บทที่ 13

โครงสร้างแฟ้มข้อมูล

วัตถุประสงค์

หลังจากเรียนจบบทที่ 13 แล้ว นักศึกษาต้องสามารถ:

- **เข้าใจและอธิบายวิธีการเข้าถึงแฟ้มข้อมูลได้**
- **เข้าใจและอธิบายคุณสมบัติของ sequential file**
- **เข้าใจและอธิบายคุณสมบัติของ indexed file**
- **เข้าใจและอธิบายคุณสมบัติของ hashed file**
- **เข้าใจและอธิบายความแตกต่างระหว่าง text file และ binary file**

แฟ้มข้อมูลเบื้องต้น

- **แฟ้มข้อมูล (file)** คือกลุ่มของข้อมูลที่สัมพันธ์กันและถือว่าเป็นหนึ่งหน่วยข้อมูล วัตถุประสงค์เบื้องต้นของแฟ้มข้อมูลคือการจัดเก็บข้อมูล เนื่องจากข้อมูลในหน่วยความจำหลักจะหายไปเมื่อจบโปรแกรมหรือเมื่อปิดเครื่องคอมพิวเตอร์ จึงจำเป็นที่จะต้องเก็บแฟ้มข้อมูลไว้ในสื่อที่ถาวร นอกจากนี้จำนวนข้อมูลในแฟ้มข้อมูลมักจะมีขนาดใหญ่เกินกว่าที่จะเก็บในหน่วยความจำหลักในเวลาใดเวลาหนึ่ง ดังนั้นเราจะต้องทำการอ่านและเขียนบางส่วนของข้อมูลในหน่วยความจำหลัก ในขณะที่ข้อมูลที่เหลือจะยังถูกเก็บไว้ในแฟ้มข้อมูล

แฟ้มข้อมูลเบื้องต้น (ต่อ)

- โดยปกติแฟ้มข้อมูลจะเก็บไว้ในหน่วยความจำสำรอง (auxiliary หรือ secondary storage devices) หน่วยความจำสำรองที่ใช้กันอย่างกว้างขวางคือจานแม่เหล็ก (magnetic disk) กับเทปแม่เหล็ก (magnetic tape) แฟ้ม ข้อมูลที่เก็บอยู่ในหน่วยความจำสำรองสามารถอ่านและเขียนได้ ที่จริงแล้วคีย์บอร์ดก็ถือว่าเป็นแฟ้มข้อมูลด้วย แต่เป็นแฟ้มข้อมูลที่ไม่สามารถเก็บข้อมูลได้อย่างถาวร
- สำหรับวิชานี้ แฟ้มข้อมูลจะหมายถึงกลุ่มของข้อมูลที่จัดเก็บในรูปของเรคคอร์ดที่แต่ละเรคคอร์ดประกอบด้วยฟิลด์อย่างน้อยหนึ่งฟิลด์

13.1

วิธีการเข้าถึงแฟ้มข้อมูล

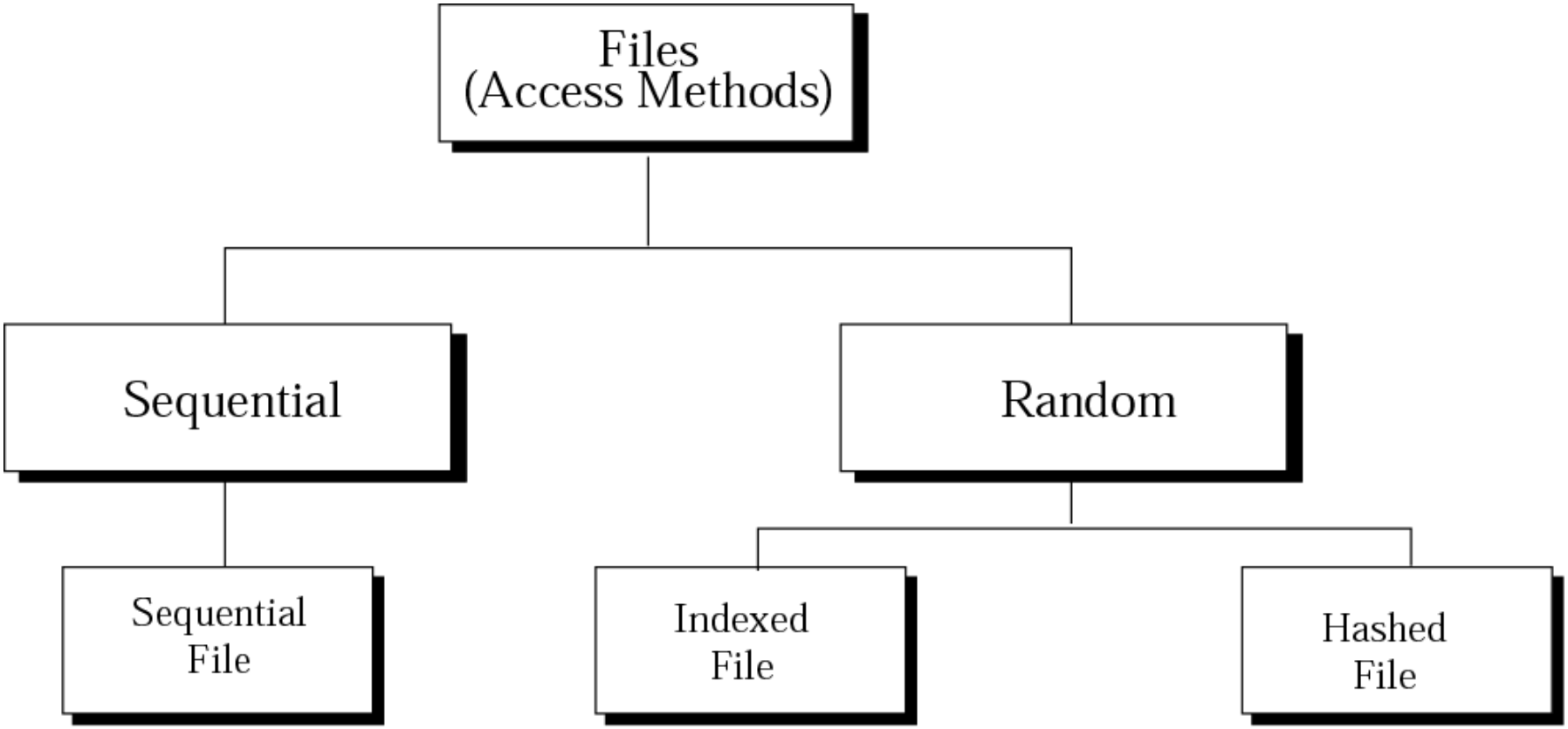
ACCESS METHODS

การเข้าถึงข้อมูล

- เมื่อเราออกแบบแฟ้มข้อมูลเราไม่เพียงแต่สนใจว่าจะเก็บแฟ้มข้อมูลอย่างไร แต่เราจะถามต่อว่าเราจะสามารถดึงหรือค้นคืนข้อมูลในแฟ้มข้อมูลได้อย่างไร บางครั้งเราต้องการประมวลผลทุกๆเรคคอร์ดในแฟ้มข้อมูล และบางครั้งเราก็ต้องการประมวลผลข้อมูลที่อยู่ในเรคคอร์ดที่เฉพาะเจาะจง วิธีการเข้าถึงข้อมูล (access method) จะบอกว่าเรคคอร์ดในแฟ้มข้อมูลจะสามารถดึงออกมาได้อย่างไร โดยปกติการเข้าถึงข้อมูลมี 2 แบบคือการเข้าถึงตามลำดับ (sequential access) กับการเข้าถึงแบบสุ่ม (random access) ซึ่งรายละเอียดมีดังนี้

วิธีการเข้าถึงแฟ้มข้อมูล

- 1. การเข้าถึงตามลำดับ ถ้าเราต้องการเข้าถึงแฟ้มข้อมูลตามลำดับของเรคคอร์ดที่จัดเรียงอยู่ในแฟ้มข้อมูลจากเรคคอร์ดแรกถึงเรคคอร์ดสุดท้าย เราจะใช้แฟ้มข้อมูลที่มีโครงสร้างเป็นแบบ sequential file
- 2. การเข้าถึงแบบสุ่ม ถ้าเราต้องการเข้าถึงเรคคอร์ดใดเรคคอร์ดหนึ่งโดยตรงโดยไม่ต้องผ่านทุกๆเรคคอร์ดก่อนหน้า เราจะใช้แฟ้มข้อมูลที่มีโครงสร้างแบบ random access ซึ่งมี 2 ลักษณะคือ แฟ้มข้อมูลแบบดัชนี (indexed file) กับแฟ้มข้อมูลแบบแฮช (hashed file) การแบ่งประเภทของแฟ้มข้อมูลแสดงตามรูปที่ 13.1



รูปที่ 13-1 การแบ่งประเภทของแฟ้มข้อมูล

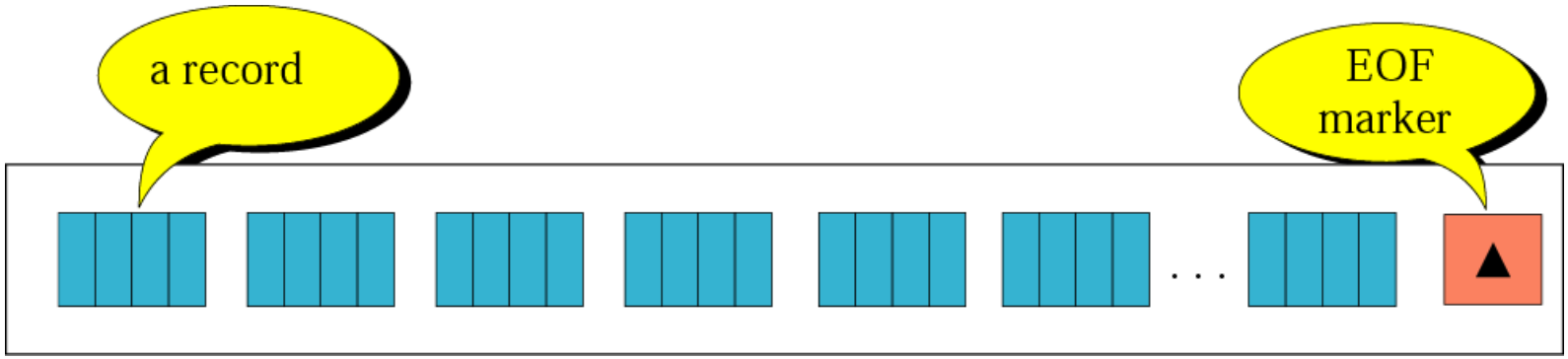
13.2

แฟ้มข้อมูลลำดับ

SEQUENTIAL FILES

แฟ้มข้อมูลลำดับ

- แฟ้มข้อมูลลำดับคือแฟ้มข้อมูลที่เรคคอร์ดทั้งหมดสามารถเข้าถึงได้โดยวิธีการเข้าถึงตามลำดับเท่านั้น นั่นคือเข้าถึงเรคคอร์ดที่ 1 เรคคอร์ดที่ 2 ... เรคคอร์ดสุดท้าย ตามลำดับ รูปที่ 13.2 แสดงโครงร่างของแฟ้มข้อมูลลำดับ เรคคอร์ดทั้งหมดถูกจัดเก็บตามลำดับในหน่วยความจำสำรอง (เทป หรือ ดิสค์) และมีเครื่องหมาย **EOF** (end-of-file) หลังเรคคอร์ดสุดท้าย ระบบปฏิบัติการไม่รู้อะไรเกี่ยวกับแฟ้มข้อมูลเลย ยกเว้นรู้ว่าในแฟ้มข้อมูลแบบลำดับนั้น เรคคอร์ดจัดเรียงกันตามลำดับเท่านั้น



รูปที่ 13-2 แฟ้มข้อมูลลำดับ

การเข้าถึงแฟ้มข้อมูลลำดับ

- แฟ้มข้อมูลแบบลำดับใช้กับการประยุกต์ที่ต้องการเข้าถึงเรคคอร์ดทั้งหมดจากเรคคอร์ดแรกจนถึงเรคคอร์ดสุดท้าย เช่นในแฟ้มข้อมูลพนักงานโดยทั่วไป ตอนสิ้นเดือนที่มีการประมวลผลเงินเดือน ทุกๆเรคคอร์ดที่แทนข้อมูลพนักงานจะต้องถูกเข้าถึงเพื่อประมวลผลตามลำดับ เมื่อประมวลผลเสร็จ ถ้าองค์กรกำหนดให้จ่ายเงินเดือนพนักงานเป็นเช็ค ระบบก็จะพิมพ์เช็คพร้อมรายละเอียดเงินเดือนตามลำดับ แต่ถ้าองค์กรกำหนดให้จ่ายเงินเดือนโดยผ่านธนาคาร กระบวนการโอนเงินผ่านธนาคารก็จะดำเนินการตามลำดับเรคคอร์ดของพนักงานเช่นเดียวกันจะเห็นว่าสะดวกกว่าแฟ้มข้อมูลแบบสุ่ม

การเข้าถึงแฟ้มข้อมูลลำดับ (ต่อ)

- อย่างไรก็ตาม แฟ้มข้อมูลแบบลำดับไม่เหมาะสำหรับการเข้าถึงแบบสุ่ม นั่นคือหากเราต้องการเข้าถึงเรคคอร์ดใดเรคคอร์ดหนึ่งโดยเฉพาะ เช่นถ้าข้อมูลลูกค้าของธนาคารจัดเก็บในแฟ้มข้อมูลแบบลำดับ เมื่อลูกค้าไปกด ATM เพื่อถอนเงิน ลูกค้าจะต้องรอให้ระบบทำการอ่านและตรวจสอบเรคคอร์ดทั้งหมดที่อยู่ก่อนหน้าเรคคอร์ดลูกค้าคนนั้น ลองคิดดูว่าถ้าธนาคารมีลูกค้าอยู่ 2,000,000 คน โดยเฉลี่ยแล้วเรรคอร์ดจะถูกเข้าถึงประมาณ 1,000,000 เรคคอร์ด ก่อนที่เรคคอร์ดที่ต้องการจะถูกเข้าถึงจะเห็นว่าเสียเวลามากและไม่มีประสิทธิภาพเลย
- รหัสเทียมในรูปที่ 13.1 แสดงการเข้าถึงเรคคอร์ดในแฟ้มข้อมูลลำดับ

```
While Not EOF
{
    Read the next record
    Process the record
}
```

รูปที่ 13.1 การประมวลผลเรคคอร์ดในแฟ้มข้อมูลลำดับ

การปรับปรุงแก้ไขข้อมูลในแฟ้มข้อมูลลำดับ

- แฟ้มข้อมูลลำดับจะต้องมีการปรับปรุงแก้ไข (update) เป็นระยะๆ เพื่อสะท้อนถึงการเปลี่ยนแปลงของข้อมูล กระบวนการปรับปรุงแก้ไขเป็นส่วนสำคัญเพราะทุกๆเรคคอร์ดในแฟ้มข้อมูลจะต้องถูกตรวจสอบและแก้ไขตามลำดับ
- แฟ้มข้อมูลที่เกี่ยวข้องกับการปรับปรุงแก้ไขมีอยู่ทั้งหมด 4 แฟ้มคือ

  1. **New Master File**: เป็นแฟ้มข้อมูลที่เก็บข้อมูลใหม่ล่าสุด
  2. **Old Master File**: เป็นแฟ้มข้อมูลที่จะทำการแก้ไข
  3. **Transaction File**: เป็นแฟ้มข้อมูลที่เก็บรายการที่จะทำการเปลี่ยน

การปรับปรุงแก้ไขข้อมูลในแฟ้มข้อมูลลำดับ(ต่อ)

- แปลงกับ master file โดยปกติการเปลี่ยนแปลงมี 3 รูปแบบคือ

  3.1 Add transaction เก็บข้อมูลของเรคคอร์ดใหม่ที่จะเพิ่มเข้าไปเก็บใน master file

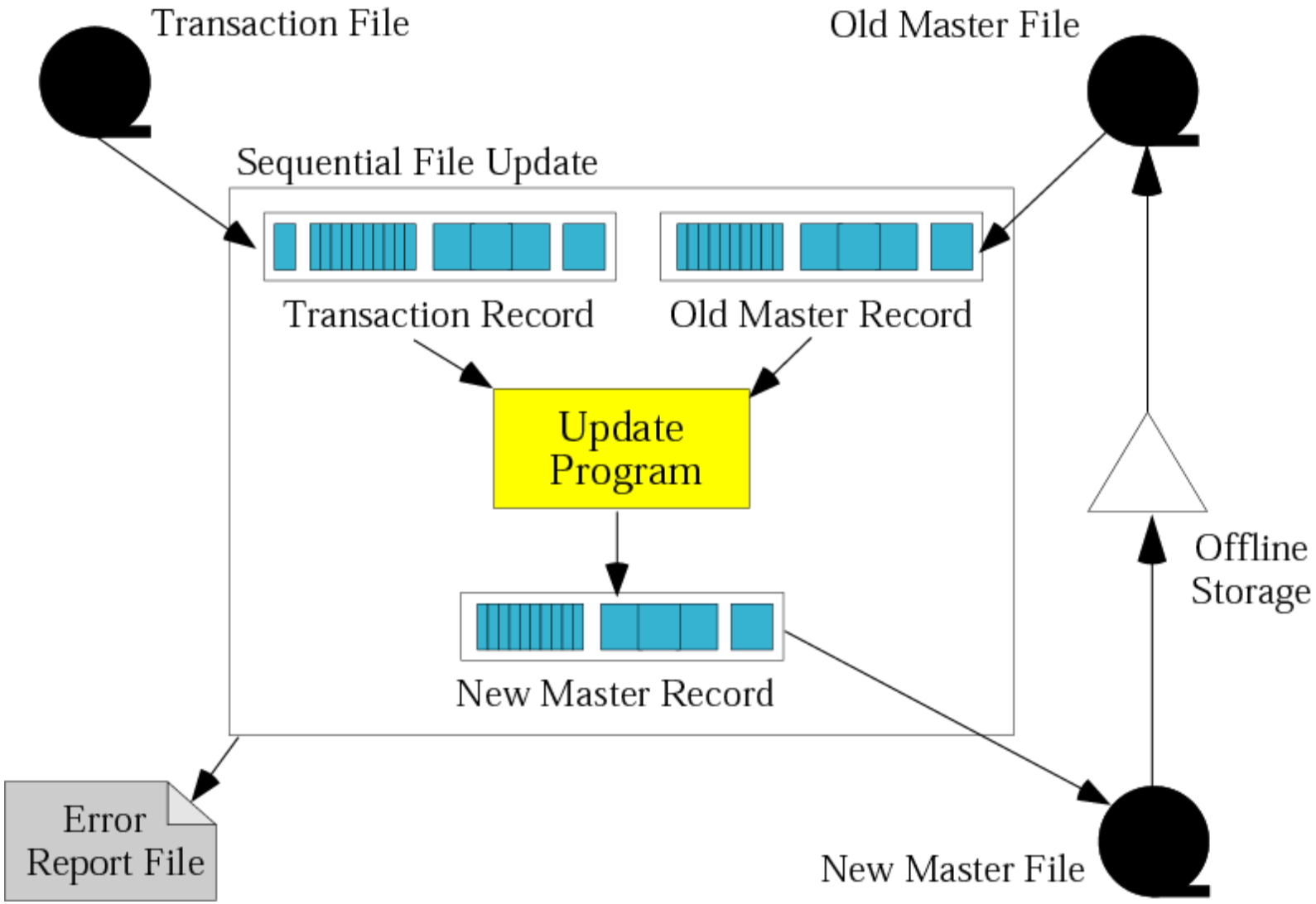
  3.2 Delete transaction ระบุรายละเอียดของเรคคอร์ ดที่จะลบออกจาก master file

  3.3 Change transaction เก็บรายละเอียดข้อมูลใหม่ที่จะใช้สำหรับการเปลี่ยนแปลงกับเรคคอร์ดใน master file

  การปรับปรุงแก้ไขใดๆก็ตามจะต้องใช้คีย์ (key) เป็นตัวค้นหาเรคคอร์ด

การปรับปรุงแก้ไขข้อมูลในแฟ้มข้อมูลลำดับ(ต่อ)

- **4. Error Report Files: เป็นแฟ้มข้อมูลที่ใช้เก็บความผิดพลาดที่เกิดจากการปรับปรุงแก้ไขข้อมูลเพื่อรายงานให้ผู้ใช้ทราบและแก้ไข**
- **กระบวนการปรับปรุงแก้ไขแฟ้มข้อมูลแบบลำดับแสดงดังรูปที่ 13.3 ในรูปนี้จะเห็นว่ามีแฟ้มข้อมูลที่เกี่ยวข้องอยู่ 4 ชนิดดังที่ได้อธิบายมาแล้ว สัญลักษณ์ที่เป็นเทปอาจเป็นดิสค์ก็ได้ นอกจากนี้จะเห็นว่าหลังจากการปรับปรุงเสร็จ new master file จะถูกจัดเก็บแบบ offline และจะเก็บไว้อย่างนี้จนกว่าจะต้องการนำมาใช้อีก เมื่อต้องการจะปรับปรุงแก้ในอนาคต แฟ้มข้อมูลนี้จะถูกนำมาจากที่เก็บ offline และจะกลายเป็น old master file**



รูปที่ 13-3 การปรับปรุงแก้ไขข้อมูลในแฟ้มข้อมูลลำดับ

การปรับปรุงแก้ไขข้อมูลในแฟ้มข้อมูลลำดับ(ต่อ)

- เพื่อให้การปรับปรุงแก้ไขแฟ้มข้อมูลมีประสิทธิภาพ แฟ้มข้อมูลทั้งหมดที่เกี่ยวข้องจะต้องจัดเก็บเรรคอร์ดที่ใช้คีย์เดียวกัน กระบวนการการปรับปรุงแก้ไขต้องใช้การเปรียบเทียบคีย์ระหว่างคีย์ใน transaction file กับคีย์ใน old master file สมมติว่าไม่มีความผิดพลาดเกิดขึ้น การกระทำ 1 ใน 3 จะเกิดขึ้น:

  1. ถ้าคีย์ใน transaction file มีค่าน้อยกว่าค่าของคีย์ใน master file ให้ทำการเพิ่มเรคคอร์ดเข้าไปใน master file
  2. ถ้าคีย์ใน transaction file มีค่าเท่ากับค่าของคีย์ใน master file ให้กระทำหนึ่งในสองกรณีต่อไปนี้

การปรับปรุงแก้ไขข้อมูลในแฟ้มข้อมูลลำดับ(ต่อ)

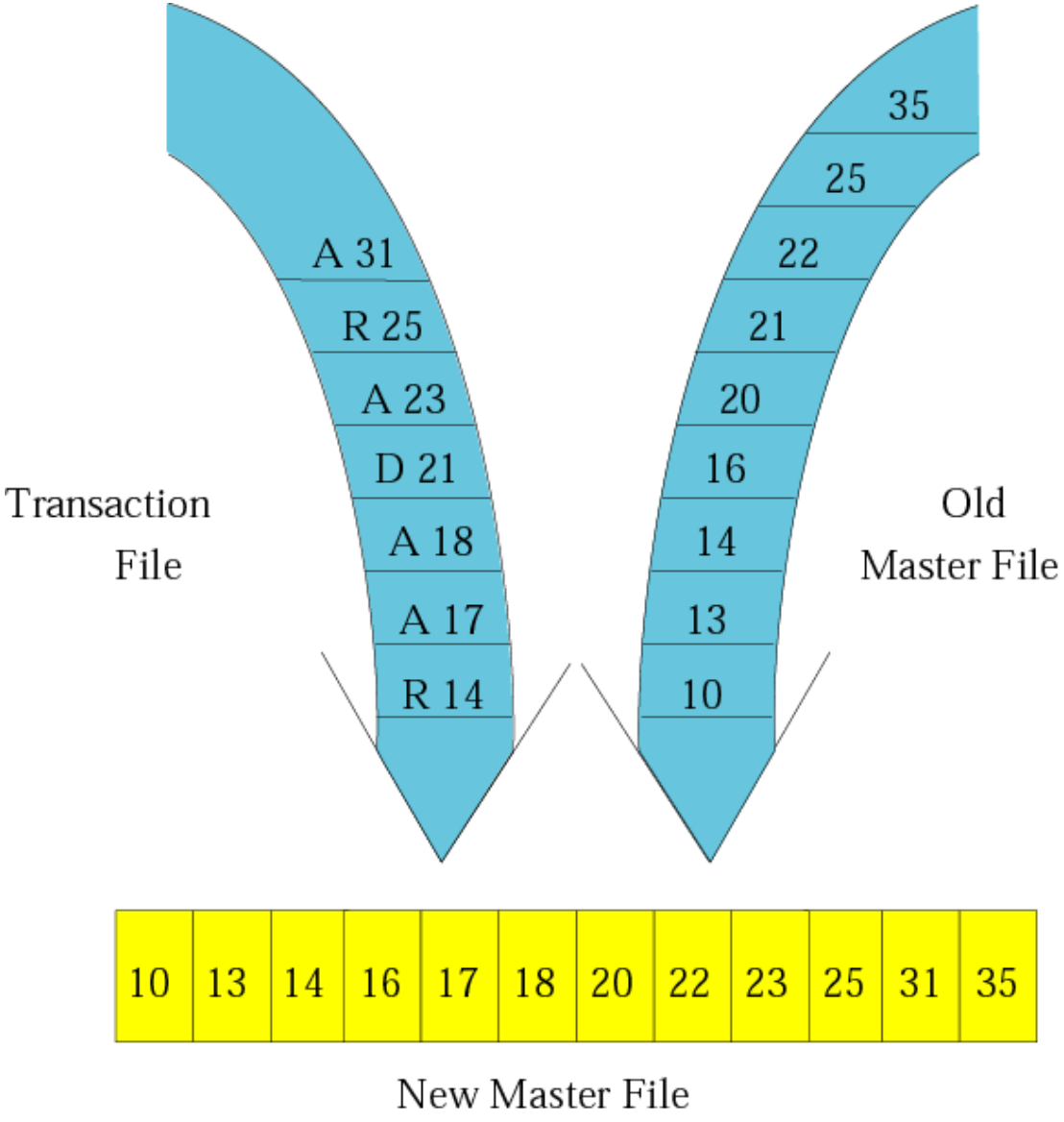
2.1 เปลี่ยนแปลงข้อมูลใน master file ถ้ารหัส transaction เป็น R

2.2 ตัดเรคคอร์ดใน master file ออก ถ้ารหัส transaction เป็น D

3. ถ้าคีย์ใน transaction file มีค่ามากกว่าค่าของคีย์ใน master file

ให้ทำการ copy เรคคอร์ดของ old master file ไปเก็บลงใน new master file ในกรณีนี้ รหัส transaction ต้องเป็น A (addition) มิฉะนั้นก็จะเกิดความผิดพลาดขึ้น

ตัวอย่างกระบวนการปรับปรุงแก้ไขแสดงในรูปที่ 13.4 จากรูป รหัส A แทน add, รหัส D แทน delete, และรหัส R แทน revise. กระบวนการเริ่มขึ้นด้วยการเปรียบเทียบคีย์ตัวแรกของทั้งสองแฟ้มข้อมูล



รูปที่ 13-4 กระบวนการแก้ไขปรับปรุง

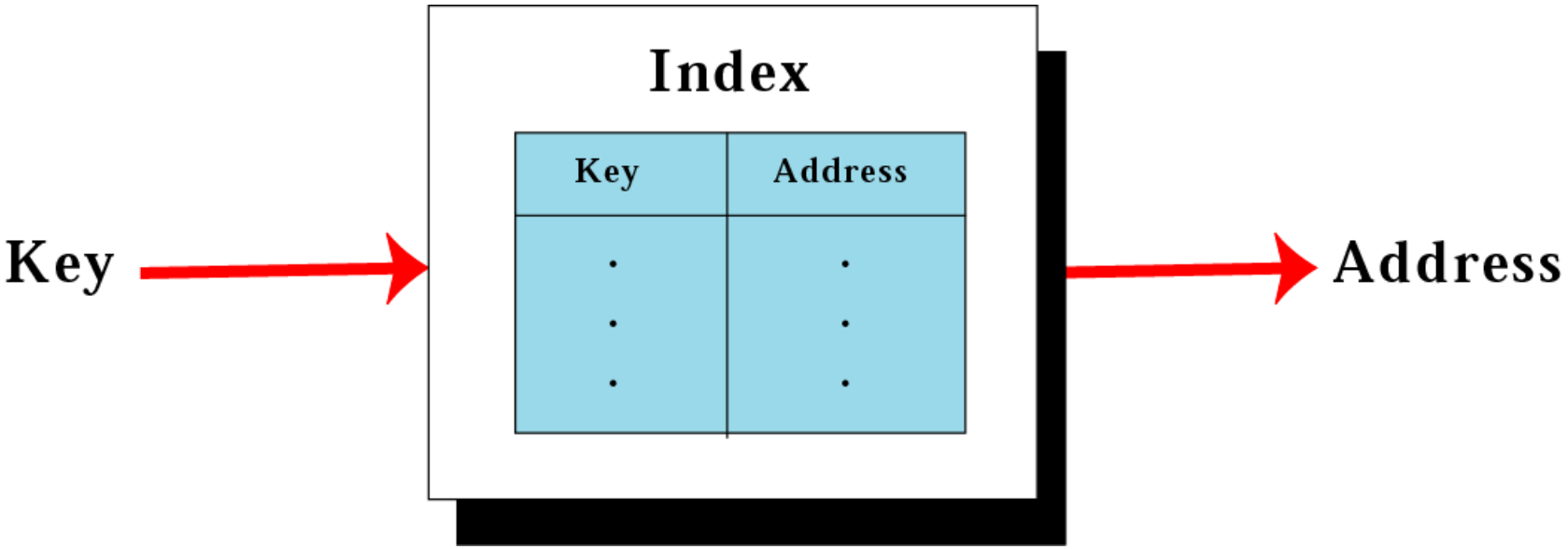
13.3

แฟ้มข้อมูลดัชนี

INDEXED FILES

แฟ้มข้อมูลดัชนี

การเข้าถึงเรคคอร์ดในแฟ้มข้อมูลแบบสุ่ม เราจะต้องทราบตำแหน่งที่อยู่ (address) ของเรคคอร์ดนั้น ตัวอย่างเช่นสมมติว่าลูกค้าต้องการตรวจสอบบัญชีธนาคารของตนว่ามียอดคงเหลือเท่าไร ทั้งเจ้าหน้าที่ธนาคารและลูกค้าต่างก็ไม่ทราบตำแหน่งที่อยู่ของเรคคอร์ดในแฟ้มข้อมูล ในทางปฏิบัติลูกค้าจะบอกเลขที่บัญชี (key) กับเจ้าหน้าที่ ณ จุดนี้แฟ้มข้อมูลแบบดัชนีจะบอกความสำพันธ์ระหว่างเลขที่บัญชี(key) กับตำแหน่งที่อยู่ของเรคคอร์ดในแฟ้มข้อมูลได้ (ดังรูปที่ 13.5)



รูปที่ 13-5 การจับคู่ในแฟ้มข้อมูลดัชนี

แฟ้มข้อมูลดัชนี (ต่อ)

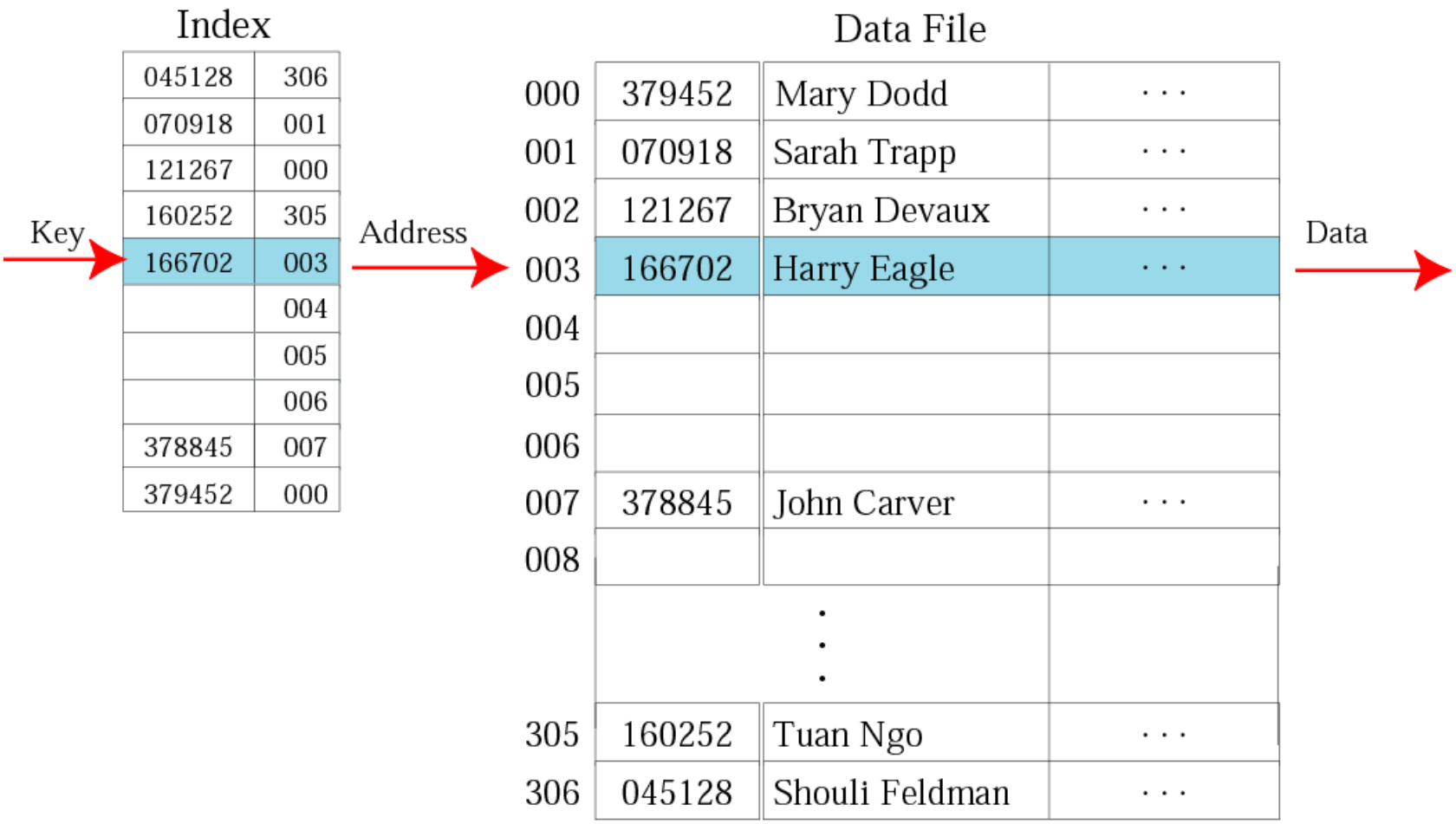
- แฟ้มข้อมูลดัชนีทำจากแฟ้มข้อลำดับมูลปกติผสมกับดัชนี ซึ่งดัชนีเองก็เป็นแฟ้มข้อมูลขนาดเล็กที่มีสองฟิลด์คือคีย์ของแฟ้มข้อมูลลำดับและตำแหน่งที่อยู่ของเรคคอร์ดในหน่วนความจำบนหน่วยความจำสำรอง รูปที่ 13.6 แสดงให้เห็นถึงแนวคิดเชิงตรรกะ(logical view) ของแฟ้มข้อมูลดัชนี การเข้าถึงเรคคอร์ดใดๆในแฟ้มข้อมูล ดำเนินตามขั้นตอนต่อไปนี้

  1. โหลด (load) แฟ้มข้อมูลดัชนีทั้งหมดลงในหน่วยความจำหลัก (ทำได้เพราะแฟ้มข้อมูลดัชนีมีขนาดเล็ก ใช้หน่วยความจำไม่มาก)
  2. ค้นหาข้อมูลในฟิลด์ที่เป็นคีย์เพื่อหาคีย์ที่ต้องการ โดยใช้วิธีการค้นหาที่มีประสิทธิภาพเช่น การสืบค้นแบบไบนารี

แฟ้มข้อมูลดัชนี (ต่อ)

3. ตำแหน่งที่อยู่ของเรคคอร์ดที่ต้องการจะถูกค้นคืนขึ้นมา (ถ้ามี)
4. ใช้ตำแหน่งที่อยู่จากขั้นตอนที่ 3 เพื่อเข้าถึงเรคคอร์ดและส่งข้อมูลที่ต้องการไปยังผู้ใช้

ข้อดีประการหนึ่งของแฟ้มข้อมูลดัชนีคือเราอาจมีมกกว่าหนึ่งดัชนี แต่ละดัชนีจะมีฟิลด์ที่เป็นคีย์ที่ต่างกันตัวอย่างเช่น แฟ้มข้อมูลพนักงานสามารถค้นคืนได้โดยใช้เลขประจำตัวพนักงาน หรือเลขประจำตัวผู้เสียภาษี หรือเลขประจำตัวประชาชน แฟ้มข้อมูลลักษณะนี้เรียกว่า “**Inverted File**”



รูปที่ 13-6 มุมมองเชิงตรรกะของแฟ้มข้อมูลดัชนี

13.4

HASHED FILES

Hashed Files

- ในแฟ้มข้อมูลดัชนีนั้น ดัชนีจะจับคู่ (map) ระหว่างคีย์กับตำแหน่งที่อยู่ (address) ของเรคคอร์ด สำหรับแฟ้มข้อมูลแบบแฮชจะใช้ฟังก์ชั่นทำการจับคู่แทนดัชนี โดยผู้ใช้เป็นผู้ระบุคีย์ ฟังก์ชั่นจะทำการคำนวณหาตำแหน่งที่อยู่ของเรคคอร์ดเพื่อการเข้าถึงข้อมูลต่อไป (ดังรูปที่ 13.7)
- แฟ้มข้อมูลแบบแฮชแตกต่างจากแฟ้มข้อมูลดัชนีคือแฟ้มข้อมูลแบบแฮชไม่มีดัชนี การค้นหาเรคคอร์ดทำได้โดยใช้ฟังก์ชั่นทำการคำนวณหาตำแหน่งที่อยู่ของเรคคอร์ดจากคีย์โดยตรง ทำให้ประหยัดหน่วยความจำที่ไม่ต้องเก็บดัชนี อย่างไรก็ตามแฟ้มข้อมูลแบบแฮชก็ยังมีปัญหาในตัวของมันเองซึ่งจะอธิบายต่อไป

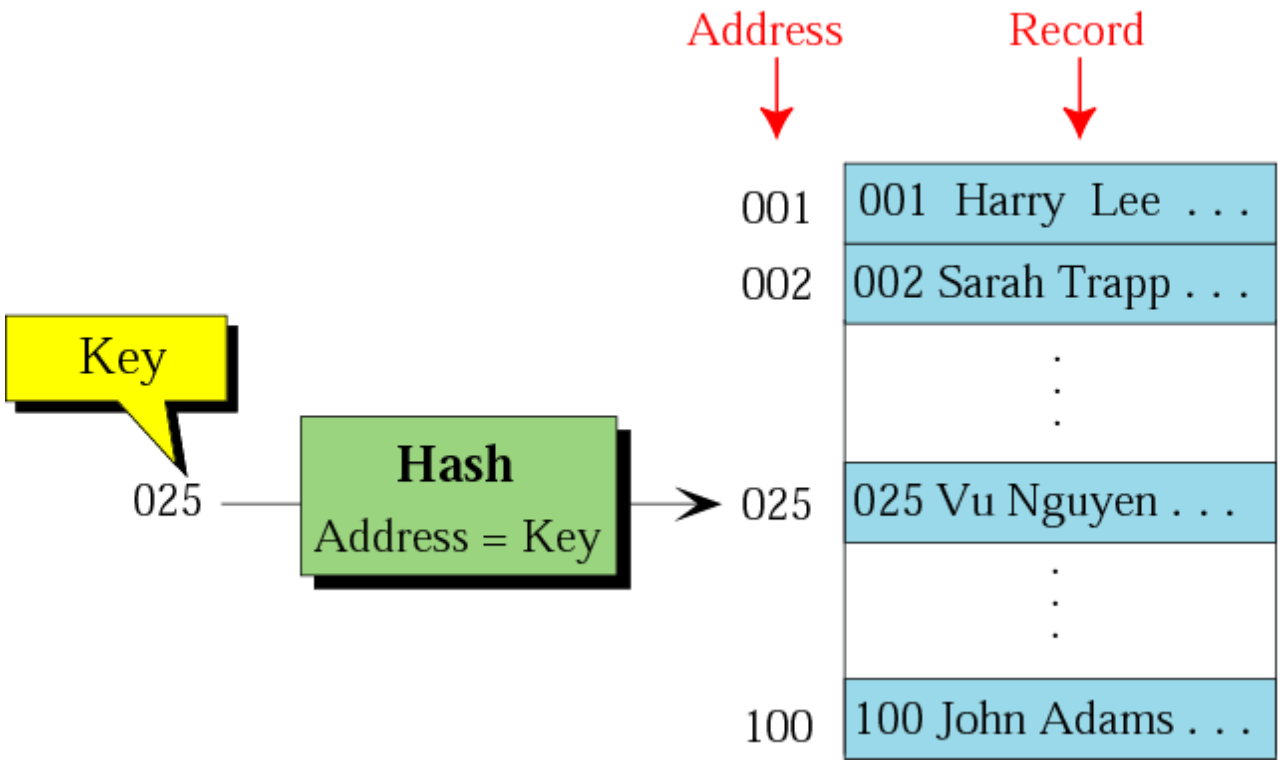
รูปที่ 13-7 การจับคู่ใน hashed file

วิธีการคำนวณหาตำแหน่งที่อยู่

- การคำนวณหาตำแหน่งที่อยู่ของเรคคอร์ดจากคีย์สามารถทำได้หลายวิธี ผู้ใช้สามารถเลือกได้ตามความเหมาะสม ในที่นี้จะอธิบายบางวิธีดังนี้

  1.1 วิธีจับคู่โดยตรง (Directed Hashing) เป็นวิธีที่ใช้คีย์เป็นตำแหน่งที่อยู่โดยตรงโดยไม่มีการคำนวณหรือเปลี่ยนแปลงใดๆ ดังนั้นแฟ้มข้อมูลมีเรคคอร์ดสำหรับทุกๆค่าของคีย์ที่เป็นไปได้ ถึงแม้ว่าวิธีนี้จะมีข้อจำกัดหลายประการ แต่ก็เป็นวิธีที่มีประสิทธิภาพสูงกับการประยุกต์บางประเภท เพราะวิธีนี้ประกันได้ว่าจะไม่มีเรคคอร์ดใดที่จัดเก็บไว้ ณ ตำแหน่งที่อยู่เดียวกัน (เกิดการชนกันของเรคคอร์ด) ซึ่งจะอธิบายต่อไป

- เพื่อจะได้เข้าใจมากขึ้น ขอยกตัวอย่างดังนี้ สมมติว่าหน่วยงานแห่งหนึ่งมีพนักงานน้อยกว่า 100 คนโดยพนักงานแต่ละคนกำหนดให้มีเลขประจำตัวอยู่ในช่วงตั้งแต่ 1 ถึง 100 ในกรณีนี้ ถ้าเราสร้างแฟ้มข้อมูลพนักงานจำนวน 100 เรคคอร์ด เราสามารถใช้เลขประจำตัวพนักงานเป็นตำแหน่งที่อยู่ของเรคคอร์ดของพนักงานแต่ละคนได้เลย ดังแสดงในรูปที่ 13.8 เช่นเรคคอร์ดที่มีคีย์เท่ากับ 025 (Vu Nguyen…) ถูกแฮชไปยังตำแหน่งที่อยู่ 025 (sector บนดิสค์) ขอให้สังเกตว่าในกรณีที่มีจำนวนพนักงานไม่ครบ 100 คน อาจเกิดจากไม่ครบตั้งแต่แรกหรือมีพนักงานลาออก จะมีช่องว่างเกิดขึ้นในแฟ้มข้อมูลซึ่งทำให้สูญเสียเนื้อที่บนดิสค์



รูปที่ 13-8 Direct hashing

วิธีการคำนวณหาตำแหน่งที่อยู่ (ต่อ)

- 1.2 วิธี Modulo Division หรือรู้จักกันในอีกชื่อหนึ่งคือ Division Remainder Hashing (DRH) วิธีนี้จะทำการหารคีย์ของเรคคอร์ดด้วยขนาดของแฟ้มแล้วบวกด้วย 1 ผลลัพธ์ที่ได้ใช้เป็นตำแหน่งที่อยู่ สามารถเขียนเป็นสมการความสัมพันธ์ได้ดังนี้:

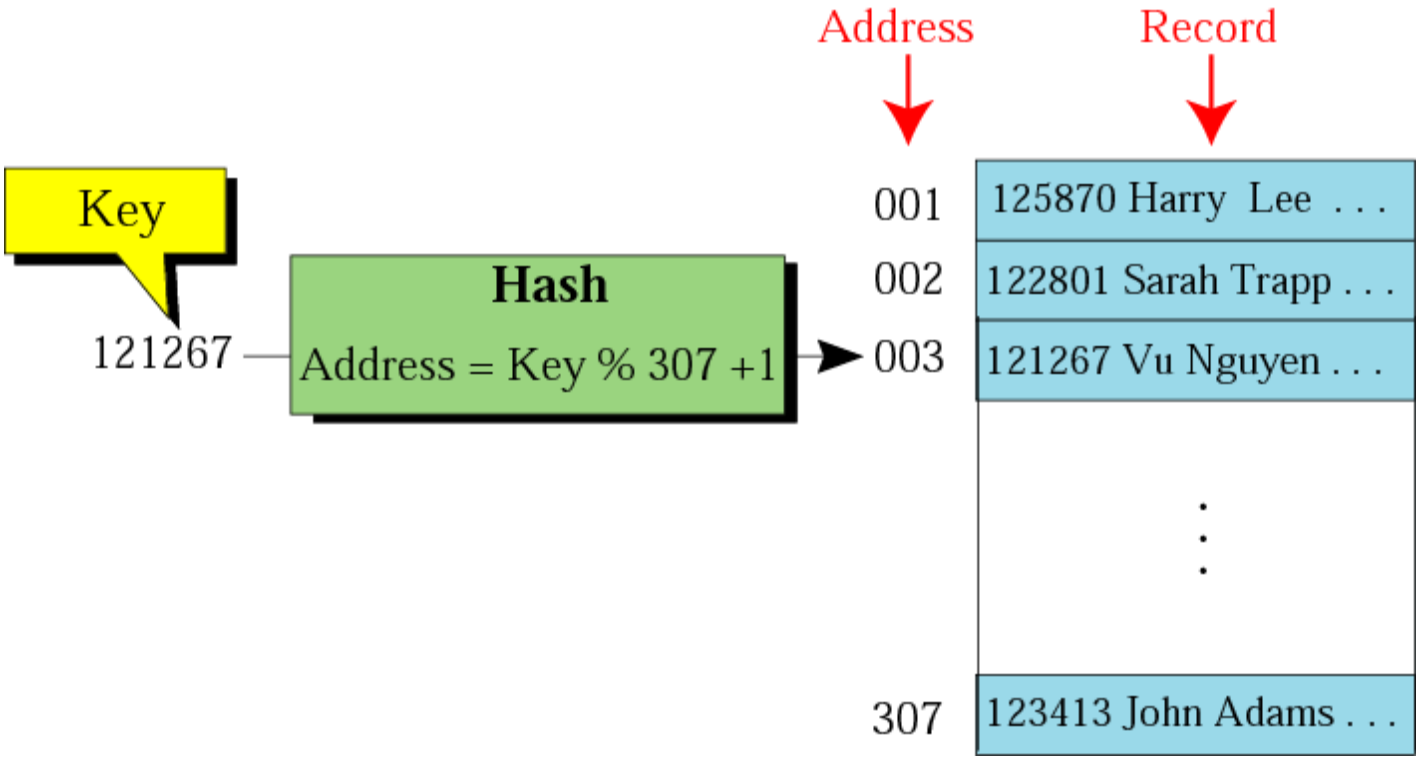
  ตำแหน่งที่อยู่ = (key DIV file-size) + 1

  เมื่อ file-size คือขนาดของแฟ้มข้อมูล (จำนวนเรคคอร์ดทั้งหมด)

  ถึงแม้ว่าวิธี DRH จะใช้ได้กับทุกขนาดของแฟ้มข้อมูล แต่ถ้าขนาดของแฟ้มข้อมูลเป็นเลขจำนวนเฉพาะ จะทำให้เรคคอร์ดต่างๆถูกจับคู่ไปยังตำแหน่งเดียวกันมีน้อยลง (ชนกันน้อยลง)

วิธีการคำนวณหาตำแหน่งที่อยู่ (ต่อ)

- ตัวอย่าง: สมมติว่าท่านก่อตั้งบริษัทขนาดเล็กที่วางแผนว่าในอนาคตจะมีพนักงานมากกว่า 100 คน ท่านอาจวางแผนล่วงหน้าโดยกำหนดให้เลขที่พนักงานสามารถรองรับพนักงานได้ถึง 1 ล้านคน ในอนาคตอันใกล้ท่านตัดสินใจที่จะเตรียมที่ไว้สำหรับพนักงาน 300 คน เลขจำนวนเฉพาะตัวแรกที่มากกว่า 300 คือ 307 ดังนั้นท่านควรเลือกขนาดของแฟ้มข้อมูลเท่ากับ 307 ตัวอย่างแฟ้มข้อมูลแสดงในรูปที่ 13.9 ในกรณีนี้ Vu Nguyen ที่มีคีย์เท่ากับ 121267 ถูกแฮชไปเก็บยังตำแหน่ง 003 เพราะ 121267 DIV 307 +1 = 3 ซึ่งตรงกับตำแหน่งที่อยู่ 003



รูปที่ 13-9 Modulo division

วิธีการคำนวณหาตำแหน่งที่อยู่ (ต่อ)

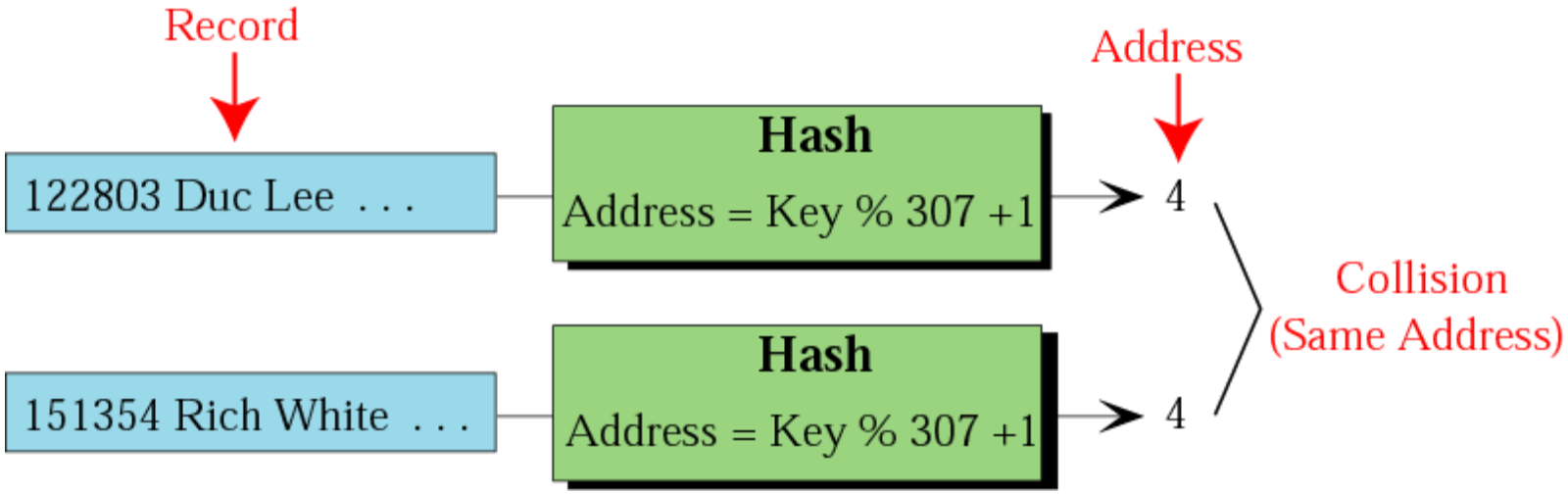
- 1.3 วิธี Digit Extraction เป็นวิธีที่เราเลือกตัวเลขบางตัวจากคีย์เพื่อนำมาเป็นตำแหน่งที่อยู่ของเรคคอร์ด เช่นถ้าเราใช้เลข 6 หลักเป็นรหัสของพนักงาน เพื่อที่จะแปลงไปเป็นตำแหน่งที่อยู่ที่ประกอบด้วยเลข 3 ตัว (000-999) เราสามารถกำหนดว่าเราจะเลือกเลขจากรหัสซึ่งอยู่ตำแหน่งที่ 1, 3, และ 4 จากทางซ้ายมือแล้วใช้เป็น address ดังตัวอย่าง:

  125870 ------> 158

  นอกจากนี้ยังมีวิธีอื่นๆที่นิยมใช้อีกเช่น Midsquare method, Folding method, Rotational method, และ Pseudorandom method เป็นต้น ให้นักศึกษาไปค้นคว้าหารายละเอียดเอง

การชนกัน : Collision

- โดยทั่วไปแล้ว จำนวนคีย์ที่ทำการแฮชจะมีมากกว่าจำนวนเรคคอร์ดในแฟ้มข้อมูล ตัวอย่างเช่นถ้าเรามีแฟ้มข้อมูลของนักศึกษาจำนวน 50 คน โดยที่นักศึกษาแต่ละคนใช้คีย์เป็นเลขสี่ตัวสุดท้ายของเลขประจำตัวประชาชน (มีค่าได้ตั้งแต่ 0000-9999 จำนวน 10,000 ตัว) ดังนั้นจะมี 200 คีย์ (10000/50) ที่เป็นไปได้สำหรับแต่ละตำแหน่งที่อยู่ภายในแฟ้มข้อมูล (ซึ่งมีอยู่ 50 ตำแหน่งเท่านั้น) เนื่องจากมีคีย์หลายคีย์ต่อหนึ่ง address ในแฟ้มข้อมูล เราเรียกกลุ่มของคีย์ที่แฮชไปยัง address เดียวกันว่าเป็นคีย์ที่ synonym กัน เหตุการณ์ที่คีย์ชนกันแสดงอยู่ในรูปที่ 13.10



รูปที่ 13-10 การชนกัน

การแก้ปัญหาการชนกันของเรคคอร์ด

- จากรูปที่ 13.10 เมื่อเราคำนวณหา address ของ 2 เรคคอร์ดที่ต่างกันจะพบว่าได้ค่า address ของทั้งสองเรคคอร์ดเท่ากับ 4 แน่นอนว่าเรคคอร์ดทั้งสองไม่สามารถจัดเก็บที่ address เดียวกันได้ เราจำเป็นจะต้องหาทางแก้สถานการณ์เช่นนี้เพื่อให้เรคคอร์ดทั้งสองสมารถเก็บในแฟ้มข้อมูลได้
- ถ้าข้อมูลที่เรามีอยู่มีเรคคอร์ดที่ synonym กันอยู่ตั้งแต่สองเรคคอร์ดขึ้นไป เราเรียกว่าเกิดสถาการณ์การชนกัน (collision) การชนกันเกิดขึ้นเมื่อฟังก์ชั่นแฮชหรืออัลกอริธึมทำการแฮชเรคคอร์ดที่จะนำเข้าไปจัดเก็บในแฟ้มข้อมูลที่ address นั้นมีเรคคอร์ดอื่นเก็บอยู่ก่อนแล้ว ค่า address ที่คำนวณจากฟังก์ชั่นแฮชเรียกว่า home address

การแก้ปัญหาการชนกันของเรคคอร์ด(ต่อ)

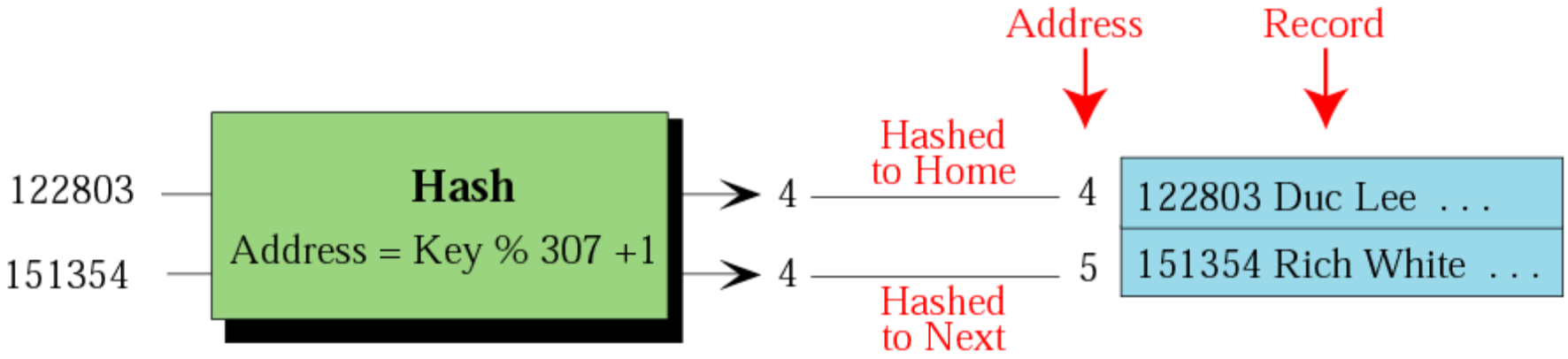
- ส่วนของแฟ้มข้อมูลที่ประกอบด้วย home address ทั้งหมดเราเรียกว่า prime area เมื่อสองคีย์ใดๆเกิดการชนกันขึ้นที่ home address เราจะต้องแก้ปัญหาการชนนี้โดยการเก็บเรคคอร์ดที่นำเข้าไปเก็บไว้ใหม่ ณ ตำแหน่งที่อยู่อื่น
- จากวิธีการคำนวณหาตำแหน่งที่อยู่ที่กล่าวมาแล้ว มีวิธีเดียวเท่านั้นที่ไม่ก่อให้เกิดการชนขึ้น วิธีนั้นคือวิธี Direct method วิธีนอกนั้นมีโอกาสชนทั้งสิ้น วิธีแก้การชนกันของเรคคอร์ดมีหลายวิธี แต่ละวิธีไม่ขึ้นอยู่กับอัลกอริธึมหรือฟังก์ชั่นที่ใช้แฮช นั่นคือเมื่อเกิดการชนกันขึ้น วิธีต่อไปนี้สามารถนำมาใช้ได้ ไม่ว่าเราใช้ฟังก์ชั่นหรืออัลกอริธึมแฮชใดๆก็ตาม

การแก้ปัญหาการชนกันของเรคคอร์ด(ต่อ)

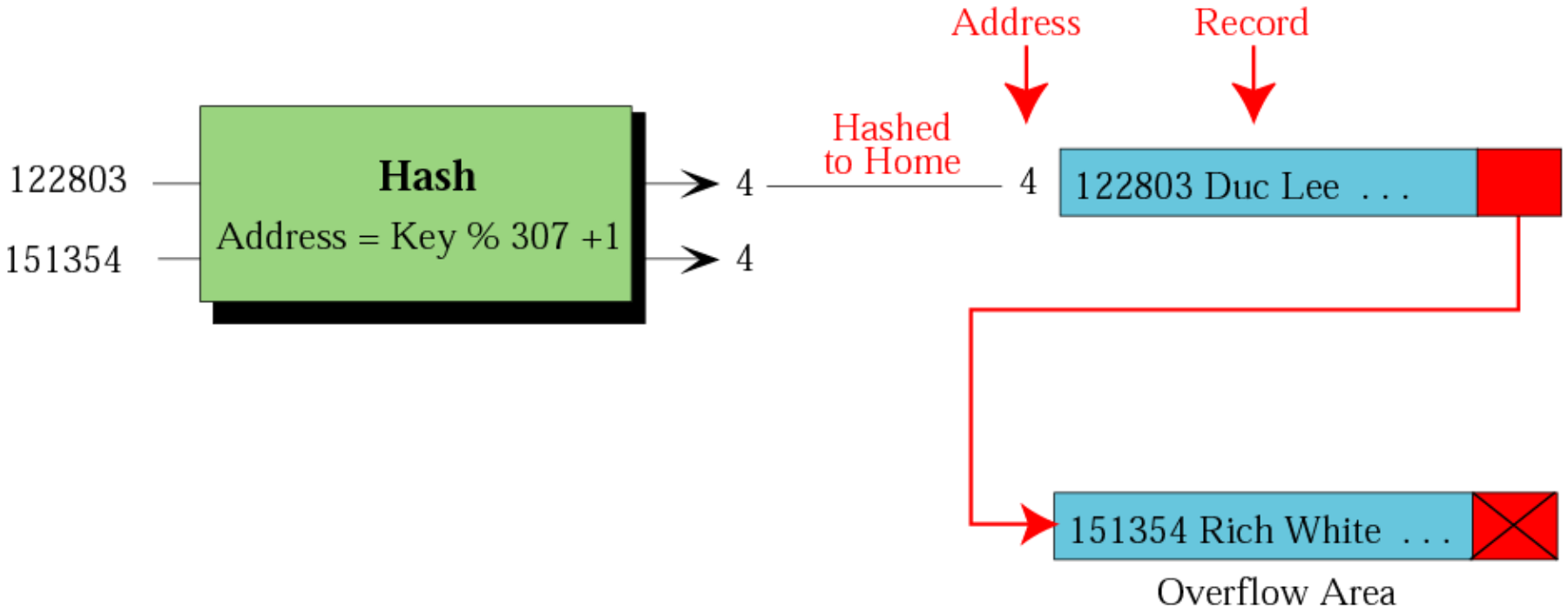
- **1. Open Addressing: เมื่อเกิดการชนกันขึ้น พื้นที่ในส่วน prime area จะถูกค้นหา address ที่ว่างเพื่อที่จะได้นำเรคคอร์ดใหม่เข้าไปเก็บ วิธีที่ง่ายที่สุดคือหา home address ถัดไป (home address + 1) ตัวอย่างในรูปที่ 13.11 แสดงการแก้ปัญหาการชนจากรูปที่ 13.10 โดยใช้วิธีนี้ เรคคอร์ดแรกจัดเก็บที่ address 4 ส่วนเรคคอร์ดที่นำเข้าไปเก็บใหม่จัดเก็บที่ address 5**
- **2. Linked List: ข้อเสียของวิธี Open addressing คือการชนแต่ละครั้งที่เกิดขึ้นทำให้โอกาสที่จะเกิดการชนในครั้งต่อๆไปมีมากขึ้นตามไปด้วย วิธี Linked List จะแก้ปัญหานี้ได้**

การแก้ปัญหาการชนกันของเรคคอร์ด(ต่อ)

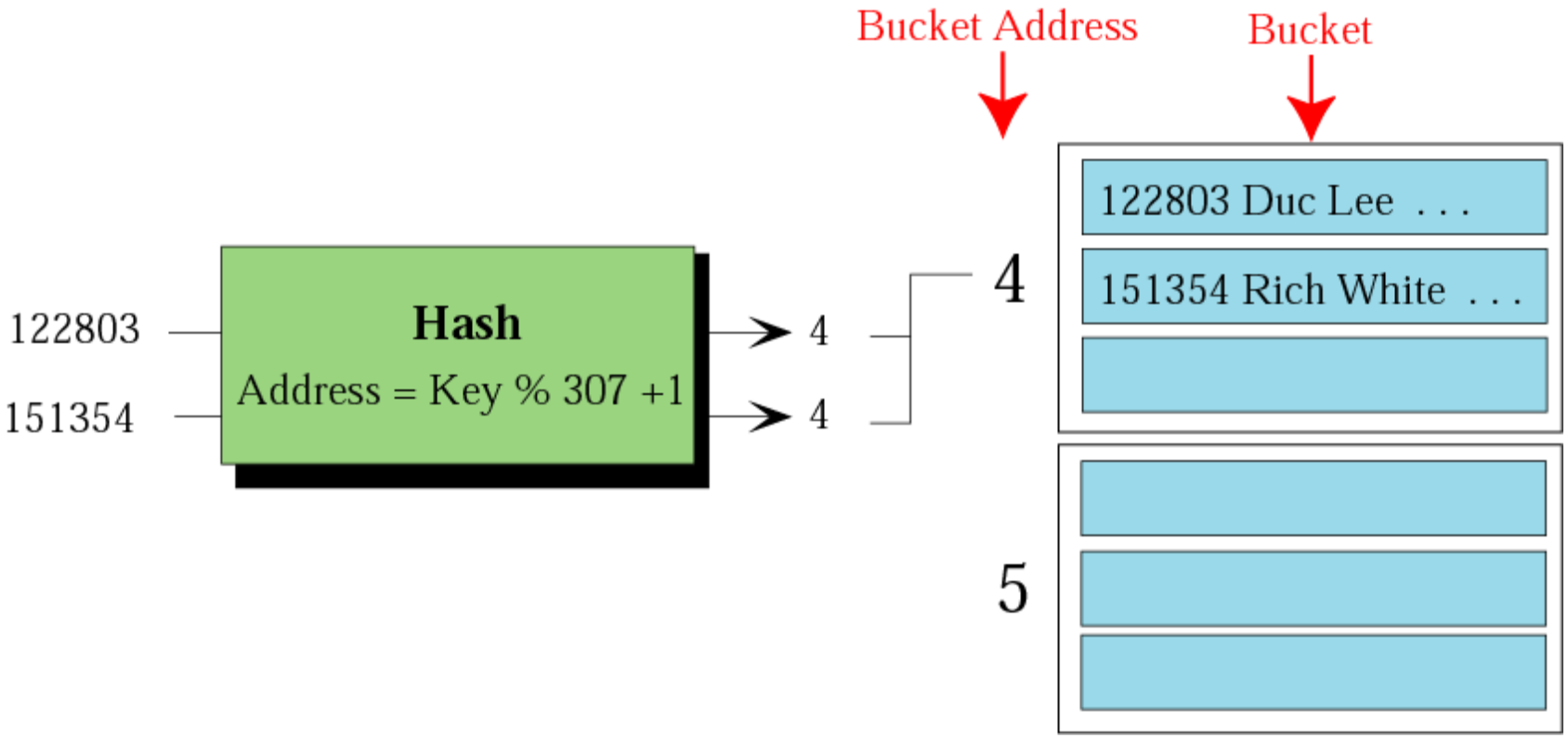
- โดยที่เรคคอร์ดแรกจะเก็บที่ home address และในเรคคอร์ดแรกจะมี pointer ชี้ไปยังเรคคอร์ดที่เข้ามาใหม่ ดังรูปที่ 13.12 (แก้ปัญหาจากรูปที่ 13.10)
- 3. Bucket Hashing: Bucket เปรียบเสมือนถังหรือกล่องที่สามารถเก็บเรคคอร์ดได้มากกว่า 1 เรคคอร์ด วิธี Bucket Hashing เป็นการแฮชคีย์ของเรคคอร์ดไปเก็บไว้ในกล่อง ดังรูปที่ 13.13



รูปที่ 13-11 Open addressing resolution



รูปที่ 13-12 Linked list resolution



รูปที่ 13-13 Bucket hashing resolution

13.5

TEXT FILES และ BINARY FILES

แฟ้มข้อมูลประเภทเท็กซ์ : Text Files

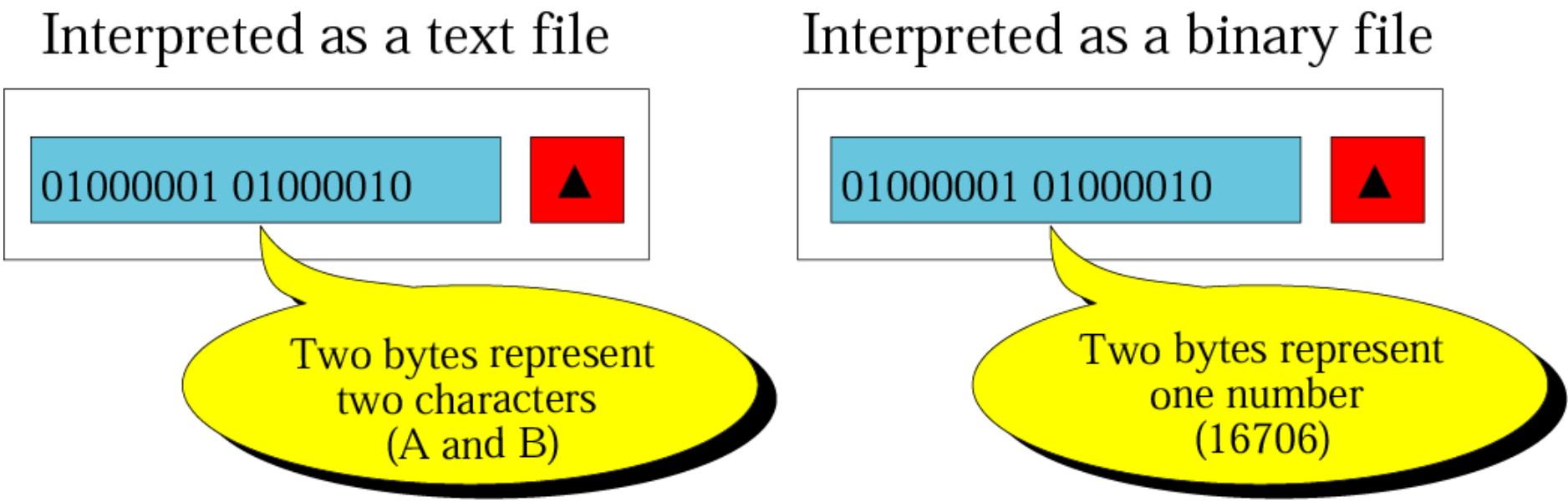
- ก่อนที่จะจบบทนี้ จะอธิบายคำสองคำที่ใช้แบ่งประเภทของแฟ้มข้อมูล ได้แก่ Text files กับ Binary files แฟ้มข้อมูลที่เก็บอยู่ในอุปกรณ์หน่วยความจำมีลักษณะเป็นลำดับของบิตที่เรียงกัน สามารถแปลความหมายโดยโปรแกรมประยุกต์ว่าเป็นแฟ้มข้อมูลประเภทใด ดังรูปที่ 13.14
- Text file คือแฟ้มข้อมูลที่เก็บตัวอักษร (characters) ไม่สามารถเก็บเลขจำนวนเต็ม จำนวนจริง หรือข้อมูลที่มีโครงสร้าง ตามรูปแบบภายในของหน่วยความจำ การเก็บข้อมูลทั้งสามประเภทใน text file จะต้องเปลี่ยนรูป (convert) ไปเป็นตัวอักษรของข้อมูลเหล่านั้น

แฟ้มข้อมูลประเภทเท็กซ์ : Text Files (ต่อ)

- แฟ้มข้อมูลบางชนิดสามารถใช้กับข้อมูลประเภทตัวอักษรเท่านั้น ที่รู้จักกันทั่วไปคือ แฟ้มสายอักษร (streams) ของคีย์บอร์ด จอภาพ (monitors) และ เครื่องพิมพ์ (printers) เป็นต้น อันนี้เป็นเหตุผลว่าทำไมเราจึงต้องใช้ฟังก์ชั่นพิเศษในการจัดรูปแบบ (format) ข้อมูลที่นำเข้าหรือส่งผลไปยังอุปกรณ์ I/O เช่นเมื่อส่งงานไปพิมพ์ยังเครื่องพิมพ์ ตัวเครื่องพิมพ์จะรับข้อมูลทีละ 8 บิต ตีความหมายว่าเป็น 1 ไบท์ แล้วจึงเปลี่ยนเป็นตัวอักษร ตามระบบโค๊ดของเครื่องพิมพ์ (ASCII หรือ EBCDIC) ถ้าตัวอักษรนั้นอยู่ในกลุ่มที่พิมพ์ได้ เครื่องพิมพ์ก็จะพิมพ์ออกมา แต่ถ้าเป็นอย่างอื่น เครื่องพิมพ์ก็จะก็จะทำตามความหมายของอักษรนั้น วนอย่างนี้จนจบ

แฟ้มข้อมูลแบบไบนารี่ : Binary Files

- **Binary file** เป็นแฟ้มข้อมูลที่เก็บข้อมูลที่เป็นรูปแบบภายในของเครื่องคอมพิวเตอร์ ข้อมูลอาจเป็นจำนวนเต็ม จำนวนจริง ตัวอักษร หรือข้อมูลโครงสร้างอื่น binary file แตกต่างจาก text file คือ ข้อมูลใน binary file จะมีความหมายต่อเมื่อมีการตีความจากโปรแกรมประยุกต์เท่านั้น ถ้าข้อมูลเป็นเท็กซ์ แล้ว 1 ไบท์จะใช้แทน 1 ตัวอักษร แต่ถ้าข้อมูลเป็นตัวเลขแล้ว ตัวเลข 1 ค่าจะต้องใช้สองไบท์ขึ้นไปเพื่อแทนเลขนั้น สมมติว่าท่านใช้คอมพิวเตอร์ส่วนบุคคลที่ใช้ 2 ไบท์เพื่อเก็บจำนวนเต็ม ในกรณีนี้เมื่อโปรแกรมอ่านหรือพิมพ์เลขจำนวนเต็ม 1 จำนวน เครื่องจะตี ความว่า 2 ไบท์แทนเลข 1 จำนวน



รูปที่ 13-14 Text and binary interpretations of a file

คำสำคัญ

- Access method, Auxiliary storage device, binary file, bucket, bucket hashing, collision, collision resolution, data file, digit extraction hashing, direct hashing, division remainder hashing, home address, ….