บทที่ 2
การอัดข้อมูล

ในบทนี้จะแบ่งออกเป็น 3 ส่วน ส่วนแรกจะกล่าวถึงศึกษาถึงทฤษฎีพื้นฐานของการอัดข้อมูลที่ควรทราบได้แก่ รหัส(Code) คุณสมบัติของรหัสที่นำมาใช้ในการอัดข้อมูล การแทนรหัส (Code Mapping) อัลกอริทึมการอัดข้อมูล วิธีการวัดประสิทธิภาพของการอัดข้อมูล นอกจากนี้จะอธิบายถึงคำศัพท์เฉพาะที่ใช้ในวิชาทฤษฎีสารสนเทศ (Information Theory) ได้แก่ สารสนเทศ (Information) เอนโทรปี (Entropy) ความเหลือเฟือ (Redundancy) ในบทความเกี่ยวกับการอัดข้อมูลจะพบคำเหล่านี้เสมอ เพราะการอัดข้อมูลเป็นสาขาหนึ่งของวิชาทฤษฎีสารสนเทศ ส่วนที่สองจะกล่าวถึงวิธีการอัดข้อมูลวิธีต่าง ๆ ที่น่าสนใจจากบทความเกี่ยวกับการอัดข้อมูล ส่วนที่สามจะกล่าวถึงโปรแกรมอัดข้อมูลที่มีใช้อยู่ในปัจจุบัน

2.1 ทฤษฎีพื้นฐานของการอัดข้อมูล

2.1.1 รหัสและการแทนรหัส (Code and Code Mapping)

การอัดข้อมูลเป็นการใช้วิธีการบางอย่างเพื่อให้ระบบการเก็บข้อมูลในเครื่องคอมพิวเตอร์สามารถใช้เนื้อที่ของหน่วยเก็บข้อมูล หรือใช้เวลาในการรับ-ส่งข้อมูลน้อยลงจากเดิม วิธีการอัดข้อมูลแต่ละวิธีจะมีวิธีการในการสร้างรหัสใหม่ที่แตกต่างกันไป เมื่อสร้างรหัสใหม่ได้แล้วรหัสเดิมก็จะถูกแทนที่ด้วยรหัสใหม่ที่สร้างขึ้น การแทนที่รหัสนี้เราเรียกว่า การแทนรหัส ซึ่งมีผลทำให้แฟ้มข้อมูลที่ใช้รหัสรูปแบบใหม่มีขนาดลดลง การแทนรหัสแบ่งออกเป็น 4 ลักษณะ ได้แก่

1. การแทนรหัสแบบคงที่-คงที่ (Block-Block Code Mapping)
2. การแทนรหัสแบบคงที่-แปรได้ (Block-Variable Code Mapping)
3. การแทนรหัสแบบแปรได้-คงที่ (Variable-Block Code Mapping)
4. การแทนรหัสแบบแปรได้-แปรได้ (Variable-Variable Code Mapping)

จากข้อมูล "aa bbb cccc ddddd eeeeee fffffffggggggggg" ประกอบด้วยรหัสแอสกี (ASCII Code) มีขนาดคงที่ 8 บิตความยาว 40 ตัว ดังรูปที่ 2.1(ก) เป็นการแทนรหัสแบบคงที่-คงที่ จากรหัสขนาดคงที่ 8 บิต ด้วยรหัสใหม่ขนาดคงที่ 3 บิต สำหรับการแทนรหัสแบบแปรได้-แปรได้ เป็นการแทนรหัสจากรหัสขนาดไม่คงที่ด้วยรหัสใหม่ที่มีขนาดไม่คงที่ ดังรูปที่ 2.1 (ข)

รหัสเดิม	รหัสใหม่
a	000
b	001
c	010
d	011
e	100
f	101
g	110
Space	111

(ก)

รหัสเดิม	รหัสใหม่
aa	0
bbb	1
cccc	10
ddddd	11
eeeeee	100
fffffff	101
ggggggggg	110
Space	111

(ข)

รูปที่ 2.1 ตัวอย่างการแทนรหัส

ถ้าข้อมูลถูกแทนรหัสโดยใช้รหัสใหม่ ดังรูปที่ 2.1 (ก) ความยาวของข้อมูลที่ถูกอัดคือ 120 บิต (3 บิต x 40 ตัว) แต่ถ้าใช้รหัสใหม่ ดังรูปที่ 2.1 (ข) จะเหลือเพียง 30 บิต จากขนาดเดิม 320 บิต (8 บิต x 40 ตัว)

เมื่อมีแฟ้มข้อมูลที่จะนำมาอัดข้อมูล สิ่งที่ต้องพิจารณาคือ การแบ่งข้อมูลออกเป็นหน่วยย่อย ๆ เรียกว่า รหัส รหัสที่ใช้ในการอัดข้อมูลอาจเป็นรหัสที่ใช้อยู่ในระบบคอมพิวเตอร์ เช่น รหัสแอสกี รหัสแอบซีดิก (EBCDIC Code) หรือรหัสที่กำหนดขึ้นใหม่โดยการใช้ข้อมูลในลักษณะข้อมูลไบนารี (Binary Data) การแบ่งข้อมูลมี 2 ลักษณะคือ แบบกำหนดรูปแบบ (Defined-Word Scheme) และแบบกระจายอิสระ (Free Parse)

การแบ่งข้อมูลแบบกำหนดรูปแบบ วิธีการอัดข้อมูลจะกำหนดรหัสที่มีในข้อมูลต้นกำเนิดไว้ก่อนเข้ารหัสข้อมูล เช่น วิธีการของฮัฟแมน วิธีการของชานนอน-ฟาโน วิธีการทั้ง 2 นี้จะอ่านข้อมูล 2 รอบ อ่านข้อมูลรอบแรกเพื่อศึกษาดูว่าข้อมูลประกอบด้วยรหัสใดบ้าง แต่ละรหัสมีจำนวนครั้งในการใช้งานเท่าใด แล้วสร้างรหัสใหม่จากความถี่ของรหัสที่รวบรวมได้จากข้อมูลต้นกำเนิดอ่านข้อมูลรอบที่ 2 เพื่อเข้ารหัสข้อมูล รหัสในลักษณะนี้จะมีรูปแบบรหัสไม่เปลี่ยนแปลงตลอดเวลาของการเข้ารหัส

สำหรับการแบ่งข้อมูลแบบกระจายอิสระ ข้อมูลต้นกำเนิดจะกำหนดรหัสและสร้างรหัสใหม่ในขณะที่กำลังเข้ารหัสข้อมูล เช่น วิธีการอแดปทีฟฮัฟแมน วิธีการของแลมเพล-ซีฟ การอัดข้อมูลที่ใช้วิธีการแบ่งข้อมูลในลักษณะนี้ จะมีรูปแบบของรหัสไม่แน่นอนเปลี่ยนแปลงไปตามลักษณะของข้อมูลในแต่ละช่วง และเป็นวิธีการอัดข้อมูลที่ทำงานเพียงรอบเดียว

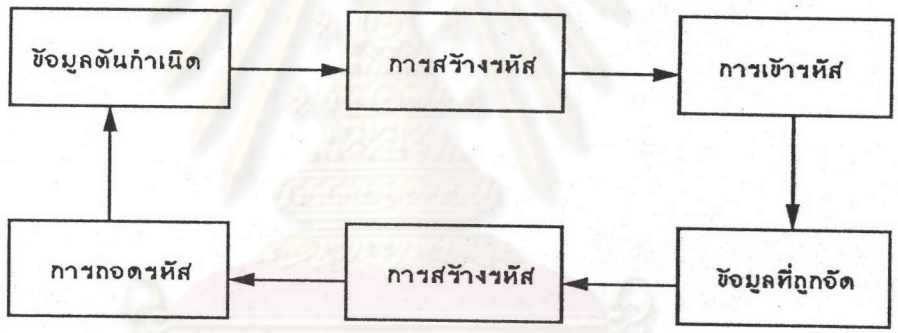
2.1.2 อัลกอริทึมการอัดข้อมูล (Data Compression Algorithm)

อัลกอริทึมการอัดข้อมูล ประกอบด้วย 3 อัลกอริทึม คือ

2.1.2.1 อัลกอริทึมการสร้างรหัส (Code Construction Algorithm) เป็นขั้นตอนที่ศึกษาและรวบรวมลักษณะของข้อมูล แล้วนำลักษณะของข้อมูลที่รวบรวมได้มาใช้ในการสร้างรหัสใหม่

2.1.2.2 อัลกอริทึมการเข้ารหัส (Encoding Algorithm) เป็นขั้นตอนการแทนรหัสเดิมในแฟ้มข้อมูลต้นกำเนิด (Source File) ด้วยรหัสใหม่ที่ได้จากอัลกอริทึมการสร้างรหัส เมื่อจบขั้นตอนการเข้ารหัสจะได้แฟ้มข้อมูลที่ถูกอัดแล้ว (Compressed File)

2.1.2.3 อัลกอริทึมการถอดรหัส (Decoding Algorithm) เป็นขั้นตอนการถอดรหัสข้อมูลที่ถูกอัดให้กลับคืนเป็นข้อมูลต้นกำเนิด



รูปที่ 2.2 อัลกอริทึมการอัดข้อมูล

วิธีการอัดข้อมูลที่มีการทำงานแบบรอบเดียว อัลกอริทึมการสร้างรหัสและอัลกอริทึมการเข้ารหัสเป็นขั้นตอนที่ทำควบคู่พร้อมกันไป ส่วนวิธีการอัดข้อมูลที่มีการทำงานแบบ 2 รอบ เช่น วิธีการของฮัฟแมน อัลกอริทึมการสร้างรหัสและอัลกอริทึมการเข้ารหัสจะทำแยกจากกันทีละขั้นตอน

อัลกอริทึมการอัดข้อมูลโดยวิธีการของฮัฟแมน ประกอบด้วยขั้นตอนสร้างรหัสใหม่จากค่าความน่าจะเป็นของรหัสที่เกิดขึ้นทั้งหมดในข้อความ ต่อจากนั้นจึงเข้ารหัสข้อมูลด้วยรหัสใหม่ที่สร้างขึ้นได้แฟ้มข้อมูลที่ถูกอัดแล้ว เมื่อต้องการแฟ้มข้อมูลต้นกำเนิดมาใช้งานก็สามารถทำได้โดยใช้อัลกอริทึมการถอดรหัส (ศึกษารายละเอียดของอัลกอริทึมฮัฟแมน ในหัวข้อ 2.2.1)

ตัวอย่างที่ 2.1 : จากข้อมูล 'DEAAABBCDABBCCCAAEDB'

- สร้างรหัสใหม่ตามหลักการของอัลกอริทึมฮัฟแมนได้ดังนี้

รหัส	ความน่าจะเป็น	รหัสใหม่
A	.30	00
B	.25	01
C	.20	10
D	.15	111
E	.10	110

- เข้ารหัสข้อมูลโดยการแทนที่รหัสเดิมด้วยรหัสใหม่ ได้ข้อมูลที่ถูกอัดคือ '1111100000000101 .....' (เป็นข้อมูลไบนารี)

- ถอดรหัสข้อมูลโดยการแทนที่รหัสใหม่ด้วยรหัสเดิม ได้ข้อมูลต้นกำเนิดคือ 'DEAAABBCDABBCCCAAEDB'

2.1.3 คุณสมบัติของรหัส

รหัสใหม่ที่นำมาใช้ในการอัดข้อมูลต้องมีคุณสมบัติดังต่อไปนี้

2.1.3.1 สามารถถอดรหัสได้รหัสเดียว (Uniquely Decodable) หมายความว่า รหัสใหม่สำหรับแต่ละรหัสจะต้องไม่ซ้ำกัน และในการถอดรหัสจะได้ความหมายของรหัสเพียงรหัสเดียว โดยไม่สามารถถอดรหัสได้ในความหมายของรหัสอื่นอีก เช่น จากตัวอย่างที่ 2.1 รหัส '111' ถอดรหัสได้เป็น 'D' เท่านั้น เพราะไม่มีรหัส '1' '11' หรือ '1111' รวมอยู่ในกลุ่มรหัสใหม่

2.1.3.2 รหัสที่มีคุณสมบัติถอดได้รหัสเดียว จะเป็นรหัสที่คุณสมบัติเป็นรหัสนำหน้า (Prefix Code) หมายความว่า รหัสแต่ละรหัสจะต้องไม่นำหน้าด้วยรหัสใด ๆ ที่มีอยู่ เช่น จากตัวอย่างที่ 2.1 รหัส 'D' (คือ '111') มีคุณสมบัติรหัสนำหน้า เนื่องจากส่วนหน้าของรหัส(คือ '1' หรือ '11') ไม่ใช่รหัสใด ๆ ที่มีอยู่

2.1.3.3 ในการถอดรหัสของข้อมูลที่เข้ารหัสด้วยรหัสที่มีคุณสมบัติเป็นรหัสนำหน้าจะสามารถถอดรหัสได้ทันที (Instantaneously Decodable) โดยไม่ต้องอ่านข้อมูลส่วงหน้าไปก่อน เช่น ในการถอดรหัสข้อมูล '1000000001' โดยใช้รหัส {1, 00, 100000} รหัสแรกของข้อมูลคือ 1 ซึ่งตามความเป็นจริงแล้วไม่สามารถกำหนดลงไปได้จนกว่าจะอ่านข้อมูลถึงรหัสสุดท้ายของข้อมูล เพราะรหัสแรกของข้อมูลจะเป็น 100000 ทันที ถ้าจำนวนของ 0 เป็นจำนวนคี่ สำหรับรหัสตามตัวอย่างที่ 2.1 เป็นรหัสที่สามารถถอดรหัสได้ทันที (Lelewer and Hirschberg 1987 : 264)

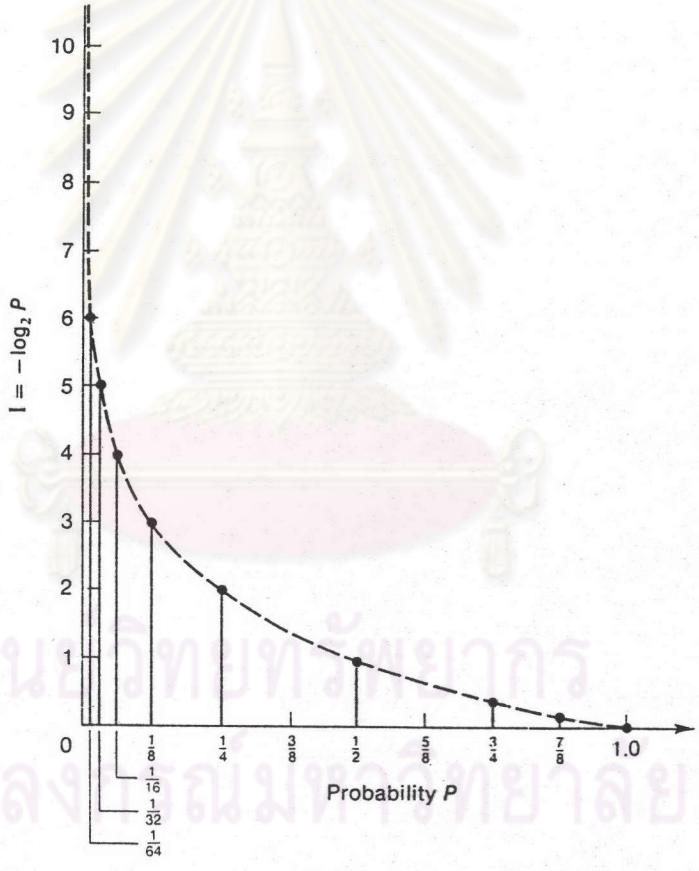
2.1.4 สารสนเทศและเอนโทรปี

สารสนเทศความหมายที่ใช้กับการอัดข้อมูล หมายถึง ค่าที่วัดจากปริมาณการใช้งานของรหัสในข้อความ ถ้ารหัสใดใช้งานน้อยจะมีสารสนเทศมาก และรหัสใดใช้งานมากก็จะมีสารสนเทศน้อย สารสนเทศของข้อความสามารถแสดงได้ด้วยฟังก์ชันค่าลดของความน่าจะเป็นของรหัสต่าง ๆ ที่เกิดขึ้น ดังรูปที่ 2.3

กำหนดให้ I_i เป็นสารสนเทศของรหัสตัวที่ i

p_i เป็นความน่าจะเป็นของการเกิดรหัสตัวที่ i ($0 <= p_i <= 1$)

จะได้ $I_i = - \log_2 p_i$



รูปที่ 2.3 ฟังก์ชันของสารสนเทศ $I = -\log_2 p$

ถ้า p_i มีค่ายิ่งน้อยสารสนเทศของรหัสยิ่งมาก ถ้า p_i มีค่าเป็น 1 หมายความว่า สารสนเทศมีค่าเท่ากับ 0 ($-\log_2 1 = 0$) ฟังก์ชัน $-\log_2 p_i$ สามารถใช้กับเลขฐานอื่น ๆ ก็ได้ แต่การใช้เลขฐาน 2 จะเหมาะสมมากที่สุดเพราะเป็นเลขฐานที่ใช้ในการแทนความหมายของรหัสต่าง ๆ ในระบบคอมพิวเตอร์ (Lynch 1985 : 13-14)

กำหนดให้ข้อมูลต้นกำเนิดประกอบด้วยกลุ่มรหัส n รหัส $a_1, a_2, \ldots\ldots, a_n$ รหัสแต่ละรหัสถูกเลือกโดยการสุ่มอย่างอิสระจากกลุ่มรหัสด้วยความน่าจะเป็น $p_1, p_2, \ldots\ldots, p_n$

จะได้ สารสนเทศของ $a_i = -\log_2 p_i$

ดังนั้น สารสนเทศเฉลี่ย $= \sum_{i=1}^{n} -p_i \log_2 p_i$

สารสนเทศเฉลี่ย เรียกว่า เอนโทรปี (H) เป็นค่าที่แสดงจำนวนหน่วยเฉลี่ยต่ำสุดของรหัสที่ใช้ในการเข้ารหัส

ในกรณีที่ความน่าจะเป็นของการใช้งานของรหัส $a_1, a_2, \ldots\ldots, a_n$ รหัสแต่ละรหัสมีการใช้งานเท่า ๆ กันคือ ความน่าจะเป็น $p1, p_2, \ldots\ldots, p_n$ มีค่าเท่ากันเท่ากับ 1/n ดังนั้นจะได้ ค่าเอนโทรปีสูงสุด (Maximum Entropy) มีค่าเท่ากับ $\log_2 n$ (H_{max})

ตัวอย่างที่ 2.2 จากข้อมูลต้นกำเนิดประกอบด้วยรหัส 8 รหัส และมีการใช้งานของรหัสเป็นดังนี้

รหัส (a_i)	ความน่าจะเป็น (p_i)	สารสนเทศ ($-\log_2 p_i$)
A	1/2	1
B	1/4	2
C	1/8	3
D	1/16	4
E	1/32	5
F	1/64	6
G	1/128	7
H	1/128	7

- เอนโทรปีของข้อมูล

$$H = \sum_{i=1}^{8} -p_i \log_2 p_i$$

$$= 1/2\ (1) + 1/4\ (2) + 1/8\ (3) + 1/16\ (4) + 1/32\ (5) + 1/64\ (6) + 1/128\ (7) + 1/128\ (7)$$

$$= 1\ 63/64$$

- เอนโทรปีสูงสุดของข้อมูล

$$H_{max} = \log_2 8$$

$$= 3$$

2.1.5 ความเหลือเฟือ

ถ้าเราพิจารณาถึงข้อมูลที่มีการใช้งานทั่ว ๆ ไป จะประกอบด้วยสารสนเทศซึ่งเป็นส่วนสำคัญจริง ๆ ของข้อมูล อีกส่วนหนึ่งเป็นส่วนที่ไม่มีความจำเป็นซึ่งสามารถกำจัดทิ้งไปได้โดยวิธีการอัดข้อมูล เราเรียกส่วนนี้ของข้อมูลว่า ความเหลือเฟือ ลักษณะความเหลือเฟือที่พบได้ในข้อมูลประเภทข้อความ เช่น รหัสสำหรับรหัส 1 รหัสในตารางแอสกีใช้เนื้อที่ 8 บิต สมมติว่ามีแฟ้มข้อมูลแฟ้มหนึ่งซึ่งมีเฉพาะข้อมูลอักษรมีอักษรแตกต่างกันไม่เกิน 64 รหัสจากรหัสทั้งหมด 256 รหัส ดังนั้นเราสามารถแทนรูปแบบใหม่ให้กับรหัสในแฟ้มข้อมูลได้โดยใช้เนื้อที่เพียง 6 บิต โดยวิธีการเช่นนี้จะลดขนาดข้อมูลลงได้ 25 เปอร์เซนต์

ความเหลือเฟือที่พบในข้อมูล (Welch 1984 : 9-10) แบ่งออกได้ 4 ลักษณะ คือ

2.1.5.1 การกระจายอักษร (Character Distribution) ในกลุ่มอักษรกลุ่มหนึ่ง ๆ อักษรบางตัวจะใช้มากกว่าตัวอื่น ๆ หรือบางตัวใช้น้อยมาก ในแฟ้มข้อมูลเฉพาะประเภทจะมีการกระจายของอักษรแตกต่างกัน เช่น ในข้อความภาษาอังกฤษ อักษร e และช่องว่างจะใช้มากที่สุด ในขณะที่ข้อมูลประเภทสินค้าคงคลัง ตัวเลข 0-9 จะใช้มากกว่าอักษรอื่น ๆ

2.1.5.2 การซ้ำอักษร (Character Repetition) เมื่อข้อความมีการเกิดซ้ำของอักษรตัวเดียวกันต่อกันหลาย ๆ ตัว การลดความเหลือเฟืออาจทำได้โดยใช้สัญญลักษณ์อื่นแทนการกลุ่มของอักษรที่เกิดซ้ำ ความเหลือเฟือลักษณะนี้มักจะพบในข้อมูลประเภทรูปภาพ

2.1.5.3 รูปแบบที่ใช้งานมาก (High-usage Patterns) กลุ่มอักษรที่มีการเกิดซ้ำค่อนข้างสูงจะพบมากในแฟ้มข้อความ เช่น คำที่ใช้บ่อย ๆ

2.1.5.4 ความเหลือเฟือเฉพาะตำแหน่ง (Positional Redundancy) ในข้อมูลประเภทข้อความไม่มีความเหลือเฟือลักษณะนี้ แต่จะพบมากในฐานข้อมูล

ในวิชาทฤษฎีสารสนเทศ ความเหลือเฟือเป็นค่าที่สามารถคำนวณออกมาได้โดยมีสูตรที่เกี่ยวข้องกับเอนโทรปี และสารสนเทศ ซึ่งเป็นความเหลือเฟือในลักษณะที่เกิดจากการใช้งานไม่เท่ากันของรหัสต่าง ๆ

ความเหลือเฟือมีการกำหนดค่าที่แตกต่างกัน 2 ลักษณะ ลักษณะแรกกำหนดโดยโทมัส เจ ลินช์ กำหนดว่า ค่าความเหลือเฟือมีค่าเท่ากับผลต่างระหว่างค่าเอนโทรปีสูงสุดกับค่าเอนโทรปี ค่าความเหลือเฟือที่ได้จากสูตรนี้เป็นความเหลือเฟือทั้งหมดที่มีอยู่ในข้อมูล

$$R = H_{max} - H$$

$$= \log_2 n - H$$

โดยที่ R เป็นค่าความเหลือเฟือ (Lynch 1985 : 50)

การกำหนดค่าความเหลือเฟืออีกลักษณะหนึ่ง เป็นค่าความเหลือเฟือที่เหลืออยู่หลังจากทำการอัดข้อมูลแล้วคือ ผลต่างระหว่างความยาวเฉลี่ยของรหัสใหม่กับค่าเอนโทรปี ถ้าความเหลือเฟือเท่ากับ 0 แสดงว่าวิธีการอัดข้อมูลนั้นสามารถลดความเหลือเฟือออกได้หมด ดังนั้นความเหลือเฟือยิ่งมีค่าต่ำย่อมแสดงถึงประสิทธิภาพการอัดข้อมูลที่ดี การกำหนดความเหลือเฟือในลักษณะนี้สามารถนำไปใช้เป็นค่าสำหรับวัดประสิทธิภาพการอัดข้อมูลวิธีหนึ่ง (Lelewer and Hirschberg 1987 : 267)

กำหนดให้ความยาวของรหัสใหม่ที่ได้จากวิธีการอัดข้อมูลของรหัส $a_1, a_2, \ldots\ldots, a_n$ คือ $l_1, l_2, \ldots\ldots, l_n$ หน่วย ตามลำดับ

$$\text{ความยาวเฉลี่ยรหัสใหม่} = \sum_{i=1}^{n} p_i l_i$$

$$\text{ความเหลือเฟือ} = \sum_{i=1}^{n} p_i l_i - H$$

2.1.5 ประสิทธิภาพการอัด (Compression Efficiency)

ในการที่จะศึกษาวิเคราะห์ข้อดีข้อเสียของวิธีการอัดข้อมูล มีปัจจัยที่นำมาพิจารณาคือ ขนาดของโปรแกรม ความซับซ้อนของอัลกอริทึมที่ใช้ ประสิทธิภาพการอัด และเวลาที่ใช้ ถ้าการอัดข้อมูลมีการใช้งานเกี่ยวข้องกับการรับส่งข้อมูลด้วย จะต้องคำนึงถึงความเร็วที่ใช้ในการอัดข้อมูลเพราะเป็นการทำงานแบบเวลาจริงดังนั้นวิธีการที่ใช้ควรเป็นวิธีที่มีความเร็วที่ยอมรับได้และในขณะเดียวกันก็ให้ประสิทธิภาพที่ดีด้วย แต่ถ้าการอัดข้อมูลมีวัตถุประสงค์สำหรับหน่วยเก็บข้อมูลก็ควรจะพิจารณาถึงประสิทธิภาพการอัดก่อนเป็นปัจจัยแรก และควรจะพิจารณาปัจจัยอื่น ๆ ควบคู่ไปด้วย

การวัดประสิทธิภาพการอัดข้อมูลสามารถวัดได้หลายวิธี ได้แก่ การพิจารณาประสิทธิภาพจากความยาวเฉลี่ยรหัสใหม่ที่ได้จากวิธีการอัดข้อมูล เช่น จากงานวิจัยการสร้างพจนานุกรมสำหรับอัลกอริทึมไทยได้ใช้วิธีการของฮัฟแมนในการลดขนาดข้อมูล เพื่อประหยัดเนื้อที่ของหน่วยเก็บข้อมูล สามารถใช้ขนาดรหัสใหม่ 5.27 บิต จากขนาดของรหัสเดิม 7 บิต (ยืน

ภู่วรวรรณ 2527 : 282-284) การวัดประสิทธิภาพอีกวิธีหนึ่งโดยการหาค่าความเหลือเฟือที่เหลืออยู่ ดังที่กล่าวมาแล้วในข้อ 2.1.4 การวัดประสิทธิภาพโดยใช้ค่าเฉลี่ยรหัสใหม่ หรือค่าความเหลือเฟือ เหมาะสำหรับวิธีการอัดข้อมูลที่มีการใช้ความน่าจะเป็นสร้างรหัสใหม่ เช่น วิธีการของฮัฟแมน วิธีการของชานนอน-ฟาโน

นอกจากนี้อาจวัดประสิทธิภาพได้จาก ค่าอัตราส่วนการอัด (Compression Ratio) เป็นการหาอัตราส่วนของขนาดข้อมูลที่สามารถลดลงได้ทั้งหมด เปรียบเทียบกับขนาดของข้อมูลเดิม การวัดประสิทธิภาพโดยใช้สูตรนี้สามารถนำไปใช้กับวิธีการใด ๆ ก็ได้ โดยไม่ต้องใช้ค่าเอนโทรปี หรือความยาวเฉลี่ยรหัสใหม่ ซึ่งต้องขึ้นอยู่กับวิธีการที่ใช้ค่าความน่าจะเป็น

$$\text{อัตราส่วนการอัด} = \frac{\text{ขนาดข้อมูลต้นกำเนิด} - \text{ขนาดข้อมูลที่ผ่านการอัด}}{\text{ขนาดข้อมูลต้นกำเนิด}}$$

2.2 อัลกอริทึมการอัดข้อมูลที่ไม่ขึ้นกับซีแมนติค

2.2.1 วิธีการของชานนอน-ฟาโน

วิธีการนี้มีการอ่านข้อมูล 2 รอบ รอบแรกอ่านข้อมูลเพื่อหาความน่าจะเป็นของรหัสแต่ละรหัสที่เกิดขึ้นในข้อความ การสร้างรหัสใหม่มีขั้นตอนดังนี้

- ให้จัดเรียงความน่าจะเป็นของรหัสทั้งหมดที่เกิดขึ้นจากมากไปน้อย
- แบ่งรหัสออกเป็น 2 กลุ่มโดยให้ผลรวมความน่าจะเป็นของรหัสทั้ง 2 กลุ่มมีค่าเท่ากันหรือใกล้เคียงกันมากที่สุด
- กำหนดให้กลุ่มแรกมีค่าบิตเป็น 0 และกลุ่ม 2 มีค่าบิตเป็น 1 ต่อจากนั้นในแต่ละกลุ่มให้ทำซ้ำเช่นนี้อีก แล้วเอาค่าบิตตัวต่อไปเข้าไปต่อท้ายจนกว่าแต่ละกลุ่มมีสมาชิกเป็น 1 ก็จะได้รหัสใหม่ครบทุกรหัสที่ปรากฏในข้อมูล

ตัวอย่างการสร้างรหัสดังรูปที่ 2.4 จากข้อมูลต้นกำเนิดคือ aa bbb cc cc ddddd eeeeee fffffffggggggggg ความยาวของข้อความคือ 40 ตัว นับจำนวนรหัสแต่ละรหัสที่เกิดขึ้นในข้อมูล คำนวณเป็นค่าความน่าจะเป็น แล้วจัดเรียงอันดับตามค่าความน่าจะเป็น แบ่งกลุ่มรหัสออกเป็น 2 กลุ่มคือ กลุ่ม (e, f, g) มีความน่าจะเป็นรวมเท่ากับ 21/40 และกลุ่ม (a, b, c, d, space) มีความน่าจะเป็นรวมเท่ากับ 19/40 กำหนดให้กลุ่ม (e, f, g) ได้บิตแรกของรหัสใหม่เป็น 0 และกลุ่ม (a, b, c, d, space) ได้บิตแรกของรหัสใหม่เป็น 1 ในแต่ละกลุ่มให้ทำซ้ำวิธีการเช่นเดิมจนแต่ละกลุ่มมีสมาชิกเป็น 1

รหัส	ความน่าจะเป็น		บิตแรก	รหัสใหม่
g	8/40		0	0 0
f	7/40	21/40	0	0 1 0
e	6/40		0	0 1 1
d	5/40		1	1 0 0
space	5/40		1	1 0 1
c	4/40	19/40	1	1 1 0
b	3/40		1	1 1 1 0
a	2/40		1	1 1 1 1

รูปที่ 2.4 วิธีการของชานนอน-ฟาโน

วิธีการนี้จะมีความยาวของรหัสใหม่เท่ากับ $-\log p(a_i)$ ซึ่งเป็นจริงต่อเมื่อสามารถแบ่งกลุ่มรหัสแล้วได้ความน่าจะเป็นของแต่ละกลุ่มเท่ากันจริง ๆ แต่ในทางปฏิบัติแล้วไม่สามารถทำได้ รหัสใหม่บางตัวจะมีความยาวเท่ากับ $-\log p(a_i) + 1$

ถ้ากำหนดให้ S เป็นความยาวเฉลี่ยรหัสใหม่

H เป็นความยาวเฉลี่ยสารสนเทศ (เอนโทรปี)

ดังนั้นจะได้ $H <= S <= H + 1$ วิธีการนี้ไม่รับประกันค่าต่ำสุดของ S

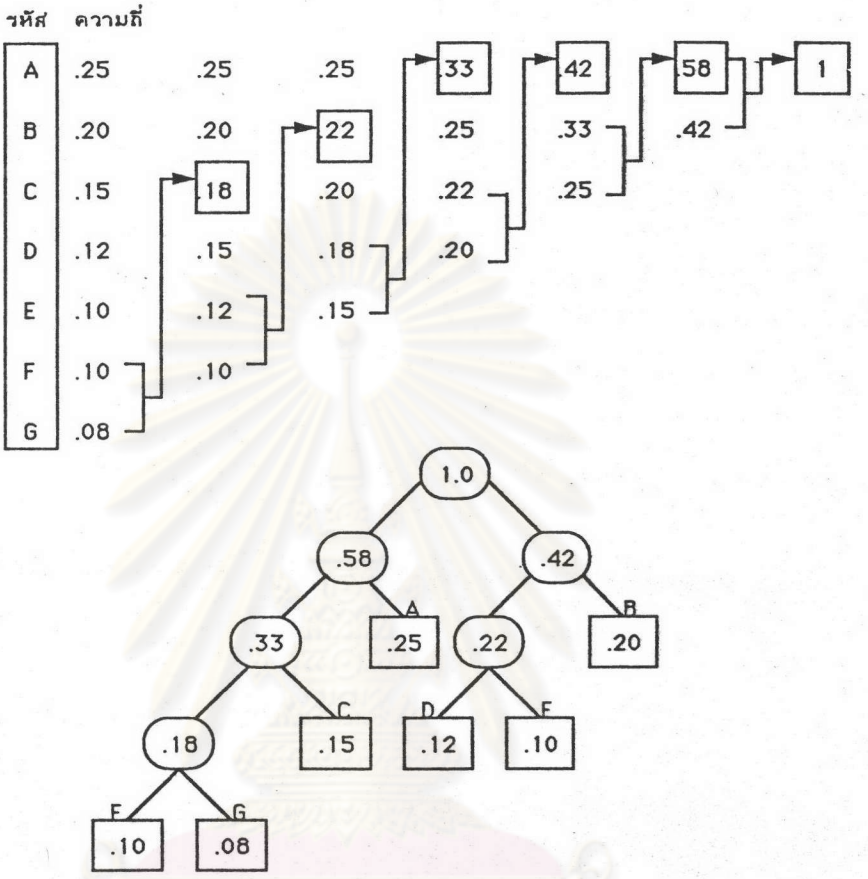
2.2.2 วิธีการของฮัฟแมน

การสร้างรหัสใหม่จะสร้างจากความน่าจะเป็นของรหัสต่าง ๆ ที่ปรากฏในข้อมูลต้นกำเนิด ตามขั้นตอนดังนี้

กำหนดให้ข้อมูลต้นกำเนิดประกอบด้วยรหัส n รหัส $\{a_1, a_2 \ldots\ldots a_n\}$ มีความน่าจะเป็น $\{w_1, w_2, \ldots, w_n\}$ โดยที่ $w_1 <= w_2 \ldots\ldots <= w_n$ นำความน่าจะเป็นของรหัสมาสร้างฟูลไบนารี (Full Binary Tree) คือ ทรีที่ทุกโหนดที่มีโหนดลูกจะต้องมีโหนดลูก 2 โหนด การสร้างทรีเริ่มจากการนำค่าความน่าจะเป็นต่ำสุด 2 ค่า คือ w_i และ w_j นำมารวมกันเป็น (w_i+w_j) ตัด w_i w_j ออกจากกลุ่มและแทรกค่า (w_i+w_j) เข้าไปในกลุ่มแทน ทำซ้ำเช่นนี้ไปเรื่อย ๆ จนกระทั่งเหลือเพียงค่าเดียวดังรูปที่ 2.5

รหัสใหม่จะได้จาก บิตทั้งหมดที่เดินจากรูทไปลีฟมาเรียงต่อกันของรหัสแต่ละรหัสในแต่ละลีฟ โดยที่โหนดพ่อไปหาโหนดลูกทางซ้ายได้ค่า 1 บิตของรหัสเป็น 1 และไปหาโหนด

ลูกทางขวาได้ค่าเป็น 0 (ในทางปฏิบัติจะสลับค่ากันก็ได้)



รูปที่ 2.5 วิธีการของฮัฟแมน

จากรูปที่ 2.5 รหัสที่ได้จากทรีดังกล่าวคือ {01, 11, 001, 100, 101, 0000, 0001} เป็นรหัสใหม่ของรหัส {A, B, C, D, E, F, G}

เมื่อพิจารณาความน่าจะเป็นและความยาวของรหัสจะมีความสัมพันธ์กันดังนี้คือ ความน่าจะเป็นของรหัสเรียงจากน้อยไปมาก คือ

$$W_1 \Rightarrow W_2 \Rightarrow W_3 \Rightarrow \ldots\ldots\ldots W_{n-1} \Rightarrow W_n$$

เมื่อสร้างรหัสใหม่ตามอัลกอริทึมฮัฟแมนได้ความยาวของรหัสใหม่เป็น

$$l(a_1) \leq l(a_2) \leq l(a_3) \ldots\ldots\ldots \leq l(a_{n-1}) \leq l(a_n)$$

นั่นคือ รหัสที่มีความน่าจะเป็นสูงจะมีรหัสใหม่ที่สั้น และรหัสที่มีความน่าจะเป็นต่ำจะมีรหัสใหม่ที่ยาว วิธีการนี้จะได้รหัสนำหน้าที่มีคุณสมบัติเป็นรหัสนำหน้าต่ำสุด และรับประกันความเหลือเฟือต่ำสุดอีกด้วย

วิธีการของฮัฟแมนจะต้องอ่านข้อมูล 2 รอบ อ่านข้อมูลรอบแรกหาความน่าจะเป็นของรหัสสำหรับการสร้างรหัสใหม่ อ่านข้อมูลรอบสองเพื่อเข้ารหัสข้อมูลด้วยรหัสใหม่ และในส่วนของข้อมูลที่เข้ารหัสจะต้องเก็บรหัสนำหน้านำหน้าข้อมูลด้วย เพื่อนำมาใช้อีกในการถอดรหัสข้อมูล เราสามารถทำได้อีกวิธีหนึ่ง คือ แทนที่จะต้องมีเก็บรหัสนำหน้าเฉพาะของข้อมูลแต่ละชุดทุกครั้ง อาจกำหนดลักษณะความน่าจะเป็นของรหัสขึ้นมาชุดหนึ่งให้เป็นตัวแทนของการกระจายข้อมูล ดังนั้น อัลกอริทึมของฮัฟแมนที่มีการกำหนดรูปแบบการกระจายของรหัสจะอ่านข้อมูลเพียงรอบเดียว นอกจากนี้ในส่วนของข้อมูลที่เข้ารหัสก็ไม่จำเป็นต้องเก็บรหัสนำหน้า เพราะโปรแกรมการเข้ารหัสและการถอดรหัสจะมีรูปแบบการกระจายของรหัสรวมอยู่ด้วย อย่างไรก็ตามถ้ารูปแบบตัวแทนไม่มีความสัมพันธ์กับข้อมูลดีพอ ประสิทธิภาพการอัดจะไม่ดีเท่าที่ควร และไม่สามารถรับประกันถึงค่าความเหลือเฟือต่ำสุด

แมคอินไทร์และพีชูรา (MacIntype and Pechura 1985) ได้ทดลองใช้วิธีการฮัฟแมนที่มีการกำหนดรูปแบบการกระจายของรหัส กับแฟ้มข้อมูลโปรแกรมภาษาคอมพิวเตอร์ 4 ภาษา จำนวนทั้งหมด 530 โปรแกรม

หลักการพื้นฐานของวิธีการของฮัฟแมนจะใช้ทรีในการสร้างรหัสใหม่ ต่อมาทานากาได้ทดลองนำไฟไนท์สเตทแมชีน (Finite-State Machine) มาใช้ในการสร้างรหัสใหม่ (Tanaka 1987 : 154-156)

จำนวนครั้งของการแทนรหัสโดยวิธีการฮัฟแมนและชานนอนฟาโน คือ $O(n)$ ครั้ง โดยที่ n เป็นจำนวนข้อมูลของข้อความต้นกำเนิด การแทนรหัสสามารถแทนรหัสโดยใช้ทรีหรืออาจสร้างตารางเก็บค่ารหัสทั้งหมดไว้ก็ได้ เวลาที่ใช้ในการเข้ารหัสและการถอดรหัสคือ $O(l)$ โดยที่ l เป็นความยาวของรหัส (คือ จำนวนทางเดินจากรูทไปยังลีฟของรหัสนั้น ๆ)

2.2.3 วิธีการเชิงคำนวณ

หลักการพื้นฐานของวิธีการเชิงคำนวณมาจากความคิดของอีไลแอส (Elias) เป็นการเข้ารหัสข้อมูลโดยใช้ช่วงค่าของเลขจำนวนจริง $[0, 1)$ โรบินได้นำวิธีการนี้มาพัฒนาต่อโดยใช้รีจิสเตอร์ความเที่ยงคงที่ (Fixed Precision Register) ในทางปฏิบัติ วิธีการนี้จะมีปัญหาเกี่ยวกับความเที่ยง (Precision) ต่อมาริทเทนได้นำไปใช้งานร่วมกับการรับส่งข้อมูล และเปลี่ยนช่วงค่าให้ใช้เลขจำนวนเต็มแทนเลขจำนวนจริง ถึงแม้ว่าได้มีการเสนออัลกอริทึมเชิงคำนวณหลายครั้งแล้วก็ตาม แต่วิธีการนี้กลับไม่แพร่หลายนัก หนังสือและบทความทางวิชาการเกี่ยวกับการอัดข้อมูลจะกล่าวถึงวิธีการนี้น้อยมาก

หลักการของวิธีการเชิงคำนวณให้รวบรวมความน่าจะเป็นของรหัสในข้อความแล้วเปลี่ยนให้อยู่ในลักษณะของความน่าจะเป็นสะสมจาก 0 ถึง 1 การเข้ารหัสข้อมูลจะใช้ความน่าจะเป็นของข้อมูลตัวที่ตามมาเป็นค่าที่ใช้สร้างช่วงค่าใหม่ที่แคบลง ถ้ารหัสใดมีความน่าจะเป็นสูงจะให้ช่วงค่าใหม่ที่แคบลงน้อยกว่ารหัสที่ความน่าจะเป็นต่ำ ดังนั้นรหัสที่ความน่าจะเป็นสูงจึงมีส่วนช่วยเพิ่มจำนวนบิตน้อยกว่าให้กับข้อความที่เข้ารหัส ข้อความจะถูกแทนค่าให้อยู่ในช่วงค่า 0 ถึง 1 ของเลขจำนวนจริง เมื่อข้อความมีความยาวเพิ่มขึ้นช่วงค่าจะยิ่งแคบลงและจำนวนบิตสำหรับการแทนช่วงค่าจะเพิ่มขึ้นตามลำดับ

ตัวอย่าง กำหนดให้ข้อความต้นกำเนิดประกอบด้วยรหัส {A, B, C, D, #} มีความน่าจะเป็น {.2, .4, .1, .2, .1} สามารถแสดงช่วงค่าและความน่าจะเป็นสะสมดังรูปที่ 2.6 รหัส A มีความน่าจะเป็นเท่ากับ .2 ของช่วงค่า [0,1) รหัส B ความน่าจะเป็นเท่ากับ .4 ของช่วงค่า [0,1) โดยมีค่าเริ่มต้นที่ .2 ถึง .6 เมื่อเริ่มลงรหัสข้อมูล ช่วงค่าเริ่มแรกเป็น [0,1)

รหัส	ความน่าจะเป็น	ความน่าจะเป็นสะสม	ช่วงค่า
A	.2	.2	[0, .2)
B	.4	.6	[.2, .6)
C	.1	.7	[.6, .7)
D	.2	.9	[.7, .9)
#	.1	1.0	[.9, 1.0)

รูปที่ 2.6 รูปแบบความน่าจะเป็นของข้อมูล

สำหรับการเข้ารหัสข้อมูล AADB# ข้อมูลตัวแรกคือ A จะลดขนาดของช่วงค่าลงเป็น [0,.2) และข้อมูลตัวที่ 2 คือ A ช่วงค่าก็จะลดลงเป็น [0,.4) (1/5 ของช่วงค่าก่อนหน้า) เมื่ออ่านข้อมูลถึง D ช่วงค่าก็ยิ่งแคบลง ได้ช่วงค่าใหม่เป็น [.028, .036) B ได้ช่วงค่าเป็น [.0296, .0328) เมื่ออ่านข้อมูลถึงตัวสุดท้าย ช่วงค่าสุดท้ายที่ได้คือ [.03248, .0328)

ขนาดของช่วงค่าสุดท้าย จะเป็นตัวกำหนดจำนวนบิตที่ต้องการใช้สำหรับการเก็บช่วงค่านี้ ถ้ากำหนดให้ S เป็นขนาดช่วงค่าสุดท้าย ดังนั้นจำนวนบิตที่ต้องการคือ $-\log S$ แต่ขนาดของช่วงค่าสุดท้ายคือ ผลคูณความน่าจะเป็นของข้อมูลทั้งหมดที่เข้ารหัส N ตัว

จะได้ $$-\log S = \sum_{i=1}^{N} \log p \text{ (ข้อมูลตัวที่ } i\text{)}$$

กำหนดให้ n เป็นจำนวนรหัสทั้งหมด

a_i เป็นรหัสตัวที่ i ของกลุ่มรหัส $\{a_1, a_2, \ldots, a_n\}$

จะได้ $$-\log S = \sum_{i=1}^{n} p(a_i) \log p(a_i)$$

ดังนั้นจำนวนบิตที่สร้างโดยการเข้ารหัสแบบเชิงคำนวณ มีค่าเท่ากับเอนโทรปีของข้อมูล ในการถอดรหัสผู้ถอดรหัสต้องรู้รูปแบบรหัสที่ใช้เข้ารหัส (ความน่าจะเป็น) และค่า 1 ค่า (ช่วงค่าสุดท้าย) ซึ่งกำหนดไปโดยผู้เข้ารหัส จากตัวอย่าง ช่วงค่าสุดท้ายที่ผู้ถอดรหัสได้รับคือ .0325 เนื่องจากค่าอยู่ระหว่าง [0, .2) ดังนั้นอักษรตัวแรกที่ถอดรหัสได้คือ A ต่อจากนั้นผู้ถอดรหัสก็ลดช่วงค่าลงอีกโดยวิธีการเช่นเดียวกับการเข้ารหัส โดยพิจารณาจากรหัสที่อยู่ในช่วงค่าย่อย ซึ่งแบ่งเป็นรหัส A ในช่วงค่า [0, .04) รหัส B ในช่วงค่า [.04, .12) รหัส C ในช่วงค่า [.12, .14) รหัส D ในช่วงค่า [.14, .18) และรหัส # ในช่วงค่า [.18, .2] ดังนั้น ค่า .0325 อยู่ในช่วงค่า [0, .04) ทำให้ผู้ถอดรหัสสามารถทราบได้ว่าข้อมูลตัวต่อมา คือ A การถอดรหัสจะทำซ้ำเช่นนี้จนได้ข้อมูลที่เข้ารหัสครบทุกตัว

วิธีการเชิงคำนวณมีสิ่งสำคัญที่ต้องพิจารณาถึงหลายประการ คือ

1. การที่ข้อมูลที่เข้ารหัสเป็นรหัสที่เกิดซ้ำอยู่ที่รหัสรหัสเดียว ในลักษณะเช่นนี้ ช่วงค่าสุดท้ายที่ได้จะเป็น 0.0 จากรูปแบบข้อมูลดังรูปที่ 2.6 ผู้ถอดรหัสสามารถถอดรหัสข้อมูลได้เป็น A, AA, AAA, AAAA, AAAAA, ... ผู้ถอดรหัสไม่สามารถรู้ได้ว่าจะหยุดถอดรหัสเมื่อใด เราสามารถแก้ไขได้ 2 วิธี วิธีแรกให้ส่งขนาดของข้อมูลที่เข้ารหัสไปด้วย อีกวิธีหนึ่งให้ส่งรหัสพิเศษเป็นตัวบอกว่าจบข้อความเข้ารหัสแล้ว

2. วิธีการเชิงคำนวณ ควรนำไปใช้ในงานที่มีการรับส่งข้อมูลด้วย เพราะอัลกอริทึมการเข้ารหัสของวิธีการนี้จะไม่ส่งข้อมูลใด ๆ ไปเลยจนกว่าจะทำจนจบได้ช่วงค่าสุดท้าย ดังนั้นในขณะที่เข้ารหัสช่วงค่าจะแคบลงเรื่อย ๆ ถ้าข้อมูลมีขนาดใหญ่มากจะทำให้ค่าซ้ายและค่าขวาเป็นค่าเดียวกัน

3. ปัญหาที่สำคัญของวิธีการนี้คือ ความเที่ยงเมื่อข้อมูลเข้าเพิ่มขึ้นช่วงค่ามีขนาดเล็กลงมาก ๆ จึงต้องการความเที่ยงที่ไม่มีขอบเขตจำกัด ได้มีการแก้ปัญหานี้โดยการนำรีจิสเตอร์ความเที่ยงคงที่มาใช้ในการแก้ปัญหาเล็กเกินไป (Underflow) และการล้น (Overflow) แต่จะทำให้ประสิทธิภาพของการอัดไม่ได้เท่าเอนโทรปี

2.2.4 วิธีการของอแดปทีฟฮัฟแมน

วิธีการของอแดปทีฟฮัฟแมน คิดขึ้นครั้งแรกในปี ค.ศ. 1973 โดยฟอลเลอร์ ต่อมาในปี ค.ศ. 1978 กัลลาเจอร์ ได้กำหนดแนวคิดเกี่ยวกับหลักการพื้นฐานของวิธีการของอแดปทีฟฮัฟแมนคือ การใช้คุณสมบัติโหนดพี่น้อง (Sibling Property) ซึ่งกำหนดว่า ไบนารีทรีของรหัส (Binary Code Tree) จะมีคุณสมบัติโหนดพี่น้อง เมื่อแต่ละโหนดที่มีโหนดพี่น้องสามารถเรียงลำดับได้ในลักษณะไม่เพิ่มขึ้นของน้ำหนัก (Gallager 1978 : 668-674)

ต่อมาคนุท ได้นำอัลกอริทึมเริ่มแรกมาพัฒนาต่อ และตั้งชื่ออัลกอริทึมนี้ว่าอัลกอริทึมเอฟจีเค สำหรับอีกอัลกอริทึมหนึ่งมีชื่อว่า อัลกอริทึมวี เกิดขึ้นในปี ค.ศ. 1987 โดยวิทเตอร์ และในปี ค.ศ. 1989 วิทเตอร์ได้นำไปพัฒนาต่อสำหรับการนำไปใช้ในการรับส่งข้อมูล

วิธีการของอแดปทีฟฮัฟแมนมีการทำงานเพียงรอบเดียว รหัสที่สร้างโดยวิธีนี้จะเปลี่ยนแปลงตลอดเวลาเพื่อให้ได้ผลดีที่สุดสำหรับการเข้ารหัส หลักการสำคัญคือ ผู้เข้ารหัสต้องเรียนรู้ลักษณะของข้อมูลต้นกำเนิดไปพร้อม ๆ กับการเข้ารหัส ในการถอดรหัสผู้ถอดรหัสจะเรียนรู้เช่นเดียวกับผู้เข้ารหัสเพื่อให้ฮัฟแมนทรีมีการปรับปรุงรูปร่างของทรีในลักษณะที่ตรงกันกับลักษณะที่อยู่กับผู้เข้ารหัส ในขณะเดียวกันก็ถอดรหัสข้อมูลไปด้วย วิธีการนี้ไม่ต้องมีส่วนของรหัสนำหน้ารวมไปกับข้อมูลที่เข้ารหัส

กำหนดให้มี k รหัสของรหัสทั้งหมด n รหัสในข้อความ ในตอนแรกรหัสทรีจะประกอบด้วยโหนดโหนดเดียว เรียกว่า 0-node เป็นโหนดพิเศษสำหรับเป็นตัวแทนรหัส (n-k) รหัสที่ยังไม่ได้ใช้ สำหรับข้อมูลแต่ละตัวที่ส่งเข้ามาต้องมีการปรับปรุงน้ำหนักและคำนวณค่าภายในรหัสทรีใหม่เพื่อให้คงคุณสมบัติโหนดพี่น้อง เมื่อมีข้อมูลส่งเข้ามาแล้ว t ตัวเป็นรูปแบบของรหัส k รหัส ($k < n$) รหัสทรีจะมีลักษณะเป็นฮัฟแมนทรีตามปกติประกอบด้วย k+1 ลีฟ โดยที่ k ลีฟเป็นของรหัส k รหัสและอีก 1 ลีฟเป็นของ 0-node เมื่อการส่งข้อมูล a_{t+1} เข้ามา ถ้ามีค่าตรงกับรหัสตัวใดตัวหนึ่งใน k รหัสที่มีอยู่ในทรี ก็ให้เข้ารหัสข้อมูล a_{t+1} ด้วยรหัสปัจจุบัน หลังจากนั้นจะต้องคำนวณค่าภายในรหัสทรีใหม่ และเพื่อให้คงคุณสมบัติโหนดพี่น้องจะต้องปรับปรุงรูปร่างของรหัสทรีด้วย แต่ถ้า a_{t+1} มีค่าไม่ตรงกับรหัสที่มีอยู่ ดังนั้น a_{t+1} คือ รหัสใหม่ ให้แยก 0-node ออกเป็น 2 โหนด สำหรับรหัสใหม่และสำหรับ 0-node แล้วคำนวณค่าน้ำหนักพร้อมทั้งปรับปรุงรูปร่างของรหัสทรีใหม่ด้วย

อัลกอริทึมวีเป็นอัลกอริทึมที่ได้พัฒนาจากอัลกอริทึมเอฟจีเค ส่วนที่พัฒนาเพิ่มขึ้นคือ การเปลี่ยนตำแหน่งของโหนดขึ้นไปข้างบนกำหนดให้มีการเลื่อนขึ้นได้ 1 ระดับเท่านั้น วิธีการนี้จึงรับประกันค่าความสูงต่ำสุดของทรี (Minimum Height) และค่าต่ำสุดความยาวทางเดินภายนอก (External Path Length) อัลกอริทึมในการปรับปรุงรูปร่างทรีมีความซับซ้อนมาก ซึ่งสามารถศึกษารายละเอียดได้จากอัลกอริทึมวีของวิทเตอร์ (Vitter 1989 : 158-167)

2.2.5 วิธีการของบีเอสทีดับบลิว

เป็นหลักการของเบนลี สลีเตอร์ ทาจาน และไว เรียกว่า อัลกอริทึม บีเอสทีดับบลิว อัลกอริทึมนี้จะสร้างลิสต์ที่มีการจัดรูปแบบเองเป็นโครงสร้างข้อมูล ที่ใช้เก็บรหัส ถ้ารหัสใดมีการนำไปใช้บ่อยจะถูกย้ายไปอยู่ต้นลิสต์ แต่ถ้ารหัสใดการนำไปใช้งานน้อย ก็จะถูกเลื่อนไปอยู่ท้ายลิสต์ ดังนั้นรหัสที่อยู่ต้นลิสต์จะเข้ารหัสด้วยจำนวนบิตน้อยกว่ารหัสที่อยู่ท้ายลิสต์

วิธีการของอัลกอริทึมบีเอสทีดับบลิวมีขั้นตอนดังนี้ ตอนแรกลิสต์จะว่าง ไม่มีรหัสใดเก็บอยู่ เมื่ออ่านข้อมูลจากข้อความต้นกำเนิด a_i นำไปตรวจสอบดูว่ามีรหัส a_i อยู่ในลิสต์หรือยัง

- ถ้ามีรหัสของ a_i อยู่ในลิสต์ตำแหน่งที่ i ให้ส่งตำแหน่งนั้นมาเข้ารหัสแล้วย้ายรหัสจากตำแหน่งที่ i มาไว้ที่ตำแหน่งที่ 1 และรหัสตำแหน่งที่ 1, 2, ..., i-1 จะต้องถูกย้ายไปไว้ที่ตำแหน่ง 2, 3, ..., i หลักการนี้เรียกว่า การย้ายไปหน้า (Move to Front)
- ถ้าไม่มีรหัสของ a_i ในลิสต์ ซึ่งขณะนี้ลิสต์ประกอบด้วยรหัสทั้งหมด k รหัส ดังนั้น a_i จะเป็นรหัสตัวที่ k+1 ให้เพิ่มรหัสตัวที่ k+1 เข้าต้นลิสต์ที่ตำแหน่งที่ 1 และย้ายรหัสตำแหน่งที่ 1, 2, ..., k ไปไว้ที่ตำแหน่ง 2, 3, ..., k+1 สำหรับการเข้ารหัสก็จะส่งลำดับที่ของรหัสไปพร้อมกับรหัสนั้น คือ $(k+1)(a_i)$

ตัวอย่าง มีข้อความที่ต้องการจะเข้ารหัสดังนี้ "abcadeabfd" จำนวนรหัสเริ่มต้นเป็น 0 และลิสต์เริ่มต้นเป็นลิสต์ว่าง

อ่านข้อมูลตัวแรกคือ a เนื่องจาก a เป็นรหัสใหม่ดังนั้นต้องเพิ่มรหัส a เข้าไปในลิสต์และเข้ารหัสข้อมูลได้ค่าเป็น 1a ต่อไปอ่านข้อมูลตัวที่ 2 คือ b b เป็นรหัสใหม่เช่นกัน จึงเพิ่มรหัส b เข้าทางต้นลิสต์ และเข้ารหัสข้อมูลได้ค่าเป็น 2b สังเกตุว่าเมื่อเพิ่มค่า b เข้าลิสต์ทางด้านหน้า ตำแหน่งของ a ในลิสต์จะถูกเปลี่ยนเป็น 2 ต่อไปอ่านข้อมูลตัวที่ 3 คือ c ซึ่งเป็นรหัสใหม่ ให้เพิ่มรหัส c เข้าทางต้นลิสต์และเข้ารหัสข้อมูลได้ค่าเป็น 3c ตำแหน่งของ a ในลิสต์จะถูกเปลี่ยนเป็น 3 ตำแหน่งของ b ในลิสต์จะถูกเปลี่ยนเป็น 2 ดังรูปที่ 2.7

อ่านข้อมูลตัวที่ 4 คือ a และ a เป็นรหัสที่มีอยู่แล้วในลิสต์ที่ตำแหน่งที่ 3 ดังนั้นข้อมูล a จะเข้ารหัสด้วยตำแหน่งของ a ในลิสต์ และลิสต์ต้องมีการจัดรูปแบบตัวเองใหม่โดยการย้ายรหัส a จากตำแหน่งที่ 3 มาไว้ที่ตำแหน่งที่ 1 รหัส c และรหัส b จะถูกย้ายไปตำแหน่งที่ 2 และ 3 ตามลำดับดังรูปที่ 2.8

เมื่ออ่านข้อมูลจนจบจะได้ข้อมูลที่เข้ารหัสดังนี้ 1a2b3c34d5e346f5

ตำแหน่ง	1	2	3
ลิสต์	c	b	a
ข้อมูลเข้ารหัส	1a2b3c		

รูปที่ 2.7 การเข้ารหัสและการสร้างลิสต์โดยอัลกอริทึมบีเอสทีดับบลิว

ตำแหน่ง	1	2	3
ลิสต์	a	c	b
ข้อมูลเข้ารหัส	1a2b3c3		

รูปที่ 2.8 การย้ายไปหน้าของรหัส a ในลิสต์

2.2.6 วิธีการของแลมเพล-ซิฟ

เป็นวิธีการอัดข้อมูลที่มีสร้างตารางรหัสที่ได้จากการเรียนรู้จากข้อมูลโดยหลักการกระจายอิสระ มีหลักการดังนี้คือ ข้อมูลที่นำมาเข้ารหัสจะถูกอ่านในลักษณะกลุ่มข้อมูล 1, 2, ... ตัว แต่ความยาวไม่เกิน L_1 ตัว กลุ่มข้อมูลจะถูกกำหนดให้เป็นรหัสใหม่เก็บลงในตารางรหัสเมื่อกลุ่มข้อมูลนั้นมีค่าไม่ซ้ำกับกลุ่มข้อมูลใด ๆ ในตารางโดยที่รหัสใหม่มีความยาวได้ไม่เกิน L_2 บิต รหัสที่ใช้ในการเข้ารหัสเป็นรหัสแบบแปรได้-คงที่คือ การเข้ารหัสข้อมูลจะแทนค่าข้อมูลขนาดไม่แน่นอนแต่ความยาวไม่เกิน L_1 ตัวด้วยรหัสขนาดคงที่ยาว L_2 บิต

วิธีการของแลมเพล-ซิฟสามารถลดความเหลือเฟือได้หลายด้าน คือการเกิดบ่อยของรหัสการเกิดรหัสเดียวซ้ำ ๆ กัน หรือกลุ่มของรหัสที่มักจะอยู่ร่วมกัน วิธีการนี้เป็นวิธีการที่มีประสิทธิภาพมากแต่จะให้ประสิทธิภาพต่ำในช่วงแรก ดังนั้นถ้าข้อมูลต้นกำเนิดมีขนาดไม่ใหญ่พอ

อาจได้ประสิทธิภาพการอัดไม่ดีเท่าที่ควร การสร้างตารางรหัสที่ใช้ในการเข้ารหัสข้อมูลจะใช้คุณสมบัติการนำหน้า (Prefix Property) หมายความว่า กลุ่มข้อมูลที่ยาวที่สุดที่ตรงกับข้อมูลในตารางรหัสจะเป็นส่วนหน้าของรหัส (Prefix) และข้อมูลอีก 1 ตัวจะเป็นส่วนท้ายของรหัส (Suffix)

วิธีการของแลมเพล-ซิฟ มีสิ่งสำคัญที่ต้องพิจารณาคือ วิธีการเก็บรหัสและการค้นหารหัสในตาราง เวลได้เสนอแนวความคิดเพิ่มเติมเกี่ยวกับการกำหนดให้ใช้ตำแหน่งของตารางเป็นค่าของรหัส รวมทั้งนำวิธีการแฮช (Hashing) มาช่วยในการเก็บรหัสและค้นหารหัสจากตาราง และได้เปลี่ยนวิธีการเข้ารหัสเป็นแบบแปรได้-แปรได้ อัลกอริทึมที่พัฒนาใหม่นี้เรียกว่า อัลกอริทึมแอลแซดดับบลิวเป็นอัลกอริทึมที่ได้รับการยอมรับอย่างมาก มีการนำไปใช้งานอย่างแพร่หลาย

ตัวอย่าง จากข้อมูล ababccabcc สร้างตารางรหัสได้ดังรูปที่ 2.9 ในตารางกำหนดให้มีรหัสเริ่มต้น 3 รหัส คือ a, b, c

กลุ่มข้อมูล	รหัส	ตำแหน่ง
a	a	1
b	b	2
c	c	3
ab	1b	4
ba	2a	5
abc	4c	6
cc	3c	7
ca	3a	8
abcc	6c	9

รูปที่ 2.9 การสร้างรหัสใหม่ที่ได้จากการอ่านข้อมูล ababccabcc

อ่านข้อมูลตัวแรกคือ a นำรหัส a ไปตรวจสอบกับตารางรหัส ปรากฏว่ารหัสนี้มีแล้วมีค่ารหัสเท่ากับ 1

อ่านข้อมูลตัวที่ 2 คือ b ได้กลุ่มข้อมูลเป็นรหัส 1b นำไปตรวจสอบกับตารางรหัสปรากฏว่ารหัสนี้ยังไม่มี ดังนั้นให้นำรหัส 1b เก็บลงตารางรหัสตำแหน่งที่ 4 นำส่วนหน้าของรหัส (คือ 1) เข้ารหัส และเปลี่ยนส่วนหลังของรหัส (คือ b) เป็นส่วนหน้าของรหัส

อ่านข้อมูลตัวที่ 3 คือ a ได้กลุ่มข้อมูลเป็นรหัส 2a นำไปตรวจสอบกับตารางรหัสปรากฏว่ารหัสนี้ยังไม่มี ดังนั้นให้นำรหัส 2a เก็บลงตารางรหัส ตำแหน่งที่ 5 นำส่วนหน้าของรหัส (คือ 2) เข้ารหัสและเปลี่ยนส่วนหลังของรหัส (คือ a) เป็นส่วนหน้าของรหัส

อ่านข้อมูลตัวที่ 4 คือ b ได้กลุ่มข้อมูลเป็นรหัส 1b นำไปตรวจสอบกับตารางรหัสปรากฏว่ารหัสนี้มีแล้วมีค่ารหัสเท่ากับ 4

อ่านข้อมูลตัวที่ 5 คือ c ได้กลุ่มข้อมูลเป็นรหัส 4c นำไปตรวจสอบกับตารางรหัสปรากฏว่ารหัสนี้ยังไม่มี ดังนั้นให้นำรหัส 4c เก็บลงตารางรหัสตำแหน่งที่ 6 นำส่วนหน้าของรหัส (คือ 4) เข้ารหัส และเปลี่ยนส่วนหลังของรหัส (คือ c) เป็นข้อมูลตัวแรก ให้ทำซ้ำขั้นตอนเช่นนี้จนจบข้อมูลจะได้ข้อมูลที่เข้ารหัสแล้วดังรูปที่ 2.10

ข้อมูล	a	b	a	b	c	c	a	b	c	c
ข้อมูลเข้ารหัส	1	2	4	3	3	6	3			
รหัสที่เพิ่มเข้าไปในตาราง	4	5	6	7	8	9				

รูปที่ 2.10 การเข้ารหัสข้อมูล

2.3 อัลกอริทึมที่มีการนำไปใช้ในปัจจุบัน

ในโปรแกรมจัดการระบบยูนิกซ์ (Unix Operating System) มีโปรแกรมที่ใช้สำหรับการอัดข้อมูล 3 โปรแกรม คือโปรแกรมคอมเพลส (Compress Program) ใช้อัลกอริทึมแอลแซดดับบลิว โปรแกรมคอมแพค (Compact Program) ที่ใช้อัลกอริทึมอแดปทีฟฮัฟแมน และโปรแกรมแพค (Pact Program) ซึ่งใช้อัลกอริทึมฮัฟแมน โปรแกรมคอมเพลสเป็นโปรแกรมให้ประสิทธิภาพการอัดดีที่สุด (Nemeth. Synder and Seebass 1989 : 461)

นอกจากนี้ได้มีการนำวิธีการอัดข้อมูลไปพัฒนาเป็นโปรแกรมสำเร็จรูป สำหรับใช้จัดเก็บแฟ้มข้อมูล (File Archives) ในเครื่องไมโครคอมพิวเตอร์ ได้แก่ โปรแกรมจัดเก็บเอกสาร

พีเคอาค (PKARC 1987) ซึ่งพัฒนาโดยบริษัทพีเคแวร์ (PKWARE) มีวิธีการอัดข้อมูลหลาย ๆ วิธี วิธีแรกเรียกว่า สควีชชิง (Squeezing) เป็นการอัดข้อมูลโดยอัลกอริทึมฮัฟแมน วิธีที่สองมีชื่อว่า ครันชิง (Crunching) เป็นวิธีการอัดข้อมูลโดยใช้อัลกอริทึมแอลแซดดับบลิว ที่มีขนาดรหัสสูงสุด 12 บิต สำหรับการอัดข้อมูลอีกวิธีหนึ่งมีชื่อว่า แพคกิง (Packing) เป็นการอัดข้อมูลแบบนับความยาว

ต่อมาได้พัฒนาโปรแกรมจัดเก็บแฟ้มเอกสารพีเคซิพ (PKZIP 1990) ขึ้น ซึ่งเป็นโปรแกรมที่ใช้วิธีการใหม่เรียกว่า อิมโพลดดิง (Imploding) โดยมีหลักการจากอัลกอริทึมของชานนอน-ฟาโน อีกวิธีการหนึ่งใช้อัลกอริทึมแอลแซดดับบลิวที่มีขนาดรหัสสูงสุด 13 บิต เรียกว่า ชิงค์กิง (Shrinking) นอกจากนี้จะใช้วิธีการที่เรียกว่า รีดิวชิง (Reducing) โดยใช้อัลกอริทึมรีดิวชิง (Reducing Algorithm) ซึ่งประกอบด้วย 2 อัลกอริทึมย่อย อัลกอริทึมแรกนับรหัสที่เกิดซ้ำ อัลกอริทึมที่สองจึงจะทำการอัดข้อมูลที่ได้จากอัลกอริทึมแรก โดยใช้ค่าความน่าจะเป็นใช้ในการอัดข้อมูล