บทที่ 2

ทฤษฎีที่เกี่ยวข้อง

ในปัจจุบันวิธีการค้นหาข้อมูลเพื่อให้ได้ผลลัพธ์ของการค้นหาที่รวดเร็ววิธีการหนึ่งที่ได้รับความนิยมคือ การใช้ดัชนี (Index) โดยในบทนี้จะกล่าวถึงความสำคัญของดัชนี พร้อมทั้งตัวอย่างดัชนีที่ได้รับความนิยมสูงมากในปัจจุบันคือ ดัชนีบีพลัสทรี (B+ tree) และดัชนีอาร์ทรี (R tree) ต่อจากนั้นจะกล่าวถึงโครงสร้าง พร้อมทั้งตัวอย่างการแทรก (Insertion) ของดัชนีวายทรี (Y-tree) ซึ่งเป็นส่วนสำคัญของวิทยานิพนธ์ฉบับนี้ และในส่วนสุดท้ายจะกล่าวถึงคุณสมบัติต่างๆ ที่สำคัญของแฟลชไดร์ฟ (Flash Drives)

2.1 ความสำคัญของดัชนี

การเข้าถึงข้อมูลของดิสก์ (Access Disk) ในแต่ละครั้งนั้นมีต้นทุน (Cost) และถ้าข้อมูลมีปริมาณมาก ต้นทุนจะมีค่ามากเช่นกัน เพื่อลดต้นทุนของการเข้าถึงข้อมูลให้น้อยลง การนำดัชนีเข้ามาช่วยเป็นวิธีการหนึ่งที่ได้รับความนิยมมาก การทำดัชนีเป็นการจัดเรียงตำแหน่งของข้อมูลเพื่อให้ง่ายต่อการค้นหา และใช้เวลาการค้นหาตำแหน่งของข้อมูลน้อยลง ผลลัพธ์ที่ได้คือเวลาการเข้าถึงข้อมูลน้อยลงเช่นกัน ในปัจจุบันนี้ดัชนีมีอยู่หลายรูปแบบ แต่ละแบบสร้างขึ้นมาเพื่อจุดประสงค์เดียวกัน คือต้องการลดเวลาของการเข้าถึงข้อมูลให้น้อยที่สุดตามลักษณะของข้อมูลนั้นๆ แต่ดัชนีที่ได้รับความนิยมสูงมากในปัจจุบันคือ ดัชนีบีพลัสทรี และดัชนีอาร์ทรี สาเหตุที่ได้รับความนิยมเนื่องมาจากวิธีการเข้าถึงข้อมูลที่ไม่ซับซ้อน และมีความรวดเร็วกว่าดัชนีแบบอื่น

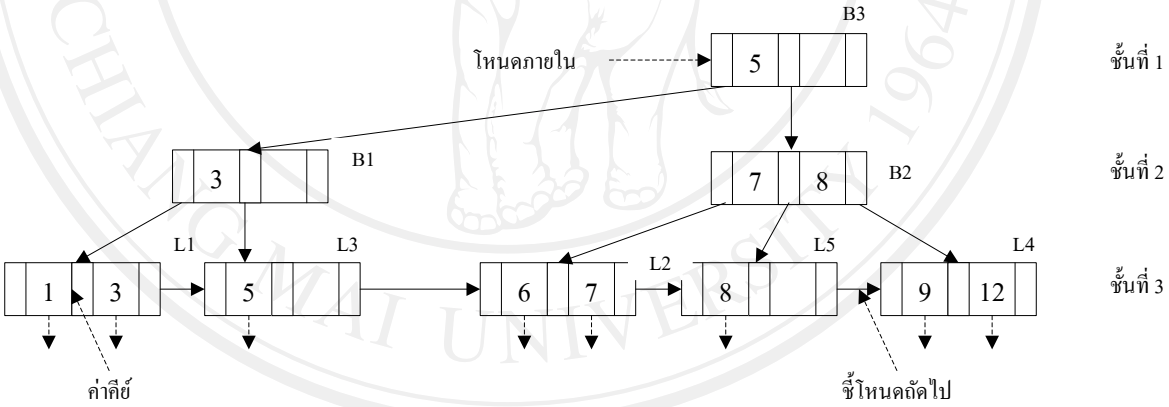
2.2 ดัชนีบีพลัสทรี

ดัชนีบีพลัสทรีได้นำคุณสมบัติเด่นของ ISAM (Indexed Sequential Access Method) และดัชนีบีทรี (B tree) มาประกอบรวมกัน โดยมีแนวคิดว่า ถ้าทรี (Tree) มีความสูงน้อยเวลาการเข้าถึงข้อมูลจะใช้เวลาน้อย นอกจากนั้นข้อมูลของดัชนีบีพลัสทั้งหมดจะเก็บอยู่ที่โหนดใบ (Leaf Node) และจุดเด่นที่สำคัญของดัชนีบีพลัสทรีคือ ที่โหนดใบจะมีพอยเตอร์ (Pointer) ซึ่งชี้ไปยังโหนดใบถัดไปทางขาวของทรี ทำให้การสอบถามแบบช่วง (Range Query) มีความรวดเร็วมาก

2.2.1 โครงสร้างดัชนีบีพลัสทรี (B+ tree Structure)

ดัชนีบีพลัสทรีเป็นโครงสร้างแบบต้นไม้ที่มีความสูงสมดุล (Height Balanced Tree) อีกชนิดหนึ่ง ซึ่งมีคุณสมบัติเด่นของการเข้าถึงข้อมูลที่มีความเร็วสูงทำให้รับความนิยมเป็นอย่างมาก โดยอาจกล่าวได้ว่าความเร็วของการเข้าถึงข้อมูลที่เกิดขึ้นนั้นมีสาเหตุหนึ่งมาจากโครงสร้างของดัชนี แสดงได้ดังรูปที่ 2.1

จากรูปที่ 2.1 เป็นโครงสร้างของดัชนีบีพลัสทรีที่มีทั้งหมด 3 ชั้น (Level) โดยในแต่ละชั้นจะมีโหนด (Node) ซึ่งบรรจุค่าคีย์ (Key Value) และโหนดเหล่านี้จะบรรจุอยู่ในทุกๆ ชั้นของโครงสร้างดัชนี ในกรณีที่เป็นชั้นที่ 1 จะเรียกว่าโหนดราก (Root Node) และชั้นที่ 3 โหนดที่บรรจุอยู่จะเรียกว่าโหนดใบ โดยในโหนดใบนี้จะมีลักษณะพิเศษคือ ด้านขวาของโหนดใบจะมีพอยเตอร์ที่ชี้ไปยังโหนดใบถัดไป ประกอบกับค่าคีย์ที่อยู่ในโหนดใบนั้นจะต้องเรียงลำดับ (Order) อีกด้วย ด้วยวิธีการเช่นนี้ทำให้การสอบถามแบบช่วง สามารถทำได้เร็วขึ้น กล่าวคือถ้าสามารถค้นหาข้อมูลเริ่มต้นของการสอบถามแบบช่วงได้แล้ว จะพบข้อมูลตัวถัดมาได้เลยโดยอาศัยพอยเตอร์ชี้ตัวถัดไป



รูปที่ 2.1 โครงสร้างของดัชนีบีพลัสทรี

2.2.2 คุณสมบัติของดัชนีบีพลัสทรี

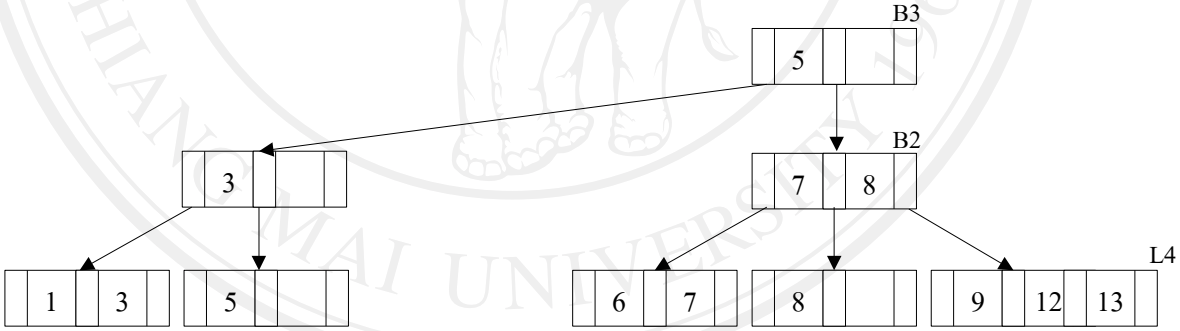
จากโครงสร้างดังรูปที่ 2.1 มีค่าความสูงเป็น 3 โดยเขียนแทนด้วย h ดังนั้นจำนวนสูงสุดของคีย์คือ $n = b^h$ ซึ่งมีค่าเป็น 8 โดยที่ b คือจำนวนลำดับ (Order) ของทรีมีค่าเป็น 2 และจำนวนน้อยที่สุดของคีย์คือ $n = 2\left(\frac{b}{2}\right)^{h-1}$ ซึ่งมีค่าเป็น 2 สมรรถนะการแทรก การค้นหา และการลบข้อมูลมีสมรรถนะสูงสุดคือ $b_p = O(\log_b n)$ และสมรรถนะของการสอบถามแบบช่วงมี

สมรรถนะสูงสุด $r_p = O(\log_b n + k)$ เมื่อ k คือ จำนวนคีย์คำตอบของบีพลัสทรี นอกจากนี้ยังกำหนดอีกว่าข้อมูลทั้งหมดจะต้องเก็บที่โหนดใบเท่านั้น พร้อมกับความสูงของทรีทุกโหนดใบต้องมีค่าเท่ากัน

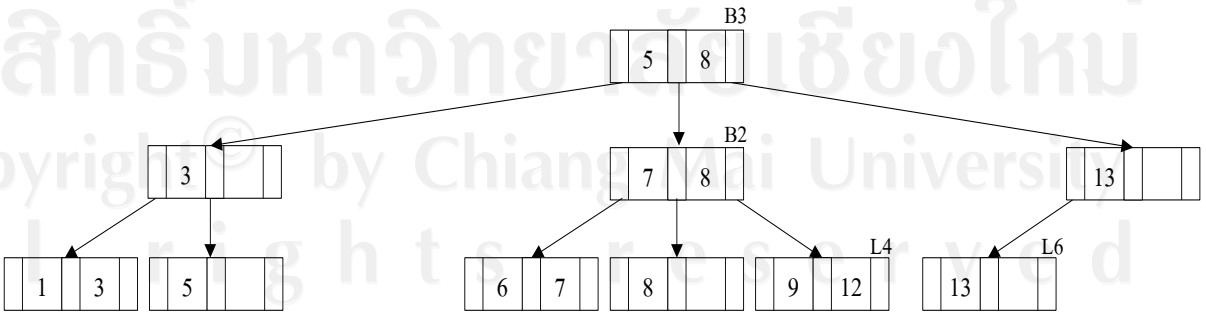
ดัชนีบีพลัสทรีอนุญาตให้กำหนดค่าออร์เดอร์ (Order) ของทรีได้เอง นั่นหมายถึงว่าถ้ากำหนดจำนวนลำดับนี้มากขึ้นผลคือจะทำให้ความสูงของทรีลดลง การเข้าถึงข้อมูลมีความเร็วเพิ่มมากขึ้น และที่สำคัญคือ จำนวนครั้งการเข้าถึงดิสก์ (Disk Access) น้อยลง

2.2.3 การแทรกข้อมูลในดัชนีบีพลัสทรี

จากรูปที่ 2.1 เมื่อต้องการแทรกค่าคีย์ 13 สามารถแสดงได้ดังรูปที่ 2.2 โดยเริ่มจากเทียบค่าคีย์ที่ต้องการแทรกในดัชนีบีพลัสทรีเพื่อหาตำแหน่งที่ต้องการแทรกในที่นี้คือ โหนด L4 เมื่อแทรกค่าคีย์เข้าไปแล้วจำนวนลำดับของโหนดนี้มากกว่าค่าที่กำหนดไว้ จึงต้องทำการแยกโหนด (Split Node) จะได้โหนดใหม่ L6 ต่อจากนั้นต้องปรับปรุง (Update) โหนดแม่ (Parent Node) B2 และยังมีผลทำให้ต้องแยกโหนดแม่นี้อีก โดยมีโหนดใหม่คือ B4 และสุดท้ายต้องปรับปรุงที่โหนดราก (Root Node) B4 เพื่อเชื่อมโยงไปหาโหนด B3 จนได้ผลสำเร็จเป็นดังรูปที่ 2.2b



a)



b)

รูปที่ 2.2 การแทรกค่าคีย์ 13 ในดัชนีบีพลัสทรี

จากการแทรกค่าคีย์ข้างต้น จะเห็นว่ามีจำนวนการเข้าถึงดิสก์ถึง 5 ครั้ง ดังนี้ 1) แทรก 13 ใน L4 2) สร้างโหนดใหม่ L6 3) ปรับปรุงโหนด B2 4) สร้างโหนดใหม่ B4 5) ปรับปรุงโหนด B3 จะเห็นได้ว่าเพียงการแทรกค่าคีย์ 13 เพียงค่าเดียวในกรณีของรูปที่ 2.1 ต้องใช้จำนวนการเข้าถึงดิสก์มากถึง 5 ครั้ง และในกรณีที่ต้องการใช้ดัชนีบีพลัสเก็บข้อมูลในปริมาณที่มาก จะทำให้ขนาดของทรีมีขนาดใหญ่ขึ้น และต้องใช้จำนวนครั้งการเข้าถึงดิสก์มากเช่นกัน จึงทำให้เกิดความล่าใช้ในกรณีการแทรกข้อมูลของดัชนีบีพลัสทรี ซึ่งคือจุดด้อยของดัชนีบีพลัสทรี

2.3 ดัชนีอาร์ทรี

ดัชนีอาร์ทรีเป็นโครงสร้างดัชนีแบบต้นไม้ที่สามารถบันทึกข้อมูลหลายมิติได้ โดยอักษรอาร์ (R) ในชื่อของดัชนีย่อมาจาก Rectangle ซึ่งใช้เป็นแนวคิดการมองวัตถุเป็นรูปสี่เหลี่ยมผืนผ้า และมีลักษณะเป็นโครงสร้างแบบต้นไม้ที่มีความสูงสมดุลอีกชนิดหนึ่ง นอกจากนี้ยังมีลักษณะคล้ายดัชนีบีทรี และสามารถรองรับการเก็บข้อมูลในดิสก์ได้เช่นเดียวกับดัชนีบีทรี

2.3.1 โครงสร้างดัชนีอาร์ทรี (R tree Structure)

โครงสร้างของดัชนีอาร์ทรีแสดงได้ดังรูปที่ 2.3 โดยข้อกำหนดของโหนดใบ และโหนดภายใน (Internal Node) ดังนี้

โหนดใบ ประกอบด้วยหน่วยข้อมูลที่เป็นระเบียนดัชนี (Index Record Entry) ที่อยู่ในรูปแบบ (*I, Tuple-identifier*) โดยที่

I คือ วัตถุเชิงพื้นที่ (Spatial Object) และ $I = (I_0, I_1, ..., I_{n-1})$ โดยที่

n คือ จำนวนมิติของข้อมูล

I_i คือ ช่วงปิด (Closed bounded interval) [a,b] ที่บอกขอบเขตของวัตถุมิติที่ i

Tuple-identifier คือ ดัชนีที่อ้างอิงไปยังระเบียนข้อมูล (Tuple) บนฐานข้อมูล

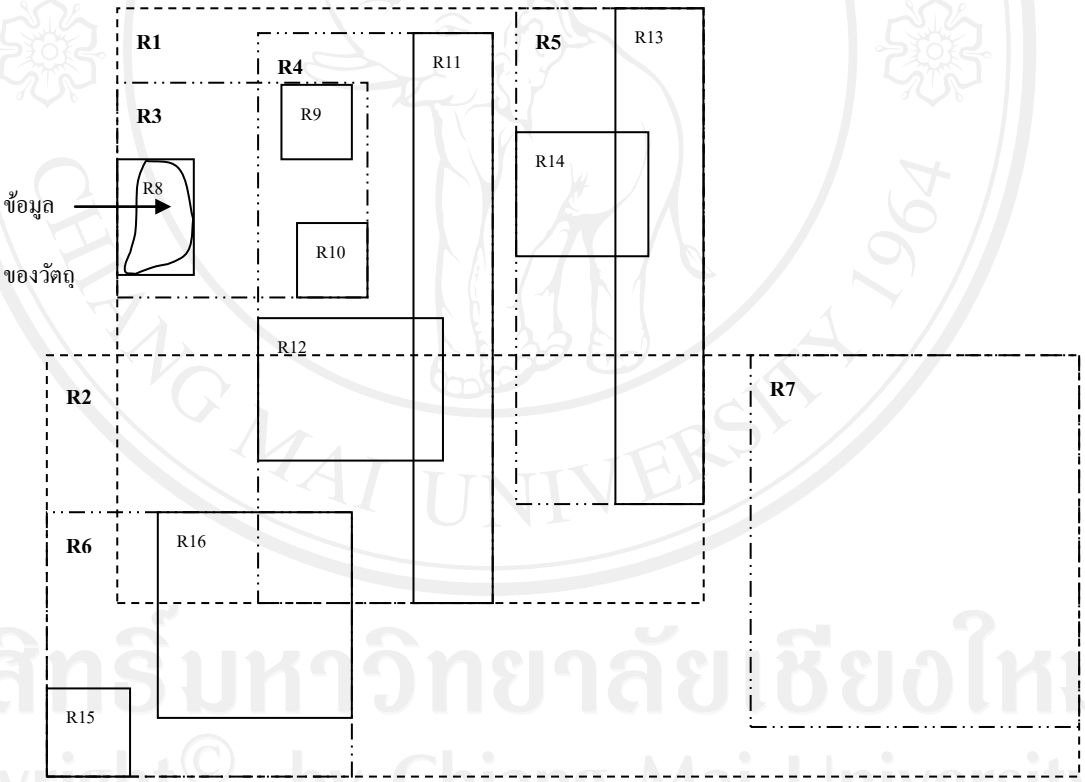
โหนดภายใน ประกอบด้วยหน่วยข้อมูล (Entry) ที่อยู่ในรูปแบบ (*I, Child - pointer*) โดยที่

I คือ ขอบเขตเชิงปริมาตร (Bounding Volume) ที่มีลักษณะเป็นรูปสี่เหลี่ยมมุม

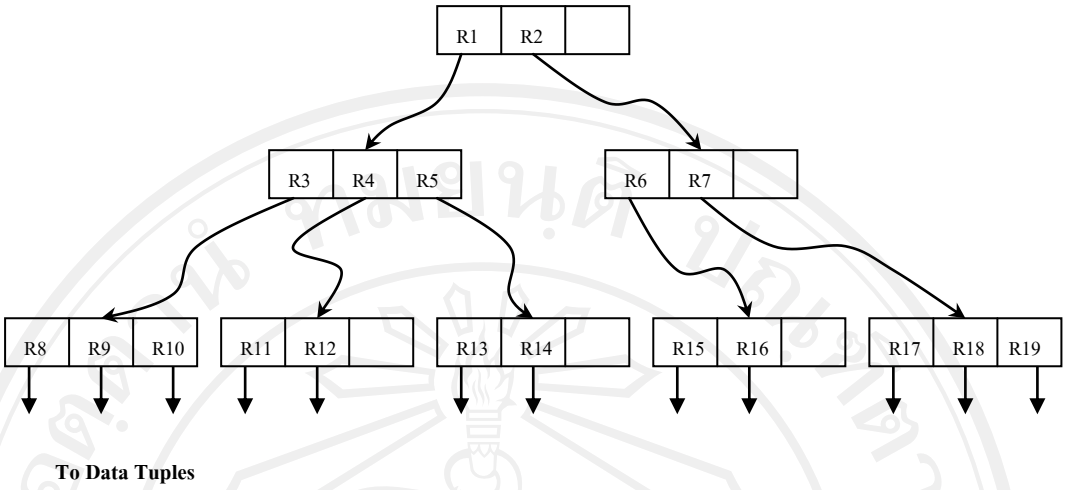
ฉากเกิน (hyper-rectangle) ที่เล็กที่สุดที่ครอบคลุมหน่วยข้อมูลทั้งหมดที่อยู่ในโหนดภายในที่ *i* ซึ่งขอบเขตเชิงปริมาตรดังกล่าวเรียกว่า Minimum Bounding Hyper-Rectangle

Child – pointer คือ ที่อยู่ของโหนดลูก หรือ ดัชนีที่ชี้ไปยังโหนดลูก

จากรูปที่ 2.3 เป็นภาพมุมมองด้านบน (Top View) ของวัตถุจะเห็นได้ว่ามีขอบเขตใหญ่ที่สุด 2 ขอบเขตที่สามารถครอบคลุมวัตถุ (Object) ได้ทั้งหมด นั่นคือ R1 และ R2 โดยกำหนดให้เป็นโหนดรากของดัชนีอาร์ทรี และนอกจากนี้ยังปรากฏขอบเขตข้อมูลที่มีขนาดเล็กลงอีกหลายขนาดของทั้ง R1 และ R2 นั่นคือโหนดภายใน และโหนดใบของดัชนีอาร์ทรีดังรูปที่ 2.4



รูปที่ 2.3 โครงสร้างของอาร์ทรี



รูปที่ 2.4 โครงสร้างของดัชนีอาร์ทรีในรูปแบบทรี

2.3.2 คุณสมบัติของดัชนีอาร์ทรี

กำหนดให้ M คือจำนวนหน่วยข้อมูลที่เก็บได้มากที่สุดในแต่ละโหนด และให้ $m \leq \left(\frac{M}{2}\right)$ โดยที่ *m* คือจำนวนหน่วยข้อมูลที่เก็บได้น้อยที่สุดในแต่ละโหนด ทำให้สามารถกำหนดคุณสมบัติของโครงดัชนีแบบอาร์ทรี ได้ดังนี้

1. โหนดใบทุก ๆ โหนดจะมีจำนวนหน่วยข้อมูลอยู่ระหว่าง *m* และ *M* ยกเว้นโหนดราก ดังนั้นโหนดรากสามารถมีจำนวนหน่วยข้อมูลได้น้อยกว่า *m*
2. โหนดใบทุกๆ โหนด *I* เป็นช่องว่างสี่เหลี่ยมผืนผ้าที่เล็กที่สุดโหนดรากที่บรรจุใน n มิติ (n-dimensional) ข้อมูลวัตถุ (Data Object) ซึ่งแทนด้วยตัวชี้ระเบียนข้อมูล (Indicated Tuple)
3. โหนดภายในทุกๆ โหนด มีโหนดลูกอยู่ระหว่าง *m* และ *M* ยกเว้นโหนดราก
4. สำหรับทุกหน่วยข้อมูลของโหนดภายใน *i* เป็นช่องว่างสี่เหลี่ยมผืนผ้าที่เล็กที่สุดที่บรรจุอยู่ในสี่เหลี่ยมผืนผ้าในโหนดลูก
5. โหนดรากมีโหนดลูก น้อยที่สุด 2 โหนดยกเว้นโหนดใบ

6. โครงสร้างแบบต้นไม้ที่มีความสูงสมดุล

2.3.3 การแทรกข้อมูลในดัชนีอาร์ทรี

การแทรกของดัชนีอาร์ทรีมีความเหมือนกับการแทรกของดัชนีบีพลัสทรี กล่าวคือ เริ่มต้นด้วยการหาโหนดที่แทรกข้อมูลได้ ต่อจากนั้นตรวจสอบว่าจะต้องแยกโหนดหรือไม่ ถ้าต้องแยกโหนดต้องปรับปรุงโหนดแม่ต่อไป ตรวจสอบทุกโหนดให้ถูกต้องตามคุณสมบัติของดัชนีอาร์ทรี ดังแสดงในรูปที่ 2.5 ดังนั้นปัญหาการเข้าถึงดิสก์ที่มีความล่าช้าในกรณีการแทรกจึงยังคงมีอยู่เช่นเดิม

2.4 ดัชนีวายทรี

ดัชนีวายทรี เป็นดัชนีประเภทหนึ่งที่ออกแบบมาเพื่อใช้แก้ปัญหาการแทรกข้อมูล ที่มีอยู่จำนวนมหาศาลเข้าสู่ระบบจัดการฐานข้อมูลโดยใช้เวลาที่น้อยที่สุด มีแนวคิดมาจากดัชนีบีพลัสทรี โดยมีสมรรถนะการแทรกสูงกว่าบีพลัสทรีถึงเกือบ 100 เท่า แต่ยังคงสมรรถนะในการสอบถามและการลบ ที่ใกล้เคียงกันกับดัชนีบีพลัสทรี [5]

แนวคิดทางการแก้ปัญหาเรื่องการแทรกที่มีรวดเร็วขึ้นนั้นจะต้องมี 2 คุณสมบัติหลักๆ คือ 1) การแทรกแบบกลุ่ม (Bulk Insertion) 2) การแทรกคู่ (key,ptr) ต้องการทราเวิร์ด (Traversal) จากโหนดรากไปยังโหนดใบที่ระบุเพียงแค่ครั้งเดียวเท่านั้น (Single-path) และนอกจากนี้ดัชนีวายทรียังอนุญาตให้มีค่าคีย์ซ้ำกันได้ ด้วยวิธีการแบบนี้จึงทำให้ดัชนีวายทรีมีสมรรถนะการแทรกเพิ่มขึ้นได้ถึงเกือบ 100 เท่า

2.4.1 โครงสร้างดัชนีวายทรี

ดัชนีวายทรีมีลักษณะหลายๆ ประการคล้ายกับดัชนีแบบ Value-list ตั้งแต่โครงสร้างของทรีรวมไปถึงโหนดภายใน และโหนดใบ โครงสร้างของดัชนีวายทรีแสดงดังรูปที่ 2.6 ซึ่งโหนดของดัชนีวายทรีประกอบด้วยโหนด 2 ประเภทคือ โหนดใบ และโหนดภายใน

1) โหนดใบ เป็นโหนดที่อยู่ในระดับล่างสุดของโครงสร้างทรี และเป็นโหนดที่ใช้เก็บตำแหน่งข้อมูลที่อยู่บนดิสก์ โหนดใบมีรูปแบบการเก็บในโหนดคือ *(key, ptr-list)* โดยที่ *key* เป็นค่าของคีย์แอตทรีบิวต์ที่กำลังทำดัชนี และ *ptr-list* คือ รายการพอยเตอร์ของข้อมูลของ *key* ที่ชี้ไปยังไฟล์ข้อมูล โดยต้องเรียงลำดับจากค่าคีย์น้อยไปหามาก

1. Start at the Root Node.
2. Select the child that needs the least enlargement in order to fit the new geometry.
3. Repeat until at a leaf node.
4. **If** leaf node has available space **then,**
   Insert.
5. **Else**
   Split the entry into two nodes.
   Update parent nodes.
   Update the entry that pointed to the node with a new MBR.
   Add a new entry for the second new node.
6. **If** there is no space in the parent node **then,**
7. Split and repeat.

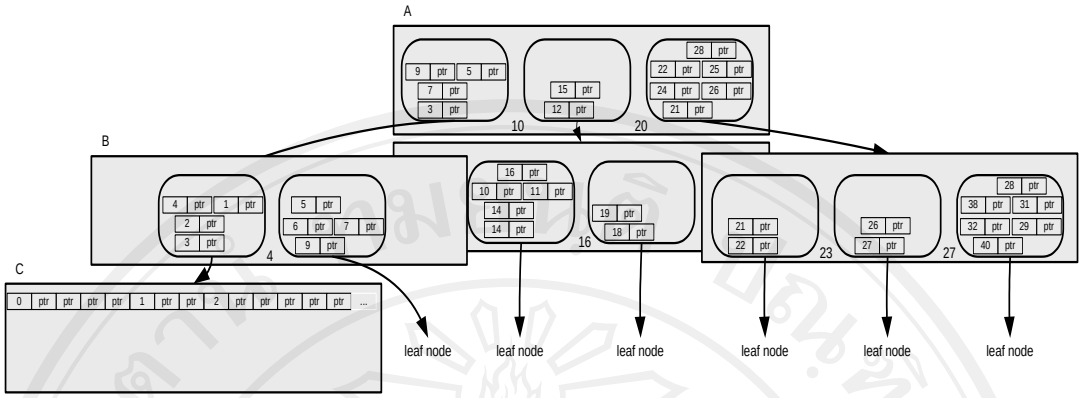
รูปที่ 2.5 ขั้นตอนวิธีการแทรกของดัชนีอาร์ทรี

2) โหนดภายใน เป็นทั้งหมดที่เหลือที่ไม่ใช่โหนดใบ ซึ่งประกอบโดย 2 องค์ประกอบหลัก คือ รายการพอยเตอร์ และฮีพ (heap) โดย *pointer-list* จะอยู่ในรูปแบบ $<P_1, K_1, P_2, K_2,..., P_{f-1}, K_{f-1}, P_f>$ โดย P_i คือบัคเก็ต (Bucket) และ K_j คือ ค่าคีย์ในบัคเก็ต และมีฮีพเป็นจำนวนเซ็ตของบัคเก็ต โดยที่ f เป็นค่าคงที่ที่ได้กำหนดไว้แล้วก่อนสร้างดัชนีวายทรีนั้นหมายถึงว่าจำนวนบัคเก็ตสูงที่สุดในโหนดจะไม่เกินค่า f

ในแต่ละบัคเก็ตจะมีพอยเตอร์ชี้ไปยังโหนดใบในระดับล่างลงมาของทรี รูปแบบที่เก็บอยู่ในบัคเก็ตอยู่ในรูปแบบ *(key,ptr)* และถือลำดับการใส่ข้อมูลลงบัคเก็ตเป็นสำคัญ และ*(key,ptr)* นี้จะต้องพบอยู่ในโหนดใบด้วย จากรูปที่ 2.6 ได้กำหนดค่า *f=3* จึงทำให้ในโหนดภายในแต่ละโหนดมีบัคเก็ตไม่เกิน 3 โหนด

2.4.2 การแทรกข้อมูลในดัชนีวายทรี

ดังที่ได้กล่าวมาแล้วข้างต้นว่า จุดประสงค์ของการออกแบบดัชนีวายทรีนั้นเพื่อต้องการให้มีการแทรกที่รวดเร็วในสภาพแวดล้อมที่มีปริมาณข้อมูลอย่างมหาศาล ซึ่งต้องอาศัยคุณสมบัติที่สำคัญ 2 ประการดังนี้



รูปที่ 2.6 โครงสร้างของดัชนีวายทรี

1) การแทรกคู่ (key,ptr) ลงดัชนีวายทรีผลลัพธ์ของการอ่าน และการเขียนโหนดมีมากที่สุด 1 พาท (Path) จากโหนดรากถึงโหนดใบในดัชนีวายทรี

2) การกำหนดขนาดของฮีพจะต้องทำให้ต้นทุนของการแทรก *d(key,ptr)* ลงดัชนีวายทรีมีคอสต์ (Cost) ที่เท่ากับคอสต์การแทรก *(key,ptr)* ลงดัชนีวายทรี และคุณสมบัติข้อนี้ถือได้ว่ามีความสำคัญเป็นอย่างมากซึ่งบ่งบอกถึงความเร็วของการแทรกลงดัชนีวายทรี และถ้าตัวแปร *d* มีขนาดใหญ่เพียงพอจะทำให้ความเร็วของการแทรกในดัชนีวายทรีมีความเร็วเพิ่มขึ้นได้

เพื่อให้การแทรกข้อมูลในดัชนีวายทรีเป็นไปตามจุดประสงค์ที่ต้องการ ต้องใช้ขั้นตอนวิธีการแทรกข้อมูลดังแสดงในรูปที่ 2.7 และการแทรกข้อมูลนี้สมมติว่าโหนดใบมีพื้นที่เพียงพอในการเก็บข้อมูล ซึ่งสามารถอธิบายได้ดังนี้

กระบวนการแทรกข้อมูลของดัชนีวายทรี จะต้องมีอินพุต (Input) ประกอบด้วย S ซึ่งหมายถึงชุดกลุ่มข้อมูลที่ต้องการแทรกลงในดัชนีวายทรี แล้วต้องกระจายออกเป็น s ที่จำนวนไม่เกินค่าของตัวแปร d ส่วน N คือโหนดปัจจุบันที่ต้องการแทรกข้อมูลและมีบัคเก็ต (Bucket) บรรจุอยู่ด้านใน กระบวนการแทรกข้อมูลจะประมวลผลแต่ละเอ็นทรีดัชนี s ใน S ต่อจากนั้นให้ตรวจสอบว่าโหนดปัจจุบันเป็นโหนดภายใน หรือโหนดใบ ถ้าเป็นโหนดใบให้เพิ่มเอ็นทรีลงโหนดปัจจุบันนั้น แล้วเขียนโหนดปัจจุบันลงสู่ดิสก์ แต่ถ้าโหนดปัจจุบันเป็นโหนดภายในให้เพิ่มเอ็นทรีดัชนีลงสู่บัคเก็ต b_i โดยพิจารณาจากเงื่อนไขว่า $K_i>=s.key$ ส่วนที่เหลือให้เพิ่มลงในบัคเก็ตสุดท้าย

ในขั้นตอนถัดมาให้เลือกบัคเก็ตที่มีจำนวน *(key,ptr)* มากที่สุด แล้วพิจารณาต่อว่าฮีพปัจจุบันมีจำนวนเอ็นทรีดัชนีมากกว่า $(f_N-1)*d$ หรือไม่ ถ้าใช่ ให้ย้ายเอ็นทรีดัชนี จำนวนน้อยที่สุดระหว่างค่า d กับ จำนวนเอ็นทรีดัชนีที่เหลือในบัคเก็ตนั้นๆ ไปสร้างชุดกลุ่มข้อมูลใหม่ S_{new} แล้วเขียนโหนดปัจจุบันลงสู่ดิสก์ กระทำด้วยวิธีการเดียวกันนี้กับบัคเก็ตที่เหลือด้วยการเรียกใช้งานฟังก์ชัน Insert(S_{new}, pointer child node b_i) อีกครั้งแบบรีเคอร์ซีฟ (Recursive) แต่ถ้าไม่ใช่ ให้เขียนโหนดปัจจุบันสู่ดิสก์

จากกระบวนการแทรกข้อมูลนั้นจะเห็นได้ว่าเป็นไปตามแนวคิดดังที่กล่าวไปแล้วข้างต้น ทั้งการแทรกข้อมูลเป็นกลุ่ม และการอ่านการเขียนโหนดที่มีมากที่สุด 1 พาท จึงทำให้ผลลัพธ์ของการแทรกข้อมูลมีความรวดเร็วมากขึ้นนั้นเอง

Algorithm Insert (parameters *S*: set of (*key, ptr*) pairs of cardinality no greater than *d*, *N: Node* having fanout f_N)

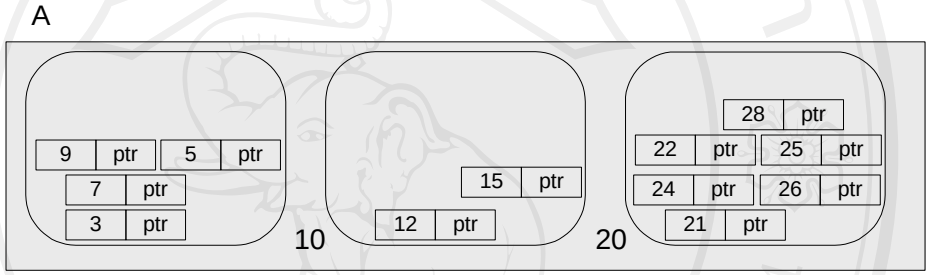
1. **If** *N* is an internal node:
2. For each element *s* of *S*, add *s* into the first heap bucket b_i such that the associated key value K_i *s.key*; or, inset into the last heap bucket if there is no such K_i.
3. Choose the bucket b_j that has the most (*key, ptr*) pairs.
4. **If** the heap contains more than $(f_N - 1) * d$ pairs,
5. Remove *min* (*d, size*(b_j)) (*key, ptr*) pairs from b_j to create S_{new}, write *N* to disk, And recursively call *Insert* (S_{new}, node pointed to by P_j).
6. **Else** write *N* to disk.
7. **Else** *N* is a leaf node:
8. Simply add *S* to the set of (*key, ptr*) pairs in *N*, then write *N* to disk.

รูปที่ 2.7 วิธีการแทรกข้อมูลของดัชนีวายทรี

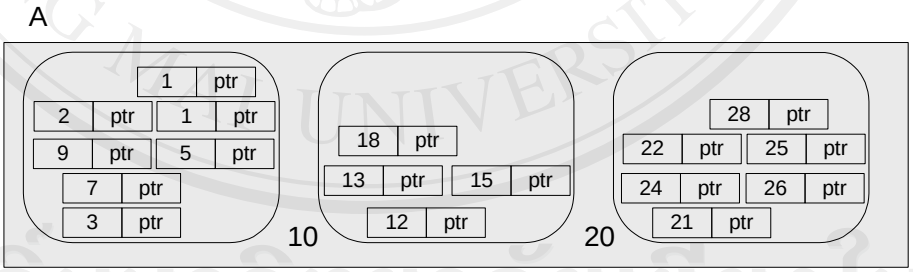
2.4.3 ตัวอย่างการแทรกข้อมูลในดัชนีวายทรี

จากรูปที่ 2.6 ซึ่งเป็นโครงสร้างของดัชนีวายทรีที่มีค่าคงที่ *f=3* และกำหนดให้ค่าคงที่ *d=5* ถ้าต้องการแทรกชุดกลุ่มข้อมูล *S={(1, ptr),(1, ptr),(2, ptr),(13, ptr),(18,ptr)}* ลงดัชนีวายทรีดังรูปที่ 2.6 สามารถอธิบายขั้นตอนตามวิธีการแทรกข้อมูลได้ดังต่อไปนี้

เริ่มต้นจากการพิจารณาค่า K ของโหนดรากดังรูปที่ 2.8 ซึ่งประกอบด้วย $K_1=10$ และ $K_2=20$ นอกจากนั้นยังมีบัคเก็ต b ประกอบด้วย $b1=\{(9,ptr),(5,ptr),(7,ptr),(3,ptr)\}$ $b2=\{(15,ptr),(12,ptr)\}$ และ $b3=\{(28,ptr),(38,ptr),(31,ptr),(32,ptr),(29,ptr),(40,ptr)\}$ ต่อจากนั้นกระจายค่าของชุดกลุ่มข้อมูล S ออกเป็น s ในรูปแบบของ (key,ptr) ดังนั้นเมื่อกระจายออกมาแล้วจะประกอบด้วย 5 คู่ดังนี้ *(1, ptr) (1, ptr) (2, ptr) (13, ptr) (18,ptr)* ในขั้นตอนต่อมาให้พิจารณาค่าของคีย์ แต่ละคู่ที่กระจายออกมา ถ้ามีค่าน้อยกว่าหรือเท่ากับค่า K_i ตัวใดให้แทรก (key,ptr) นั้นลงบัคเก็ต b_i นั้นๆ เช่น *(1, ptr) (1, ptr) (2, ptr)* จะถูกแทรกที่ b_1 และ *(13, ptr) (18,ptr)* จะถูกแทรกที่ b_2 ดังแสดงในรูปที่ 2.9



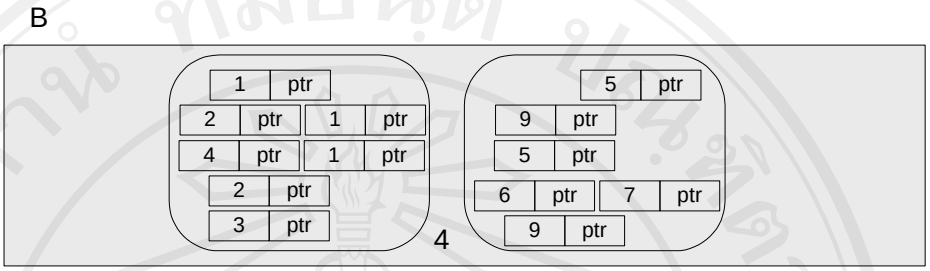
รูปที่ 2.8 โหนดรากของดัชนีวายทรี



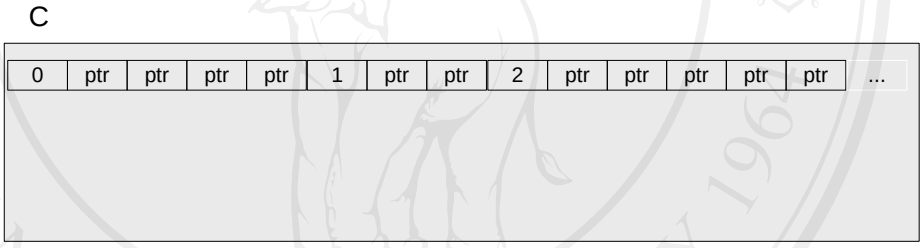
รูปที่ 2.9 โหนดรากของดัชนีวายทรีเมื่อแทรก S แล้ว

ขั้นตอนต่อมาจะเลือกบัคเก็ตที่มีจำนวนคู่ (key,ptr) มากที่สุดนั้นคือ b_1 และถ้าฮีพปัจจุบันมีจำนวนคู่ (key,ptr) มากกว่า $(f_N-1)*d$ ซึ่งในที่นี้มีค่าเป็น 10 จำนวนคู่ (key,ptr) ของฮีพนี้มีค่าเป็น 17 ดังนั้นจึงต้องย้ายคู่ (key,ptr) ของบัคเก็ต b_1 ของคู่บนสุดจำนวน 5 คู่คือ (1,ptr) (1,ptr) (2,ptr) (5,ptr) (9,ptr) ออกเก็บไว้ใน S_{new} แล้วเรียกใช้งานฟังก์ชัน Insert(S_{new},pointer child node b_i) ทำให้ได้ผลลัพธ์เป็นดังรูปที่ 2.10 และเช่นกันในขั้นตอนสุดท้ายจะต้องทำการย้ายคู่ (1,ptr) (1,ptr) (1,ptr) (2,ptr)

(4,ptr) ไปแทรกในโหนดใบดังรูปที่ 2.11 และใช้วิธีเรียกซ้ำฟังก์ชัน Insert() จนครบทุกบัคเก็ตในดัชนีวายทรี



รูปที่ 2.10 คู่ (key,ptr) หลังการแทรกจากโหนด A



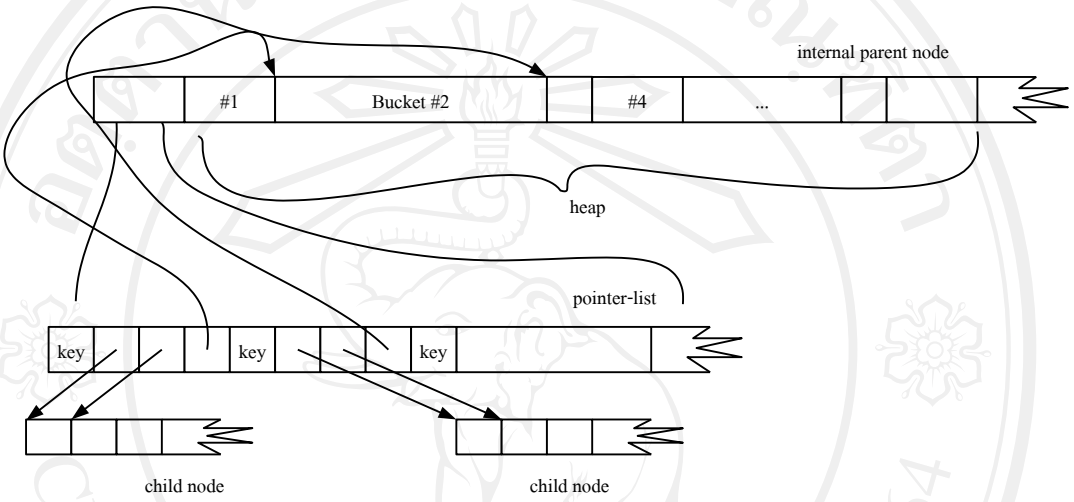
รูปที่ 2.11 ptr-list ของโหนด C หลังการแทรกจากโหนด B

2.4.4 การปรับโครงสร้างดัชนีวายทรีเพื่อรองรับการเก็บข้อมูลขนาดใหญ่

จุดประสงค์ประการหนึ่งของการสร้างดัชนีวายทรีขึ้นมาคือใช้กับระบบฐานข้อมูลขนาดใหญ่ซึ่งแน่นอนว่าดัชนีวายทรีต้องใช้พื้นที่ในการเก็บข้อมูลอย่างมากเช่นกัน แต่วิธีการที่จะทำให้ดัชนีวายทรีเก็บข้อมูลได้ในปริมาณที่มาก และยังคงคุณสมบัติในเรื่องความเร็วเช่นเดิม ซึ่งมีวิธีการปรับปรุงโครงสร้างของดัชนีวายทรีสามารถอธิบายได้ดังนี้

เริ่มต้นด้วยการเพิ่ม *end-pointer* เข้าไปในทุกๆ pointer list ดังนั้นโหนดภายในจึงมีรูปแบบเป็นดังนี้ $<(P_1,end_1), K_1, (P_2,end_2), K_2,..., (P_{f-1},end_{f-1}), K_{f-1}, (P_f, end_f)>$ แทนที่จะเป็น $<P_1, K_1, P_2, K_2,..., P_{f-1}, K_{f-1}, P_f>$ โดยที่ *end-pointer* หมายถึงจุดที่บ่งบอกถึงจุดสิ้นสุดของข้อมูลที่มาจากโหนดแม่นั้นๆ ซึ่ง end-pointer จะชี้ไปยังตำแหน่งคู่ (key,ptr) สุดท้ายของบัคเก็ตโหนดพาแลนท์ (Parent) ในลำดับต่อมามีความเป็นไปได้จากในขั้นตอนแรกที่เมื่อโหนดมีขนาดใหญ่ขึ้น จะมีผลโดยตรงกับสมรรถนะของการสอบถาม โดยสมมติว่าจะต้องมีคู่ *(key,ptr)* อยู่ในดัชนีวายที การค้นหาจะต้อง

ค้นหาในฮีพของทุกๆ โหนดภายในเพื่อให้พบสิ่งที่ต้องการ แต่ในความเป็นจริง สิ่งที่ต้องการคือโหนดลูกจะต้องมีความเกี่ยวข้องกับฮีพบัคเก็ตนั้นด้วย ซึ่งสามารถทำได้ดังนี้ เริ่มจากรวมฮีพบัคเก็ตเข้าไว้ด้วยกัน และเพิ่มตัวชี้เอาไว้ที่จุดเริ่มต้นของทุกๆบัคเก็ต แสดงดังในรูปที่ 2.12



รูปที่ 2.12 การปรับโครงสร้างของโหนดภายในสำหรับโหนดขนาดใหญ่

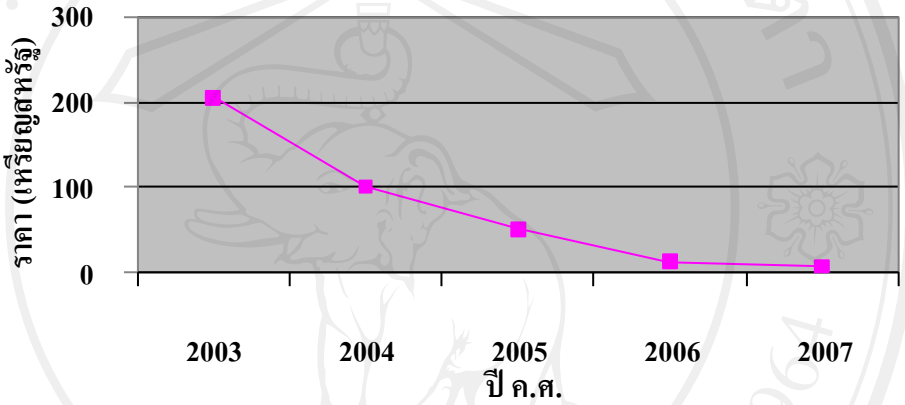
ในกรณีของการสอบถามแบบช่วงที่เมื่อนำโหนดภายในมารวมกับแบบนี้ จะสามารถกำหนดโดยดูจากคีย์ที่ 1 และ 2 ที่อยู่ใน *pointer-list* ของโหนด และต้องสามารถข้ามไปถึงจุดเริ่มต้นของบัคเก็ตของคีย์ที่ 2 ได้ จากนั้นอ่านข้อมูลของบัคเก็ตที่ 2 ไปจนกว่าจะพบจุดเริ่มต้นของบัคเก็ตที่ 3 ซึ่งสามารถพบ (key,ptr) ที่ต้องการได้

2.5 แฟลชไดรฟ์

โดยทั่วไปปริมาณข้อมูลที่มากมายซึ่งถูกจัดเก็บลงในคลังข้อมูลจะถูกจัดเก็บลงจานบันทึก (Disk) ซึ่งเป็นประเภทจานแม่เหล็ก (Magnetic Disk) อย่างไรก็ตามในช่วงเวลาประมาณ 2-3 ปีที่ผ่านมา แฟลชไดรฟ์ ซึ่งเป็นแหล่งเก็บข้อมูลประเภท Electrically-erasable Programmable Read-only Memory (EEPROM) เริ่มได้รับความนิยมในการนำมาเก็บข้อมูลมากขึ้นเรื่อยๆ ด้วยเหตุผลของราคาที่ถูกลง ประสิทธิภาพที่เพิ่มมากขึ้น และความจุที่สูงขึ้นเรื่อยๆ ดังแสดงในรูปที่ 2.13 ซึ่งแสดง

ให้เห็นว่าราคาของแฟลชไดรฟ์ต่อกิกะไบต์ เริ่มลดลงอย่างรวดเร็วในปี ค.ศ. 2004-2006 และก็เริ่มลดลงอย่างช้าๆในปี ค.ศ. 2006-2007 [4] ซึ่งตรงกันข้ามกับความจุที่เพิ่มขึ้นเรื่อยๆ อย่างรวดเร็ว

แฟลชไดรฟ์ได้ถูกนำไปใช้อย่างกว้างขวาง กับอุปกรณ์หลายๆ ชนิดมาก เช่น Thumb Drives แหล่งเก็บข้อมูลของกล้องถ่ายภาพดิจิตัล หรือฮาร์ดดิสก์ของคอมพิวเตอร์พกพา จากแนวโน้มดังกล่าว อาจกล่าวได้ว่าแฟลชไดรฟ์จะกลายมาเป็นแหล่งเก็บข้อมูลหลักของระบบคอมพิวเตอร์ในอีกไม่ช้า [11]



รูปที่ 2.13 ราคาของ แฟลชไดรฟ์ต่อกิกะไบต์ [4]

อย่างไรก็ตามลักษณะพิเศษประการหนึ่งที่ทำให้แฟลชไดรฟ์ได้รับความนิยมเพิ่มขึ้น คือ ความเร็วในเข้าถึงข้อมูลดังแสดงในตารางที่ 2.1 ซึ่งแสดงให้เห็นถึงเวลาในการอ่าน เขียน ลบ ของ Samsung Flash Memory Chip [9] โดยมีอัตราความเร็วการเขียนต่อการอ่านอยู่ที่ 1:2.5 และอัตราความเร็วในการลบต่อการอ่านเป็น 1:18.7

แฟลชไดรฟ์ที่มีใช้งานอยู่ในปัจจุบันมีอยู่ 2 แบบด้วยกันคือ แฟลชไดรฟ์แบบนอร์ นำเสนอโดยบริษัทอินเทล (Intel) ในปีค.ศ. 1988 และแฟลชไดรฟ์แบบแนนด์ซึ่งเสนอโดยบริษัทโตชิบา (Toshiba) ในปีค.ศ. 1989 ใน [7] ได้กล่าวถึงคุณสมบัติของแฟลชไดรฟ์ซึ่งกล่าวโดยสรุปได้ดังตารางที่ 2.2 ใน [7] และตารางที่ 2.2 ได้กล่าวว่ามีผู้เชี่ยวชาญแนะนำให้ใช้แฟลชไดรฟ์แบบแนนด์ด้วยเหตุผลดังนี้ แฟลชไดรฟ์แบบแนนด์มีคุณสมบัติเรื่องความจุสูงกว่า การอ่าน การเขียนข้อมูลที่เร็วกว่า มีอายุการใช้ที่ยาวกว่า และมีราคาต่ำกว่า แฟลชไดรฟ์แบบนอร์ เป็นต้น

แฟลชไดรฟ์แบ่งได้ 2 ประเภท ถ้าใช้แรงดันไฟฟ้าเป็นตัวกำหนดสถานะของเซลล์ (cell) ในแฟลชไดรฟ์สามารถแบ่งได้ดังนี้ คือ Single-Level Cell (SLC) และ Multi-Level Cell (MLC)

ตารางที่ 2.1 เวลาในการเข้าถึงข้อมูลของแฟลชไดรฟ์ และจานแม่เหล็ก [9]

Media	**Access time**		
	Read(2KB)	**Write(2KB)**	**Erase(128KB)**
Magnetic	12.7 ms	13.7 ms	N/A
Flash Memory	80 μs	200 μs	1.5 ms

SLC แฟลชไดรฟ์จะใช้ 1 บิต (Bit) ในการเก็บแรงดันไฟฟ้าทำให้มี 2 สถานะคือ 0 หมายถึงการถูกโปรแกรม (Programmed) และ 1 หมายถึงการถูกลบ (Erased) แฟลชไดรฟ์แบบ SLC มีข้อดีด้านความน่าชื่อถือสูงมาก ดังนั้นจึงถูกประยุกต์ใช้งานในโรงงานอุตสาหกรรม MLC แฟลชไดรฟ์มีอยู่ 4 สถานะคือ 00 หมายถึง fully programmed 01 หมายถึง partially programmed 10 หมายถึง partially erased และ 11 หมายถึง fully erased แฟลชไดรฟ์แบบ MLC มีข้อดีด้านความทนทานจึงถูกประยุกต์มาใช้กับลูกค้าทั่วไป เช่น แฟลชไดรฟ์แบบ ยู เอส บี (USB flash drives), portable media players, or compact flash cards [7]

การกำหนดสถานะของเซลล์ด้วยแรงดันแบบที่กล่าวมาข้างต้น มีผลทำให้เวลาของการเขียนข้อมูลใช้เวลามาก เพราะว่าจะต้องเพิ่มแรงดันไฟฟ้าให้สูงขึ้นถึงระดับที่กำหนดไว้ เพื่อระบุสถานะให้เป็นการเขียนข้อมูล และนอกจากนี้ก่อนการเขียนข้อมูลทุกครั้งจะต้องกระทำการปรับสถานะของเซลล์ให้ว่าง จึงจะสามารถทำการเขียนข้อมูลได้

ตารางที่ 2.2 ความแตกต่างระหว่างแฟลชไดรฟ์แบบนอร์ และแฟลชไดรฟ์แบบแนนด์

รายละเอียด	**แฟลชไดรฟ์แบบนอร์**	**แฟลชไดรฟ์แบบแนนด์**
ความจุ	1MB-32MB	16MB-512MB
ประสิทธิภาพ	การลบช้ามาก การเขียนช้า การอ่านเร็ว	การลบเร็ว การเขียนเร็ว การอ่านเร็ว
รอบการลบ (Erase Cycles)	10,000-100,000	100,000-1,000,000
วิธีเข้าถึง (Access Method)	แบบสุ่ม	แบบลำดับ
ความง่ายต่อการใช้งาน	ง่าย	ซับซ้อน
ราคา	สูง	ต่ำ