บทที่ 2
ทฤษฎีและงานวิจัยที่เกี่ยวข้อง

ในหัวข้อนี้จะกล่าวถึงทฤษฎีพื้นฐานต่างๆ ที่เกี่ยวข้องในการวิจัย ประกอบด้วย การบีบอัดข้อมูลแบบต่างๆ การบีบอัดข้อมูลแบบขนานด้วยวิธีบีซิพทู และการวัดสมรรถนะ (Performance Measurement) ซึ่งเนื้อหาทั้งหมดนี้จำเป็นสำหรับการศึกษาวิจัย เพื่อใช้ในการทดลองและประเมินสมรรถนะของการบีบอัดข้อมูลแบบขนานด้วยวิธีบีซิพทู

2.1 การบีบอัดข้อมูล (Data Compression)

การบีบอัดข้อมูล [2] [7] [10] คือ การลดขนาดข้อมูลเพื่อให้ข้อมูลมีขนาดเล็กลง ทำให้สามารถเก็บข้อมูลได้มากขึ้นในขณะที่พื้นที่เก็บข้อมูลเท่าเดิม ซึ่งเทคโนโลยีนี้ถูกนำไปใช้งานในด้านต่างๆเช่น การรักษาพื้นที่ว่างสำหรับดิสก์ การสำรองข้อมูล รวมถึงการลดเวลาที่ใช้ในการสื่อสารหรือเวลาที่ใช้ในการถ่ายโอนข้อมูล สามารถแบ่งออกเป็น 2 แบบ คือ

1) การบีบอัดข้อมูลแบบที่มีการสูญเสียข้อมูล (Lossy Compression) เมื่อบีบอัดข้อมูลแล้วคลายข้อมูลกลับจะมีความเพี้ยนและสูญหายของข้อมูลไปจากข้อมูลต้นฉบับด้วย ซึ่งการยอมสูญเสียข้อมูลบางส่วนนี้ก็เพื่อแลกกับการบีบอัดข้อมูลให้เล็กลงได้มากขึ้น [10] ข้อมูลประเภทรูปภาพและข้อมูลเสียงจัดเป็นข้อมูลที่เหมาะสำหรับการบีบอัดข้อมูลประเภทนี้

2) การบีบอัดข้อมูลแบบที่ไม่มีการสูญเสียข้อมูล (Lossless Compression) เมื่อบีบอัดข้อมูลแล้วคลายข้อมูลกลับจะต้องได้ข้อมูลเหมือนเดิมครบถ้วนทุกประการ ด้วยเหตุนี้ การบีบอัดข้อมูลแบบนี้จึงมักถูกใช้กับข้อมูลประเภทที่ข้อมูลจะสูญเสียไม่ได้ เช่น ข้อมูลประเภทข้อความหรือแฟ้มข้อมูลโปรแกรม เป็นต้น การบีบอัดข้อมูลรูปภาพก็ใช้แนวทางการบีบอัดแบบนี้โดยอาศัยหลักการลดการซ้ำซ้อนของข้อมูล (Redundancy Reduction) และการใช้จำนวนบิตที่เก็บค่าข้อมูลไม่คงที่ (Variable-Length Coding) เช่น การเข้ารหัสแบบฮัฟแมน (Huffman Coding) [4] และที่ใช้กันแพร่หลายก็คือการบีบอัดข้อมูลในรูปแบบแฟ้มข้อมูลรูปภาพของ PCX, GIF และ BMP ซึ่งให้อัตราการบีบอัดระหว่าง 10% ถึง 90 % สำหรับรูปภาพกราฟิก [10]

2.1.1 การเข้ารหัสแบบฮัฟแมน (Huffman Coding)

การเข้ารหัสแบบฮัฟแมน [4] เป็นการใช้รหัสที่สั้นกว่าแทนสัญลักษณ์ที่เกิดขึ้นบ่อย โดยจะใช้ต้นไม้สองทางในการสร้างรหัสของสัญลักษณ์แต่ละตัวกระจายไปกับข้อมูลของโหนดใบซึ่งเชื่อมอยู่กับต้นไม้สองทาง (Binary Tree) แต่ละโหนดจะมีน้ำหนักกำหนดอยู่ ซึ่งก็คือ ความถี่ หรือความน่าจะเป็นของการปรากฏของสัญลักษณ์นั้น

วิธีการสร้างต้นไม้สามารถทำได้ตามขั้นตอนดังนี้ [10]

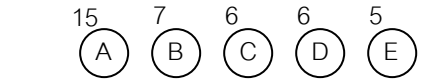
1) กำหนดให้ทุกๆสัญลักษณ์เป็นโหนดใดๆ
2) หา 2 โหนดใดๆ ที่มีน้ำหนักน้อยที่สุด
3) สร้างโหนดแม่สำหรับโหนดสองโหนดนี้ โดยมีน้ำหนักเท่ากับผลรวมของน้ำหนักของโหนดลูก
4) กำหนดให้โหนดแม่นี้เป็นโหนดใดๆ และนำโหนดลูกออกจากโหนดใดๆ
5) โหนดลูกโหนดหนึ่งจะถูกกำหนดให้มีทางผ่านจากโหนดแม่เมื่อทำการถอดรหัสด้วยบิต 0 และอีกโหนดจะถูกกำหนดด้วย บิต 1
6) กลับไปข้อ 1) จนกว่าจะเหลือโหนดใดๆ หนึ่งโหนด ซึ่งโหนดนี้จะถูกกำหนดให้เป็นรากของต้นไม้

ตัวอย่างที่ 2.1 กำหนดให้มีข้อมูลต่างๆดังนี้

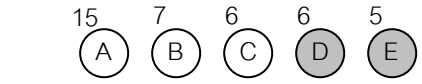
ตารางที่ 2.1 แสดงสัญลักษณ์และจำนวนของการปรากฏของข้อมูลสัญลักษณ์ทั้งหมด 5 สัญลักษณ์ คือ A, B, C, D และ E

Symbol	Count
A	15
B	7
C	6
D	6
E	5

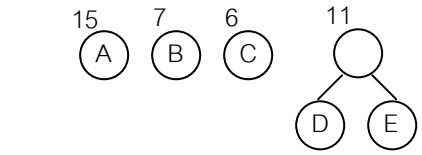
สัญลักษณ์ทั้ง 5 สัญลักษณ์นี้จะเป็นโหนดใบของต้นไม้ เมื่อกระบวนการเริ่มขึ้น โหนดทั้งหมดจะถูกกำหนดให้เป็นโหนดใดๆ ในรายการ (List)



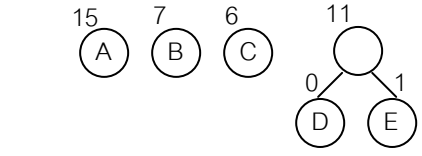
ขั้นแรก หา 2 โหนดที่มีน้ำหนักน้อยสุด คือ D และ E ซึ่งมีน้ำหนัก 6 และ 5



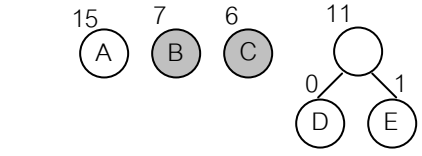
รวมโหนดทั้งสองเข้ากับโหนดแม่ และกำหนดให้โหนดแม่มีน้ำหนักเป็น 11 และนำโหนด D และ E ออกจากโหนดใดๆ ในรายการ



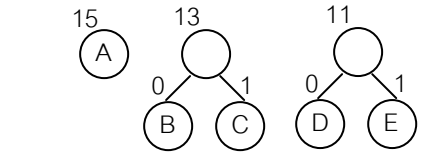
เมื่อขั้นตอนแรกเสร็จสิ้น เราจะรู้ว่าอะไรคือบิตที่น้อยที่สุดที่แสดงค่าของรหัส D และ E โดย D จะถูกแยกออกจากโหนดแม่ด้วย 0 (ซ้าย) และ E จะถูกแยกด้วย 1 (ขวา)



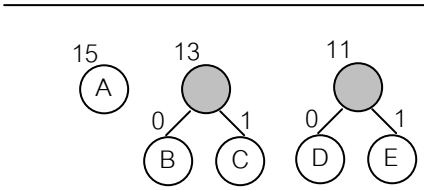
ขั้นต่อมา 2 โหนดที่มีค่าน้ำหนักน้อยที่สุดในโหนดใดๆ คือ โหนด B และ C



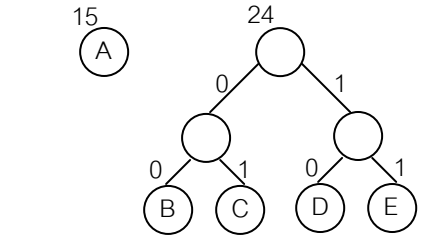
ดังนั้น ต้องเพิ่มโหนดแม่ใหม่และกำหนดให้มีน้ำหนักเป็น 13 และนำโหนด B และ C ออกจากโหนดใดๆ ในรายการ



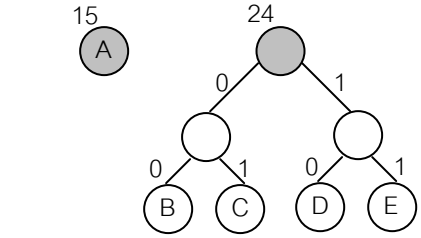
ขั้นต่อมา 2 โหนดที่มีค่าน้ำหนักน้อยที่สุด คือ โหนดแม่ของ B/C และ D/E



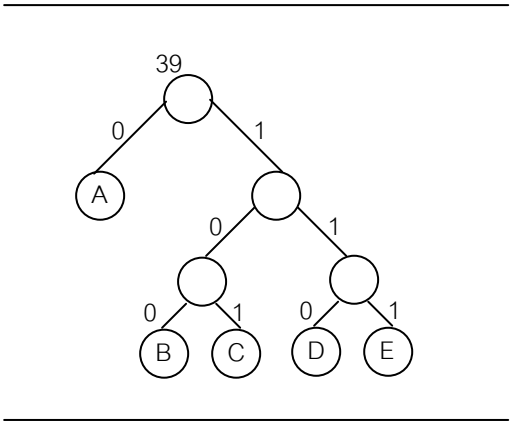
ดังนั้นต้องเพิ่มโหนดแม่ใหม่และกำหนดให้มีน้ำหนักเป็น 24 และนำโหนดลูกออกจากโหนดใดๆ ในรายการ



ในขั้นตอนสุดท้ายจะเหลือเพียง 2 โหนดใดๆ ในรายการ คือ โหนดแม่ที่มีน้ำหนัก 24 และโหนด A



สร้างโหนดแม่ใหม่ให้มีน้ำหนักเป็น 39 และนำโหนดลูกออกจากโหนดใดๆ ในรายการตอนนี้เหลือโหนดแม่เพียงโหนดเดียว นั้นหมายถึงการสร้างต้นไม้เสร็จสิ้น



ตัวอย่างโครงร่างโปรแกรมการเข้ารหัสแบบฮัฟแมน (Huffman Pseudocode)

```
Huffman (C)
    n = the size of C
    insert all the elements of C into Q,
    using the value of the node as the priority
    for i in 1..n-1 do
        z = a new tree node
        x = Extract-Minimum (Q)
        y = Extract-Minimum (Q)
        left node of z = x
        right node of z = y
        f[z] = f[x] + f[y]
        Insert (Q, z)
    end for
return Extract-Minimum (Q) as the complete tree
```

รูปที่ 2.1 ตัวอย่างโครงร่างโปรแกรมการเข้ารหัสแบบฮัฟแมน (Huffman Pseudocode)

ในการกำหนดรหัสให้กับสัญลักษณ์นั้น ทำโดยเดินจากรากของต้นไม้ไปตามโหนดใบของต้นไม้ฮัฟแมน สะสมบิตใหม่ถ้าเดินผ่านโหนดแม่แต่ละโหนด ด้วยวิธีนี้จะได้โครงสร้างของรหัสแสดงได้ดังนี้

ตารางที่ 2.2 แสดงรหัสแทนสัญลักษณ์ของการเข้ารหัสแบบฮัฟแมน

The Huffman Code Table	
A	0
B	100
C	101
D	110
E	111

จะเห็นว่ารหัสจะมีคุณสมบัติคือรหัสนำหน้าจะมีเพียงหนึ่งเดียว (Prefix code) ไม่ซ้ำกัน หมายความว่า ไม่มีรหัสใดเป็นรหัสนำหน้าของรหัสตัวอื่นๆ โดยรหัสฮัฟแมนสามารถถอดรหัสได้อย่างไม่กำกวมเมื่อรหัสนั้นมาในรูปแบบสายอักขระ สัญลักษณ์ที่มีความถี่หรือค่าความน่าจะเป็นสูงที่สุด คือ A จะถูกกำหนดด้วยจำนวนบิตที่น้อยที่สุด และสัญลักษณ์ที่มีความถี่หรือ ค่าความน่าจะเป็นน้อยที่สุด คือ E จะถูกกำหนดด้วยจำนวนบิตที่มากที่สุด

2.1.2 เบอร์โรวส์-วีเลอร์ (Burrows-Wheeler Transform)

วิธีเบอร์โรวส์-วีเลอร์ [1] เป็นวิธีที่ทำให้อัตราการบีบอัดข้อมูลมีประสิทธิภาพเพิ่มขึ้น โดยลักษณะการแปลงข้อมูลวิธีนี้ จะแบ่งข้อมูลต้นกำเนิดออกเป็นกลุ่ม หรือบล็อก (Block) ที่มีขนาดเท่าๆกัน แล้วทำการเปลี่ยนลำดับ ของอักษรภายในบล็อกดังกล่าว เพื่อให้อักษรที่เหมือนกันเรียงต่อกัน เช่น bacba ถูกเปลี่ยนเป็น bbcaa ซึ่งการเข้ารหัสและถอดรหัสด้วยวิธีเบอร์โรวส์-วีเลอร์มีดังต่อไปนี้

1) การเข้ารหัสด้วยวิธีเบอร์โรวส์-วีเลอร์

กำหนดให้ S เป็นบล็อกข้อมูลขนาด N ตัวอักษร โดยที่แต่ละตัวอักษรในบล็อกแทนด้วยสัญลักษณ์ B[0], B[1], B[2], …, B[N-2], B[N-1] ตามลำดับ และกำหนดให้เมตริก M เป็นเมตริกจัตุรัสขนาด N x N ซึ่งมีรูปแบบการเก็บอักษรในเมตริกดังนี้

$$\begin{pmatrix} B[0] & B[1] & B[2] & \cdots & B[N-3] & B[N-2] & B[N-1] \\ B[1] & B[2] & B[3] & \cdots & B[N-2] & B[N-1] & B[0] \\ B[2] & B[3] & B[4] & \cdots & B[N-1] & B[0] & B[1] \\ & \cdots & & & \cdots & & \\ & \cdots & & & \cdots & & \\ B[N-2] & B[N-1] & B[0] & \cdots & B[N-5] & B[N-4] & B[N-3] \\ B[N-1] & B[0] & B[1] & \cdots & B[N-4] & B[N-3] & B[N-2] \end{pmatrix}_{N \times N}$$

แถวแรกของเมตริก M ได้จากการนำเอาอักษรแต่ละตัวในบล็อก S มาวางแต่ละตำแหน่งในแถวตามลำดับ

แถวที่สองของเมตริก M ได้จากการย้ายตำแหน่งตัวอักษรที่อยู่หน้าสุดของแถวแรกไปไว้ในตำแหน่งหลังสุด แล้วเลื่อนอักษรทุกตัวที่เหลือในแถวไปทางซ้าย 1 ตำแหน่ง

แถวที่สามของเมตริก M ได้จากการย้ายอักษรที่อยู่หน้าสุดของแถวทีสองไปไว้ในตำแหน่งหลังสุด แล้วเลื่อนอักษรทุกตัวที่เหลือในแถวไปทางซ้าย 1 ตำแหน่ง ในทำนองเดียวกันแถวที่สี่จนถึงแถวที่ N จะทำลักษณะเดียวกันนี้ไปจนครบ ก็จะได้เมตริก M ขนาด N x N ดังแสดงในตัวอย่างที่ 2.2

ตัวอย่างที่ 2.2 สมมติว่าบล็อก S ในข้อมูลต้นกำเนิดมีอักษรเรียงต่อกันคือ bacba

จากโจทย์ S = bacba, N = 5

เลขแถว	เมตริก M
0	b a c b a
1	a c b a b
2	c b a b a
3	b a b a c
4	a b a c b

$$M = \begin{pmatrix} b & a & c & b & a \\ a & c & b & a & b \\ c & b & a & b & a \\ b & a & b & a & c \\ a & b & a & c & b \end{pmatrix}$$

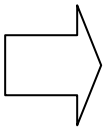
ข้อมูลแต่ละแถวในเมตริก M จะถูกมองเสมือนว่าเป็นกลุ่มข้อมูลแต่ละกลุ่ม เช่น ในตัวอย่าง 2.2 แถวทั้ง 5 แถว ในเมตริก M จะถูกมองเสมือนว่าเป็นกลุ่มข้อมูล 5 กลุ่ม โดยมีเลขแถวกำกับตั้งแต่ 0 ถึง 4 (N = 5) คือ bacba, acbab, cbaba, babac และ abacb ตามลำดับ ซึ่งในขั้นตอนต่อไปจะนำเอากลุ่มข้อมูลเหล่านี้ มาเรียงลำดับ (Sorting) ในลักษณะเดียวกับการเรียงคำศัพท์ในพจนานุกรม จากนั้นจะนำกลุ่มข้อมูลทั้งหมดที่เรียงลำดับแล้ว ไปใส่ไว้ในเมตริกใหม่ที่สร้างขึ้น โดยเริ่มใส่ข้อมูลที่ละแถวจากแถวบนมายังแถวล่างสุด และเรียกเมตริกใหม่ที่เกิดขึ้นดังกล่าวว่า เมตริก M' ดังแสดงในตัวอย่างที่ 2.3

ตัวอย่างที่ 2.3 สร้างเมตริก M' จากเมตริก M จากตัวอย่างที่ 2.2

เมตริก M มีกลุ่มข้อมูล 5 กลุ่ม และมีลำดับดังนี้ bacba, acbab, cbaba, babac และ abacb

เมื่อเรียงลำดับกลุ่มข้อมูลทั้ง 5 กลุ่มใหม่แล้ว ลำดับใหม่ของกลุ่มข้อมูลที่เกิดขึ้นจะเป็นดังนี้ abacb, acbab, babac, bacba และ cbaba

เลขแถว	เมตริก M
0	b a c b a
1	a c b a b
2	c b a b a
3	b a b a c
4	a b a c b

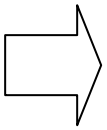


เลขแถว	เมตริก M'
4	a b a c b
1	a c b a b
3	b a b a c
0	b a c b a
2	c b a b a

ให้ L เป็นข้อมูลสดมภ์สุดท้ายของเมตริก M' ผลลัพธ์ของการเข้ารหัสคือ คู่ลำดับ (L, I) โดยที่ I คือ เลขแถวตัวแรกที่พบจากการนำเอาข้อมูลหลักสุดท้ายของแถวแรกในเมตริก M' ไปค้นหาข้อมูลที่มีค่าตรงกับหลักสุดท้ายของเมตริก M โดยจะเรียก I ว่าดัชนี ดังตัวอย่างที่ 2.4

<u>ตัวอย่างที่ 2.4</u> หาผลลัพธ์การเข้ารหัสจากตัวอย่างที่ 2.3

เลขแถว	เมตริก M
0	b a c b a
1	a c b a **b**
2	c b a b a
3	b a b a c
4	a b a c b



เลขแถว	เมตริก M'
4	a b a c **<u>b</u>**
1	a c b a **b**
3	b a b a **c**
0	b a c b **a**
2	c b a b **a**

จากตัวอย่างที่ 2.4 ข้อมูลในหลักสุดท้ายของแถวแรกในเมตริก M' คือ b ('b' ที่ขีดเส้นใต้ในเมตริก M') ซึ่งเมื่อนำไปค้นในสดมภ์สุดท้ายของเมตริก M พบว่า 'b' ตัวแรกที่เจอ ('b' ตัวหนาในเมตริก M) มีค่าเลขแถวเท่ากับ 1 ฉะนั้น I มีค่าเป็น 1 ผลลัพธ์ของข้อมูลที่ผ่านการเข้ารหัสแล้วจึงเป็น (bbcaa, 1)

2) การถอดรหัสด้วยวิธีเบอร์โรวส์-วีเลอร์

จากการเข้ารหัสในหัวข้อที่ผ่านมา ข้อมูลสดมภ์สุดท้ายของเมตริก M' (คู่ลำดับแรกของผลลัพธ์) จะมีจำนวนตัวอักษรเท่ากับข้อมูลสดมภ์สุดท้ายของเมตริก M' เสมอ เช่นในตัวอย่างที่ 2.4 จะเห็นว่าสดมภ์แรกและสดมภ์สุดท้ายของเมตริก M' มีตัวอักษรเท่ากันคือ มี'a' 2 ตัว 'b' 2 ตัว และ 'c' 1 ตัว แต่จะแตกต่างที่ข้อมูลในสดมภ์แรกของเมตริกซ์ M' จะมีการเรียงลำดับตามตัวอักษร ฉะนั้นเมื่อทราบผลลัพธ์ซึ่งเป็นสดมภ์สุดท้ายของเมตริก M' ก็จะสามารถหาค่าของสดมภ์แรกของเมตริก M' ได้ทันที

ก่อนการถอดรหัสจะต้องมีข้อมูล 3 ส่วน ได้แก่ สดมภ์สุดท้ายของเมตริก M' (คู่อันดับแรกของผลลัพธ์) ดัชนี I และเวกเตอร์ V[N] ซึ่งเวกเตอร์ V คือ เมตริกขนาด N x 1 และมีค่าตัวเลขอยู่ในช่วง 0 ถึง N-1 เมื่อ N คือ ขนาดของบล็อกข้อมูลของสดมภ์สุดท้ายของเมตริก M' ซึ่งข้อมูล 2 ส่วนแรก คือ คู่อันดับของผลลัพธ์ที่ได้จากการเข้ารหัส (ไม่ต้องคำนวณหา) แต่เวกเตอร์ V จะสามารถหาได้จากการคำนวณโดยมีขั้นตอนดังต่อไปนี้

ในการสร้างเวกเตอร์ V จะนำเอาอักษรทีละตัวในสดมภ์สุดท้ายของเมตริก M' จากบนลงล่างมาค้นหาอักษรที่มีค่าเดียวกันในสดมภ์แรกของเมตริก M' เมื่อพบตัวอักษรตัวแรกที่ตรงกันก็จะกำหนดค่าตัวเลข ณ แถวเดียวกันกับการพบตัวอักษรที่ตรงกันในสดมภ์แรกของเมตริก M' โดยให้ตัวแรกที่พบจะมีค่าเป็น 0 ตัวที่สองจะมีค่าเป็น 1 ทำเช่นนี้ไปเรื่อยๆจนกระทั่งถึง N-1 ดังตัวอย่างที่ 2.5

<u>ตัวอย่างที่ 2.5</u> ทำการสร้างเวกเตอร์ V จากคู่อันดับ (bbcaa, 1)

กำหนดให้

L แทนข้อมูลสดมภ์สุดท้ายของเมตริกซ์ M' (L=bbcaa)

F แทนข้อมูลสดมภ์แรกของเมตริกซ์ M' (F=aabbc)

เมื่อนำมาทำการสร้างเวกเตอร์เพื่อใช้ในการถอดรหัสจะมีลักษณะการทำงานดังต่อไปนี้

L	**F**	**V**
[b]	a	
b	a	
c	[b]	0
a	b	
a	c	

⇨

L	**F**	**V**
b	a	
[b]	a	
c	b	0
a	[b]	1
a	c	

⇨

L	**F**	**V**
b	a	
b	a	
[c]	b	0
a	b	1
a	[c]	2

⇨

L	**F**	**V**
b	[a]	3
b	a	
c	b	0
[a]	b	1
a	c	2

⇨

L	**F**	**V**
b	a	3
b	[a]	4
c	b	0
a	b	1
[a]	c	2

ฉะนั้นเวกเตอร์ V = $\begin{pmatrix} 3 \\ 4 \\ 0 \\ 1 \\ 2 \end{pmatrix}$

เมื่อได้เวกเตอร์ในการถอดรหัสแล้ว จะสามารถหาบล็อกข้อมูลต้นกำเนิด S ได้โดยใช้ข้อมูลจากสดมภ์สุดท้ายของเมตริกซ์ M' เวกเตอร์ V และดัชนี I หาค่าอักษรของบล็อกข้อมูลต้นกำเนิดทีละตัวดังอัลกอริทึมต่อไปนี้ และแสดงในตัวอย่างที่ 2.6

```
For j = 0 to N-1
        row = I
        S[j] = L[row]
        I = V[row]
End For
```

ตัวอย่างที่ 2.6 ทำการหาบล็อกข้อมูลต้นกำเนิด โดยใช้ข้อมูลจากตัวอย่างที่ 2.5

จากคู่ลำดับ (bbcaa, 1)

จะได้ L = bbcaa, I = 1 และ V = (3, 4, 0, 1, 2)

การหาบล็อกข้อมูลต้นกำเนิด S จะมีขั้นตอนการทำงานดังต่อไปนี้

L	V	S
b	3	b
[b]	4	
c	0	
a	1	
a	2	

⇨

L	V	S
b	3	b
b	4	a
c	0	
a	1	
[a]	2	

⇨

L	V	S
b	3	b
b	4	a
[c]	0	c
a	1	
a	2	

⇨

L	V	S
[b]	3	b
b	4	a
c	0	c
a	1	b
a	2	

⇨

L	V	S
b	3	b
b	4	a
c	0	c
[a]	1	b
a	2	a

I	=	1

I	=	4

I	=	2

I	=	0

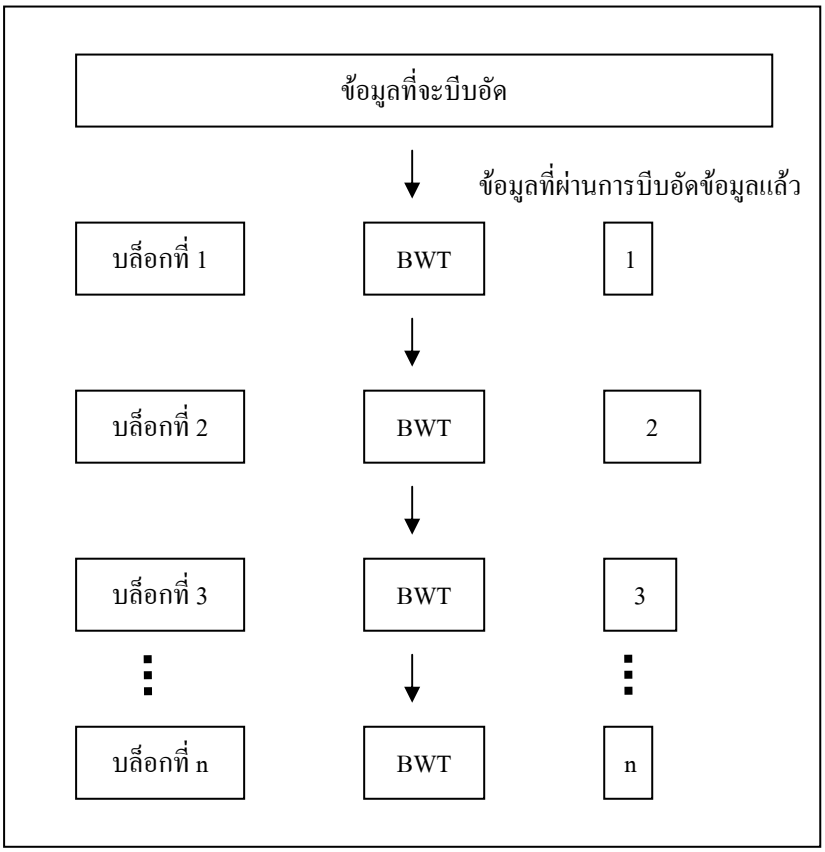
I	=	3

บล็อกข้อมูลต้นกำเนิด คือ bacba

2.1.3 บีซิพทู (Bzip2)

บีซิพทูเป็นโปรแกรมบีบอัดข้อมูลโอเพ่นซอส (Open Source) ที่มีความนิยมและหาใช้ได้ง่ายในระบบยูนิกส์ ใช้วิธีการเบอร์โรวส์-วีเลอร์ และการเข้ารหัสแบบฮัฟแมน โดยปกติการบีบอัดข้อมูลแบบนี้จะดีกว่าการบีบอัดข้อมูลแบบ LZ77/LZ78 และมีประสิทธิภาพใกล้เคียงแบบ PPM ของการบีบอัดข้อมูลโดยใช้สถิติ แต่เร็วกว่าทางด้านเวลา [14] โปรแกรมจะประมวลผลข้อมูลโดยแบ่งข้อมูลออกเป็นช่วงบล็อกขนาด 100,000 ไบต์ ถึง 900,000 ไบต์ ขึ้นอยู่กับคำสั่ง ปกติขนาดของบล็อกจะเท่ากับ 900,000 ไบต์ โปรแกรมจะอ่านข้อมูลครั้งละ 5,000 ไบต์ จนกว่าจะครบเท่ากับ

ขนาดของบล็อกที่กำหนด และประมวลผลบีบอัดข้อมูล เขียนข้อมูลที่บีบอัดแล้ว ลงบนหน่วยเก็บข้อมูล ทำเช่นนี้ต่อไปเรื่อยๆจนกว่าบล็อกข้อมูลถูกบีบอัดข้อมูลแล้วทั้งหมด ดังแสดงในรูปที่ 2.2

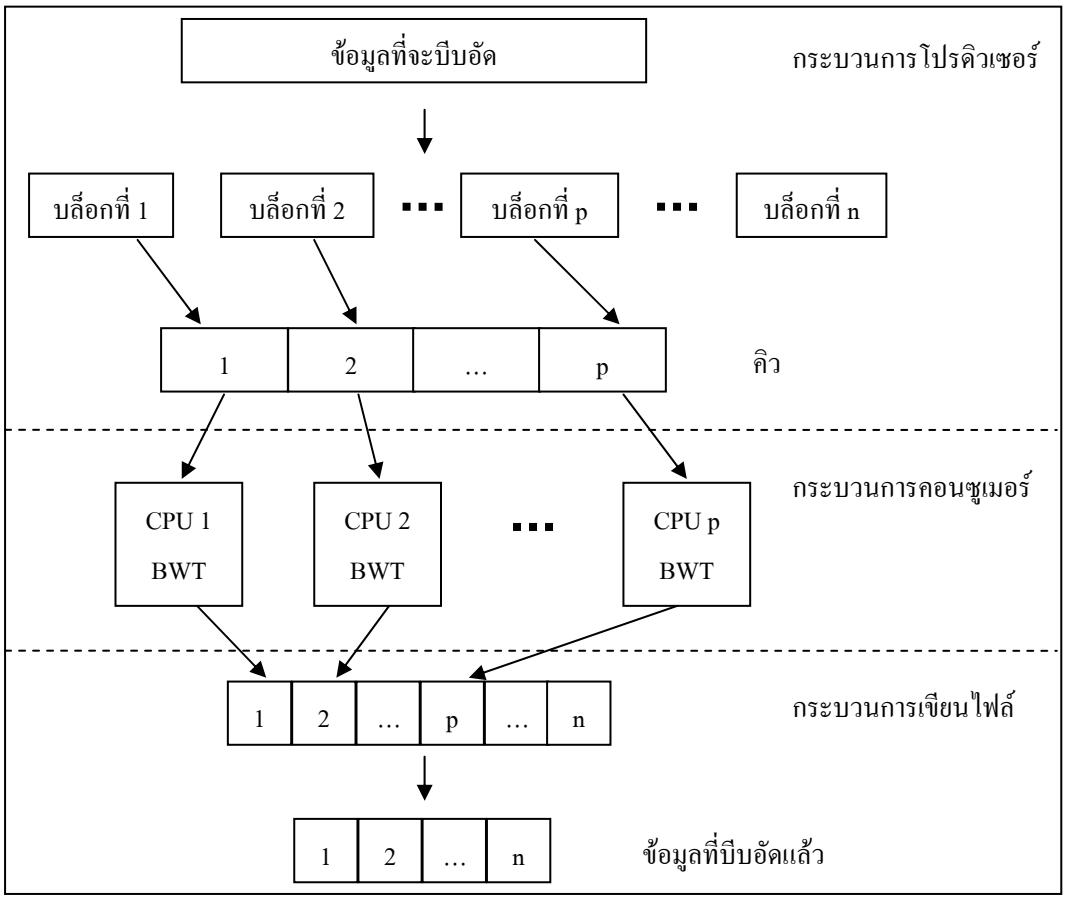


รูปที่ 2.2 แสดงการบีบอัดข้อมูลด้วยวิธีบีซิพทู

จากรูปที่ 2.2 จะพบว่า การบีบอัดข้อมูลด้วยวิธีบีซิพทูนี้เหมาะสมกับการนำไปพัฒนาเป็นการบีบอัดข้อมูลแบบขนาน เนื่องจากข้อมูลที่จะทำการบีบอัดข้อมูลถูกแบ่งออกเป็นบล็อกข้อมูลย่อยหลายๆ บล็อก จากนั้นบล็อกข้อมูลย่อยแต่ละบล็อกจะผ่านกระบวนการบีบอัดข้อมูล ซึ่งกระบวนการบีบอัดข้อมูลนี้ จะถูกใช้เหมือนกันกับทุกๆบล็อกข้อมูลย่อย ดังนั้น บล็อกข้อมูลย่อยแต่ละบล็อก สามารถนำมาประมวลผลบีบอัดข้อมูลพร้อมกันได้ในระบบประมวลผลแบบขนาน ดังจะกล่าวในหัวข้อต่อไป

2.2 การบีบอัดข้อมูลแบบขนานด้วยวิธีบีซิพทู

การบีบอัดข้อมูลแบบขนานด้วยวิธีบีซิพทู [3] ถูกพัฒนาขึ้นในปี ค.ศ. 2004 โดย เจฟ กิลเชริส์ท (Jeff Gilchrist) ซึ่งพัฒนาการบีบอัดข้อมูลนี้บนเครื่องคอมพิวเตอร์แบบขนานที่ใช้หน่วยความจำร่วม (Shared-Memory Parallel Architectures) วิธีนี้จะแบ่งข้อมูลออกเป็นบล็อกเท่าๆ กัน ซึ่งกำหนดโดยผู้ใช้ แทนที่จะอ่านข้อมูลครั้งละ 5,000 ไบต์ มีการนำระบบคิวและแบบจำลองโปรดิวเซอร์-คอนซูเมอร์(Producer-Consumer Model) ใช้ในการออกแบบพัฒนาระบบ โดยการประมวลผลแบบขนานจะประกอบด้วยคอนซูเมอร์หลายๆ คอนซูเมอร์ขึ้นอยู่กับหน่วยประมวลผล โดยโปรดิวเซอร์จะทำหน้าที่แบ่งข้อมูลออกเป็นบล็อกแล้วนำเข้าคิว และคอนซูเมอร์ จะทำหน้าที่อ่านข้อมูลจากคิว และประมวลผลบีบอัดข้อมูลด้วยวิธีบีซิพทู จากนั้นข้อมูลที่ถูกบีบอัดแล้วจะถูกเขียนลงไฟล์ผลลัพธ์ให้ถูกต้องตามลำดับโดยกระบวนการเขียนไฟล์ ดังแสดงในรูป 2.3 เมื่อแต่ละคอนซูเมอร์ทำแต่ละกระบวนการเสร็จแล้วจะไปอ่านข้อมูลจากคิว ทำเช่นนี้ไปเรื่อยๆจนกว่าคิวจะว่าง นั่นคือ ข้อมูลทั้งหมดถูกบีบอัดข้อมูลแล้ว



รูปที่ 2.3 แสดงการบีบอัดข้อมูลแบบขนานด้วยวิธีบีซิพทู

2.3 การวัดสมรรถนะ (Performance Measurement)

วิธีการวัดสมรรถนะขั้นตอนวิธีการทางคอมพิวเตอร์แบบขนาน (Parallel algorithm) จะมีความซับซ้อนกว่าขั้นตอนวิธีการทางคอมพิวเตอร์แบบอนุกรม (Sequential algorithm) โดยสามารถประเมินผลจากเวลาที่ใช้ในการประมวลผล (Response Time) อัตราการเพิ่มขึ้นของความเร็ว (Speedup) และประสิทธิภาพ (Efficiency) เมื่อใช้หน่วยประมวลผลเพิ่มขึ้น ในการประมวลผลจริงบนระบบคลัสเตอร์สามารถวัดเวลาที่ใช้ในการประมวลผลงานหนึ่งได้โดยจับเวลาที่ใช้ในการประมวลผลตั้งแต่เริ่มต้นจนสิ้นสุดการประมวลผล ซึ่งวัดได้ทั้งแบบอนุกรม (Sequential Programming) และแบบขนาน (Parallel Programming) โดยจะทำการวัดเวลาที่ใช้ในการประมวลผล ซึ่งไม่รวมเวลาที่ใช้ในการย้ายข้อมูลมาเก็บไว้ในหน่วยความจำสำรอง (Hard disk) และการตั้งค่าระบบ (Initialization) ของ MPI ซึ่งกระบวนการทั้งสองจะถูกทำครั้งเดียว

2.3.1 เวลาที่ใช้ในการประมวลผล (Response Time)

การวัดเวลาที่ใช้ในการประมวลผลสำหรับการบีบอัดข้อมูลแบบขนาน ในทางทฤษฎี (Theoretical Approach) เวลาที่ใช้ในการประมวลผลแบบขนาน (Parallel Computation) จะสามารถคำนวณได้ดังสมการที่ 2.1 [8] [13]

$$T_p = \frac{T_s}{P} \tag{2.1}$$

เมื่อ T_p คือ เวลาที่ใช้ในการประมวลผลแบบขนานด้วย P หน่วยประมวลผล

T_s คือ เวลาที่ใช้ในการประมวลผลแบบขนานด้วยหนึ่งหน่วยประมวลผล หรือเวลาที่ใช้ในการประมวลผลข้อมูลแบบอนุกรมนั้นเอง

P คือ จำนวนหน่วยประมวลผล (Processor) ที่ใช้ในระบบคลัสเตอร์

ซึ่งกรณีนี้จะเรียกว่าเป็นกรณีอุดมคติ (Ideal Case) เพราะว่ามีการสมมติให้เวลาที่ใช้ในการติดต่อสื่อสาร (Communication Time) ระหว่างหน่วยประมวลผลมีค่าเป็นศูนย์ แต่ในทางปฏิบัติ (Practical Approach) บนระบบคลัสเตอร์ การวัดเวลาที่ใช้ในการประมวลผลแบบขนานจะวัดโดยการจับเวลา ตั้งแต่เริ่มประมวลผล จนกระทั้งสิ้นสุดการประมวลผล หรือได้คำตอบที่ต้องการ ซึ่งเวลาที่วัดได้ดังกล่าวจะรวมเวลาที่ใช้ในการติดต่อสื่อสารด้วย ดังนั้นถึงแม้ว่าจำนวนของหน่วยประมวลผล (P) ในระบบจะมีเพิ่มขึ้นมากๆ แต่เวลาที่ได้ของระบบจะไม่ลดลงจนเข้าใกล้ศูนย์ เพราะสาเหตุมาจากเวลาที่ใช้ในการประมวลผลทั้งหมดประกอบด้วยเวลาที่ใช้ในการประมวลผล

(Computation Time) และเวลาที่ใช้ในการติดต่อสื่อสารระหว่างหน่วยประมวลผล (Communication Time) ซึ่งปกติจะไม่เป็นศูนย์ ดังกรณีอุดมคติ โดยจะมีลักษณะตามสมการที่ 2.2

$$T_p = t_{computation} + t_{communication} \quad (2.2)$$

เมื่อ T_p คือ เวลาที่ใช้ในการประมวลผลแบบขนานทั้งหมดด้วย P หน่วย

$t_{computation}$ คือ เวลาที่ใช้ในการประมวลผลซึ่งไม่รวมเวลาที่ใช้ในการติดต่อสื่อสารระหว่างหน่วยประมวลผล

$t_{communication}$ คือ เวลาที่ใช้ในการสื่อสารระหว่างหน่วยประมวลผล (ถ้ามี)

ดังนั้นในทางปฏิบัติเราสามารถเพิ่มหน่วยประมวลผลเข้าไปในระบบมากขึ้น และเวลาจะลดลงช่วงหนึ่งเท่านั้น เพราะเวลาที่ใช้ในการประมวลผล (Computation Time) มากกว่าเวลาที่ใช้ในการติดต่อสื่อสาร (Communication Time) ระหว่างหน่วยประมวลผล แต่ ณ จุดหนึ่งเมื่อเพิ่มหน่วยประมวลผลเข้าไปอีก เวลาจะไม่ลดลงแต่อาจจะเพิ่มขึ้นเนื่องจากเวลาที่ใช้ในการสื่อสารระหว่างหน่วยประมวลผล $t_{communication}$ มากขึ้นและมีค่ามากกว่า เวลาที่ใช้ในการประมวลผล

2.3.2 อัตราการเพิ่มของความเร็ว (Speedup)

นอกจากการวัดเวลาที่ใช้ในการประมวลผลแล้วยังสามารถวัดอัตราการเพิ่มของความเร็ว (Speedup) ของการประมวลผลแบบขนานได้ ดังสมการที่ 2.3 [8] [13]

$$S_p = \frac{T_s}{T_p} \leq P \quad (2.3)$$

เมื่อ S_p คือ อัตราการเพิ่มของความเร็วที่ใช้ในประมวลผลแบบขนาน โดยใช้ P หน่วยประมวลผล ซึ่งมีค่าสูงสุดในอุดมคติเท่ากับ P

T_s คือ เวลาที่ใช้ในการประมวลผลแบบขนานด้วยหนึ่งหน่วยประมวลผล

T_p คือ เวลาที่ใช้ในการประมวลผลแบบขนานด้วย P หน่วยประมวลผล

2.3.3 ประสิทธิภาพ (Efficiency)

นอกจากการวัดเวลาที่ใช้ในการประมวลผล และอัตราการเพิ่มขึ้นของความเร็วแล้ว ยังสามารถวัดประสิทธิภาพของการประมวลผลแบบขนานได้ ดังสมการที่ 2.4 [8] [13]

$$E_p = \frac{S_p}{P} \leq 1 \qquad (2.4)$$

เมื่อ E_p คือ ประสิทธิภาพของการประมวลผลแบบขนานมีค่าในอุดมคติเท่ากับ 1

S_p คือ อัตราการเพิ่มของความเร็วที่คำนวณได้จากสมการที่ 2.3

P คือ จำนวนหน่วยประมวลผลที่ใช้บนระบบคลัสเตอร์