บทที่ 4
โครงสร้างแฟ้มข้อมูล

วัตถุประสงค์

1. เพื่อให้นักศึกษาทราบถึงภาษาสำหรับแสดงโครงสร้างแฟ้มข้อมูล
2. เพื่อ่ให้นักศึกษาทราบถึงแฟ้มข้อมูลแบบเรียงลำดับ(Sequential File)
3. เพื่อให้นักศึกษาทราบถึงโครงสร้างของ Invert File
4. เพื่อให้นักศึกษาทราบถึงการสร้าง Invert File ในระบบค้นคืนสารสนเทศ
5. เพื่อให้นักศึกษาทราบถึง Index-Sequential File
6. เพื่อให้นักศึกษาทราบถึงโครงสร้างข้อมูลที่ไม่เป็นเชิงเส้น

สารบัญ

4.1 บทนำ

สำหรับในบทนี้จะกล่าวถึง โครงสร้างแฟ้มข้อมูลที่ถูกนำมาใช้ในระบบค้นคืนสารสนเทศ โครงสร้างของแฟ้มข้อมูลนั้นสามารถแบ่งได้เป็น 2 แบบคือ โครงสร้างทางตรรกศาสตร์(Logical Organization) และโครงสร้างทางกายภาพ (Physical Organization) ซึ่งในที่นี้เราจะสนใจความสัมพันธ์ระหว่างข้อมูลที่เก็บที่สามารถเก็บให้เหมาะสม โดยไม่สนใจการเก็บจริงๆภายในคอมพิวเตอร์ วิธีการที่แสดงหรืออธิบายเกี่ยวกับข้อมูลนั้นมีหลายวิธีได้แก่

1.Relational Model เป็นโมเดลที่ข้อมูลจะถูกอธิบายโดย n-tuple ของ คุณสมบัติ (Attribute Values)

2.Hierarchical Approach เป็นวิธีการที่ข้อมูลจะถูกแสดงในรูปของลำดับชั้นของข้อมูล

3.Network Approach เป็นวิธีการที่ข้อมูลถูกเชื่อมโยงเป็นข่ายงาน ในแต่ละลิงค์ที่เชื่อมโยงระหว่างสองไอเท็มจะเป็นไปตามเงื่อนไขที่กำหนดเช่นมีคุณสมบัติเหมือนกัน

4.2 ภาษาสำหรับแสดงโครงสร้างแฟ้มข้อมูล

จากศัพท์ของ Hsiao และ Harary ในการแสดงโครงสร้างโครงสร้างแฟ้มข้อมูล

กำหนดให้ A เป็นคุณสมบัติ(Attributes)
V เป็นเซตของค่า (Attribute Values)
R เป็นเรคอร์ด

ดังนั้น R เป็นซับเซตของ Cartesian Product AxV ซึ่งแต่ละคุณสมบัติจะมีเพียงหนึ่งค่าเท่านั้น อยู่ในรูปของเซตของ Ordered Pairs ในรูปแบบของ (Attribute,Value)

ตัวอย่างที่ 4.1 กำหนดให้ R เป็นเรคอร์ดดังนี้

R =((K1,X1), (K2,X2),…,(Kn,Xn))

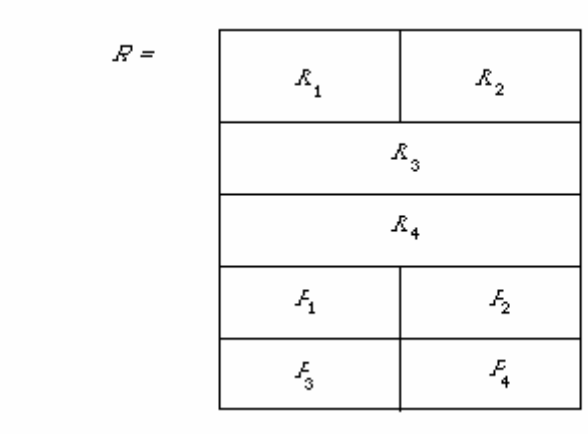
โดย K หมายถึงคีย์เวิรด์(Keyword) หรือคุณสมบัติ(Attribute)
X หมายถึงค่า(Value) หรือ ค่าน้ำหนัก(Numerical Weight)

ในบางกรณีที่เอกสารจะถูกกำหนดให้อยู่ในรูปแบบของ

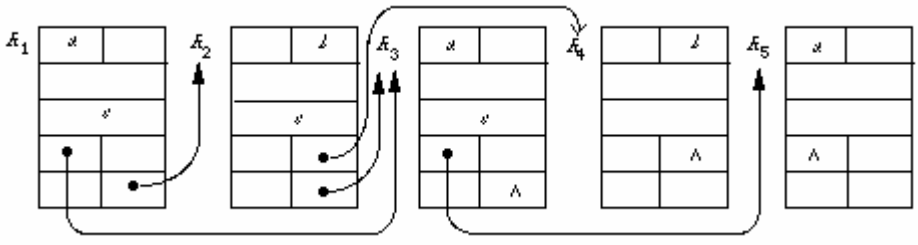
R = {Kt1, Kt2, Kt3,…, Ktn}

โดยเอกสารจะถูกทดสอบว่ามีคีย์เวิร์ดนั้นอยู่ในเอกสารหรือไม่ในกรณีที่คีย์เวิร์ดมีอยู่ในเอกสารจะมีค่า X เท่ากับ 1 แต่ถ้าไม่มีคีย์เวิร์ดในเอกสารค่าของ X จะมีค่าเท่ากับ 0

เรคอร์ดหนึ่งประกอบด้วยหลายฟิลด์แต่ละฟิลด์จะเก็บค่าคุณสมบัติ(Attribute) ที่กำหนด ความยาวของแต่ละฟิลด์ไม่จำเป็นต้องมีขนาดคงที่ และอาจจะมีบางฟิลด์ที่เก็บตำแหน่งที่อยู่(address) ของเรคอร์ดอื่นๆ แสดงตามรูปที่ 4.1 เราสามารถนำตัวแปรชนิดพอยเตอร์(Pointer) มาเชื่อมโยงเรคอร์ดที่มีค่าคุณสมบัติ(X) ของคีย์เวิร์ด K เข้าด้วยกัน โดยเรคอร์ดสุดท้ายนั้นจะถูกชี้ไปที่ Null ซึ่งพอยเตอร์ที่เชื่อมโยงกับคุณสมบัติ K ในเรคอร์ด R จะถูกเรียกว่า K-Pointer คุณสมบัติที่ถูกใช้เพื่อช่วยในการจัดสร้างแฟ้มข้อมูลเรียกว่า คีย์(Key) ดังแสดงตามรูปที่ 4.2



รูปที่ 4.1 : แสดงตัวอย่างของเรคอร์ดที่ประกอบด้วยรายการข้อมูลที่เกี่ยวข้อง



รูปที่ 4.2 : แสดงการใช้พอยเตอร์ในการเชื่อมโยงเรคอร์ด

กำหนดให้ L เป็นลิสต์ของเรคอร์ดที่เกี่ยวข้องกับคีย์เวิร์ด K

K-list เป็นเซตของเรคอร์ดที่มี K อยู่

จากรูปที่ 4.2 นั้น

a-list เป็นลิสต์ของ R1,R3,R5

b-list เป็นลิสต์ของ R2,R4

c-list เป็นลิสต์ของ R1,R2,R3

เราสามารถนิยามไดเรกทอรี่(Directory)ของแฟ้มข้อมูลได้ดังนี้

กำหนดให้ F เป็นแฟ้มข้อมูลหนึ่งที่มีเรคอร์ดที่บรรจุคีย์เวิร์ดที่แตกต่างกัน m ตัว ได้แก่ K1,K2,…,Km

Ni เป็นจำนวนเรคอร์ดที่บรรจุคีย์เวิร์ด Ki

hi เป็นจำนวนของ Ki-Lists ใน F

Aij เป็นตำแหน่งเริ่มต้นของ Kj-list

ดังนั้น เราสามารถนิยามไดเร็กทอรี่ได้ว่า เป็นชุดของ {Ki, Ni, Hi, Ai1, Ai2,…,Aihi}

4.3 Sequential File

แฟ้มข้อมูลแบบเรียงลำดับเป็นแฟ้มข้อมูลพื้นฐานที่สุดที่ไม่มีไดเร็กตอรี่และพอยเตอร์สำหรับเชื่อมโยง โดยทั่วไปการจัดเก็บหรือการสืบค้นข้อมูลจะกระทำเรียงลำดับตามคีย์ หรือเรียงตามลำดับที่เก็บในหน่วยบันทึกข้อมูล ในกรณีคีย์ซ้ำสามารถเลือกคีย์ที่สองมาจัดเรียงลำดับข้อมูล

ข้อดีของแฟ้มข้อมูลแบบเรียงลำดับคือ

- ง่ายต่อการสร้าง และมีโครงสร้างแบบง่าย
- สามารถประมวลผลเรคอร์ดถัดไปได้อย่างรวดเร็ว
- ใช้ทรัพยากรได้อย่างมีประสิทธิภาพ เหมาะสำหรับงานที่ต้องใช้ข้อมูลทุกเรคอร์ด
- ใช้ได้กับหน่วยความจำสำรองที่มีราคาถูกเช่น เทปแม่เหล็ก

ข้อเสียของแฟ้มข้อมูลแบบลำดับ

- แทรกหรือแก้ไขข้อมูลในแฟ้มข้อมูลได้ยาก และเสียเวลามาก
- การสืบค้นข้อมูลที่ต้องการนั้น ต้องอ่านข้อมูลตั้งแต่เรคอร์ดแรกไปตามลำดับจนกว่าจะพบข้อมูลที่ต้องการทำให้เสียเวลาและไม่เหมาะกับการประมวลผลแบบสุ่ม(Random)
- การบันทึกข้อมูลต้องกระทำตามลำดับตามคีย์ ดังนั้นต้องนำเรคอร์ดมาจัดเรียงลำดับก่อน(sort)
- ข้อมูลในแฟ้มข้อมูลไม่ทันสมัยหรือทันต่อเหตุการณ์ทุกขณะ

โดยทั่วไปข้อมูลที่เก็บในแฟ้มแบบลำดับนี้ มีการปรับปรุงข้อมูลในแฟ้มอยู่ 3 ลักษณะ คือ

- การเพิ่มเรคอร์ดใหม่
- การลบเรคอร์ดเก่าออกจากแฟ้มลำดับ
- การเปลี่ยนแปลงค่าของฟิลด์ในเรคอร์ดที่ต้องการ

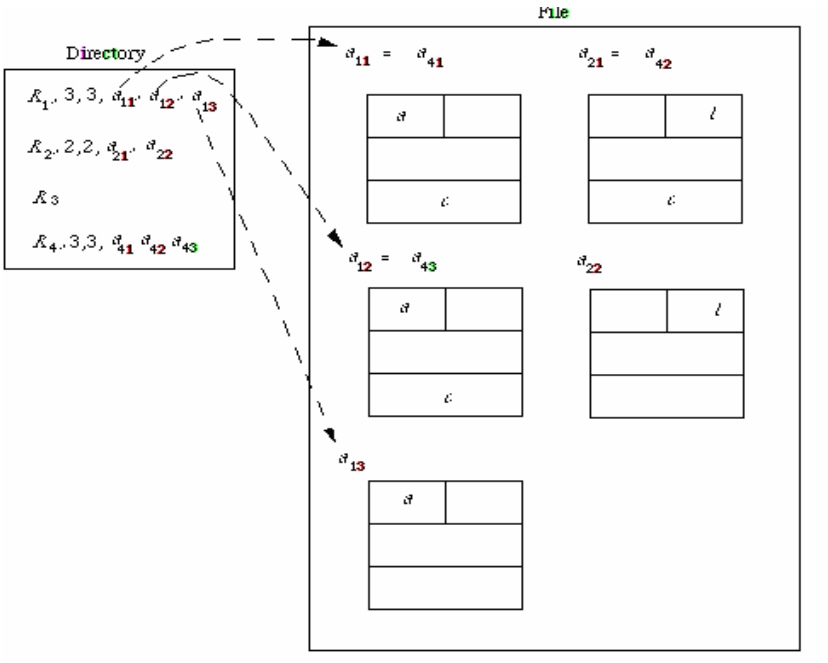
ซึ่งการปรับปรุงข้อมูลนี้ โปรแกรมเมอร์ต้องตรวจสอบข้อมูลให้ดี เพราะอาจเกิดความผิดพลาดได้ 3 ประการคือ

- เพิ่มเรคอร์ดที่มีอยู่แล้ว ทำให้ข้อมูลเก็บซ้ำกันทำให้ข้อมูลซ้ำซ้อน
- ลบเรคอร์ดที่ไม่มีในแฟ้มข้อมูล
- เปลี่ยนแปลงค่าของข้อมูลในกรณีที่ไม่มีเรคอร์ดที่ต้องการอยู่ในแฟ้ม

ดังนั้นในการปรับปรุงข้อมูลต้องมีการตรวจสอบข้อมูลที่ต้องการก่อนว่ามีอยู่ในแฟ้มข้อมูลหรือไม่

4.4 Invert File

เป็นโครงสร้างแฟ้มข้อมูลที่ทุกๆลิสต์บรรจุเพียง 1 เรคอร์ด ดังนั้นทุกๆ K-list จะมีเพียง 1 เรคอร์ด ดังนั้นไดเร็กทอรี่ของแฟ้มข้อมูลจะมี Ni = hi สำหรับทุกๆ i ซึ่งจำนวนเรคอร์ดที่บรรจุ Ki จะมีค่าเท่ากับจำนวนของ Ki-list



รูปที่ 4.3 :แสดงไดเร็คทอรี่ของ invert File

จากรูปที่ 4.3 นั้นจะเห็นได้ว่าไดเร็คทอรี่จะเก็บตำแหน่งที่อยู่ของเรคอร์ดทั้งหมดที่บรรจุ Ki ไว้เพื่อใช้ในการดึงเอกสาร ซึ่งการเก็บในลักษณะนี้จะทำให้สืบค้นหรือสามารถประมวลผลเอกสารได้อย่างรวดเร็ว ซึ่งมีการตรวจสอบดรรชนีเพื่อค้นหาเอกสารที่ตรงตามข้อเรียกร้อง

ตัวอย่างที่ 4.1 สมมุติว่า เอกสาร D1, D2, D3, D4 มีคีย์เวิร์ดดังนี้

D1 = (K1 ,K2, K3)

D2 = (K2, K3, K5)

D3 = (K1, K3, K4, K5)

D4 = (K1, K2, K3, K4)

เมื่อนำมาเก็บในโครงสร้างของ Inverted file จะเก็บข้อมูลในลักษณะนี้

K1 = (D1,D3, D5)

K2 = (D1, D2, D4)

K3 = (D1, D2, D3, D4)

K4 = (D3, D4)

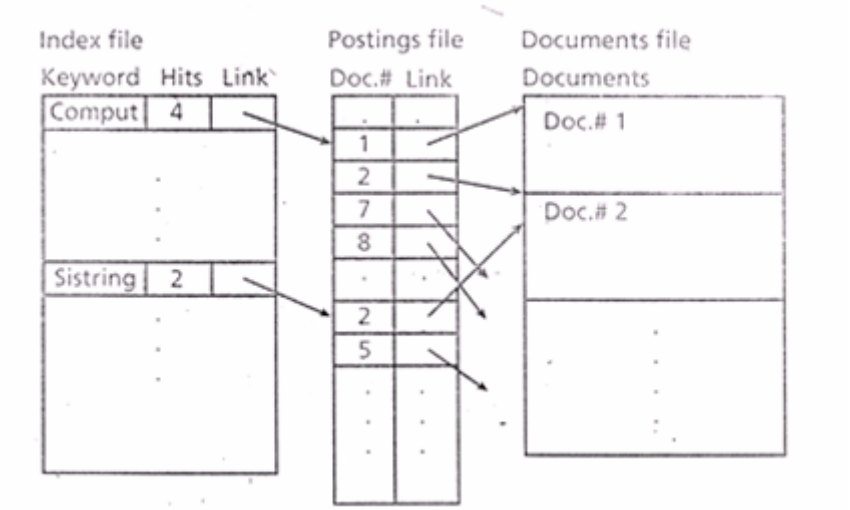
K5 = (D2, D3)

สำหรับการค้นหาเอกสารในระบบค้นคืนสารสนเทศจะใช้กลยุทธ์ Boolean Search สมมุติว่าข้อคำถาม คือ Q = K2 AND K3

ผลของการสืบค้นจะได้เอกสารดังนี้ (D1 , D2, D4) เห็นได้ว่าการค้นหาข้อมูลทำได้สะดวกและรวดเร็ว แต่ข้อเสียของแฟ้มข้อมูลชนิดนี้คือการแก้ไขเปลี่ยนแปลงแฟ้มข้อมูลทำได้ค่อนข้างช้าและเสียเวลามาก

ตัวอย่างที่ 4.2 กำหนดให้ D1, D2 เป็นชุดของเอกสารที่ประกอบด้วยคำ p1 และ p2 ตามลำดับ และ U เป็นชุดของเอกสารทั้งหมด ดังนั้น

- U – D1 ได้ผลลัพธ์เป็นเซตของเอกสารที่ไม่มีคำ p1 (not p1)
- D1 intersect D2 เซตของเอกสารที่ประกอบทั้งคำ p1 และ p2
- D1 U D2 เซตของเอกสารที่ประกอบด้วยคำ p1 หรือ p2



รูปที่ 4.4: แสดง Inverted file ที่ถูกสร้างโดยใช้วิธีการเรียงลำดับอาเรย์ (sorted array)

ตัวอย่างที่ 4.3 กำหนดให้เอกสาร 1 ถึง 5 ประกอบด้วยคีย์เวิร์ดดังต่อไปนี้

เอกสาร1	ประกอบด้วยคีย์เวิร์ดดังนี้	อัลกอริธึม, สารสนเทศ, ค้นคืน
เอกสาร 2	ประกอบด้วยคีย์เวิร์ดดังนี้	ค้นคืน ,วิทย์
เอกสาร 3	ประกอบด้วยคีย์เวิร์ดดังนี้	อัลกอริธึม, สารสนเทศ , วิทย์
เอกสาร 4	ประกอบด้วยคีย์เวิร์ดดังนี้	รูปแบบ ,ค้นคืน , วิทย์
เอกสาร 5	ประกอบด้วยคีย์เวิร์ดดังนี้	วิทย์ ,อัลกอริทึม

นำคีย์เวิร์ดต่าง ๆมาแยกคำและนับความถี่ของคีย์เวิร์ดในเอกสารต่าง ๆดังตารางดังต่อไปนี้

ดัชนี	Keyword_id
อัลกอริธึม	1
สารสนเทศ	2
ค้นคืน	3
วิทย์	4
รูปแบบ	5

Keyword_id	DocId	Freq
1	1	1
2	1	1
3	1	1
3	2	1
4	2	1
1	3	1
2	3	1
4	3	1
5	4	1
3	4	1

DocId	Document Content
1	อัลกอริธึม สารสนเทศ ค้นคืน
2	ค้นคืน วิทย์
3	อัลกอริธึม สารสนเทศ วิทย์
4	รูปแบบ ค้นคืน วิทย์
5	วิทย์ อัลกอริทึม

สมมุติว่าคำร้องขอสารสนเทศ คือ

{ เอกสาร1, เอกสาร3} intersect {เอกสาร1, เอกสาร2, เอกสาร4 }

ผลลัพธ์ที่ได้คือ {เอกสาร1}

สมมุติว่าคำร้องขอ สารสนเทศ

{ เอกสาร1, เอกสาร3} U {เอกสาร1, เอกสาร2, เอกสาร4 }

ผลลัพธ์ที่ได้คือ {เอกสาร1, เอกสาร2, เอกสาร3, เอกสาร4 }

ซึ่งคำร้องขอที่ดีควรเป็นภาษาธรรมชาติ เป็นประโยค วลี หรือ คำ ไม่ควรเขียนในรูปของคณิตศาสตร์ โดยมีเครื่องหมายแบ่ง การประมวลผลสามารถเพิ่มหรือให้ความสำคัญของคำได้ ซึ่งคำร้องขอต้องสามารถสืบค้นข้อมูลที่ตรงกับความต้องการของผู้ใช้ได้ ไม่เพียงเฉพาะคำที่ผู้ใช้ระบุ เช่น ต้องการค้นหารถราคาถูก ระบบค้นคืนไม่ได้ใช้เฉพาะคำดัชนี เพียง รถ, ราคาถูก เท่านั้นชุดคำค้นควรสามารถค้นหาในส่วนอื่นๆได้อีกเช่นคำว่า ส่วนลด ,ราคาสมเหตุผล, ไม่แพง, ยานพาหนะ, เก๋ง, รถส่วนบุคคล ถ้านำมาเขียนเป็นชุดคำร้องขอทางพีชคณิตเช่นใช้ข้อสอบถามดังนี้

q = (“ส่วนลด” or “ราคาสมเหตุผล” or “ไม่แพง” or “ราคาถูก”) and

(“รถ” or “เก๋ง” or “ส่วนบุคคล”) เป็นต้น

ตัวอย่างที่ 4.4 การสร้าง inverted file index

กำหนด Vector File เป็นดังนี้

docs	*t1*	*t2*	*t3*	RSV=Q.Di
D1	1	0	1	4
D2	1	0	0	1
D3	0	1	1	5
D4	1	0	0	1
D5	1	1	1	6
D6	1	1	0	3
D7	0	1	0	2
D8	0	1	0	2
D9	0	0	1	3
D10	0	1	1	5
D11	1	0	1	3
Q	1	2	3	
	q1	*q2*	*q3*	

Invert file คือ Vector File ที่ invert แถวให้เป็นคอลัมภ์ และเปลี่ยนคอลัมภ์ให้เป็นแถวดังนี้

Terms	**D1**	**D2**	**D3**	**D4**	**D5**	**D6**	**D7**	**…**
t1	1	1	0	1	1	1	0	
t2	0	0	1	0	1	1	1	
t3	1	0	1	0	1	0	0	

ตัวอย่างที่ 4.5 การสร้าง Invert File

สมมุติว่า มีเอกสาร DOC1 และ DOC2 ดังนี้

DOC1 ประกอบด้วย Now is the time for all good men to come to the aid of their country

DOC2 ประกอบด้วย It was a dark and stormy night in the country manor. The time was past midnight

เอกสารต่างๆเหล่านี้จะถูกดึงทีละ Token เพื่อกำหนดเป็นหมายเลขของเอกสาร สำหรับแต่ละ Token ดังนี้ =========>

Term	Doc #
now	1
is	1
the	1
time	1
for	1
all	1
good	1
men	1
to	1
come	1
to	1
the	1
aid	1
of	1
their	1
country	1
it	2
was	2
a	2
dark	2
and	2
stormy	2
night	2
in	2
the	2
country	2
manor	2
the	2
time	2
was	2
past	2
midnight	2

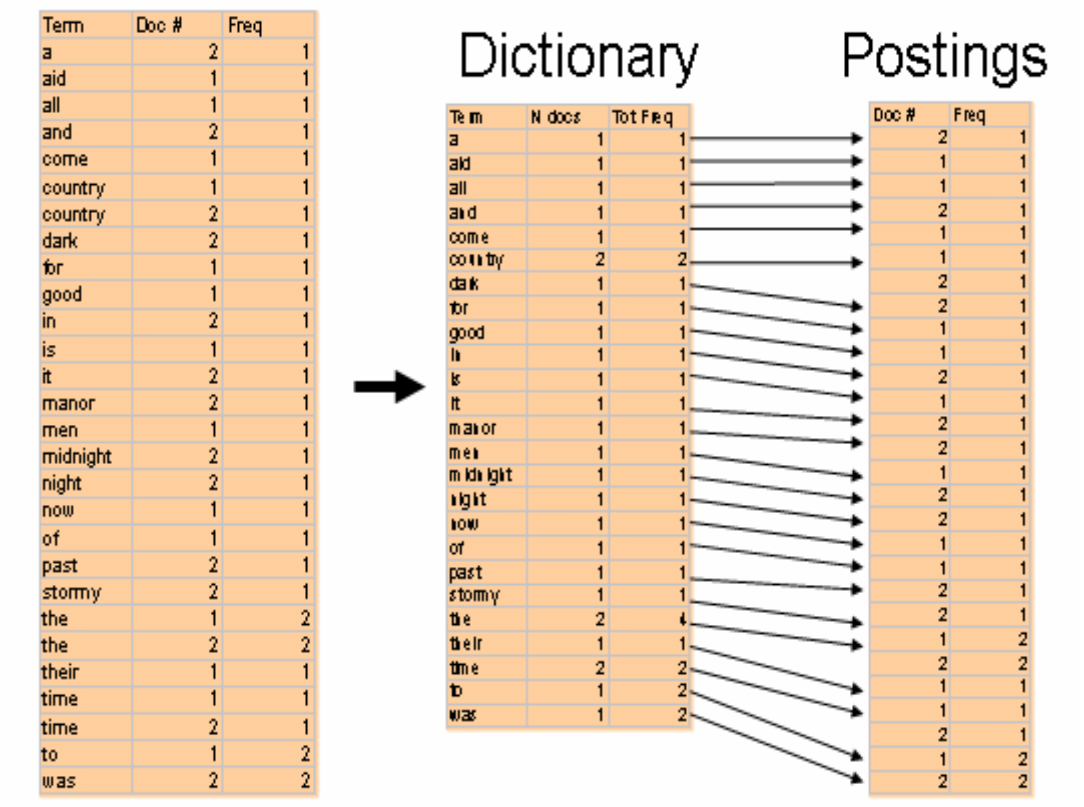
ต่อจากนั้นนำมาเรียงลำดับ และเทอมที่ซ้ำกันให้รวม(merge) เป็นเทอมเดียว ดังนี้

a	2
aid	1
all	1
and	2
come	1
country	1
country	2
dark	2
for	1
good	1
in	2
is	1
it	2
manor	2
men	1
midnight	2
night	2
now	1
of	1
past	2
stormy	2
the	1
the	1
the	2
the	2
their	1
time	1
time	2
to	1
to	1
was	2
was	2

→

Term	Doc #	Freq
a	2	1
aid	1	1
all	1	1
and	2	1
come	1	1
country	1	1
country	2	1
dark	2	1
for	1	1
good	1	1
in	2	1
is	1	1
it	2	1
manor	2	1
men	1	1
midnight	2	1
night	2	1
now	1	1
of	1	1
past	2	1
stormy	2	1
the	1	2
the	2	2
their	1	1
time	1	1
time	2	1
to	1	2
was	2	2

จากตัวอย่างจะเห็นได้ว่าเราจะทราบถึงความถี่ของเทอมต่างๆที่มีอยู่ในเอกสารทั้งหมด สำหรับการสร้าง Invert File นั้นเราจะแยกแฟ้มที่ได้นี้ออกเป็น Dictionary File และ Postings File ดังนี้



วิธีการนี้ทำให้การค้นหามีความรวดเร็วเพราะเทอมต่างๆเป็นอิสระต่อกัน การใช้งานนั้น แต่ละเทอมประกอบด้วย Document ID , Frequency of term in doc (optional) และ Position of term in doc (optional) ในการสอบถามสามารถใช้ Boolean queries หรือ statistical ranking algorithms ก็ได้ เช่น country ปรากฏทั้งเอกสารทั้ง d1 และ d2 ส่วน manor ปรากฏอยู่ในเอกสาร d2 ดังนั้น country AND manor จะทำการดึงเอกสาร d2 ออกมาให้แก่ผู้ใช้ เป็นต้น

การตรวจสอบนี้จะสืบค้นจาก Posting File เพื่อดึงเอกสารที่ตรงกับข้อคำถามดังตัวอย่าง สมมุติว่าข้อสอบถามคือ “time” AND “dark”

2 docs with “time” in dictionary -> ID 1 and 2 from posting file

1 doc with “dark” in dictionary ->ID 2 from posting file

ระบบจะทำการค้นหาจากพจนานุกรมก่อนว่าคำว่า “time” ปรากฏในเอกสารจำนวนเท่าใดในที่นี้มีด้วยกัน 2 เอกสาร และมีความถี่รวมเท่ากับ 2 มีการเชื่อมโยงไปยังตำแหน่งที่อยู่ของเอกสารทั้งสอง ในทำนองเดียวันจะมีการตรวจสอบคำว่า “dark” จากพจนานุกรมและดึงเอกสาร 2 จาก posting file นำมา AND ผลจากการสอบถามระบบจะถึงเอกสาร 2 ออกมาให้ผู้ใช้

ตัวอย่างที่ 4.6 Invert File

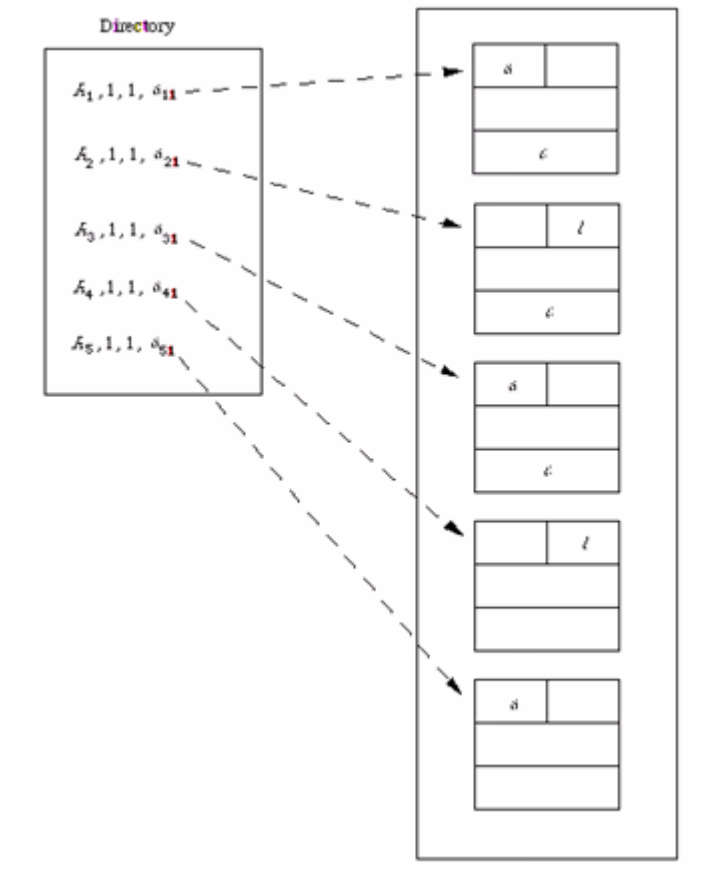
Invert File นี้มีการสร้างขึ้นจาก home page ของ Microsoft.com เพื่อใช้สำหรับการสืบค้นฐานข้อมูล และ Microsoft เป็น Text File ที่ประกอบด้วย Term String และ docID string ซึ่ง Term เหล่านี้สามารถสร้างโดยใช้ โครงสร้างต้นไม้(Tree) หรือ Hash index ก็ได้

Term	docID
data	http://www-inst.eecs.berkeley.edu/~cs186
database	http://www-inst.eecs.berkeley.edu/~cs186
date	http://www-inst.eecs.berkeley.edu/~cs186
day	http://www-inst.eecs.berkeley.edu/~cs186
dbms	http://www-inst.eecs.berkeley.edu/~cs186
decision	http://www-inst.eecs.berkeley.edu/~cs186
demonstrate	http://www-inst.eecs.berkeley.edu/~cs186
description	http://www-inst.eecs.berkeley.edu/~cs186
design	http://www-inst.eecs.berkeley.edu/~cs186
desire	http://www-inst.eecs.berkeley.edu/~cs186
developer	http://www.microsoft.com
differ	http://www-inst.eecs.berkeley.edu/~cs186
disability	http://www.microsoft.com
discussion	http://www-inst.eecs.berkeley.edu/~cs186
division	http://www-inst.eecs.berkeley.edu/~cs186
do	http://www-inst.eecs.berkeley.edu/~cs186
document	http://www-inst.eecs.berkeley.edu/~cs186
document	http://www.microsoft.com

microsoft	http://www.microsoft.com
microsoft	http://www-inst.eecs.berkeley.edu/~cs186
midnight	http://www-inst.eecs.berkeley.edu/~cs186
midterm	http://www-inst.eecs.berkeley.edu/~cs186
minibase	http://www-inst.eecs.berkeley.edu/~cs186
million	http://www.microsoft.com
monday	http://www.microsoft.com
more	http://www.microsoft.com
most	http://www-inst.eecs.berkeley.edu/~cs186
ms	http://www-inst.eecs.berkeley.edu/~cs186
msn	http://www.microsoft.com
must	http://www-inst.eecs.berkeley.edu/~cs186
necessary	http://www-inst.eecs.berkeley.edu/~cs186
need	http://www-inst.eecs.berkeley.edu/~cs186

Index-Sequential File(IS File)

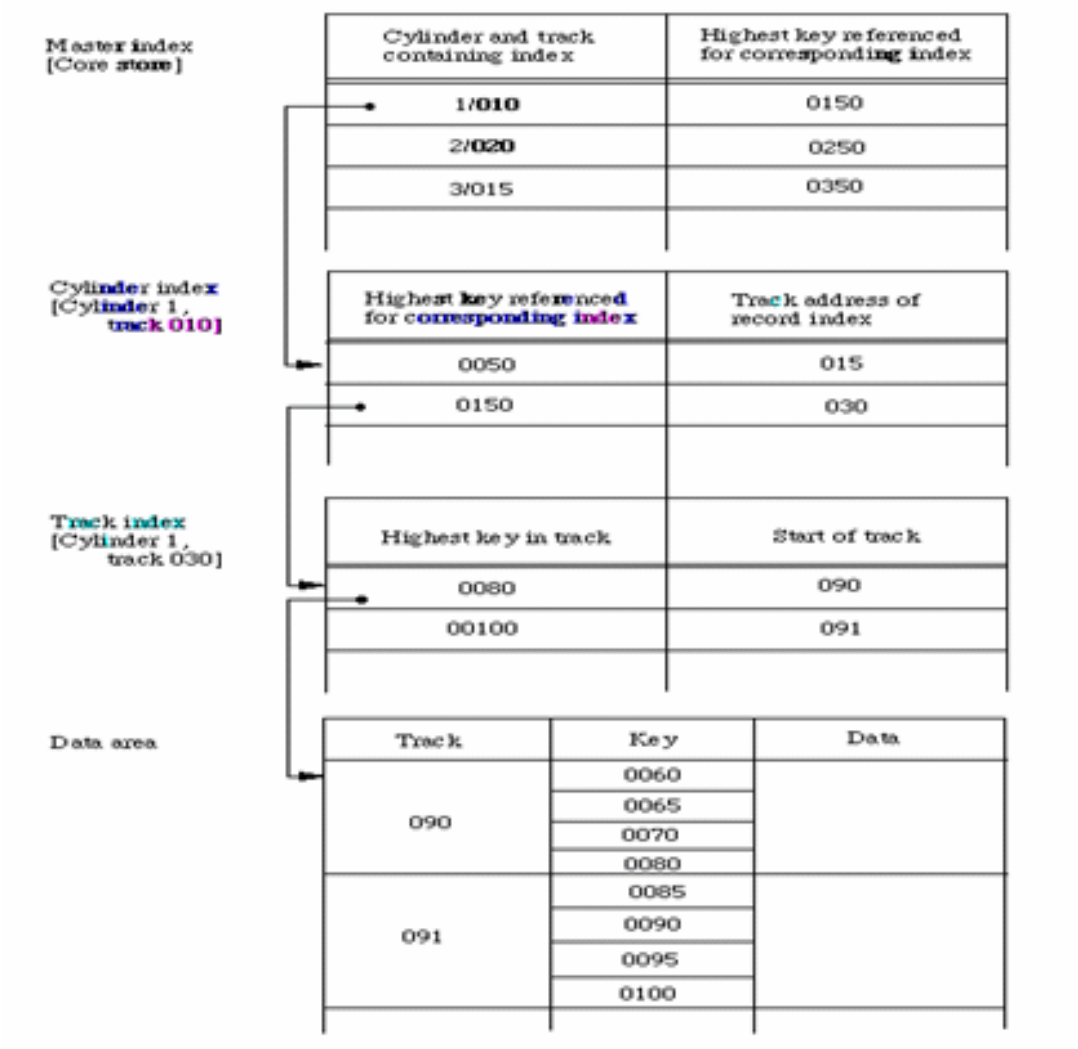
เป็นแฟ้มข้อมูลชนิด Inverted File ที่มีทุกคีย์เวิร์ด Ki ซึ่ง Ni = hi = 1 ซึ่งแต่ละเรคอร์ดจะมีเพียง 1 คีย์เวิร์ดเท่านั้น ซึ่งกลุ่มของเรคอร์ดเหล่านี้จะถูกจัดเรียงลำดับตามลำดับของคีย์ ดังรูปที่ 4.5



รูปที่ 4.5 : แสดง index-sequential file

Index-Sequential File(IS) นั้นเหมือนกับ Sequential File ที่มีดรรชนีเป็นลำดับชั้น โดยจะใช้หมายเลขเรคอร์ดเป็นคีย์ จากรูปที่ 4.6 เป็นไดเร็กทอรี่ของแฟ้ม Index-Sequential File ที่ถูกสร้างบนเครื่องคอมพิวเตอร์ ประกอบด้วยดัชนี 3 ระดับได้แก่ Master Index , Cylinder index และ Track index สมมุติว่าต้องการค้นหาคีย์ที่ 0070 การค้นหาตำแหน่งที่อยู่ของข้อมูลจะเริ่มค้นหาจาก Master index ก่อนโดยตรวจสอบจาก Highest Key referenced

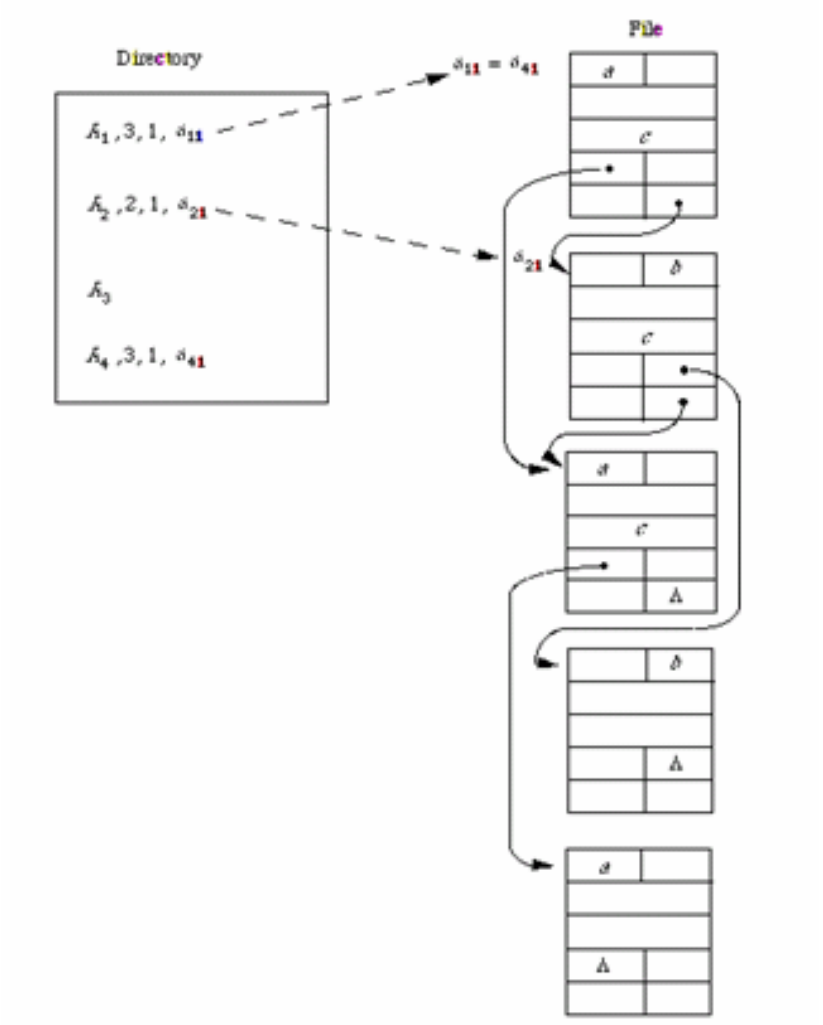
ผลลัพธ์เป็น 1/010 หมายความว่าข้อมูลที่ต้องการอยู่ Cylinder ที่ 1 Track ที่ 010 ต่อจากนั้นการสืบค้นจะไปค้นหาจากดรรชนีในระดับที่สองใน Cylinder ที่ 1 Track ที่ 010 โดยนำค่าของ Key ในที่นี้คือ 0070 ไปสืบค้น จาก Highest key referenced for corresponding index ได้ผลลัพธ์ว่า ข้อมูลที่ต้องการอยู่ใน Track ที่ 030 นำผลที่ได้ไปสืบค้นจากดรรชนีในระดับที่ 3 ต่อไป โดยเปรียบเทียบกับ Highest Key in track ได้ผลลัพธ์เท่ากับ 090 ทำให้ระบบทราบว่าข้อมูลที่ต้องการนั้นอยู่ใน track ที่ 90 นั่นเองสามารถดึงข้อมูลได้ตามต้องการ



รูปที่ 4.6 : แสดงการสร้าง Index-sequential file

มัลติลิสต์(Multi-List)

เป็น Inverted File ที่มี 1 ลิสต์ต่อ 1 คีย์เวิร์ด ซึ่งไดเรอทอรี่จะมี hi = 1 โดยไดเร็กทอรี่จะเก็บตำแหน่งที่อยู่ของเรคอร์ดแรกและมีการเชื่อมโยงเรคอร์ดที่มีคีย์เวิร์ด Ki ไว้ด้วยกัน สร้างเป็น Ki-list ตามรูปที่ 4.7

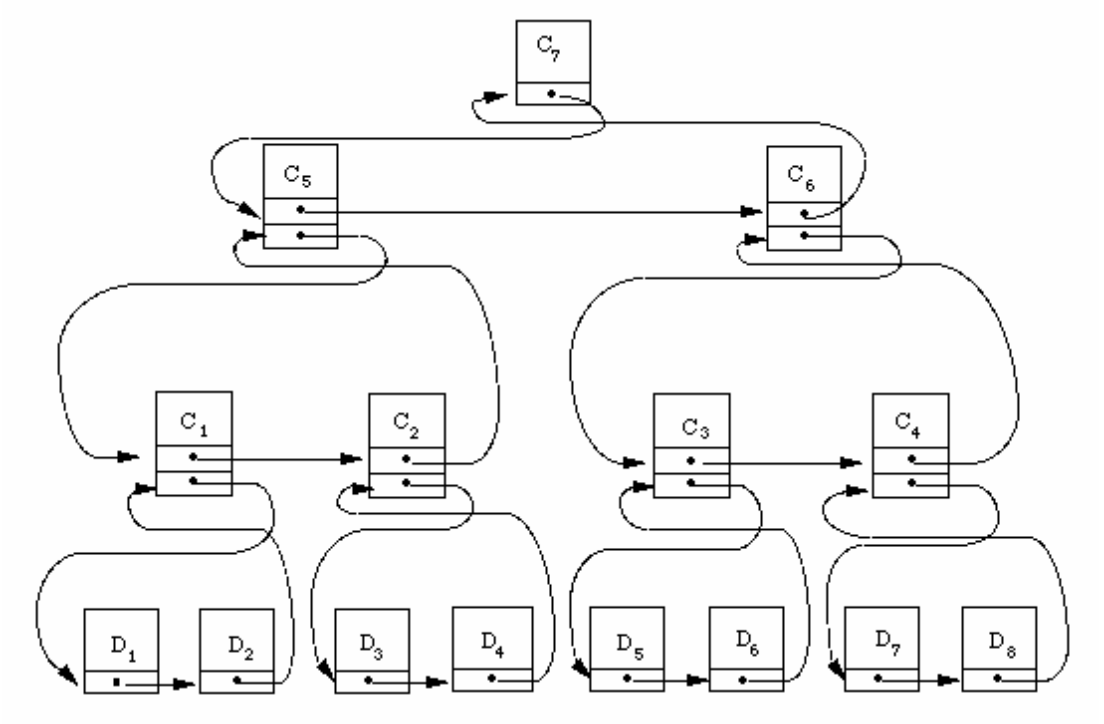


รูปที่ 4.7 : แสดง Multi-list

การพัฒนา Invert File ประเภทนี้เพื่อแก้ปัญหาความยุ่งยากของการ Update ทำให้มีการแทรกและลบได้ง่ายและใช้เวลาน้อยลง

Ring Structure

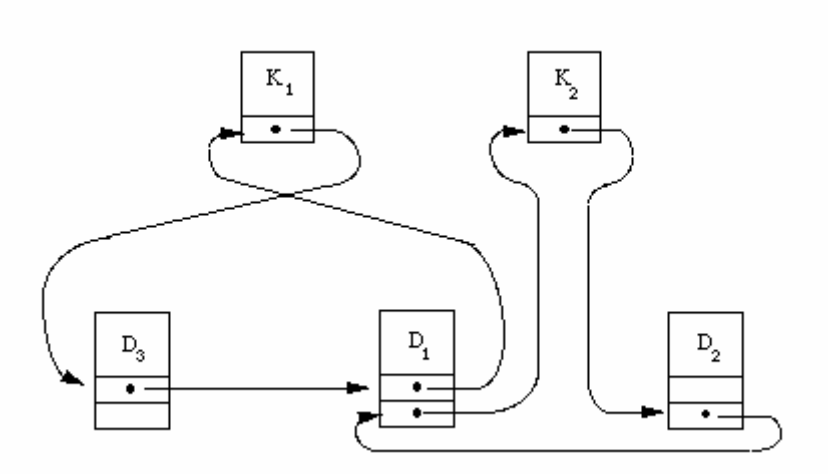
เป็น Invert File ที่ดัดแปลงอีกเล็กน้อยที่การเชื่อมโยงจะเป็นลักษณะวงแหวน กล่าวคือ ตอนต้นและตอนปลายของลิสต์จะต้องเป็นเรคอร์ดเดียวกัน ซึ่งการเก็บข้อมูลโดยใช้โครงสร้างแฟ้มข้อมูลในลักษณะนี้จะเป็นประโยชน์ในการแบ่งชนิดของข้อมูล ดังรูปที่ 4.8



รูปที่ 4.8 : แสดงการสร้าง Dendrogram ใช้ Ring Structure

จากรูป D1 ,D2 ,D3 ,D4 ,D5 ,D6 ,D7, D8 เป็นเอกสาร

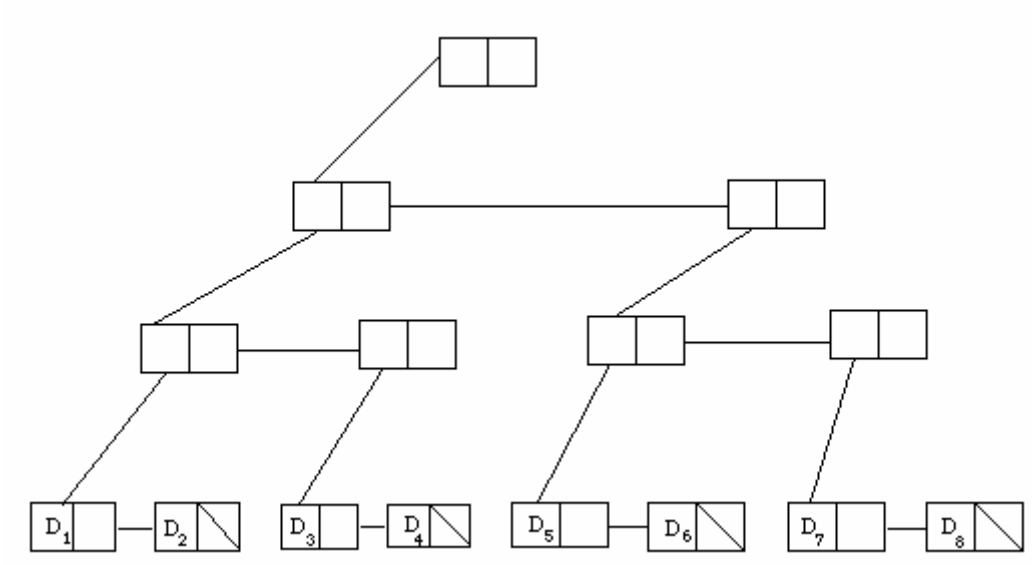
สำหรับ C1 , C2 , C3, C4 เป็นคลาสของเอกสาร ซึ่ง C1 เป็นคลาสของเอกสาร D1 และ D2 ซึ่งเอกสารที่อยู่ในคลาสเดียวกันจะมีคีย์เวิร์ดร่วมกัน ทำให้เราสามารถจัดกลุ่มของเอกสารที่ใช้สำหรับแต่ละคีย์เวิร์ดได้ แต่บางครั้งก็อาจเกิดการซ้อนทับกันคือ เอกสารหนึ่งอาจจะอยู่ได้ในหลาย ๆคลาส ดังรูปที่ 4.9 เห็นได้ว่า เอกสาร D1 อยู่ทั้งคลาส K1 และ K2



รูปที่ 4.9 : แสดง Two overlapping ring

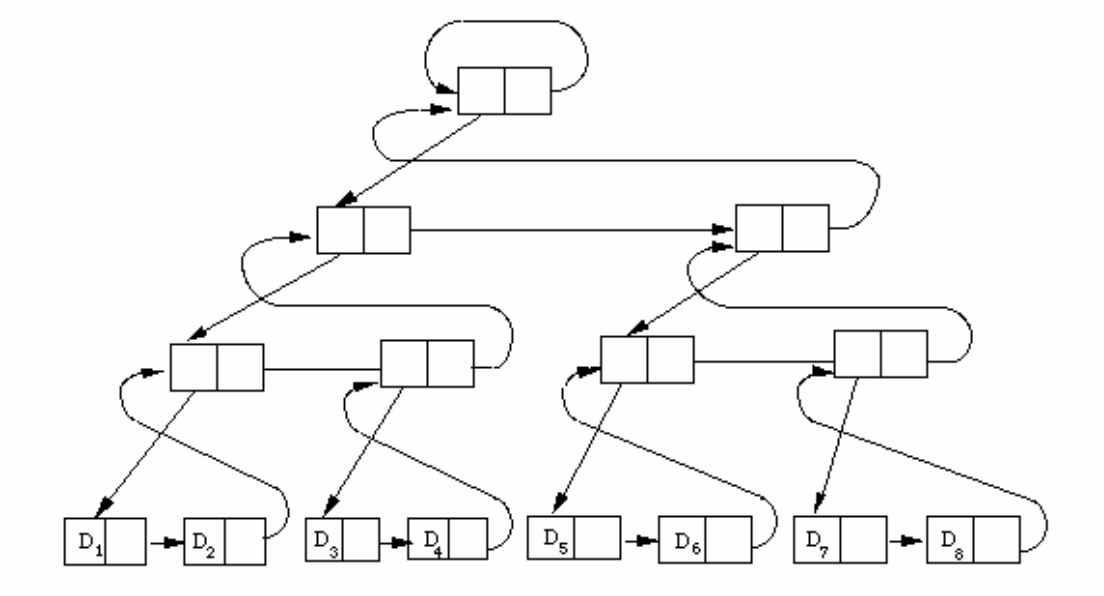
Threaded List

เป็นการแบ่งกลุ่มแบบง่ายๆ จากรูปที่ 4.9 นั้นมีการแบ่งกลุ่มของเอกสารออกเป็น 4 กลุ่มคือ (D1,D2) , (D3,D4) , (D5,D6) , (D7,D8) ซึ่งแต่ละลิสต์ย่อย(sublist)จะมีตัวชี้ตำแหน่ง 2 ตัวเท่านั้น



รูปที่ 4.10 : โครงสร้างของลิสต์ที่สร้างโดย hierarchic classification

เราสามารถเปลี่ยนแปลงโครงสร้างนี้ให้คล้ายกับ Ring Structure โดยให้ตัวชี้ที่อยู่ทางขวามือชี้กลับไปยังส่วนหัวของลิสต์ย่อย ดังรูปที่ 4.11



รูปที่ 4.11 :แสดง threaded list ที่สร้าง hierarchic classification

ประโยชน์ของโครงสร้างในลักษณะนี้นั้นคือการท่องแต่ละโหนดนั้นสามารถกระทำได้อย่างรวดเร็ว สามารถกระทำได้โดยไม่ต้องใช้โครงสร้างชนิดสแตกมาช่วยในการเข้าถึงเอกสาร แต่ก็มีข้อเสียคือการแบ่งกลุ่มนั้นต้องกระทำโดยเริ่มจากราก(root) เสมอ

Double Chained Tree

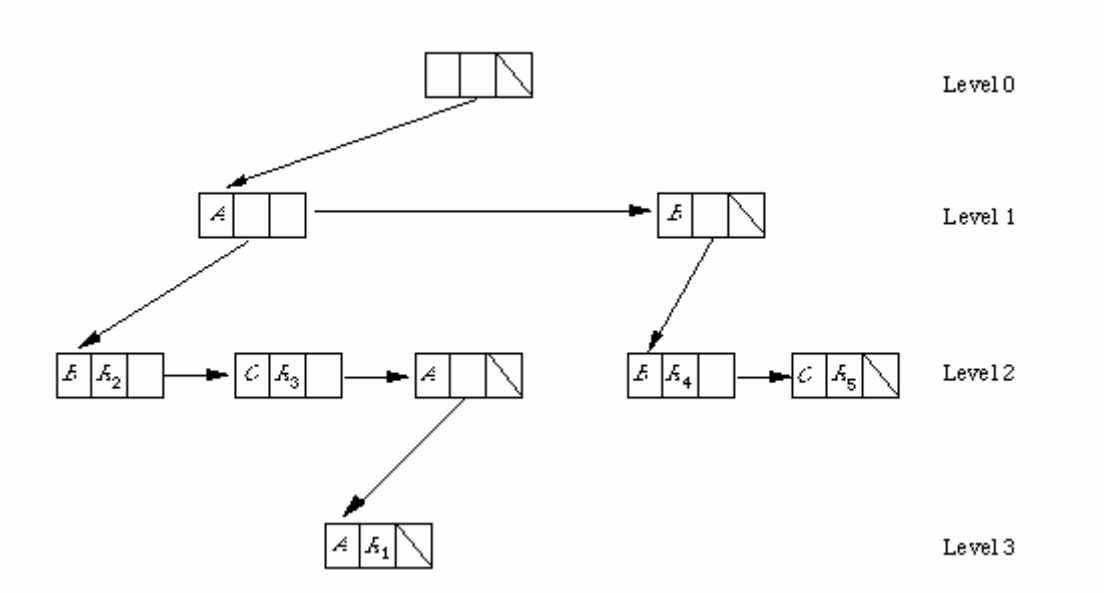
เป็นโครงสร้างของข้อมูลที่มีการปรับเปลี่ยนแต่ละโหนดโดยจากที่มีตัวชี้หรือ pointer 2 ตัวโดยมีการเพิ่มฟิลด์สัญลักษณ์ (Symbol) เข้าไปเพื่อนำไปใช้สำหรับการเก็บคีย์ที่มีขนาดไม่คงที่ ดังรูปที่ 4.11 แต่ละคีย์จะถูกสร้างขึ้นมาจากการเลือกสัญลักษณ์จากตัวอักษร

กำหนดให้ {A, B ,C } เป็นเซตของสัญลักษณ์ของคีย์

R1 ,R2, R3, R4, R5 เป็นเรคอร์ดที่เก็บบันทึกในโครงสร้างนี้

สมมุติว่าคีย์แต่ละเรคอร์ดมีขนาดไม่เท่ากัน

R1	มีคีย์ว่า	AAA
R2	มีคีย์ว่า	AB
R3	มีคีย์ว่า	AC
R4	มีคีย์ว่า	BB
R5	มีคีย์ว่า	BC

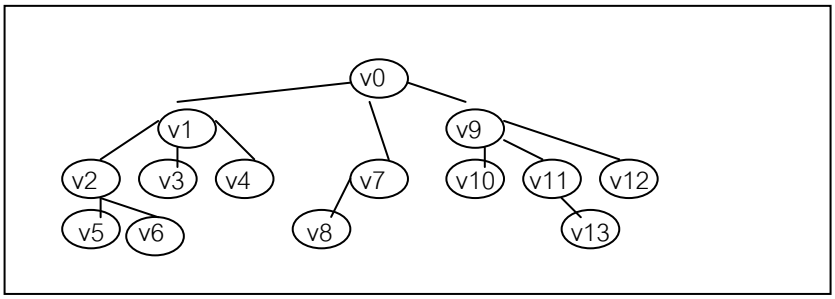


รูปที่ 4.12 : แสดง doubly chained tree

การสร้างโครงสร้างของข้อมูลนั้นจะสร้างโหนดแรกที่ไม่มีสัญลักษณ์ใดๆอยู่ใน Level 0 ซึ่งเป็น root ของโครงสร้างนี้ โดยจาก root จะชี้ไปยังโหนดที่มีการเก็บสัญลักษณ์ตัวแรกของคีย์และเชื่อมโยงไปยังสัญลักษณ์อื่นๆที่เป็นตัวแรกของคีย์ ในที่นี้สัญลักษณ์ตัวแรกของคีย์เป็น A และ B ดังนั้นจะมีการเชื่อมโยงกันใน Level 1 จำนวน 2 โหนดคือ โหนดที่มีสัญลักษณ์ A และโหนดที่มีสัญลักษณ์ B ต่อจากนั้นพิจารณาคีย์ในตำแหน่งที่สอง เพื่อสร้างโครงสร้างข้อมูลใน Level 2 ซึ่งสัญลักษณ์ตัวที่สองของสัญลักษณ์ A ได้แก่ A,B,C สำหรับสัญลักษณ์ตัวที่สองของ B ได้แก่ B ,C เราจะทำการเชื่อมโยงข้อมูลโดยแยกการเชื่อมโยงตามลำดับของคีย์

Tree (ต้นไม้)

เป็นโครงสร้างข้อมูลที่ไม่เป็นเชิงเส้น (non-linear list) ในลักษณะของลำดับชั้น (hierarchical data structure) ประกอบด้วยเซตของโหนดตั้งแต่ 1 โหนดขึ้นไปมีลักษณะ คือ มีโหนดพิเศษ 1 โหนดที่เรียกว่า ราก (root node) โหนดอื่นๆจะถูกแบ่งเป็นเซตเล็กๆ ที่ไม่มีโหนดร่วมกันเรียกแต่ละเซตว่าต้นไม้ย่อย(subtree) และในแต่ละต้นไม้ย่อย ก็จะมีรากเช่นเดียวกัน



รูปที่ 4.13 แสดงต้นไม้

จากรูปที่ 4.13

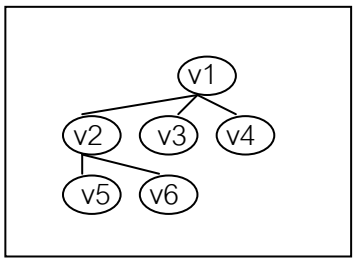
v0 เป็นราก(root node)

tree = {v0,v1,v2,v3,v4,v5,v6,v7,v8,v9,10,v11,v12,v13}

subtreeของ v0 คือ {v1,v2,v3,v4,v5,v6},{v7,v8},{v9,v10,v11,v12,v13}

ต้นไม้นี้ประกอบด้วยโหนดทั้งหมด 14 โหนด คือ v0..v13 และโหนดพิเศษคือ v0 เป็น root node ซึ่งแบ่งเป็นเซตเล็กๆซึ่งเรียกว่าต้นไม้ย่อย 3 subtree ด้วยกัน ซึ่งsubtree แรกมี v1 เป็น root ส่วน subtree ที่ 2 มี v7 เป็น root ท้ายสุด subtree ที่ 3 มี v9 เป็น root

ต้นไม้ย่อยหนึ่งๆมีลักษณะเป็นต้นไม้ซึ่งมีต้นไม้ย่อยได้ดังตัวอย่างรูปที่ 4.14



รูปที่ 4.14
มี v1 เป็นราก
ต้นไม้ย่อยประกอบด้วย
{v2,v5,v6},{v3},{v4}

ดีกรีของโหนด(Degree of node)

คำนวณจากจำนวนของต้นไม้ย่อยของแต่ละโหนด

V0 มีต้นไม้ย่อย 3 ต้น มี Degree = 3

V2 มีต้นไม้ย่อย 2 ต้น มี Degree = 2

V5 มีต้นไม้ย่อย 0 ต้น มี Degree = 0

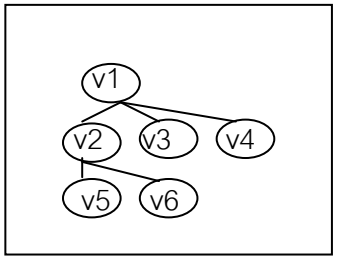
โหนดที่เป็นใบ (Leaf node)

คำนวณจากโหนดที่ไม่มีต้นไม้ย่อยหรือโหนดที่มีดีกรีเป็น 0 หรือเรียกอีกอย่างหนึ่งว่า Terminal node ตามตัวอย่างจะมีโหนดที่เป็นใบคือ v3,v4,v5,v6,v8,v10,v12,v13 ส่วนโหนดที่มีดีกรีเรียกว่าโหนดที่เป็นกิ่ง

ระดับของโหนด(Level of node)

หมายถึงระยะที่ห่างจากรากของโหนดใดๆตามแนวดิ่ง กำหนดให้โหนดที่เป็นรากมีระดับเป็น 1 โหนดอื่นๆที่อยู่ห่างจากรากมาจะมีระดับมากขึ้นอีก 1

ความสูงของต้นไม้(Height หรือ Depth) หมายถึงระดับสูงสุดของต้นไม้ใดๆ



ระดับที่1
ระดับที่2
ระดับที่3 สูงสุด ดังนั้น Depth=3

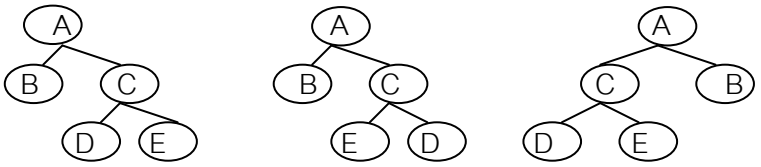
การเรียกชื่อโหนดต่าง ๆในโครงสร้างต้นไม้

- root ของต้นไม้แต่ละต้นเรียกว่า father หรือ parent
- โหนดที่เกิดจากพ่อเดียวกันเรียกว่า brother หรือ sibling โดยถือโหนดทางซ้ายมือสุดเรียกว่าลูกชายคนโต (oldest brother) ถัดมาทางขวาเป็นคนรอง (younger brother) ถัดมาทางขวาอีกเป็นคนสุดท้อง (youngest brother)
- ลูกหลาน (decendents)หมายถึงโหนดที่อยู่ต่อจากโหนดที่เป็น parent ลงมาทั้งหมด
- บรรพบุรุษ(ancestors) หมายถึงโหนดที่อยู่ก่อนโหนดที่กำหนดขึ้นไปเป็นสายเดียวกัน

ชนิดของต้นไม้

พิจารณาจากการจัดลำดับของโหนดในต้นไม้ย่อย แบ่งได้เป็น 2 ชนิดคือ

1. Ordered tree เป็นลักษณะของต้นไม้ที่ถือว่าการจัดลำดับของต้นไม้ย่อยมีความสำคัญ ถ้ามีการสลับที่กันของโหนด จะถือว่าเป็นต้นไม้นั้นมีลักษณะแตกต่างกัน



รูปที่ 4.15: แสดงภาพของต้นไม้

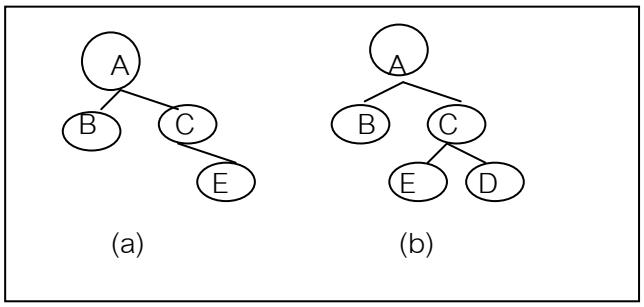
จากรูปที่ 4.15 ต้นไม้ทั้ง 3 ต้นจัดว่ามีลักษณะแตกต่างกันถึงแม้จะมีโหนดที่เหมือนกันก็ตาม

2. Oriented tree เป็นลักษณะของต้นไม้ที่ถือว่าการจัดลำดับของต้นไม้ย่อยไม่มีความสำคัญ โดยตามตัวอย่างข้างบนถือว่าเป็นต้นไม้ต้นเดียวกัน คือประกอบด้วยโหนด A,B,C,D,E

ตัวอย่างที่ 4.7 การนำโครงสร้างต้นไม้มาใช้ในระบบค้นคืนสารสนเทศ สมมุติว่ามีเอกสารอยู่ชุดหนึ่ง ได้แก่ {D1,D2, D3, D4, D5, D6, D7, D8 } เราแบ่งกลุ่มเป็น 4 กลุ่มได้แก่ {(D1,D2) ,(D3,D4) ,(D5,D6),(D7,D8)} และมีการจัดกลุ่มของ 4 กลุ่มนี้ออกเป็นอีก 2 กลุ่มใหญ่ดังนี้

(((D1,D2),(D3,D4)) , ((D5,D6),(D7,D8)))

เราสามารถใช้ Binary Tree เป็นต้นไม้ที่เป็น ordered tree และมีกิ่งที่ยื่นออกมา(out node) ของทุกๆโหนดน้อยกว่าหรือเท่ากับ 2 ถ้า out node ทุกๆโหนดเท่ากับ 2 หรือ 0 จะเรียกต้นไม้นั้นว่า full binary tree



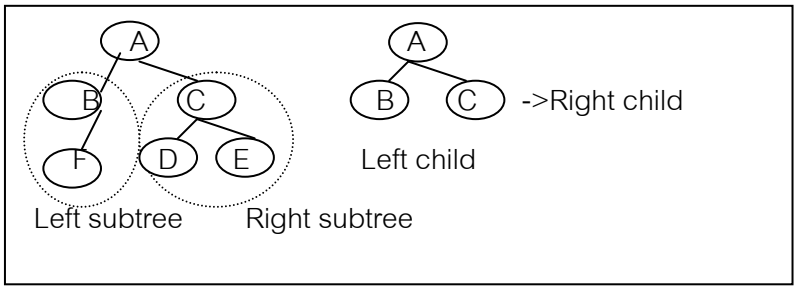
รูปที่ 4.16 แสดงต้นไม้

(a) เป็น Binary tree

(b) เป็น full Bianry tree

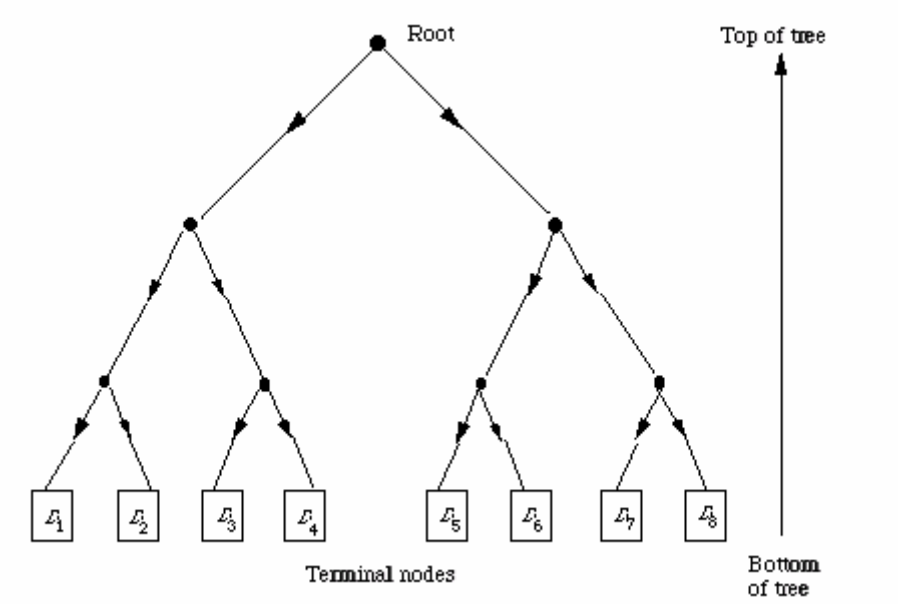
ต้นไม้แบบไบนารี แต่ละโหนดจะมีดีกรีสูงที่สุดคือ 2 ดังนั้นแต่ละโหนดจะมีต้นไม้ย่อยได้มากที่สุดเพียง 2 ต้นไม้

- ต้นไม้ย่อยทางซ้ายว่า Left subtree หรือ left child กรณีมีต้นไม้ย่อยเพียง 1 โหนด
- ต้นไม้ย่อยทางขวาเรียกว่า Right subtree หรือ Right child กรณีมีต้นไม้ย่อยเพียง 1 โหนด



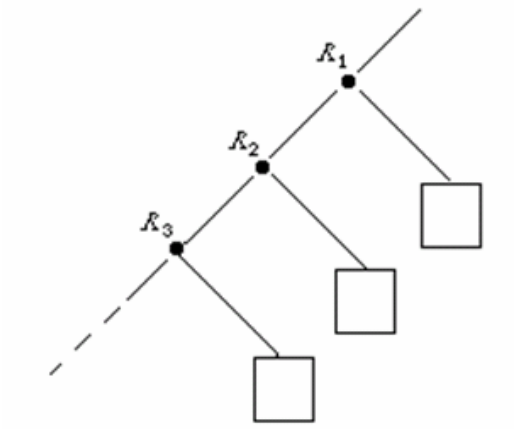
รูปที่ 4.17 : แสดงต้นไม้ที่มีต้นไม้ย่อยทางซ้ายและทางขวา

ดังนั้นจากตัวอย่างสามารถสร้างต้นไม้ได้ดังนี้ ในการค้นหาเอกสารที่ต้องการนั้นจะเริ่มค้นหาจาก root โดยทั่วไปเพื่อให้มีประสิทธิภาพมากต้องนำคีย์มาเรียงลำดับก่อนทำให้แบ่งคีย์ออกเป็น 2 ชุด และการค้นหาในแต่ละระดับจะทิ้งชุดที่ไม่มีคีย์ออกไปครึ่งหนึ่ง ดังนั้นเวลาเฉลี่ยในการค้นหาจึงมีค่าเท่ากับ log2 N



รูปที่ 4.18 : แสดง โครงสร้างต้นไม้ที่แทน dendrogram

เราสามารถนำโครงสร้างต้นไม้แทน dendrogram ของเอกสารได้ดังรูปที่ 4.18 ปัญหาของการค้นหาข้อมูลในโครงสร้างนี้คือต้นไม้มีการเอียงไปด้านใดด้านหนึ่ง ดังรูปที่ 4.19 ทำให้เสียเวลาในการสืบค้น การแก้ไขนั้นต้องพยายามที่จะรักษาต้นไม้ให้คงคุณสมบัติการเรียงอันดับไว้ในรูปแบบใกล้กับความเต็มต้น เพื่อความรวดเร็วในการใส่และการค้นหา พยายามที่จะทำให้ต้นไม้มีความสูงได้ดุลย์ (Balanced Tree) โดยสร้างต้นไม้ใหม่



รูปที่ 4.19 : ตัวอย่างของ degenerate tree

ในความเป็นจริง เราสามารถหาเอกสารที่ใกล้เคียงกับความต้องการมากที่สุด ซึ่งอาจจะจัดให้เอกสารที่คล้ายคลึงกันจัดเป็นกลุ่มใกล้เคียงกัน ซึ่งส่งผลให้ต้นไม้ไม่ได้ดุล ดังนั้นการใช้โครงสร้างต้นไม้ในการจัดแบ่งกลุ่มเอกสารนั้นจึงมักใช้โครงสร้างแบบต้นไม้ทั่วไป (General Tree Structure) มากกว่า Binary Tree เพราะไม่มีการจำกัดจำนวนกิ่งที่ออกมาจากโหนดนั้น

กราฟ (Graph)

โครงสร้างแบบต้นไม้หรือทรี ซึ่งเป็นกราฟชนิดพิเศษอย่างหนึ่ง โดยความสัมพันธ์ระหว่างโหนดเป็นแบบลำดับชั้น โดยชั้นบนเป็นพ่อแม่และระดับชั้นล่างเป็นลูก ซึ่งทฤษฎีของกราฟสามารถนำมาประยุกต์ใช้ในการแก้ปัญหางานอื่นๆได้อีกเช่น ปัญญาประดิษฐ์ วงจรไฟฟ้า ฯลฯ

กราฟประกอบด้วยเซตของโหนดและเอจ (edge) โดยที่โหนดเป็นข้อมูลเบื้องต้นของกราฟ และเอจ เป็นเส้นเชื่อม (link) ระหว่างโหนดในกราฟ ดังนั้นเราสามารถกล่าวอีกนัย คือ

กราฟ G ประกอบด้วย

1. เซต V ของข้อมูลเรียกว่า โหนด (node) ,จุด (point) หรือ vertice

2. เซต E ของเส้นเชื่อม ซึ่งแต่ละเส้นเชื่อม e ใน E เทียบเท่ากับคู่หน่วยที่ไม่เรียงลำดับของ [u ,v] ของโหนดใน V ซึ่งสามารถแสดงได้โดย e = [u ,v]

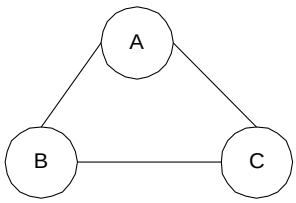
บางครั้งเราแสดงส่วนของกราฟด้วยสมการ G = (V , E)

ถ้า e = [u ,v] แล้วโหนด u และ v จะเรียกว่าเป็น endpoint ของ e โดย u และ v นั้นกล่าวได้ว่าเป็น adjacent nodes หรือ neighbors โดย degree ของโหนด u เรียกว่า deg(u) เป็นจำนวนของเส้นเชื่อมที่อยู่บน u ถ้า deg(u) = 0 แสดงว่า u เป็นโหนดที่ไม่เชื่อมต่อกับโหนดใดบนกราฟ ซึ่งเรียกว่า isolated node

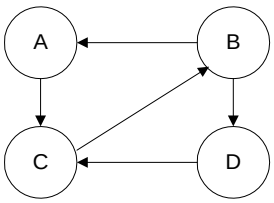
วิถี (path) P มีความยาวเท่ากับ n หมายความว่า จำนวนโหนดจากโหนด u ถึงโหนด v ซึ่งเชื่อมด้วยเส้นเชื่อมที่เป็นอันดับต่อเนื่องกันจำนวน n + 1 โหนด แทนด้วย

$$P = (v_1 , v_2 , v_3 , . . . , v_n)$$

กราฟแบ่งออกเป็น 2 ชนิด คือ กราฟแบบไม่มีทิศทาง (undirected graph) และกราฟแบบมีทิศทาง (directed graph)



กราฟแบบไม่มีทิศทาง



กราฟแบบมีทิศทาง

รูปที่ 4.20 แสดงตัวอย่างกราฟแบบไม่มีทิศทางและมีทิศทาง

รูปที่ 4.20 แสดงตัวอย่างความสัมพันธ์ระหว่างโหนดและเอจด้วยแผนภาพ รูปวงกลมแทนโหนดของกราฟ (node) เส้นตรงที่มีลูกศรและไม่มีลูกศรแทนเอจที่เชื่อมระหว่างโหนด โดยกราฟที่มีเส้นเชื่อมเป็นลูกศรจะเป็นกราฟแบบมีทิศทาง พิจารณารูปกราฟที่มีทิศทาง โหนด B , deg (B) มีค่าเท่ากับ 2 , โหนด C มี deg (C) เท่ากับ 1

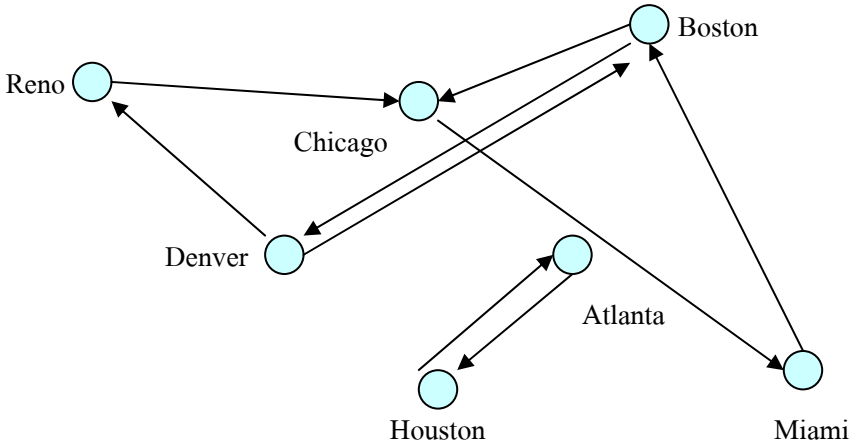
กราฟแบบไม่มีทิศทาง เส้นเชื่อมระหว่างโหนดจะไม่มีทิศทางกำหนด ดังนั้น ถ้า AB คือ path ระหว่างโหนด A กับ B จะมีความหมายเหมือนกับ path ระหว่างโหนด B กับ A

กราฟแบบมีทิศทาง เส้นเชื่อมระหว่างโหนดจะมีทิศทางกำหนด ดังนั้น ถ้า AB คือ path ระหว่างโหนด A กับ B แต่ BA ไม่เป็น path ระหว่าง B กับ A เราสามารถใช้สัญลักษณ์ A---> B แสดงการเชื่อมระหว่างโหนด A ไปโหนด B ซึ่งการเชื่อมนี้มีด้วยกัน 2 ลักษณะ คือ เชื่อมแบบทางตรง (directly connect) และเชื่อมแบบทางอ้อม (indirectly connect) เช่น A ต่อทางตรงกับ B สามารถใช้สัญลักษณ์ได้ดังนี้ A--->B ส่วน A เชื่อมทางอ้อมกับ B สามารถใช้สัญลักษณ์ได้ดังนี้ A--->C , C--->B หรือ A--->C----->B เป็นต้น

ตัวอย่างที่ 4.8 สายการบิน Friendly Airways มีเที่ยวบินประจำวัน 9 เที่ยว ดังต่อไปนี้

103 Atlanta to Houston
104 Houston to Atlanta
201 Boston to Chicago
203 Boston to Denver
204 Denver to Boston
301 Denver to Reno
305 Chicago to Miami
308 Miami to Boston
402 Reno to Chicago

จากรายละเอียดที่ให้มาสามารถเขียนเป็นกราฟได้ดังนี้

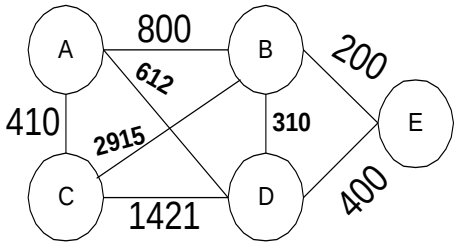


ปกติการประยุกต์ใช้กราฟในงานต่างๆนั้นไม่เพียงแต่มี edge ที่เป็นเส้นเชื่อมระหว่างโหนด เท่านั้น แต่ว่าอาจมีการระบุค่าให้กับ edge แต่ละเส้นบนกราฟด้วย เช่น ปัญหาการหาเส้นทางนั้นเราไม่สนใจเพียงเส้นทางเดินระหว่างโหนดเท่านั้น แต่เราสนใจระยะทางที่เชื่อมระหว่างโหนดด้วย ค่าที่เป็นตัวเลขซึ่งแทนระยะทางนี้เรียกว่า ค่าถ่วงน้ำหนัก (weight) เราสามารถปฏิบัติการกับกราฟที่แต่ละเอจมีค่า weight อยู่ใน ข่ายงาน (network) ได้ โดยค่าของ edge ที่เชื่อมระหว่างโหนดในข่ายงาน เราสามารถใช้ แถวอันดับ adjacency matrix เก็บค่าของ weight นั่นเอง

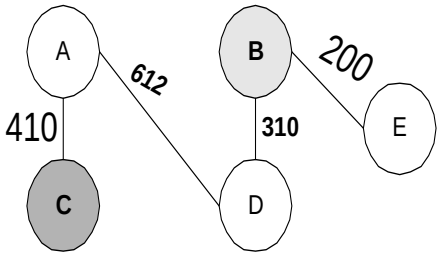
Spanning tree ที่มีค่าต่ำสุด (Minimum Spanning tree)

สมมุติว่าปัญหาของเราคือการหาเส้นทางที่เชื่อมทุกโหนดเข้าด้วยกัน โดยที่ผลรวมของระยะทางบนกราฟที่เชื่อมทุกโหนดต้องมีระยะที่สั้นที่สุด การแก้ปัญหานี้เราสามารถใช้อัลกอลิทึมซึ่งเปลี่ยนรูปทางมโนภาพของข่ายงานเป็นโครงสร้างแบบต้นไม้ ซึ่งเรียกต้นไม้แบบนี้ว่า minimum spanning tree

อัลกอลิทึมนี้ให้ผลลัพธ์เฉพาะเส้นที่เชื่อม (edge) ระหว่างโหนดบนกราฟ เมื่อนำค่าของ edge มารวมกันแล้วจะได้ระยะทางที่สั้นที่สุดบนกราฟ รูปที่ 4.21 แสดงตัวอย่างของข่ายงานและ Minimum Spanning บนข่ายงาน รูปที่ 4.21.1 แสดงตัวอย่างรูปของข่ายงาน และรูปที่ 4.21.2 แสดงตัวอย่างแถวอันดับที่เก็บค่า weight ของกราฟรูปนี้



รูปที่ 4.21.1 ตัวอย่างกราฟ



รูปที่ 4.21.2 minimum spanning tree

รูปที่ 4.21 แสดงตัวอย่างของข่ายงานและ Minimum Spanning

4.5 Direct File

โครงสร้างแฟ้มข้อมูลชนิดนี้อาจเรียกว่า Scatter Storage หรือ Hash Addressing เทคนิคนี้เป็นการค้นหาแบบ Hashing ซึ่งใช้หลักการโดยมีค่าคีย์แบบหนึ่งในการค้นหาข้อมูล วิธีนี้เหมือนกับการอ้างข้อมูลชนิดแถว โดยค่าคีย์เป็นเสมือนดัชนีใน Array วิธีนี้จะแปลงค่าคีย์ให้เป็นค่าแอดเดรสของคีย์ โดยใช้ฟังก์ชั่นที่เรียกว่า Hashing function หรือ key-to-address transformation โดยใช้สัญลักษณ์ H(k) ซึ่ง K คือค่าคีย์ที่ใช้ในการค้นหาข้อมูล

ค่าคีย์ที่ใช้ในการค้นหาข้อมูลนั้นอาจเป็นตัวอักษรล้วน เช่น ชื่อบุคคล หรือเป็นตัวเลขล้วนๆก็ได้ เราต้องเลือกฟังก์ชั่นในการแปลงค่าคีย์ให้เป็นตัวเลขซึ่งตัวเลขนี้คือตำแหน่งที่อยู่ของข้อมูลที่เราต้องการบันทึกหรือแก้ไขหรือค้นหานั่นเอง ค่าคีย์ที่เลือกใช้ต้องมีค่าเดียวไม่ซ้ำกัน ปัญหาที่เกิดขึ้นในการแปลงค่าคีย์ให้เป็นตำแหน่งที่อยู่ของข้อมูลคือการซ้ำกันของตำแหน่งที่อยู่ข้อมูล กล่าวคือเมื่อแปลงค่าคีย์ที่ใช้ในการค้นหาเป็นตำแหน่งที่อยู่ของข้อมูลแล้วเป็นค่าตำแหน่งที่มีข้อมูลบรรจุอยู่แล้ว คือได้ค่าตำแหน่งที่ตรงกัน ซึ่งเราเรียกการเกิดในลักษณะนี้ว่า การชนกัน (collisions) ดังนั้นการเลือกใช้ฟังก์ชั่นในการแปลงเป็นสิ่งที่ต้องคำนึงถึง โดยเลือกใช้ฟังก์ชั่นที่มีการกระจายข้อมูลหรือค่าคีย์ไปสู่ตำแหน่งต่างๆให้มากที่สุด

ในกรณีที่เกิดการชนกันเกิดขึ้น เราสามารถแก้ไขได้หลายวิธีการ วิธีการที่ง่ายที่สุดในกรณีที่ต้องการบันทึกข้อมูลใหม่ คือเลื่อนไปยังระเบียนว่างในลำดับต่อไปแล้วนำข้อมูลใหม่บันทึกยังที่ว่างที่ใกล้กับตำแหน่งที่คำนวณได้มากที่สุด และในกรณีค้นหาถ้าตำแหน่งที่คำนวณได้ไม่ใช่ข้อมูลที่ต้องการเราสามารถค้นหาข้อมูลที่ต้องการในระเบียนถัดไป ซึ่งยังนับว่าการค้นหาข้อมูลยังคงรวดเร็วและมีประสิทธิภาพ เพราะข้อมูลที่ต้องการยังคงอยู่ในลำดับถัดไปที่ไม่ไกลจากตำแหน่งที่คำนวณได้อยู่ดี

ฟังก์ชั่นต่างๆที่ใช้ในการแปลงค่าคีย์ให้เป็นตำแหน่งที่อยู่นี้ มีด้วยกันหลายวิธี ที่นิยมใช้กันมาก มีดังนี้

(a) Division method

เป็นวิธีการหาร รูปแบบของวิธีการนี้คือ

$$H(k) = k \bmod m + 1$$

โดย K คือคีย์ที่ใช้ในการค้นหา

M คือตัวเลขจำนวนเต็มที่ใช้ในการหาร ส่วนมากเลือกเป็นเลขจำนวนเฉพาะที่ใกล้เคียงกับจำนวนข้อมูลมากที่สุด

วิธีการนี้เป็นวิธีการที่ง่ายที่สุด เพราะเพียงแต่นำคีย์ซึ่งเป็นเลขจำนวนเต็มหารด้วย m แล้วบวกด้วย 1 เท่านั้น

ตัวอย่างที่ 4.9

สมมุติว่าหนังสือของหน่วยงานหนึ่งมีจำนวนสูงสุดไม่เกิน 100 เล่ม แต่การกำหนดรหัสหนังสือป็นตัวเลข 4 หลัก เช่น

รหัสหนังสือ	ชื่อหนังสือ	สำนักพิมพ์
3205	Pascal Programming	ซีเอ็ดบุ๊ค
7148	C++	มหาวิทยาลัยรามคำแหง
2345	Programming Language	คุรุสภา
9524	JAVA	สยาม
..	.. ..	..

ถ้าเราเก็บข้อมูลแบบเรียงลำดับข้อมูลของหนังสือจะเก็บเรียงลำดับไปเรื่อยๆ การค้นหาจะเริ่มจากระเบียนแรกไปเรื่อยๆ กรณีที่ต้องการค้นหาแบบ Binary จะต้องทำการเรียงลำดับข้อมูลเสียก่อนโดยเรียงตามรหัสหนังสือ ในกรณีของ Hashing นี้การเก็บข้อมูลจะแตกต่างจากแบบแรกคือข้อมูลไม่เก็บต่อเนื่องกันไป ซึ่งข้อมูลจะกระจัดกระจายมีช่องว่างระหว่างระเบียนเกิดขึ้น ในที่นี้มีหนังสือทั้งหมด 100 คนต้องจัดสรรเนื้อที่เพื่อเก็บข้อมูลได้ 100 ระเบียนเต็มโดยทั่วไปการจัดเก็บในลักษณะนี้ต้องเผื่อให้เกินไว้อย่างน้อย 20 % ของปริมาณข้อมูลทั้งหมด เพื่อความรวดเร็วในการค้นหาและบันทึก เพราะการคำนวณอาจจะเกิดการชนกันได้มากนั่นเอง

จากตัวอย่างรหัสหนังสือเป็นตัวเลข 4 หลัก แต่จำนวนหนังสือ หรือจำนวนระเบียนที่จัดเก็บไม่เกิน 100 คน การเลือกค่า m เราควรเลือกค่า 97 เพราะเป็นเลขจำนวนเฉพาะที่ไม่มีตัวเลขใดหารลงตัว เมื่อนำมาหารกับตัวเลขใดๆ ความน่าจะเป็นในการหารแล้วเหลือเศษจะซ้ำกันน้อยกว่าค่าใดๆนั่นเอง จากตัวอย่างนี้สามารถคำนวณตำแหน่งที่อยู่ได้ดังนี้

H(3205) = 3205 mod 97 + 1
= 5

H(7148) = 7148 mod 97 +1
= 68

H(2345) = 2345 mod 97 + 1
= 8

เมื่อนำค่าคีย์ซึ่งคือรหัสหนังสือมาแปลงเป็นตำแหน่งที่อยู่นั้น เมื่อนำมาหารด้วย 97 เศษที่ได้จะมีค่าระหว่าง 0 ถึง 96 ในกรณีที่เกิดการชนกันเราสามารถนำข้อมูลไปเก็บในตำแหน่งว่างถัดไปได้

(b) Midsquare

เป็นวิธีการที่นำค่าคีย์มายกกำลังสอง แล้วนำค่าหลักที่อยู่ตรงกลางของผลลัพธ์มาเป็นค่าตำแหน่งของข้อมูล

H(k) = I

ตัวอย่างที่ 4.10

สมมุติว่าหนังสือของหน่วยงานหนึ่งมีจำนวนสูงสุดไม่เกิน 100 เล่ม แต่การกำหนดรหัสหนังสือป็นตัวเลข 4 หลัก เช่น

รหัสหนังสือ	ชื่อหนังสือ	สำนักพิมพ์
3205	Pascal Programming	ซีเอ็ดบุ๊ค
7148	C++	มหาวิทยาลัยรามคำแหง
2345	Programming Language	คุรุสภา
9524	JAVA	สยาม
..	.. ..	..

วิธีนี้จะนำรหัสหนังสือมายกกำลังสอง แล้วทำการตัดตัวเลขทางซ้ายสุดและตัวเลขทางขวาสุด และนำค่าหลักที่อยู่ตรงกลางมาเป็นค่าตำแหน่งของข้อมูล

K	3205	7148	2345
K2	10272025	51093904	5499025
H(k)	72	93	99

วิธีการนี้ในกรณีที่ค่าคีย์มีจำนวนหลักมากๆ การยกกำลังสองอาจจะส่งผลให้ยุ่งยากได้

(c) Folding

ในกรณีที่คีย์มีจำนวนหลักมากๆ สมมุติว่า รหัสหนังสือมีจำนวนตัวเลขหลายหลักเช่น 4251047856 เป็นต้น การนำค่ารหัสหนังสือมายกกำลังสอง อาจจะยุ่งยากเพราะเกินขีดความสามารถของเครื่องคอมพิวเตอร์ได้ จึงมีวิธีการที่พัฒนาการแปลงค่าโดยนำค่าคีย์ที่มีจำนวนหลักมากๆ แบ่งออกเป็นส่วนๆ ส่วนแล้วนำมาพับรวมกัน ก็ได้

H(k) = k1 + k2 + k3 + … + kr

วิธีนี้แบ่งค่าคีย์ เป็นส่วนๆ โดยที่แต่ละส่วนมีจำนวนหลักของเลขสำหรับชี้ตำแหน่งเท่ากัน ยกเว้นส่วนสุดท้าย ต่อจากนั้นนำส่วนต่างๆมารวมกันแล้วตัดตัวทดออก

ตัวอย่างที่ 4.11

สมมุติว่าหนังสือของหน่วยงานหนึ่งมีจำนวนสูงสุดไม่เกิน 100 เล่ม แต่การกำหนดรหัสหนังสือป็นตัวเลข 4 หลัก เช่น

รหัสหนังสือ	ชื่อหนังสือ	สำนักพิมพ์
3205	Pascal Programming	ซีเอ็ดบุ๊ค
7148	C++	มหาวิทยาลัยรามคำแหง
2345	Programming Language	คุรุสภา
9524	JAVA	สยาม
..	.. ..	..
..	.. ..	

วิธีการ Folding นี้ เนื่องจากคีย์คือรหัสหนังสือเพียง 4 หลัก เราอาจแบ่งค่าคีย์ออกเป็น 2 ส่วน แล้วนำมารวมกันจะได้

H(3205) = 32+05 = 37

H(7148) = 71+48 = 119 = 19

H(2345) = 23+45 = 68

หรืออาจกลับค่าส่วนที่สองก่อนการบวกก็ได้

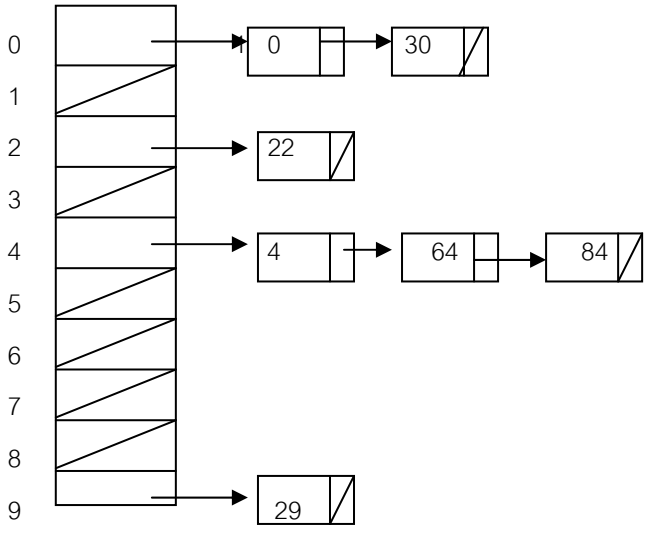
H(3205) = 32+50 = 82

H(7148) = 71+84 = 155 = 55

H(2345) = 23+54 = 77

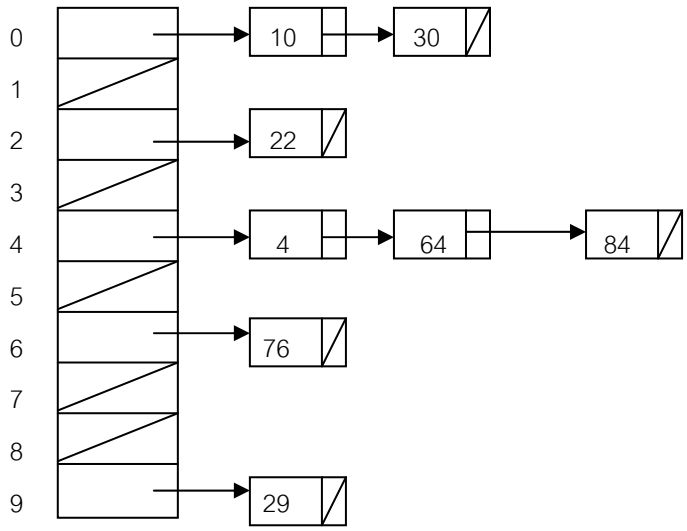
(d) Open Hashing (Separate Chaining)

การบันทึกข้อมูลหรือการค้นหาข้อมูลในกรณีที่เป็นโครงสร้างข้อมูลชนิด linked list สามารถกระทำได้หลายวิธี วิธีที่จะกล่าวถึงคือ Open hashing เป็นการนำตำแหน่งโหนดของข้อมูลเก็บเชื่อมโยงกัน โดยตำแหน่งโหนดข้อมูลตัวแรกของลิสต์เก็บในโครงสร้างชนิดแถว ดังรูปที่ 4.21



รูปที่ 4.22 : แสดง Open Hashing

จากรูปที่ 4.22 เห็นได้ว่า ต้องมีการแปลงค่าคีย์ให้เป็นตำแหน่งที่อยู่ในที่นี้ตำแหน่งที่อยู่มีค่าเพียง 10 ค่าเริ่มจาก 0 ถึง 9 ในที่นี้ใช้วิธีการแปลงค่าวิธี Division method โดยนำค่า 10 หารค่าคีย์เศษที่ได้คือตำแหน่งที่อยู่ของข้อมูล ในกรณีที่ค่าคีย์คือ 76 เมื่อนำ 10 ไปหารได้เศษเท่ากับ 6 จะนำโหนดข้อมูลนี้มาเชื่อมโยงในลิสต์ของช่องที่มีดัชนี 6 โหนดของข้อมูลจะ



รูปที่ 4.23 : แสดง การแทรกข้อมูลใน Open Hashing

ในการค้นหาโหนดของข้อมูลนั้น เราต้องนำคีย์ไปแปลงเป็นตำแหน่งที่อยู่ของโหนด และ ค้นหาจากโหนดแรกไปเรื่อยๆจนกระทั่งพบข้อมูล หรืออีกกรณีคือไม่พบข้อมูล

ในการแทรกข้อมูลใหม่เข้าไปใน Open hashing นี้สามารถกระทำได้ไม่ยาก เพียงแต่คำนวณตำแหน่งของโหนดข้อมูลว่าอยู่ในช่องใดของตัวแปรชนิดแถว H และทำการแทรกโหนดใหม่เข้าไปใน ลิสต์ ซึ่งการแทรกอาจเป็นการแทรกโหนดใหม่เป็นตัวแรก หรือเป็นตัวสุดท้าย หรือ ระหว่างกลางโดยเรียงลำดับก็ได้ ขึ้นอยู่กับความต้องการและความเหมาะสมในการดำเนินการ

แบบฝึกหัด

1. จงอธิบายถึงภาษาสำหรับแสดงโครงสร้างแฟ้มข้อมูล
2. จงอธิบายถึงโครงสร้างของแฟ้มข้อมูลแบบเรียงลำดับ(Sequential File) พร้อมทั้งอธิบายถึงข้อดีข้อเสียของแฟ้มชนิดนี้
3. จงอธิบายถึงโครงสร้างของ Invert File พร้อมยกตัวอย่างประกอบการอธิบายมาพอเข้าใจ
4. จงอธิบายถึงการสร้าง Invert File ในระบบค้นคืนสารสนเทศ พร้อมยกตัวอย่างประกอบการอธิบายมาพอเข้าใจ
5. จงอธิบายถึง Index-Sequential File มาพอเข้าใจ
6. จงอธิบายถึงโครงสร้างข้อมูลที่ไม่เป็นเชิงเส้น ชนิด Tree และ Graph มาพอเข้าใจ
7. จงอธิบายถึงโครงสร้างแฟ้มข้อมูลที่เรียกว่า Scatter Storage มาพอเข้าใจ

บรรณานุกรม

สุมาลี เมืองไพศาล ,"การจัดการข้อมูล และการเรียกใช้ข้อมูล" ,สำนักพิมพ์มหาวิทยาลัยรามคำแหง ,CS337 ,2535

ดร.ชุลีรัตน์ จรัสกุลชัย ,"**Texual Application** ", intelligentce information Retrieval and database, ภาควิชาวิทยาการคอมพิวเตอร์ มหาวิทยาลัยเกษตรศาสตร์

Lawrence, S., Giles, C.L. 1999. Accessibility of Information on the Web. Nature. Vol.400.pp.107-109.

Ribeiro-Neto, B.A., Barbosa, R.A. 1998. Query Performance for Tighly Coupled Distributed Digital Libraries. Digital Libraries 98. pp.182-190.

Ribeiro-Neto, B.A., Moura, E.S., Neubert, M.S., Ziviani, N. 1999. **Efficient Distributed Algorithms to Build Inverted Files**. SIGIR'99. pp.105-112

http://www.cs.sci.ku.ac.th/~chulee/course/45/wu/handout/wu-1-text.ppt#310,33,การจัดการแฟ้มข้อมูล

www.cs.sci.ku.ac.th/~chulee/course/45/wu/handout/wu-1-text.ppt

http://www.cs.su.ac.th/~sirak/517632/