บทที่ 4

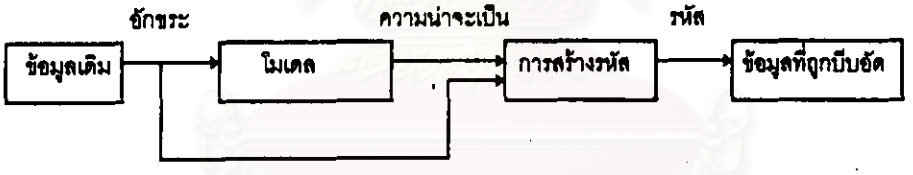
การลดขนาดข้อมูล

การบีบอัดข้อมูล โดยทั่วไปประกอบด้วยการอ่านสายข้อมูลเข้ามาแล้วทำการแปลงข้อมูลเหล่านี้ให้อยู่ในรูปแบบรหัสต่างๆ ถ้าหากการบีบอัดข้อมูลมีประสิทธิภาพ รหัสที่ได้จะต้องมีขนาดสั้นกว่าข้อมูลเดิม

เทคนิคการบีบอัดเพื่อลดขนาดข้อมูลแบ่งได้เป็น 2 ประเภท คือ

1. ใช้เทคนิคและหลักการประมวลผลภาษาธรรมชาติ เช่น ศึกษาสถิติการใช้ตัวอักษร การใช้คำและพยางค์ การใช้คู่ตัวอักษร เป็นต้น สามารถแบ่งออกเป็น 2 ชนิด คือ

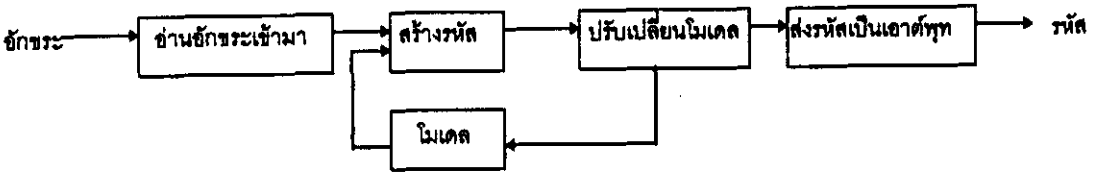
   1.1 แบบสถิต (Static) วิธีนี้จะต้องอ่านข้อมูล 2 ครั้ง ครั้งแรกเพื่อหาค่าความถี่ของตัวอักษรแต่ละตัวในข้อมูล และครั้งที่สองเพื่อแทนค่าตัวอักษรต่างๆด้วยรหัส วิธีสร้างตารางจัดเก็บค่าความถี่จะคงที่และในการเข้ารหัสจะต้องเก็บตารางค่าความถี่เอาไว้ใช้ในการถอดรหัส เช่น วิธีฮัฟแมน



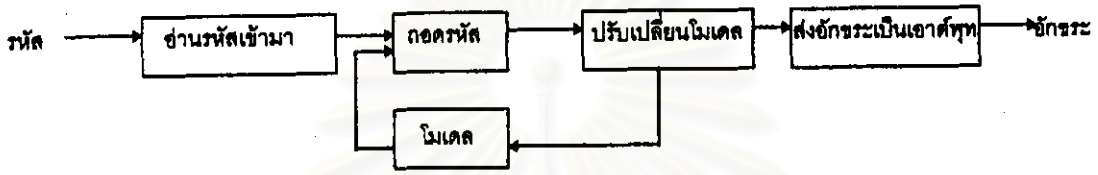
รูปที่ 4-1 การเข้ารหัสโดยวิธีสถิต

วิธีนี้มีข้อเสียคือ ถ้าอักษรที่ปรากฏในข้อมูลไม่ซ้ำกันเลยจะทำให้ข้อมูลที่ผ่านการเข้ารหัสมีขนาดใหญ่กว่าข้อมูลก่อนการเข้ารหัส และตารางรหัสที่ใช้ก็เฉพาะเจาะจงสำหรับแต่ละแฟ้มข้อมูลเท่านั้น ข้อที่ควรคำนึงอีกประการ คือ การจัดเก็บตารางรหัสในลักษณะ order - 0 ทำให้สามารถเก็บรหัสได้เพียง 256 ไบต์ แต่ถ้าเปลี่ยนไปใช้ order - 1 จะสามารถเก็บรหัสได้ถึง 65,536 ไบต์แต่ก็ทำให้เกิดโอเวอร์เฮดมาก

   1.2 แบบอะแดปทีฟ (Adaptive) วิธีนี้จะอ่านข้อมูลเข้ามาเพียงครั้งเดียวและตารางจัดเก็บค่าความถี่จะเปลี่ยนไปตามข้อมูลนำเข้าซึ่งจะใช้วิธีเดียวกันทั้งการเข้ารหัสและถอดรหัส จึงไม่จำเป็นต้องเก็บตารางรหัสไว้พร้อมกับข้อมูลเข้ารหัส เช่น วิธีอะแดปทีฟฮัฟแมน



รูปที่ 4-2 การบีบอัดข้อมูลโดยวิธีอะแดปทีฟ



รูปที่ 4-3 การขยายข้อมูลโดยวิธีอะแดปทีฟ

2. ใช้หลักของพจนานุกรม

วิธีใช้หลักของพจนานุกรมนี้ต่างกับวิธีใช้ค่าทางสถิติคือ ในการเข้ารหัสของวิธีทางสถิติจะเข้ารหัสทีละตัวอักษรแต่วิธีพจนานุกรมนี้จะอ่านข้อมูลจากแฟ้มข้อมูลเข้ามาและตรวจสอบกับข้อมูลที่อ่านเข้ามาก่อนหน้าถ้าหากพบอักษรหรือวลีที่ตรงกันจะให้ตัวชี้ที่ชี้ไปยังจุดที่ซ้ำกันนั้น ทำให้การบีบอัดข้อมูลทำได้ดีขึ้น วิธีใช้หลักของพจนานุกรม สามารถแบ่งออกได้เป็น 3 ชนิด คือ

2.1 แบบสถิต (Static) วิธีการนี้ใช้รูปแบบการบีบอัดข้อมูลและขยายข้อมูลคงที่ ไม่แปรตามข้อมูลที่เข้ารหัส ตารางเทียบรหัสหรือพจนานุกรมของคำหรือกลุ่มตัวอักษรคงที่ทั้งตอนเข้ารหัส และถอดรหัส วิธีนี้มีข้อด้อย คือ ถ้าข้อมูลมีความแตกต่างกันมากตารางเทียบรหัสจะโตมากตามไปด้วย

2.2 เซมิอะแดปทีฟ (Semiadaptive) วิธีนี้การขยายข้อมูลจะใช้ตารางเทียบรหัสซึ่งมีขนาดเล็กกว่า ข้อด้อยของวิธีนี้คือ ใช้เวลาในการอ่านและตรวจข้อมูลว่าเป็นข้อมูลประเภทใดก่อน แต่ยังต้องมีการเก็บตารางเทียบรหัสไว้ภายในด้วย

2.3 อะแดปทีฟ (Adaptive) วิธีนี้ทั้งรูปแบบการเข้ารหัสและถอดรหัสจะใช้ตารางเทียบรหัสซึ่งได้จากข้อความที่ผ่านมาแล้ว เช่น วิธี แอลแซดดับบลิว

การลดขนาดข้อมูลวิธีต่างๆ มี ดังนี้

1. วิธีการของชานนอน-ฟาโน

วิธีการเข้ารหัสข้อมูลชานอน-ฟาโนเป็นวิธีการเข้ารหัสข้อมูลวิธีแรกที่เป็นที่นิยมกันอย่างแพร่หลาย โดย คลอด ชานอน (Claude Shannon) จากห้องวิจัยเบลล์และ อาร์ เอ็ม ฟาโน (R.M. Fano) จากมหาวิทยาลัยเอ็มไอที ได้พัฒนาขึ้นพร้อมๆกัน โดยนำความถี่ในการปรากฏซ้ำของอักษรในข้อความ นำมาสร้างตารางรหัส ซึ่งมีคุณสมบัติ ดังนี้ (Shannon, 1948 and Fano, 1985)

1. แต่ละรหัสจะมีจำนวนบิตต่างกัน
2. แทนที่อักษรที่มีความถี่ในการใช้งานมาก ด้วยรหัสที่มีจำนวนบิตน้อย และแทนที่อักษรที่มีความถี่ในการใช้งานน้อยด้วยจำนวนบิตมาก
3. แต่ละรหัสจะไม่ซ้ำกัน เนื่องจากมีจำนวนบิตไม่เท่ากัน

ในการจัดเก็บรหัสจะเก็บในลักษณะไบนารี ทรี (Binary Tree) ซึ่งทำให้ง่ายในการถอดรหัส

การเข้ารหัสโดยวิธีชานอน-ฟาโน

1. วิธีนี้จะอ่านข้อมูลสองรอบ รอบแรกเพื่อหาค่าความถี่ของอักษรแต่ละตัวในข้อความ
2. เรียงลำดับตามค่าความถี่จากมากไปน้อย
3. แบ่งข้อมูลรหัสเป็น 2 กลุ่ม โดยผลรวมค่าความถี่ของทั้ง 2 กลุ่มมีค่าใกล้เคียงกันมากที่สุด
4. กำหนดรหัสในกลุ่มแรกมีค่าบิตเป็น 0 และรหัสในกลุ่มที่สองมีค่าบิตเป็น 1
5. ทำซ้ำในข้อ 3-4 ในข้อมูลทั้งสองกลุ่ม จนกว่าแต่ละกลุ่มจะเหลือสมาชิกเพียง 1 ตัว ก็จะได้รหัสใหม่ครบทุกตัวอักษร

ตัวอย่างการเข้ารหัสวิธีชานอน-ฟาโน(Nelson and Gailly , 1996)

สมมุติว่าอ่านข้อความรอบแรกพบอักษร 5 ตัว และนำอักษรแต่ละตัวมาจัดเรียงตามความถี่ได้ดังตารางที่ 4-1

อักษร	ความถี่
A	15
B	7
C	6
D	6
E	5

ตารางที่ 4-1 แสดงการจัดเรียงความถี่ของตัวอักษรจากมากไปน้อย

ต่อไปแบ่งรหัสออกเป็น 2 กลุ่ม จะได้ กลุ่มแรกประกอบด้วยอักษร A และ B ซึ่งมีผลรวมค่าความถี่เป็น 22 และกลุ่มที่สอง ประกอบด้วยอักษร C D และ E มีผลรวมค่าความถี่เป็น 17 ต่อไปกำหนดค่าบิตให้อักษรในกลุ่มแรกเป็น 0 และค่าบิตในอักษรในกลุ่มที่สองเป็น 1 ดังตารางที่ 4-2

อักษร	ความถี่		
A	15	0	
B	7	0	
			การแบ่งครั้งแรก
C	6	1	
D	6	1	
E	5	1	

ตารางที่ 4-2 แสดงการแบ่งกลุ่มรหัส (ครั้งที่ 1)

ต่อไปแบ่งกลุ่มแรกออกเป็น 2 ส่วน ตอนนี้จะได้ กลุ่มแรกประกอบด้วยอักษร A และเป็นโหนดใบที่มีรหัสเป็น 00 ส่วนกลุ่มที่สอง ประกอบด้วยอักษร B เป็นโหนดใบที่มีรหัสเป็น 01 และแบ่งกลุ่มที่สองต่อไปเรื่อยๆจะได้ผลดัง ตารางที่ 4-3

อักษร	ความถี่			
A	15	0	0	
				การแบ่งครั้งที่สอง
B	7	0	1	
				การแบ่งครั้งแรก
C	6	1	0	
				การแบ่งครั้งที่สาม
D	6	1	1	0
				การแบ่งครั้งที่สี่
E	5	1	1	1

ตารางที่ 4-3 แสดงการแบ่งกลุ่มรหัสทุกขั้นตอน

วิธีการนี้จะมีความยาวของรหัสใหม่เท่ากับ -log p (a_i) ซึ่งจะเป็นจริงเมื่อสามารถแบ่งกลุ่มรหัสแล้วได้ค่าความน่าจะเป็นของแต่ละกลุ่มเท่ากันจริงๆ แต่ในทางปฏิบัติแล้วมักไม่สามารถทำได้ รหัสใหม่บางตัวจึงอาจมีความยาวเท่ากับ -log p (a_i) + 1

วิธีการนี้ไม่ต้องจัดเรียงอันดับความน่าจะเป็น ค่าความน่าจะเป็นของรหัสที่อยู่ในช่วงค่าจำนวนจริง [0, 1] จะถูกนำมาใช้ในการเข้ารหัสข้อมูล เนื่องจากการใช้เลขจำนวนจริงเป็นรหัส ดังนั้นวิธีการนี้มักจะมีปัญหาเรื่องความเที่ยง (Precision) จึงมีการนำไปใช้กับการรับส่งข้อมูล และเปลี่ยนช่วงค่าของรหัสให้ใช้เลขจำนวนเต็มแทนเลขจำนวนจริง

2. วิธีเชิงคำนวณ

หลักการพื้นฐานของวิธีการเชิงคำนวณมาจากความคิดของ อีไลเอส(Elias) ซึ่งเป็นการเข้ารหัสข้อมูลโดยใช้เลขจำนวนจริง ที่มีค่าระหว่าง 0 ถึง 1

ตัวอย่างการเข้ารหัสข้อความ "BILL GATES" (Nelson and Gailly , 1996)

ตัวอักษร	ความน่าจะเป็น (Probability)
SPACE	1/10
A	1/10
B	1/10
E	1/10
G	1/10
I	1/10
L	2/10
S	1/10
T	1/10

ตารางที่ 4-4 แสดงความน่าจะเป็นของอักษรในข้อความ "BILL GATES"

อักษร	ความน่าจะเป็น (Probability)	ช่วงของความน่าจะเป็น
SPACE	1/10	0.00>= r < 0.10
A	1/10	0.10>= r < 0.20
B	1/10	0.20>= r < 0.30
E	1/10	0.30>= r < 0.40
G	1/10	0.40>= r < 0.50
I	1/10	0.50>= r < 0.60
L	2/10	0.60>= r < 0.80
S	1/10	0.80>= r < 0.90
T	1/10	0.90>= r < 1.00

ตารางที่ 4-5 แสดงช่วงความน่าจะเป็นของอักษรต่างๆ

จากตารางที่ 4-5 แสดงความน่าจะเป็นของอักษรแต่ละตัวในข้อความ ซึ่งมีค่าอยู่ระหว่าง 0 ถึง 1 ส่วนที่สำคัญของการเข้ารหัสวิธีนี้ คือ อักษรตัวแรกของข้อความ สำหรับในข้อความนี้คือ อักษร "B" จะเห็นว่ามีความน่าจะเป็นอยู่ระหว่าง 0.20 ถึง 0.30 ฉะนั้นจะกำหนดให้อักษรตัวที่ตามมาจะต้องมีความน่าจะเป็นอยู่ในช่วง 0.20 ถึง 0.30 เท่านั้น ตัวอักษรตัวต่อไปคือ "I" พบว่ามีความน่าจะเป็นอยู่ระหว่าง 0.50 ถึง 0.60 แต่ภายในช่วงใหม่ที่กำหนด คือ 0.20 ถึง 0.30 ดังนั้นความน่าจะเป็นของอักษร "I" จะอยู่ในช่วงเปอร์เซ็นไทล์ที่ 50 ถึง 60 ฉะนั้นจะได้ค่าอยู่ระหว่าง 0.25 ถึง 0.26 อัลกอริทึมในการหาค่าความน่าจะเป็นของอักษรต่างๆ มีดังนี้

1. กำหนดให้ ค่าต่ำสุดและสูงสุด เท่ากับ 0.00 และ 1.00
2. อ่านอักษรจากข้อความเข้ามาทีละตัวจนกว่าจะหมดข้อความ
3. หาค่าช่วงความน่าจะเป็นซึ่งเท่ากับ ค่าสูงสุด - ค่าต่ำสุด
4. หาค่าสูงสุดใหม่ ซึ่งเท่ากับ ค่าต่ำสุด + ช่วงความน่าจะเป็น * ค่าสูงสุดของอักษร
5. หาค่าต่ำสุดใหม่ ซึ่งเท่ากับ ค่าต่ำสุด + ช่วงความน่าจะเป็น * ค่าต่ำสุดของอักษร
6. กลับไปทำข้อ 2 จนกว่าจะสิ้นสุดข้อความ

อักษร	ค่าต่ำสุด	ค่าสูงสุด
	0.00	1.00
B	0.2	0.3
I	0.25	0.26
L	0.256	0.258
L	0.2572	0.2576
SPACE	0.25720	0.25724
G	0.257216	0.257220
A	0.2572164	0.2572168
T	0.25721676	0.2572168
E	0.257216772	0.257216776
S	0.2572167752	0.2572167756

ตารางที่ 4-6 แสดงช่วงความน่าจะเป็นของอักษรต่างๆ ภายใต้ช่วง 0.2 – 0.3

จากตารางที่ 4-6 พบว่าค่าต่ำสุดของข้อความเป็น 0.2572167752

การถอดรหัสวิธีการเชิงคำนวณ

ขั้นตอนแรกหาว่าตัวอักษรตัวใดเป็นอักษรตัวแรกของข้อความโดยดูจากช่วงต่ำสุดและสูงสุดใดเท่ากับช่วงต่ำสุดและสูงสุดของอักษร จากตัวอย่างพบว่าตัวอักษร "B" มีคุณสมบัติดังกล่าว คือมีค่าช่วงความน่าจะเป็นเท่ากับ 0.2 ถึง 0.3 ต่อไปนำค่า 0.2572167752 ลบด้วยค่าต่ำสุดของ "B" คือ 0.2 จะได้ 0.0572167752 และหารด้วยช่วงระหว่างค่าต่ำสุดและสูงสุดของ "B" คือ 0.1 ผลลัพธ์จะได้ 0.572167752 ทำเช่นนี้ต่อไปกับรหัสตัวอื่นๆอัลกอริทึม มีดังนี้

1. กำหนดให้ number เท่ากับ รหัสจากข้อความที่เข้ารหัสแล้ว
2. กำหนด symbol เท่ากับ ค่าความน่าจะเป็นของรหัสในข้อ 1
3. หาค่าช่วงความน่าจะเป็น โดย ค่าความน่าจะเป็นสูงสุดของ symbol - ค่าความน่าจะเป็นต่ำสุดของ symbol
4. หาค่า number จาก number - ค่าความน่าจะเป็นต่ำสุดของ symbol
5. หาค่า number จาก number หาร ช่วงความน่าจะเป็นจากข้อ 3
6. กลับไปทำข้อ 2 จนกว่าจะหมดแฟ้มข้อมูล

รหัส	อักษรจากการถอดรหัส	ค่าต่ำสุด	ค่าสูงสุด	ช่วงระหว่างค่าต่ำสุดและค่าสูงสุด
0.2572167752	B	0.2	0.3	0.1
0.572167752	I	0.5	0.6	0.1
0.72167752	L	0.6	0.8	0.2
0.6083876	L	0.6	0.8	0.2
0.04198	SPACE	0.0	0.1	0.1
0.4198	G	0.4	0.5	0.1
0.1938	A	0.2	0.3	0.1
0.98	T	0.9	1.0	0.1
0.38	E	0.3	0.4	0.1
0.8	S	0.8	0.9	0.1
0.0				

ตารางที่ 4-7 แสดงการถอดรหัสวิธีเชิงคำนวณ

ปัญหาของวิธีการเชิงคำนวณ คือ ไม่ทราบจุดสิ้นสุดในการถอดรหัส ฉะนั้นจึงมีวิธีแก้ปัญหาอ 2 วิธี คือ กำหนดรหัสสิ้นสุดแฟ้มข้อมูลหรือระบุความยาวของข้อความไปพร้อมกับการเข้ารหัส

3. วิธีการบีบอัดข้อมูลโดยวิธีฮัฟแมน

วิธีฮัฟแมนโค้ดดิง นี้ถูกคิดค้นโดย ดี.เอ ฮัฟแมน และได้ถูกตีพิมพ์ในบทความ "A Method for the Construction of Minimum Redundancy Codes " ในปี ค.ศ. 1952 โดยมีหลักการ คือ คำหรือตัวอักษรแต่ละหน่วยจะมีค่าความน่าจะเป็นของความถี่ในการใช้ไม่เท่ากัน แทนที่จะแทนรหัสตัวอักษรให้มีจำนวนบิตหรือหลักเท่าๆกัน ดังเช่นรหัสแอสกี้มี 8 บิต อาจแทนด้วยตัวอักษรที่มีความถี่การใช้งานมากด้วยจำนวนบิตน้อย และตัวอักษรที่มีความถี่ของการใช้น้อยด้วยจำนวนบิตมาก การบีบอัดข้อมูลวิธีฮัฟแมน นอกจากจะพิจารณาความถี่ของตัวอักษรที่ใช้ อาจพิจารณาจากความถี่ของคำที่ใช้ คำที่พบบ่อยจะได้จำนวนบิตที่น้อยกว่า และที่สำคัญรหัสทุกตัวที่ได้จะมีแอตทริบิวต์ทำหน้าที่ไม่เหมือนกัน ดังนั้นโค้ดที่ได้จึงสามารถถูกถอดรหัสได้อย่างถูกต้อง

วิธีนี้มีข้อด้อย คือ จะต้องเก็บตารางค่าความถี่ไว้พร้อมกับข้อมูลที่ถูกบีบอัดไว้ เพื่อการขยายข้อมูล จึงทำให้สิ้นเปลืองเนื้อที่ในการจัดเก็บ และหากข้อมูลนำเข้ามีขนาดเล็กมากเมื่อผ่านการบีบอัดข้อมูลแล้วอาจได้ขนาดของข้อมูลใหญ่กว่าเดิมได้

ขั้นตอนการเข้ารหัสข้อมูลแบบฮัฟแมน

1. การสร้างแผนภูมิต้นไม้ โดยวิธีฮัฟแมนมีขั้นตอน ดังนี้ (สุทธิศักดิ์ พงศ์ธนาพาณิช 2537:110, 297-301)
   1.1 นับค่าความน่าจะเป็นของข้อมูลทุกตัวและเรียงลำดับข้อมูลตามค่าความน่าจะเป็นจากน้อยไปหามาก
   1.2 โหนดจะถูกสร้างทีละคู่โดยเริ่มจากคู่ข้อมูลที่มีค่าความน่าจะเป็นมากที่สุด แต่ถ้าหากข้อมูลตัวสุดท้ายไม่สามารถจัดเข้าคู่ได้ คือเหลือเพียงตัวเดียวก็จะถูกแยกเป็นโหนดอิสระที่มีกิ่งเพียงกิ่งเดียว
   1.3 สร้างโหนดแม่ (parent node) ของโหนดคู่ที่ได้จากขั้นตอนแรก และจะมีค่าน้ำหนักเท่ากับผลรวมของค่าความน่าจะเป็นของโหนดลูก (child node) ทั้งสอง
   1.4 โหนดแม่ที่ได้ก็จะถูกเพิ่มเข้ามาเป็นโหนดอิสระ และโหนดลูกก็จะถูกยกเลิก
   1.5 โหนดลูกที่ถูกกำหนดให้เป็นเส้นทางของโหนดแม่ โหนดหนึ่งจะถูกกำหนดให้มีค่าเป็นไบนารี 0 ส่วนโหนดลูกที่เหลือก็จะถูกกำหนดให้มีค่าเป็น 1
   1.6 ทำซ้ำขั้นตอนทั้งหมดที่ผ่านมาจนกระทั่งเหลือโหนดอิสระเพียงโหนดเดียว ซึ่งโหนดที่เหลือนี้ก็จะถูกกำหนดเป็นโหนดราก (root node)

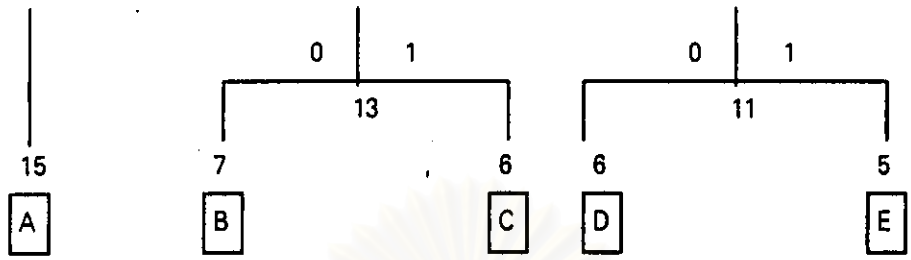
สมมุติว่ากลุ่มข้อมูลประกอบด้วยตัวอักษร A ถึง E เมื่อนำมาเรียงลำดับตามค่าความถี่แล้วจะมีค่าความน่าจะเป็น ดังตารางที่ 4-8

กลุ่มข้อมูล	A	B	C	D	E
ค่าความน่าจะเป็น	15	7	6	6	5

ตารางที่ 4-8 แสดงกลุ่มข้อมูลเมื่อนำมาเรียงลำดับตามค่าความถี่

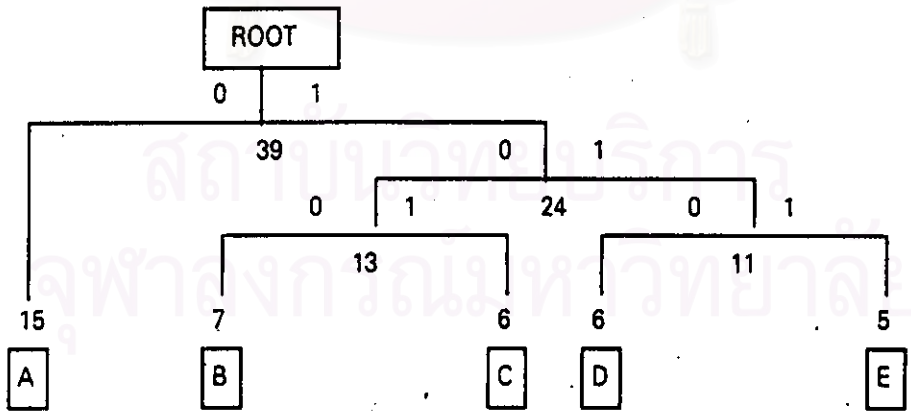
จากกลุ่มข้อมูลทั้ง 5 ตัว จะได้โหนดลูกทั้งหมด 5 โหนด และเนื่องจากจำนวนข้อมูลเป็นเลขคี่ ดังนั้นจึงสามารถจับคู่เพื่อสร้างโหนดคู่ได้ 2 คู่และโหนดเดี่ยวอีก 1 โหนด โดยการจับคู่โหนดจะต้องเริ่มต้นจากคู่ที่มีค่าผลรวมของค่าความน่าจะเป็นน้อยที่สุดไปยังค่ามากที่สุด ดังนั้นโหนดคู่แรกจึงสามารถจับคู่ได้ระหว่าง D และ E (หรือจะเป็น C และ E ก็ได้ เพราะ C และ D มีค่าความถี่เท่ากัน) ซึ่งน้ำหนักเท่ากับผลรวมระหว่าง 6 กับ 5 เป็น 11 โหนดคู่ต่อมาจะเป็นการจับคู่ระหว่าง B และ C ซึ่งมีน้ำหนักเท่ากับผลรวมระหว่าง 7 กับ 6 เป็น 13 ส่วนตัวอักษร A จะถูกปล่อยเป็นโหนดเดียวอิสระ ต่อจากนั้นก็จะทำการสร้างโหนดแม่โดยเกิดจากผลรวมของโหนดลูกคู่ต่างๆ ดังนั้นโหนดแม่ของโหนดคู่ D/E ก็จะมีค่าน้ำหนักเป็น 11 และโหนดแม่ของโหนดคู่ B/C ก็จะมีค่าน้ำหนักเป็น 13 และต่อจากนั้นก็ทำการกำหนดค่าบิตให้

กับแต่ละกิ่งก้าน ดังนั้นกิ่ง B และ D จึงมีค่าบิตเป็น 0 และกิ่ง C และ E ก็จะมีค่าบิตเป็น 1 ซึ่งแผนภูมิต้นไม้ที่ได้เป็นดังรูปที่ 4-4



รูปที่ 4-4 แสดงแผนภูมิต้นไม้ฮัฟแมนขั้นตอนแรก

ขั้นตอนต่อมาก็เป็นการรวมโหนดคู่จากโหนดแม่ที่ได้ โดยสมมติว่าโหนดแม่ที่ได้ก็คือโหนดลูกของลำดับต่อไปนั่นเอง ดังนั้นโหนดแม่ของโหนดคู่ B/C และ D/E ก็จะถูกสร้างขึ้นมาโดยมีค่าน้ำหนักเท่ากับ 13+11 เท่ากับ 24 และกิ่งของโหนดแม่ B/C ก็จะถูกกำหนดให้มีค่าบิตเท่ากับ 0 ส่วนกิ่งของโหนดแม่ D/E ก็จะถูกกำหนดให้มีค่าบิตเท่ากับ 1 ส่วนโหนดเดียวอิสระ A นั้นก็จะถูกนำมาสร้างโหนดคู่ในขั้นตอนสุดท้ายของการสร้างแผนภูมิต้นไม้ เพื่อให้เหลือเพียงโหนดรากเพียงโหนดเดียว ดังนั้นโหนดกิ่ง A ก็จะถูกนำมาจับคู่กับโหนดที่ได้จากการรวมโหนดแม่ของโหนด B/C และ D/E ซึ่งโหนดรากที่ได้ก็จะมีค่าน้ำหนักเป็น 15+24 เท่ากับ 39 และเช่นเดิมกิ่งของโหนด A ก็จะถูกกำหนดให้มีค่าบิตเป็น 0 ส่วนโหนดที่ได้จากการจับคู่โหนดแม่ของ B/C และ D/E ก็จะถูกกำหนดให้มีค่าบิตเป็น 1 ซึ่งแผนภูมิที่ได้เป็นดังรูปที่ 4-5



รูปที่ 4-5 แสดงแผนภูมิต้นไม้ฮัฟแมนที่สมบูรณ์

จากรูปที่ 4-5 สามารถที่จะเข้ารหัสตัวอักษรแต่ละตัว โดยการเดินตามกิ่งสาขาของต้นไม้ซึ่งจะได้รหัสดังในตารางที่ 4-9

ตัวอักษร	A	B	C	D	E
โค้ด	0	100	101	110	111
ค่าความถี่	15	7	6	6	5
Huffman Bit	1	3	3	3	3
รวม	15	21	18	18	15

ตารางที่ 4-9 แสดงอักขระที่ได้จากการถอดรหัส

จะเห็นได้ว่าข้อมูลที่มีค่าความถี่มากกว่าจะมีจำนวนบิตที่น้อยกว่า และเมื่อเปรียบเทียบกับเมื่อใช้ตัวอักษรแอสกี้ ขนาด 8 บิต จะเห็นว่าข้อมูลทั้งหมดมีความยาวเท่ากับ 8*39 หรือ 312 บิต แต่โค้ดที่ได้จากวิธีฮัฟแมนจะทำให้ข้อมูลทั้งหมดมีความยาวเพียง 15+21+18+18+15 หรือ 87 บิตเท่านั้น

จะเห็นว่าโหนดหนึ่งๆของต้นไม้ประกอบด้วยข้อมูล 3 ตัว คือ ค่าน้ำหนัก และตัวชี้ไปยังโหนดลูกทั้งซ้ายและขวาโดยกำหนดให้มีค่าบิตเป็น 0 และ 1 ตามลำดับ

2. การบีบอัดข้อมูลของแต่ละตัวอักษรโดยวิธีฮัฟแมนโค้ดติ้ง วิธีการคือ

2.1 อ่านตัวอักษรแต่ละตัวจากแฟ้มข้อมูลและนับค่าความถี่ของข้อมูลแต่ละตัวเก็บไว้ในตัวแปร

2.2 จัดเก็บค่าความถี่ของข้อมูลแต่ละตัวลงในแฟ้มบีบอัดข้อมูล เพื่อการขยายข้อมูลจะได้สามารถสร้างโมเดลเดิมขึ้นมาอย่างถูกต้อง และเพื่อเป็นการประหยัดเนื้อที่ จึงจัดเก็บค่าที่ไม่เป็น 0 เฉพาะช่วงดังนี้ start, stop, counts, start, stop, counts,.....,0 สุดท้ายจัดเก็บข้อมูล 0 เพื่อแสดงจุดสิ้นสุดของข้อมูลส่วนนี้

2.3 สร้างแผนภูมิต้นไม้ฮัฟแมนโดยเริ่มจากค้นหา 2 โหนดที่มีค่าความถี่น้อยที่สุดแล้วบวกค่าความถี่ทั้งสองเข้าด้วยกันเพื่อกำหนดให้เป็นโหนดใหม่ ซึ่งโหนดใหม่ที่ได้นี้ก็จะมีโหนดลูก 2 โหนด ซึ่งโหนดทั้งสองก็จะจัดเก็บค่าความถี่ของโหนดที่น้อยที่สุดข้างต้น ต่อไปก็จะประมวลผลซ้ำอีกแต่มีการขยับโหนดที่จะทำการประมวลไปเรื่อยๆ

2.4 ถอดรหัสโค้ดฮัฟแมน จากแผนภูมิต้นไม้

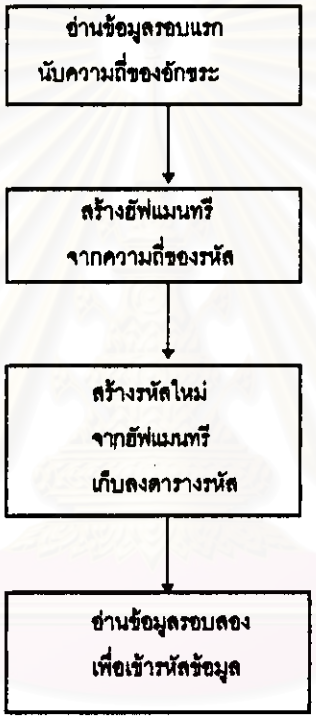
2.5 ทำการบีบอัดข้อมูลโดยการแทนที่ข้อมูลแต่ละตัวด้วยโค้ดฮัฟแมนที่ได้ แต่เนื่องจากโค้ดฮัฟแมนแต่ละตัวมีความยาวไม่เท่ากัน แต่ในระบบการจัดเก็บข้อมูลจะใช้โค้ดแอสกีซึ่งมีจำนวนบิตคงที่ ดังนั้นเราจึงต้องทำการนำโค้ดฮัฟแมน แต่ละตัวมาต่อกันและจัดเก็บลงในแฟ้มข้อมูลเพื่อให้จำนวนบิตเท่ากับ 8 โดยในการจัดเก็บก็ยังคงใช้รหัสแอสกีอยู่เหมือนเดิม

3. การขยายข้อมูล มีวิธีการ ดังนี้

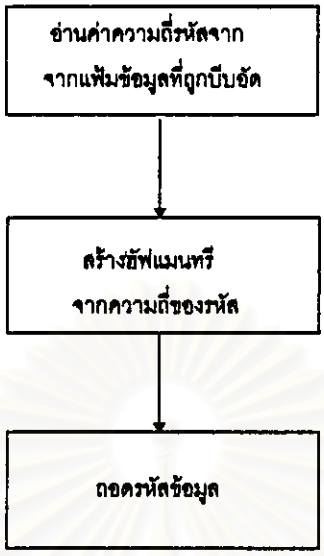
3.1 นำค่าความถี่ของข้อมูลที่ได้จัดเก็บลงแฟ้มบีบอัดข้อมูล จากขั้นตอนการบีบอัดข้อมูลมาสร้างแผนภูมิต้นไม้

3.2 ขยายขนาดข้อมูลทุกตัว โดยเริ่มจากการอ่านข้อมูลจากแฟ้มข้อมูลแล้วประมวลผลครั้งละ 1 บิต เริ่มจากรากของแผนภูมิต้นไม้แล้วเดินตามกิ่งก้าน ซึ่งจะได้โค้ดแต่ละตัวก็ต่อเมื่อเราอ่านบิตแล้วเลื่อนลงมาจนถึงโหนดล่างสุดของต้นไม้

3.3 จัดเก็บรหัสแอสกีลงในแฟ้มข้อมูลขยายขนาด



รูปที่ 4-6 แสดงการบีบอัดข้อมูลโดยวิธีฮัฟแมน



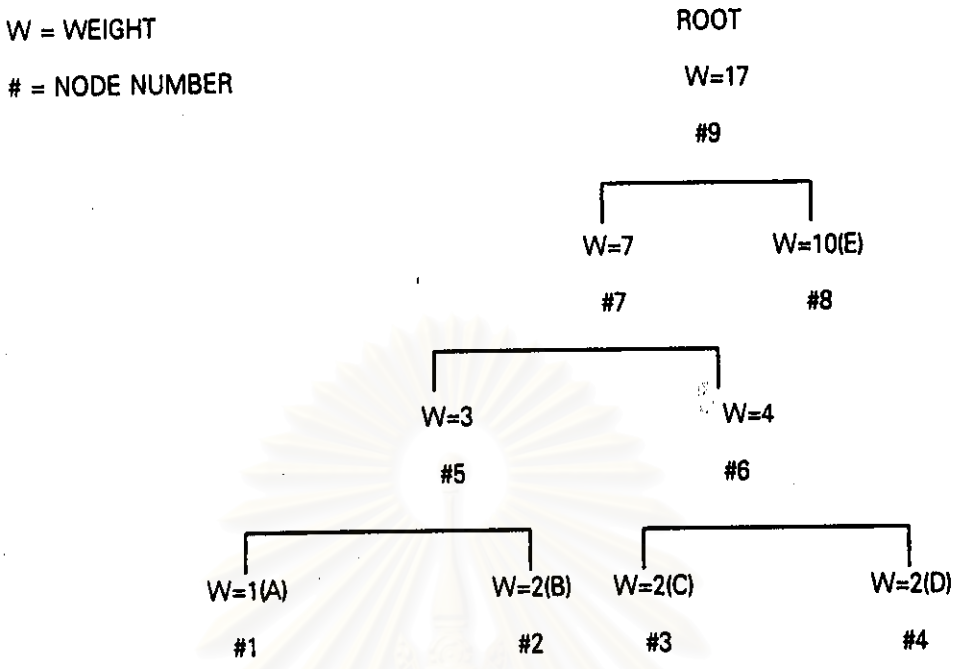
รูปที่ 4-7 แสดงการถอดรหัสข้อมูลโดยวิธีฮัฟแมน

4. **วิธีการอะแดปทีฟฮัฟแมน** (Adaptive Huffman)

การเข้ารหัสโดยวิธีอะแดปทีฟฮัฟแมน คิดขึ้นครั้งแรกในปี ค.ศ. 1973 โดยฟอลเลอร์ ต่อมาในปี ค.ศ. 1978 กัลลาเจอร์ ได้กำหนดหลักการพื้นฐานของวิธีเข้ารหัสโดยวิธีอะแดปทีฟฮัฟแมน คือ การใช้คุณสมบัติโหนดพี่น้อง (sibling) ซึ่งกำหนดไว้ว่าแผนภูมิต้นไม้ฮัฟแมน จะมีคุณสมบัติโหนดพี่น้อง เมื่อแต่ละโหนดมีโหนดแม่ร่วมกันและแต่ละโหนดจะต้องเรียงกันตามค่าความถี่ที่มีทิศทางเพิ่มขึ้น (Gallager , 1978)

(สุทธิศักดิ์ พงศ์ธนาพาณิช, 2538)การเข้ารหัสข้อมูลโดยวิธีฮัฟแมน จะเห็นว่าตัวอักษรในแฟ้มข้อมูลที่ซ้ำๆ กันจะถูกนำมาเข้ารหัสใหม่ ให้มีจำนวนบิตน้อยลง โดยขึ้นอยู่กับค่าความถี่ของตัวอักษรแต่ละตัว และวิธีการบีบอัดข้อมูลดังกล่าวได้ใช้วิธีการสร้างโมเดลแบบลำดับ 0 (order 0 model) ซึ่งตัวอักษรตัวที่ผ่านมาจะไม่ถูกนำมานับค่าความถี่ของตัวถัดไป จึงจำเป็นต้องเก็บตารางความถี่ของตัวอักษรในแฟ้มที่ถูกบีบอัดข้อมูลด้วย เพื่อใช้ในการขยายข้อมูล ดังนั้นถ้าหากข้อมูลที่ถูกบีบอัดมีขนาดเล็กก็จะเป็นการทำให้ข้อมูลมีขนาดใหญ่ขึ้น แต่ถ้าหากเราเปลี่ยนวิธีการสร้างโมเดลจากลำดับ 0 มาเป็นโมเดลลำดับ 1 ก็จะลดพื้นที่จัดเก็บตารางค่าความถี่ลงได้มาก แต่การใช้โมเดลลำดับสูงๆ จะทำให้ความเร็วในการทำงานของโปรแกรมลดลง

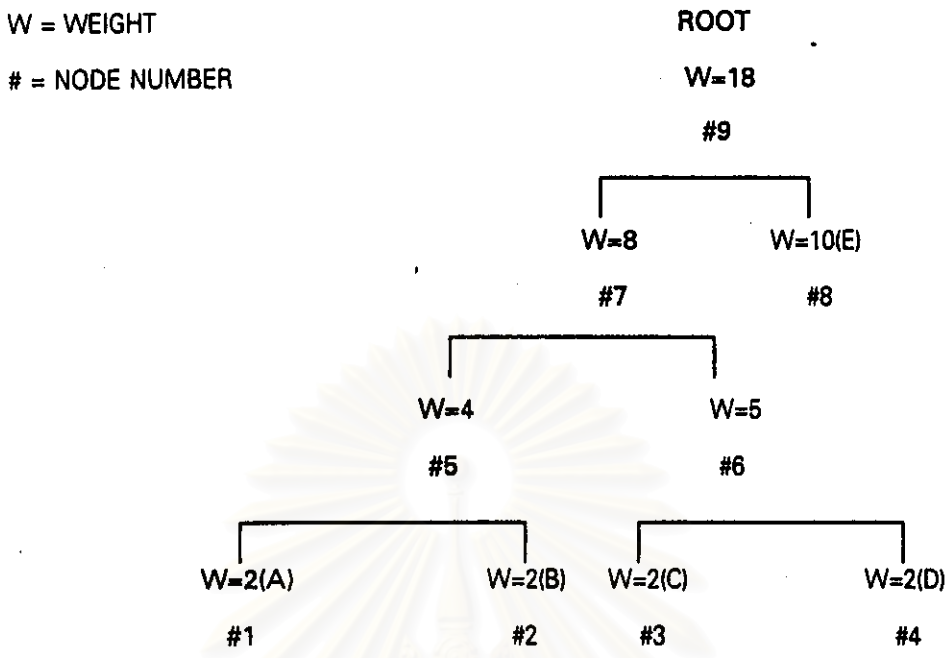
วิธีการเข้ารหัสข้อมูลแบบ อะแดปทีฟ เป็นวิธีที่ช่วยให้สามารถสร้างโมเดลลำดับที่สูงกว่าได้โดยไม่จำเป็นต้องเพิ่มตารางสถิติ โดยการจัดแต่งแผนภูมิต้นไม้ของฮัฟแมนไปพร้อมๆ กับการสร้างโมเดลอ้างอิงจากข้อมูลที่ผ่านมาแต่จะไม่คำนึงถึงข้อมูลตัวถัดไป



รูปที่ 4-8 แสดงแผนภูมิต้นไม้ฮัฟแมน

<u>การปรับปรุงแผนภูมิต้นไม้ของฮัฟแมน</u>

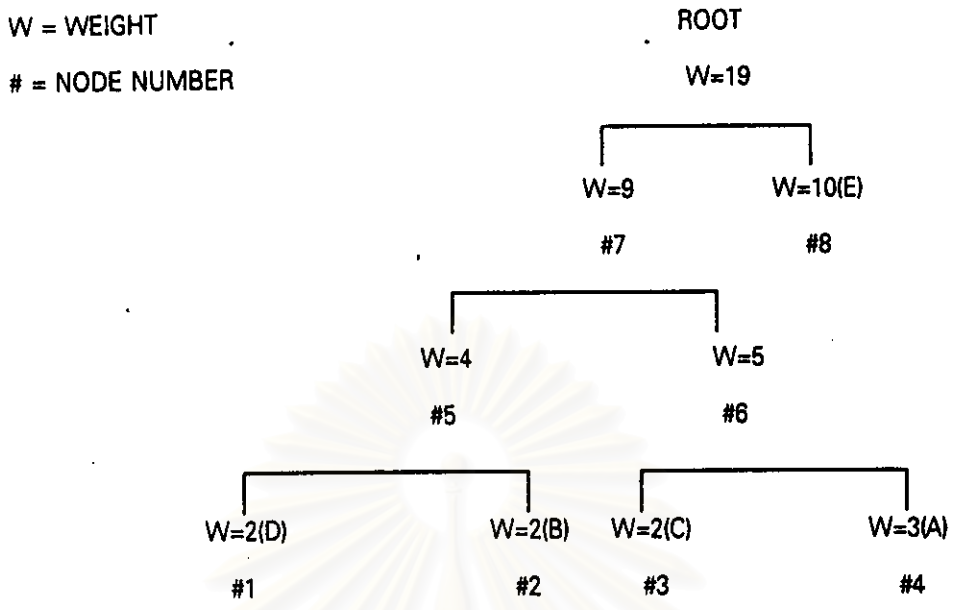
แผนภูมิต้นไม้ของฮัฟแมน เป็นแผนภูมิต้นไม้ที่มีค่าความถี่ของตัวอักษรให้กับทุกๆโหนด ซึ่งทุกๆ โหนดยกเว้นโหนดรากจะมีคุณสมบัติ การเป็นพี่น้อง (sibling) จากรูปที่ 4-8 ประกอบด้วยข้อมูลค่าความถี่ ของตัวอักษร A = 1, B = 2, C = 2, D = 2, E = 10 จากแผนภูมิต้นไม้เริ่มจากโหนดลูกที่ 1 เป็นค่าความถี่ของ A ไปจนถึงโหนดลูกที่ 4 เป็นค่าความถี่ของ D และโหนดภายในที่ 5 และ 6 จะมีค่าความถี่เท่ากับ 3 และ 4 ตามลำดับ โดยค่าความถี่ที่ได้จะเป็นผลรวมความถี่ของโหนดที่ 1, 2 และ 3, 4 ตามลำดับ และโหนดภายในที่ 7 ก็จะมีค่าความถี่เท่ากับ 7 ซึ่งเป็นค่าความถี่ที่ได้จากผลรวมค่าความถี่ของโหนดที่ 5 และ 6 จะเห็นว่าโหนดภายในที่ 7 จะอยู่ในระดับเดียวกับโหนดที่ 8 ซึ่งเป็นโหนดใบของตัวอักษร E ซึ่งมีค่าความถี่เป็น 10 สุดท้ายจะเป็นโหนดราก โหนดที่ 9 เป็นโหนดสูงสุดของแผนภูมิต้นไม้นี้ ซึ่งมีค่าความถี่เป็น 17 การเข้ารหัสโดยวิธี อะแดปทีฟฮัฟแมนนี้ คุณสมบัติความเป็นพี่น้อง (sibling) นี้มีความสำคัญมากเมื่อมีการปรับค่าความถี่ของโหนดต่างๆ ภายในแผนภูมิ



รูปที่ 4-9 แสดงการปรับแผนภูมิต้นไม้เมื่อโหนดถูกปรับค่าความถี่ (กรณีแรก)

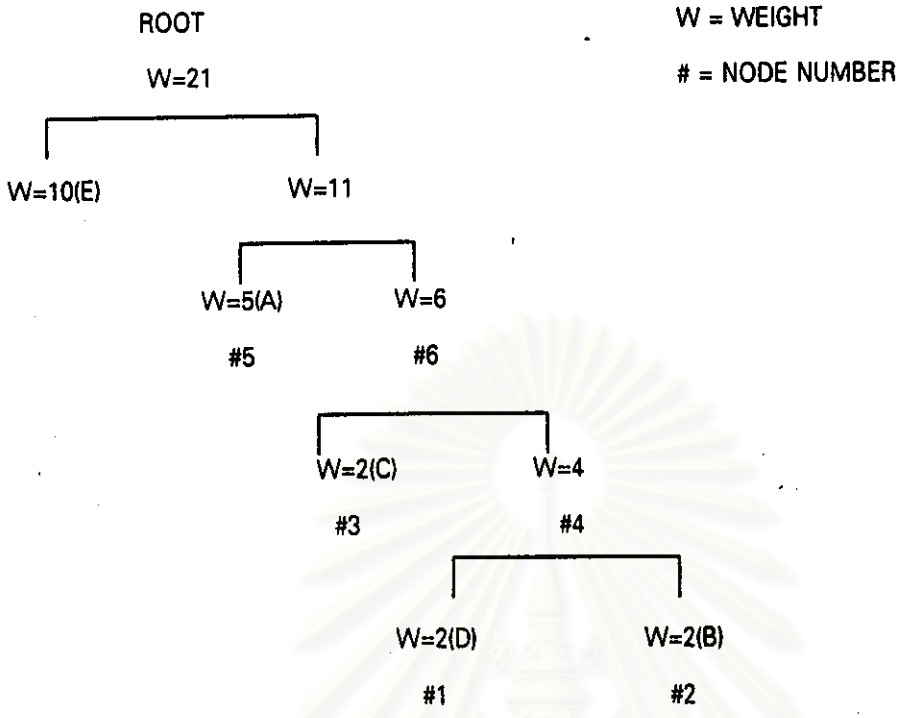
การเพิ่มค่าความถี่ให้กับโหนดจะทำให้เกิดเหตุการณ์ขึ้น 2 ประการ ดังนี้ ประการแรกถ้าหากโหนดที่ถูกเพิ่มค่าความถี่ยังคงรักษาคุณสมบัติโหนดพี่น้อง คือ ค่าความถี่มีค่าไม่มากกว่าโหนดถัดไปในระดับเดียวกัน เช่น ในรูปที่ 4-9 ถ้าเราเพิ่มค่าความถี่ของโหนดที่ 1 (A) ขึ้นอีก 1 โหนดจะมีค่าความถี่เป็น 2 ซึ่งจะเห็นว่ายังคงรักษาคุณสมบัติโหนดพี่น้อง ต่อไปทำการปรับปรุงค่าความถี่ของโหนดแม่ ซึ่งเป็นโหนดที่ได้รับผลกระทบจากโหนดที่ถูกปรับค่าความถี่เท่านั้น จะเห็นว่าโหนดที่ 5, 7 และ 9 เท่านั้นที่ถูกปรับค่าความถี่เสียใหม่

ประการที่สอง สมมติว่าโหนดที่ 1 (A) ถูกเพิ่มค่าความถี่อีก 1 ครั้ง จะทำให้โหนดนี้มีค่าความถี่เป็น 3 ซึ่งจะเห็นว่าได้มีการทำลายคุณสมบัติโหนดพี่น้อง เนื่องจากความถี่ของโหนดที่ 1 (A) มีค่ามากกว่าความถี่ของโหนดที่ 2 (B) ดังนั้นจึงต้องสลับโหนดที่ 1 (A) กับโหนดที่ 4 (D) ต่อไปปรับปรุงค่าความถี่ของโหนดแม่ของโหนดที่ถูกสลับตำแหน่ง แต่การเปลี่ยนแปลงนี้จะไม่มีผลต่อความยาวบิตของแต่ละตัวอักษร คือ ทั้งตัวอักษร A และ D ยังคงมีความยาวบิตเท่ากับ 3 บิตเหมือนเดิม ทั้งนี้เนื่องจากแผนภูมิต้นไม้ไม่มีการเปลี่ยนแปลงรูปร่างหลังการเพิ่มความถี่



รูปที่ 4-10 แสดงการปรับแผนภูมิต้นไม้เมื่อโหนดถูกปรับค่าความถี่(กรณีที่สอง)

ต่อไป สมมุติว่าโหนด A ถูกเพิ่มความถี่อีก 2 ครั้ง จนมีค่าความถี่เป็น 5 ซึ่งเมื่อเป็นดังนี้ จะเห็นว่าแผนภูมิต้นไม้ จะมีรูปร่างที่เปลี่ยนแปลงไปจากเดิมทันที ดังปรากฏในรูปที่ 4-10 การเปลี่ยนแปลงครั้งนี้ส่งผลต่อความยาวบิตของตัวอักษรทันที ดังนี้ A มีความยาว 2 บิต B และ D มีความยาว 4 บิต และ C มีความยาว 3 บิต ซึ่งจะเห็นว่าการเพิ่มความถี่ให้กับโหนด A ในครั้งนี้ส่งผลต่ออัตราส่วนการบีบอัดข้อมูลโดยตรง



รูปที่ 4-11 แสดงการเปลี่ยนแปลงรูปร่างแผนภูมิต้นไม้

5. <u>วิธี LZ77</u>

ในปี ค.ศ. 1977 Ziv และ Lempel ได้ตีพิมพ์เรื่อง " A Universal Algorithm for Sequential Data Compression " ในวารสาร IEEE Transactions on Information Theory ซึ่งจากหลักการดังกล่าวต่อมาได้พัฒนาเป็น LZ77

วิธีการบีบอัดข้อมูล LZ77 จะใช้ข้อความที่พบก่อนหน้าเป็นพจนานุกรม ซึ่งจะแทนค่าวลีขนาดต่างๆด้วยพอยเตอร์ที่ชี้ไปยังข้อความที่พบก่อนหน้า โครงสร้างหลักๆของวิธีการ LZ77 ประกอบด้วย ส่วนของข้อความที่อ่านเข้ามาเพื่อทำการบีบอัดข้อมูล และส่วนของข้อความที่พบก่อนหน้า ตัวอย่างเช่น

for (I=0 ; i<MAX-1;i++)\r for(j=i+1 ;j	<MAX ;j++)\r

ส่วนของข้อความที่อ่านเข้ามาใหม่ ส่วนของข้อความที่พบก่อนหน้า

รูปที่ 4-12 แสดงหน้าต่างข้อความของวิธี LZ77

ในส่วนของข้อความที่พบก่อนหน้ามีข้อความ "<MAX;j++)\r" เมื่อตรวจสอบกับส่วนข้อความที่อ่านเข้ามาเพื่อจะทำการบีบอัดข้อมูล พบว่าตรงกับวลี "<MAX" ที่ตำแหน่ง 14 ในข้อความที่จะทำการบีบอัดข้อมูล และ ตรงกับอักษร 4 ตัวแรกในส่วนของข้อความที่พบก่อนหน้า และ " " คือตัวอักษรแรกที่ไม่ตรงกัน ฉะนั้นเราจะได้รหัส 14, 4, ' ' ต่อไปจะขยับการอ่านข้อความเข้ามาอีก 5 ตัวอักษร ทั้งในส่วนของข้อความที่พบก่อนหน้าและส่วนที่ต้องการบีบอัดข้อมูล และทำเช่นเดิมไปเรื่อยๆจนหมดข้อมูลนำเข้า

(i = 0 ; i<MAX-1;i++)\r for(j=i+1 ;j<MAX	; j++)\r a[i

ส่วนของข้อความที่อ่านเข้ามาใหม่ ส่วนของข้อความที่พบก่อนหน้า

รูปที่ 4-13 แสดงหน้าต่างข้อความหลังจากเข้ารหัส 14, 4, ' '

ข้อเสียของวิธีนี้คือ ข้อจำกัดของพื้นที่จัดเก็บของข้อความทั้ง 2 ส่วน แต่การขยายขนาดพื้นที่ของข้อความทั้ง 2 ส่วนจะมีผลให้ขนาดของข้อมูลที่ผ่านการบีบอัดมีขนาดใหญ่ขึ้นตามไปด้วย นอกจากนี้ขนาดของวลีที่สามารถจับคู่กับข้อความที่ต้องการบีบอัดข้อมูลขึ้นกับขนาดของส่วนที่เก็บข้อความที่พบก่อนหน้า นอกจากนี้วิธีนี้ก็ไม่เหมาะกับข้อความที่พบมานานแล้วเพราะจะไม่ปรากฏในส่วนของข้อความที่พบก่อนหน้า

6. วิธี LZ78

เนื่องจาก LZ77 มีข้อด้อยหลายประการ Ziv และ Lempel จึงได้คิดค้นวิธีการบีบอัดข้อมูลใหม่ โดยตีพิมพ์ในวารสาร IEEE Transactions on Information Theory (ฉบับเดือนกันยายน 1978) เรื่อง "Compression of Individual Sequences via Variable-Rate Coding" ซึ่งต่อมาได้พัฒนาเป็น วิธีการบีบอัดข้อมูล LZ78

วิธี LZ78 ทั้งการบีบอัดข้อมูลและการขยายข้อมูล ในตอนเริ่มต้นพจนานุกรมจะว่าง แต่จะมีการสร้างพจนานุกรมไปพร้อมๆกับการบีบอัดและขยายข้อมูล ดังตัวอย่าง

ข้อมูลนำเข้า "DAD DADA DADDY DADO...."

วลีส่งออก	อักขระส่งออก	รหัส
0	'D'	"D"
0	'A'	"A"
1	' '	"D "
1	'A'	"DA"
4	' '	"DA "

วลีส่งออก	อักขระส่งออก	รหัส
4	'D'	"DAD"
1	'Y'	"DY"
0	' '	" "
6	'O'	"DADO"

ตารางที่ 4-10 แสดงการสร้างรหัสโดยวิธี LZ78

จากตัวอย่างขั้นแรกจะอ่านตัวอักษร 'D' เข้ามา เนื่องจากพจนานุกรมว่างอยู่ จึงได้เอาต์พุต เป็น 0 ส่งค่า (0+character) ซึ่งเท่ากับ "D" เก็บในพจนานุกรม เป็น phrase 1 ต่อมาอ่านอักษร 'A' เข้ามาซึ่งไม่พบในพจนานุกรมเช่นกัน จึงเก็บ "A" เป็น phrase 2 และส่ง 0 เป็นเอาต์พุต ต่อมาเมื่ออ่านอักษร 'D' เข้ามา พบว่าตรงกับ phrase 1 ที่อยู่ในพจนานุกรม จึงอ่าน ' ' เข้ามาและรหัส 1 เป็นเอาต์พุตเนื่องจากอักษร 'D' ที่อ่านเข้ามาก่อนหน้าตรงกับ pharase 1 ในพจนานุกรม และนำ "D " เก็บในพจนานุกรม ทำเช่นเดิมต่อไปจนกว่าจะหมดข้อมูลนำเข้า จะได้พจนานุกรม ดังนี้

0	""
1	"D"
2	"A"
3	"D"
4	"DA"
5	"DA "
6	"DAD"
7	"DY"
8	" "
9	"DADO"

ตารางที่ 4-11 แสดงตารางรหัสที่สร้างโดยวิธี LZ78

ส่วนการขยายข้อมูลจะต้องสร้างพจนานุกรมเช่นเดียวกัน

ข้อเสียของวิธี LZ78 คือ ขนาดของพจนานุกรมจะโตขึ้นอย่างรวดเร็วจนเต็มเนื้อที่จัดเก็บ เช่น ถ้าใช้ รหัส 16 บิต พจนานุกรมจะเต็มเมื่อเราเพิ่มรหัสเข้าไป 65,535 ตัว ซึ่งทำให้ไม่สามารถเพิ่มรหัสใหม่ได้ และจะมีผลให้ข้อมูลที่ผ่านการบีบอัดไม่ถูกต้อง

7. การบีบอัดข้อมูลโดยวิธี LZW

การบีบอัดและขยายข้อมูลโดยวิธี LZW เป็นวิธีการบีบอัดข้อมูลแบบอะแดปตีฟ ซึ่งวิธีการบีบอัดข้อมูลแบบอะแดปตีฟนี้พัฒนามาจากแนวคิดของ Jacob Ziv และ Abraham Lempel วิธีการของ Ziv และ Lempel รวมเรียกว่า การเข้ารหัส LZ ถือเป็นต้นแบบของการบีบอัดข้อมูล ซึ่งมีผู้ปรับปรุงและพัฒนาต่อไปมากมาย เช่น LZR, LZSS, LZB, LZH, LZW และ LZJ เป็นต้น

สำหรับ LZW นั้น จากการที่ Terry Welch ตีพิมพ์เรื่อง "A Technique for High-Performance Data Compression " ในวารสาร IEEE (Computer) ในปี ค.ศ. 1984 เทคนิคที่ Welch ได้กล่าวถึงต่อมาได้พัฒนาเป็น LZW Compression

วิธีการบีบอัดข้อมูล โดยวิธี LZW นี้พัฒนามาจาก วิธี LZ78 แต่จะสร้างพจนานุกรมสำหรับอักขระที่มีอยู่แล้ว เช่น A , B และ C เป็นต้น และเก็บเป็นพจนานุกรมเริ่มต้น การบีบอัดข้อมูลจะอ่านคำหรือวลีจากอินพุตเข้ามาและตรวจสอบกับตารางรหัสที่มีอยู่ ถ้าหาไม่พบก็จะนำคำหรือวลีนั้นไปสร้างเป็นรหัสใหม่ และส่งอักษรนั้นเป็นเอาต์พุต ถ้าหากตรวจสอบพบว่ามีรหัสอยู่แล้ว จะส่งรหัสนั้นๆเป็นเอาต์พุต สำหรับการขยายข้อมูลก็จะอ่านรหัสจากอินพุตเข้ามาและแปลเป็นอักษรหรือวลีส่งเป็นเอาต์พุต ในขณะเดียวกันก็สร้างรหัสใหม่ไปด้วย จึงเป็นข้อดีของวิธีการบีบอัดข้อมูลแบบ LZW ที่ไม่ต้องเก็บตารางรหัสจากการบีบอัดข้อมูล เพื่อนำมาใช้ในการขยายข้อมูล ทำให้ประหยัดเนื้อที่ได้มาก

ตัวอย่างวิธีการบีบอัดข้อมูลวิธี LZW

ข้อมูลนำเข้า " WED WE WEE WEB WET "

อักขระที่อ่านเข้ามา	รหัสส่งออก	รหัสใหม่
" W"	' '	256 = " W"
"E"	'W'	257 = "WE"
"D"	'E'	258 = "ED"
" "	'D'	259 = "D "
"WE"	256	260 = " WE"
" "	'E'	261 = "E "
"WEE"	260	262 = " WEE"
" W"	261	263 = "E W"
"EB"	257	264 = "WEB"
" "	B	265 = "B "
"WET"	260	266 = " WET"

อักขระที่อ่านเข้ามา	รหัสส่งออก	รหัสใหม่
<EOF>	T	

ตารางที่ 4-12 แสดงการสร้างรหัสโดยวิธี LZW

ขั้นแรกจะอ่าน " W" จากอินพุต ตรวจสอบกับตารางรหัส เมื่อตรวจสอบแล้วไม่พบจะส่ง " " เป็นเอาต์พุท และสตริง " W" จะถูกเพิ่มเป็นรหัสใหม่ในตาราง โดยกำหนดเป็นรหัส 256 เนื่องจากรหัส 0-255 ได้สงวนไว้สำหรับตัวอักษรโดดๆ ต่อไปอ่านอักษร "E" เข้ามา ตอนนี้ "WE" จะถูกเพิ่มเป็นรหัสใหม่ในตาราง และส่งรหัสของ "W" เป็นเอาต์พุท ต่อไปอ่าน "D" เข้ามา string code ตอนนี้คือ "ED" ซึ่งจะถูกเพิ่มเป็นรหัสใหม่ในตาราง และส่งรหัสของ "E" เป็นเอาต์พุท ต่อไปอ่าน " " เข้ามา string code ตอนนี้คือ "D " ซึ่งถูกเพิ่มเป็นรหัสใหม่ และส่งรหัสของ "D" เป็นเอาต์พุท ต่อไปอ่าน "WE" เข้ามาเมื่อตรวจสอบกับตารางพบว่าตรงกับรหัส 256 ก็จะส่ง 256 เป็นเอาต์พุท และเพิ่ม " WE" ไปในตาราง ต่อไปทำเช่นเดิมจนกว่าจะหมดข้อมูลนำเข้า

จากตัวอย่างจะเห็นว่า หลังจากบีบอัดข้อมูลแล้วจะได้รหัสแทนวลี 5 รหัสและตัวอักษร 7 ตัว ถ้าหากใช้รหัส 9 บิต จากข้อมูลนำเข้า 19 ตัวอักษร จะทำให้ลดเนื้อที่จัดเก็บลงเหลือ 13.5 ไบต์

การขยายข้อมูล

รหัสนำเข้า " WED<256>E<260><261><257>B<260>T"

อินพุต/		ข้อความ/		ตาราง
รหัสใหม่	รหัสเดิม	เอาต์พุท	อักษร	รหัส
' '	' '	" "		
'W'	' '	"W"	"W"	256 = " W"
'E'	'W'	"E"	'E'	257 = "WE"
'D'	'E'	"D"	'D'	258 = "ED"
256	'D'	" W"	' '	259 = "D "
'E'	256	"E"	'E'	260 = " WE"
260	'E'	" WE"	' '	261 = "E "
261	260	"E"	'E'	262 = " WEE"
257	261	"WE"	'W'	263 = "E W"
'B'	257	"B"	'B'	264 = "WEB"
260	'B'	" WE"	' '	265 = "B "

อินพุต/		ข้อความ/		ตาราง
รหัสใหม่	รหัสเดิม	เอาต์พุท	อักษร	รหัส
'T'	260	"T"	'T'	266 = " WET"

ตารางที่ 4-13 แสดงการขยายข้อมูลโดยวิธี LZW

การขยายข้อมูลขั้นแรกจะอ่าน ' ' จากอินพุตเข้ามาเนื่องจากรหัสไม่เกิน 255 จึงส่ง ' ' เป็นเอาต์พุต ต่อไป อ่าน "W" เข้ามา รหัสไม่เกิน 255 ก็จะส่ง "W" เป็นเอาต์พุต ขณะเดียวกันนำ " W" สร้างเป็นรหัสใหม่ คือ 256 เพื่อเก็บไว้ใช้ต่อไป ต่อไปอ่าน "E" เข้ามา นำ "WE" มาสร้างรหัสใหม่ และส่ง "E" เป็นเอาต์พุต ต่อไปอ่าน "D" เข้ามา นำ "ED" มาสร้างเป็นรหัสใหม่ และส่ง "D" เป็นเอาต์พุต ต่อไปอ่าน 256 เข้ามาเนื่องจากมีค่าเกิน 255 จึงตรวจสอบกับตารางรหัส ซึ่งจะได้ " W" ส่งเป็นเอาต์พุต และนำ "D " สร้างเป็นรหัสใหม่ ทำเช่นนี้ไปจนกระทั่งหมดข้อมูลนำเข้า