกับปัญหาส่วนของระเบียนที่ Collision

มีได้ 2 แบบ

- **Open Addressing** ภายในเนื้อที่ของแฟ้มสุ่มมีเฉพาะ Home Bucket อย่างเดียว เป็นที่เก็บระเบียนปกติ และระเบียนส่วนที่ล้นปะปนกันไป
- **Separate Overflow** คือส่วน Home Bucket และ Overflow Bucket แยกจากกัน โดย Overflow Bucket จะจัดเก็บระเบียนส่วนที่ล้นโดยเฉพาะ

ข้อดี-ข้อเสีย ของ Separate Overflow

ข้อดี

- สามารถลดปัญหา ระเบียนที่ล้นหรือที่ชนกันไปแย่ง Home Address ของระเบียนอื่นได้

ข้อเสีย

- จะต้องเพิ่มค่าใช้จ่ายในการบำรุงรักษา Separate File

เทคนิคในการแก้ปัญหา Collision

มี 3 แบบ

- Linear Probing หรือ Progressive Overflow
- Double Hashing
- Synonym Overflow หรือ Synonym Chain

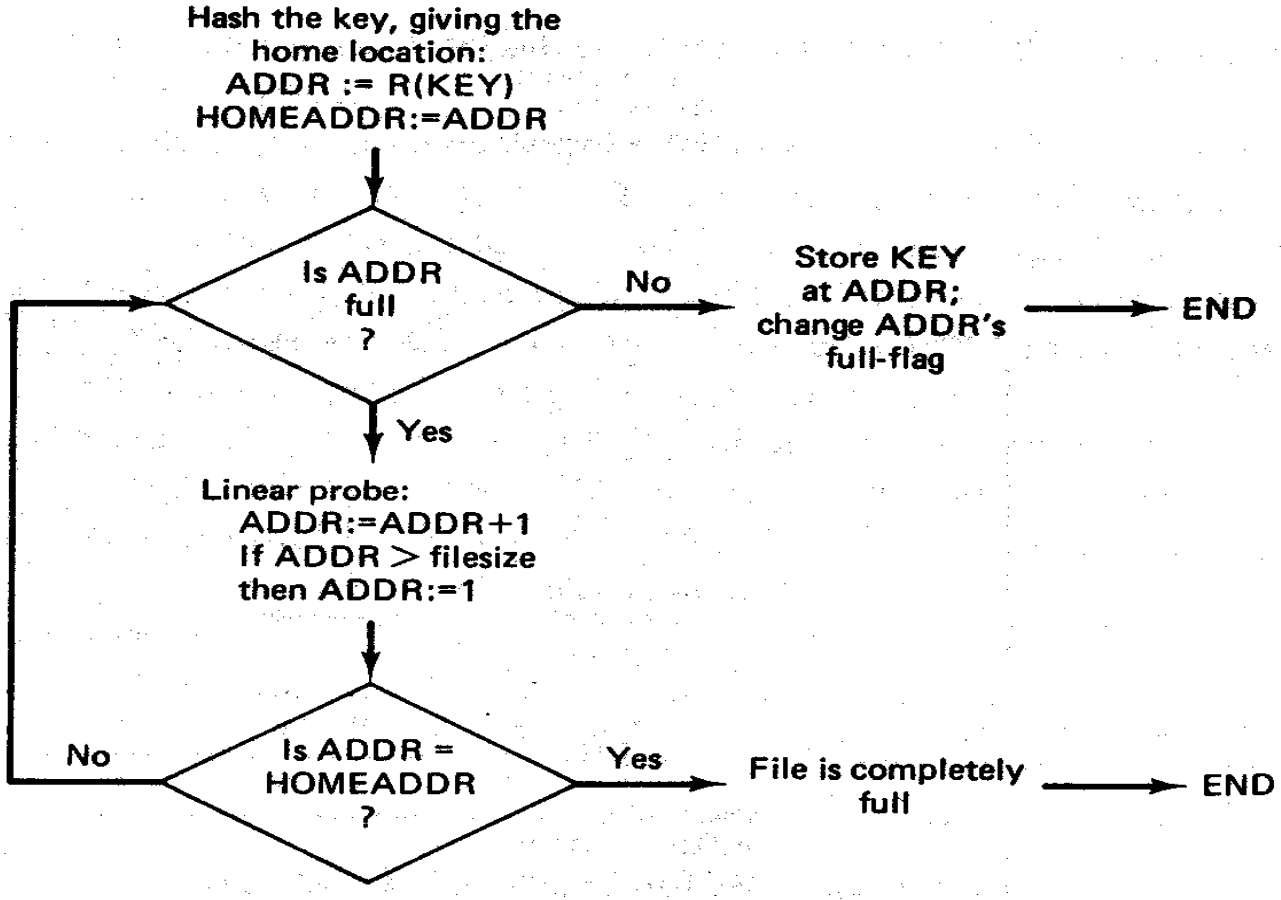


Figure 13–12 Logic for record storage by hashing with linear probing.

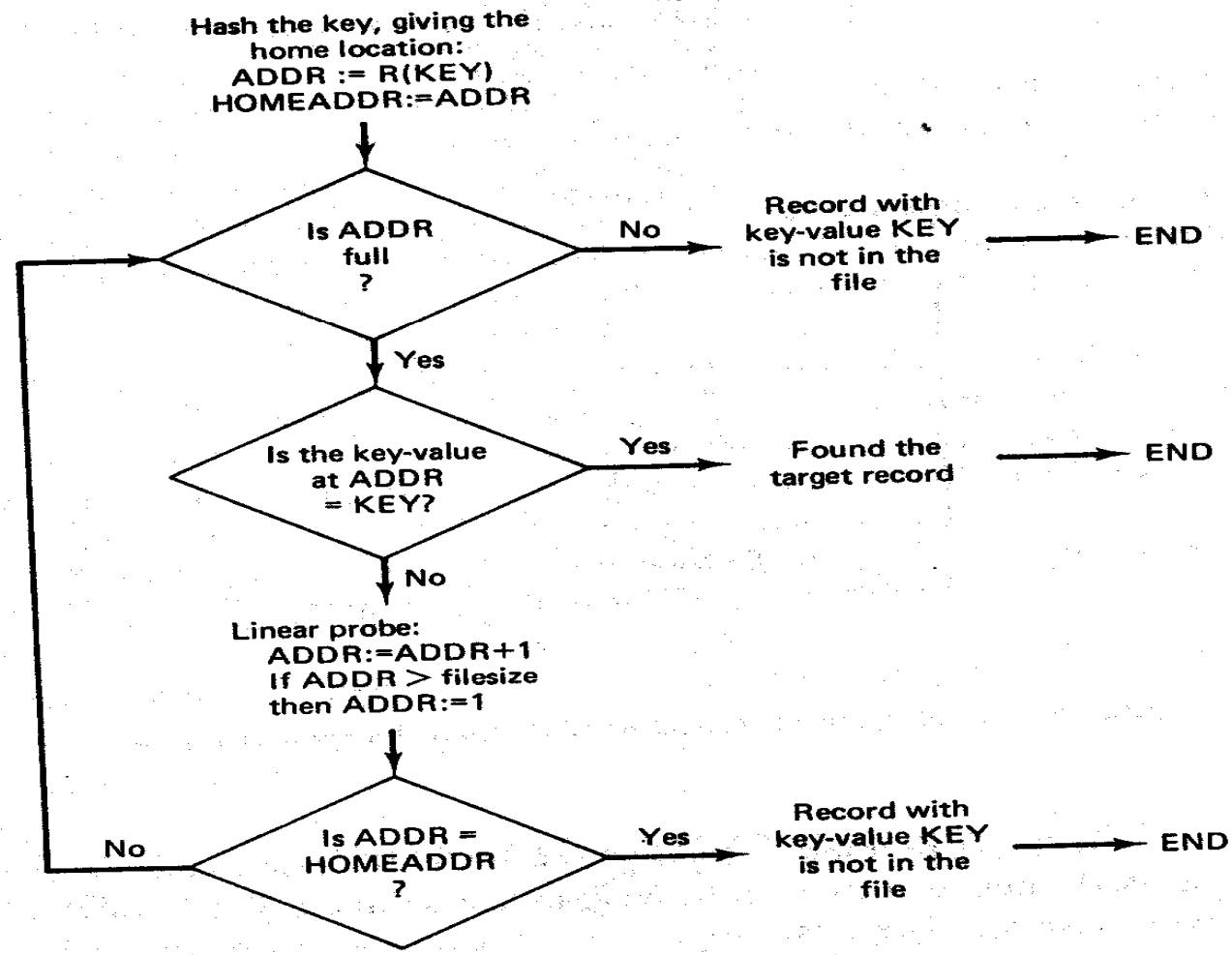


Figure 13–13 Logic for record retrieval by hashing with linear probing.

Linear Probing หรือ Progressive Overflow

- เป็นการแก้ปัญหาง่ายๆ โดยการเก็บระเบียนส่วนที่ล้นไว้ในตำแหน่งถัดจาก Home Bucket อาจจะเป็น Bucket ติดกัน หรือถ้าหากที่ติดกันไม่ว่างก็ขยับไปยัง Bucket ต่อไปอีก

Bucket Capacity = 4 Record

	Home Bucket				
	0	20	160		
	1				
	2	42	22	12	
	3	53	43	23	73
Key 33	4	14	33	24	54
Key 74	5	35	85	74	

Double Hashing

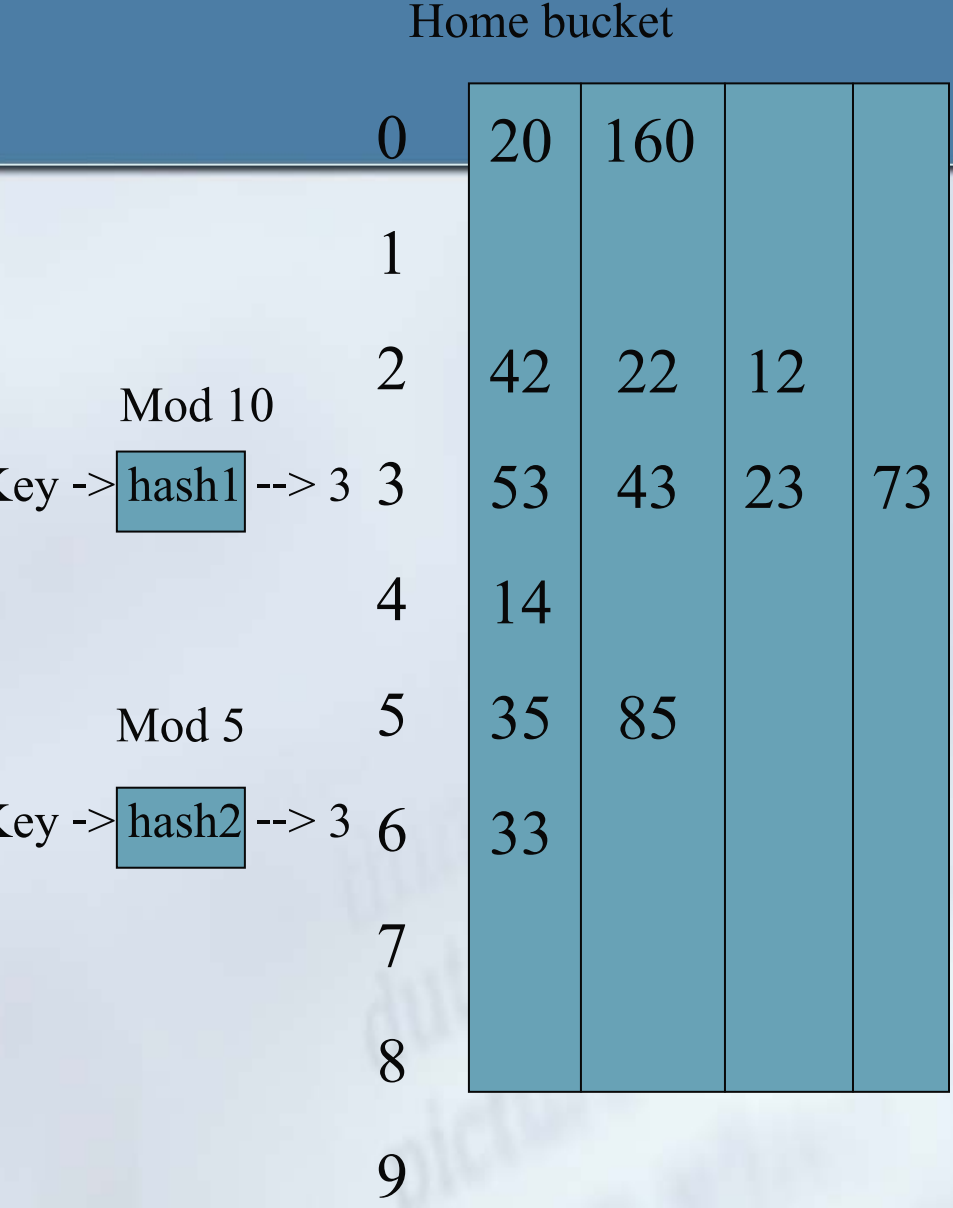
- เป็นการนำระเบียนส่วนที่ล้นไม่สามารถเก็บใน Home Bucket ได้มาใช้ **Hashing Function คำนวณตำแหน่งใหม่อีกครั้งหนึ่ง** และอาจจะนำค่าใหม่ที่ได้บวกกับตำแหน่งเดิมที่คำนวณได้ในครั้งก่อน

Mod 10

Key -> hash1 --> 3

Mod 5

Key -> hash2 --> 3



Double Hashing (ต่อ)

- Hashing ครั้งที่ 2 จะใช้ Function เดิม หรือ Function ใหม่ก็ได้
- การจัดเก็บจะจัดแบบ Open Addressing หรือ Separate ก็ได้
- ในส่วนของ Overflow จะใช้ Linear Probing มาแก้ปัญหาร่วมด้วยก็ได้

Synonym Chaining

- การเอา synonym ทั้งหลายมาโยงต่อกันเป็นลูกโซ่
- ใช้ separate overflow ในการเก็บระเบียนส่วนล้น และเพิ่ม link field ในทุก ๆ home bucket และ overflow bucket ซึ่ง link field เป็นตัวเชื่อม synonym เข้าด้วยกัน

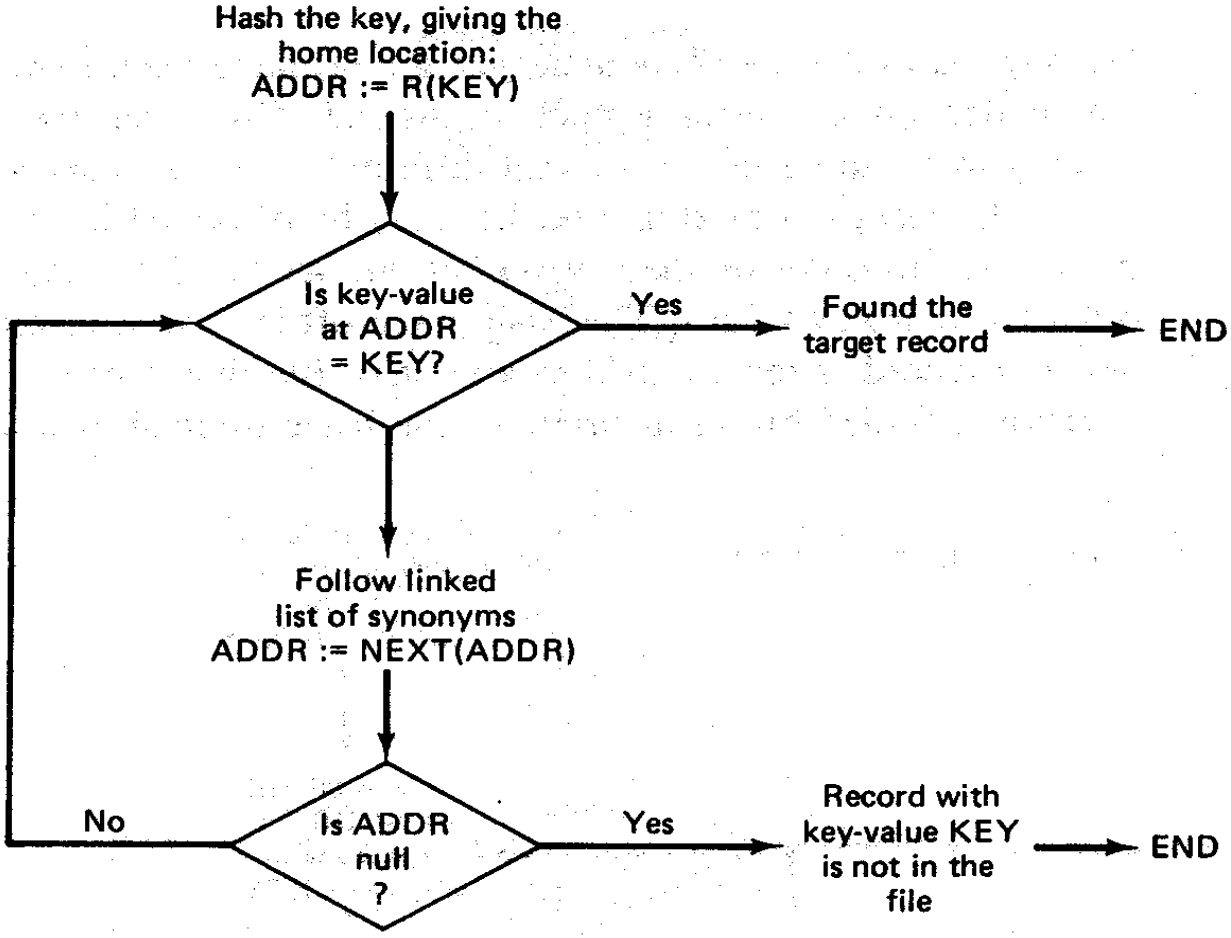
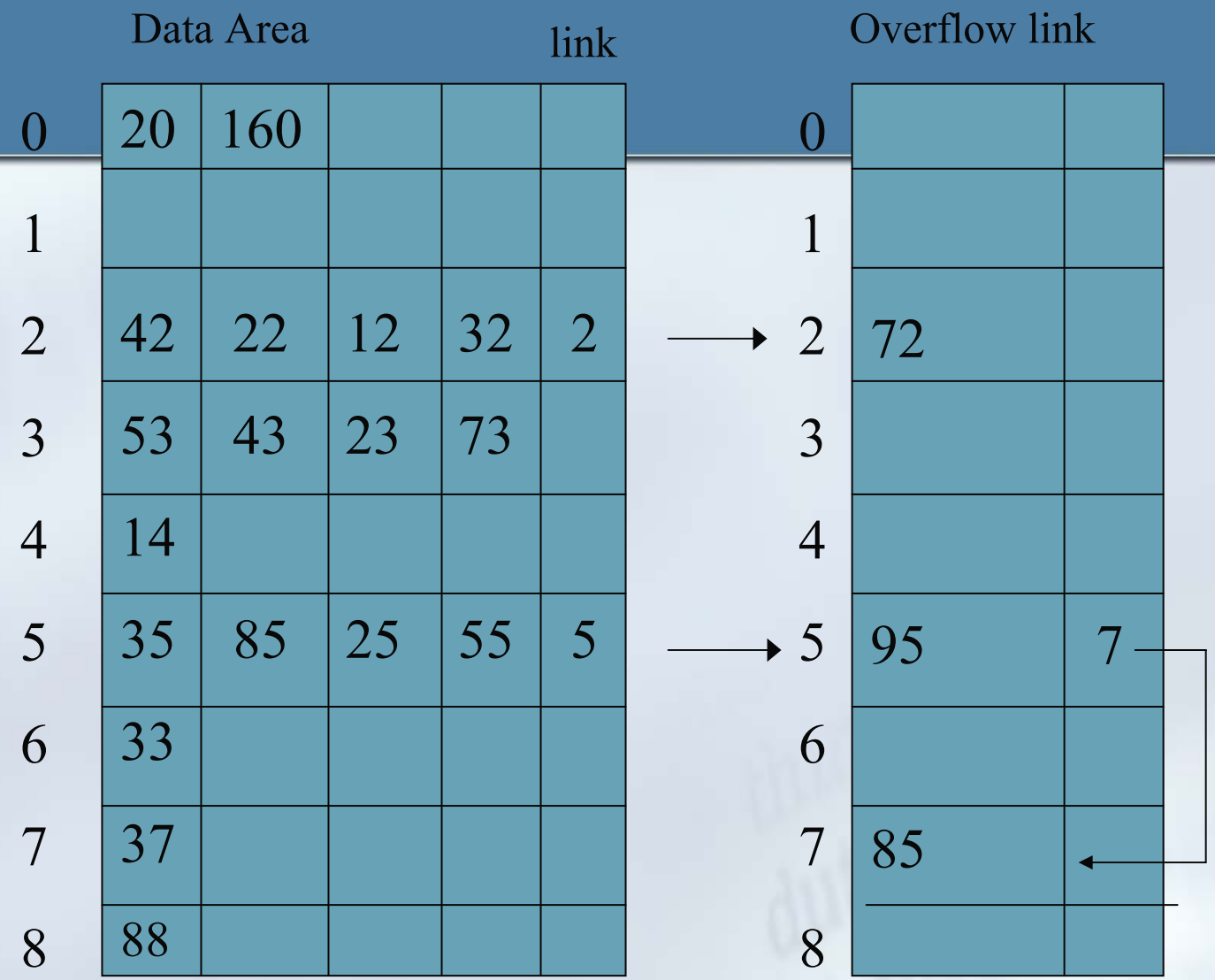


Figure 13–17 Logic for record retrieval by hashing with synonym chaining.



การแก้ปัญหา collision โดยใช้ Synonym Chaining

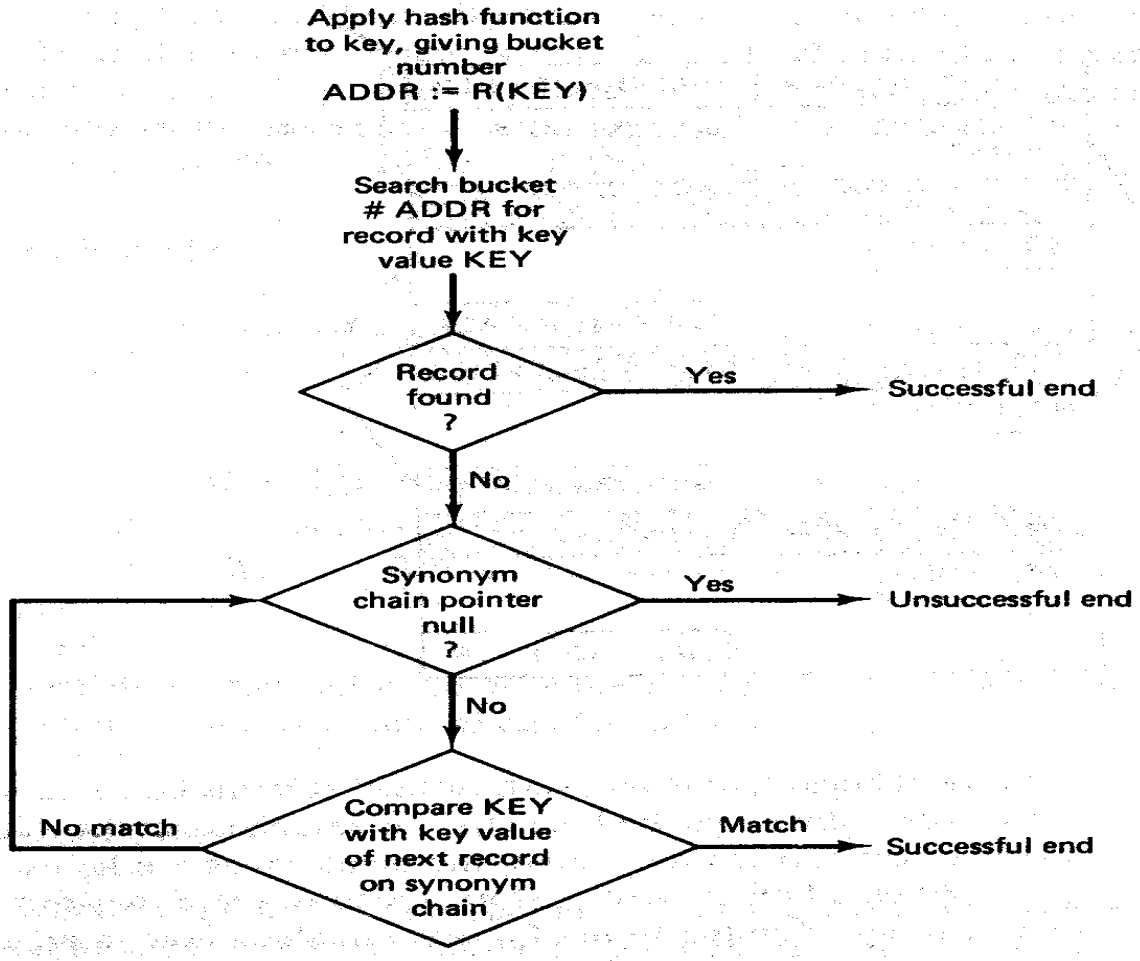


Figure 13–19 Logic for record retrieval by hashing with bucket addressing, with bucket overflow into overflow area and synonym chaining in the overflow area.

การออกแบบแฟ้มข้อมูลแบบสุ่ม

สิ่งที่ต้องคำนึงถึงในการออกแบบ

- มีเขตข้อมูลใดบ้างที่ประกอบกันเป็นระเบียน
  - ควรใช้ Fixed Length Record
- จะใช้เขตข้อมูลใดประกอบกันเป็น Primary Key
  - เขตข้อมูลนั้นจะต้องเป็นตัวบ่งบอกตำแหน่งของระเบียน จะต้องเป็นเขตข้อมูล ที่สามารถจำแนกความแตกต่างของระเบียนได้
- ขนาดของ Bucket ที่เหมาะสมควรจะใช้เท่าไร
  - ควรเลือกตามขนาดของ Block Size เพื่อสามารถอ่านเขียนข้อมูลได้เร็ว

การออกแบบแฟ้มข้อมูลแบบสุ่ม (ต่อ)

ควรจะใช้วิธีการแปลงตำแหน่งชนิดใดจึงจะเหมาะสม

- ถ้า Primary key ไม่เป็นตัวเลข ไม่ควรใช้ Direct Mapping
- ให้เปรียบเทียบช่วงของ key value กับระเบียนที่มีอยู่จริง
  - ถ้ามีจำนวนใกล้เคียงกัน ใช้ Direct Mapping (แบบ Relative Address)
  - ถ้ามีจำนวนแตกต่างกันมาก อาจใช้ Dictionary lookup หรือ Hashing Function
- ถ้าการขยายตัวของแฟ้มข้อมูลสูง ควรใช้ Loading Factor ไม่เกิน 80%
- ถ้าต้องการเข้าถึงข้อมูลเร็วควรใช้ Dictionary Lookup (เร็วกว่า Hashing)
- ถ้ารู้การกระจายของ Key Value สามารถเลือก Hashing Function ที่เหมาะสมได้

การออกแบบแฟ้มข้อมูลแบบสุ่ม (ต่อ)

- จะจัดการกับระเบียนส่วนล้นด้วยวิธีใด จึงจะเหมาะสม
  - Linear Probing : Access ช้า, ประหยัดที่, เขียนโปรแกรมง่าย
  - Double Hashing : Access เร็วกว่า, ประหยัดที่, คำนวณหลายครั้ง
  - Synonym : Access เร็ว, แต่อาจจะต้อง Seek หลายครั้ง
- มีภาษาใดที่อำนวยความสะในการใช้แฟ้มสุ่มบ้าง
  - COBOL, C ตามที่ตนเองถนัด โดยต้องเช็คภาษาเหล่านั้น

ลักษณะงานที่เหมาะสำหรับงานแฟ้มสุ่ม

- งานระบบ Online
- ต้องการปรับปรุงแฟ้มข้อมูลหลายๆชุดพร้อมกัน
- มีอัตราการเปลี่ยนแปลงต่ำ รายการปรับในแต่ละรอบมีน้อย
- ต้องการคำตอบรวดเร็ว
- ตัวอย่างระบบงานที่ต้องใช้แฟ้มชนิดนี้
  - ระบบธนาคารแบบ Online
  - ระบบการสำรองที่นั่งของสายการบิน