เทคนิคการลดขนาดข้อมูล

DATA COMPRESSION TECHNIQUE

โดย

นายภานุ ชุมสิทธิ์

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2535

เทคนิคการลดขนาดข้อมูล

นายกานุ ชุมสิทธิ์

อาจารย์ที่ปรึกษา

อาจารย์เกรียงไกร วงศ์โรจน์ภรณ์

ปีการศึกษา 2535

บทคัดย่อ

วิทยานิพนธ์นี้เป็นการศึกษาเทคนิคการลดขนาดข้อมูลของโปรแกรม PKZIP โดยจำลองการทำงานและเทคนิคของโปรแกรม PKZIP แยกออกเป็นส่วนๆพร้อมคำอธิบาย โฟลว์ชาร์ท โปรแกรมและผลของโปรแกรม โดยอัลกอริทึมของ PKZIP ดัดแปลงมาจากอัลกอริทึมของ LZW , Huffman / Shannon-fano coding และ Sliding Dictionary โดยเพิ่มเทคนิคพิเศษบางอย่างทำให้อัตราการลดขนาด(compression ratio) ดีขึ้น โปรแกรมภาษา C ที่เขียนขึ้นจากอัลกอริทึมของ PKZIP มีความเข้ากันได้กับโปรแกรม PKZIP เป็นอย่างดี

DATA COMPRESSION TECHNIQUE

Mr.Phanu Choomsit

Advisor:

Mr.Kriengkai Vonglodjanaporn

1992

Abstract

The objective of this project is to study the data compression technique from the PKZIP program. The study of this program can be divided into three parts, each part describes in flowchart form of the subprogram and its result. The used algorithm in PKZIP is modified from LZW , Huffman / Shonnon-Fano coding and sliding dictionary. It also includes some special technique which is the better compression data. The data compression program of PKZIP is rewriten with C language which can compress and de-compress data from PKZIP file format.

สารบัญ

บทที่ 1
บทนำ

เทคโนโลยีทางด้านการลดขนาดข้อมูล มีการนำไปใช้เป็นประโยชน์หลายด้าน เช่น

1 ทำให้การส่งข้อมูลได้เร็วขึ้น เมื่อมีการลดขนาดข้อมูลก่อนการส่งไปยังด้านรับ จะทำให้เสมือนกับว่า สามารถส่งข้อมูลได้เร็วขึ้น หรือส่งข้อมูลได้มากขึ้นในเวลาเท่าเดิม เช่น เมื่อข้อมูลผ่านการลดขนาดแล้วมีขนาดเหลือเพียงครึ่งหนึ่ง เวลาที่ใช้ในการส่งลดลงเหลือครึ่งหนึ่งจากปกติ หรืออาจคิดว่าถ้าส่งข้อมูลในเวลาที่เท่าเดิม แต่สามารถส่งข้อมูลมากขึ้นเป็นสองเท่า

2 ทำให้พื้นที่ว่างของอุปกรณ์เก็บข้อมูลมีมากขึ้น เพราะข่าวสารและข้อมูลมีมากขึ้นกว่าก่อน อุปกรณ์เก็บข้อมูลที่เข้าใจว่าเพียงพอแก่การเก็บข้อมูล เริ่มไม่เป็นความจริง ทำให้เกิดความจำเป็นต้องจัดหาอุปกรณ์เก็บข้อมูลมาเพิ่ม เพียงแต่หากใช้โปรแกรมลดขนาดข้อมูล ช่วยในการลดข้อมูลก่อนจัดเก็บลงอุปกรณ์เก็บข้อมูลเป็นหนทางหนึ่งที่ช่วยให้ยืดระยะเวลาในการจัดหาอุปกรณ์เก็บข้อมูลมาเพิ่มได้

3 ลดปัญหาในการพัฒนาระบบที่มีความต้องการข้อมูลมากๆ เช่นในระบบมัลติมีเดีย(Multi meadia system) ต้องมีการส่งข้อมูลทีละมากๆ เพื่อสร้างภาพกราฟิกเคลื่อนไหวได้เหมือนจริง การลดขนาดข้อมูล ทำให้ลดปัญหาของการถ่ายเทข้อมูลเป็นปริมาณมากๆ ปัจจุบันการลดขนาดข้อมูล ของภาพกราฟิก ใช้เทคนิคของ JPEGซึ่งใช้อัลกอริทึมของ Discrete Cosine Transform (DCT) สำหรับลดขนาดข้อมูล ซึ่งมีประสิทธิภาพในการลดขนาดข้อมูลดีมาก โดยที่ไม่ทำให้คุณภาพของภาพเสียไปมาก

จะเห็นได้ว่าเทคนิคการลดขนาดข้อมูลสามารถนำไปใช้ให้เกิดประโยชน์ ได้หลายทาง โดยส่วนใหญ่จะเกี่ยวข้องกับการลดพื้นที่ในการเก็บข้อมูล

โปรแกรมลดขนาดข้อมูลเริ่มมีการใช้ครั้งแรก ราวปี ค.ศ. 1980 ในระบบยูนิกซ์ (UNIX)มีการใช้โปรแกรม COMPACT ซึ่งใช้อัลกอริทึมของ ฮัฟแมนแบบปรับตัวได้ (Adaptive Huffman) ซึ่งให้ผลการลดขนาดน่าพอใจแต่ทำงานช้าต่อมาได้มีการพัฒนาโปรแกรมชื่อ COMPRESS บนยูนิกซ์ขึ้นมาโดยใช้อัลกอริทึมของ LZW ซึ่งให้ประสิทธิภาพการลดข้อมูลได้ดีกว่าและเร็วกว่าทำให้ COMPRESSเป็นที่นิยมกันอย่างแพร่หลาย ไม่นานนักมีโปรแกรมชื่อ ARC ซึ่งเลียนแบบโปรแกรม COMPRESS บนยูนิกซ์และใช้อัลกอริทึมของฮัฟแมนอีกด้วย ARC เป็นที่นิยมใช้กันอย่างกว้างขวาง จนกระทั่ง โปรแกรมPKZIP เสนอโปรแกรมที่เร็วมากและลดขนาดได้มาก ทำให้ PKZIP เป็นที่นิยมขึ้นมาแทน ARC อย่างรวดเร็ว การศึกษา PKZIP จึงเป็นการศึกษาโปรแกรมที่มีประสิทธิภาพ ทางด้านการลดข้อมูล ที่ดีที่สุดขณะนี้

รายงานฉบับนี้แบ่งออกเป็น 4 บท โดยที่บทที่ 2 กล่าวถึงทฤษฎีและหลักการ ซึ่งเป็นพื้นฐานบางส่วนของทฤษฎีข่าวสาร เช่นคำจำกัดความของข่าวสาร ข่าวสารเฉลี่ยและอธิบายถึงหลักการพื้นฐานของอัลกอริทึมการลดขนาดข้อมูล 3 ชนิดที่ใช้ในโปรแกรมที่ศึกษา คือ ฮัฟแมน , LZW และ อัลกอริทึมแบบพจนานุกรมแบบเลื่อนได้ บทที่ 3 กล่าวถึงผลการศึกษา เทคนิคการลดขนาดข้อมูลของโปรแกรม PKZIP บทนี้จะอธิบายโครงสร้างแฟ้มข้อมูลของ Zipfile ว่าจัดเรียงแฟ้มข้อมูลอย่างไร อธิบาย

เทคนิคการเลือกใช้อัลกอริทึมในการลดขนาดข้อมูล อธิบายอัลกอริทึมทั้ง 2 ที่ใช้ใน PKZIP พร้อมด้วย โฟลว์ชาร์ท บทที่ 5 กล่าวถึง วิเคราะห์ผลการศึกษา บทนี้รวบรวมผลการเปรียบเทียบคุณสมบัติต่างๆของโปรแกรม เช่น อัตราการลดขนาดข้อมูล (compression ratio) อัตราเร็วของการลดขนาดข้อมูล (compression speed) อัตราเร็วของการขยายขนาดข้อมูลกลับ (expandsion speed)

บทที่ 2
ทฤษฎีและหลักการ

เทคนิคการลดขนาดข้อมูล คือ เทคนิคที่ใช้ในการลดปริมาณของพื้นที่ที่ใช้เก็บข้อมูล ทำให้ลดเวลาการส่งข้อมูล หรือ ลดแบนด์วิดธ์ที่ต้องใช้ในการส่งข้อมูลได้ เทคนิคการลดขนาดข้อมูลมี 2 ชนิดกว้างๆ คือ

1. เทคนิคที่ไม่สามารถนำข้อมูลกลับคืนมาได้ เหมือนเดิมทุกประการ ซึ่งมีชื่อเรียกต่างๆกัน เช่น Entropy reduction , Lossy compression

2. เทคนิคที่สามารถนำข้อมูลกลับคืนมาได้ เหมือนเดิมทุกประการ มีชื่อเรียกต่างๆกัน เช่น Noiseless coding , Redundancy reduction ,Lossless compression

1 เทคนิคที่ไม่สามารถนำข้อมูลกลับคืนมาได้เหมือนเดิมทุกประการ

การลดขนาดข้อมูลเกิดขึ้น เนื่องจากการตัดทอนข่าวสาร(Information)บางส่วนทิ้งไปในระดับที่ยอมรับได้ ข่าวสารที่ทิ้งไปไม่สามารถนำกลับคืนมาได้ ทำให้ไม่สามารถนำข้อมูลกลับมาได้เหมือนเดิมทุกประการ เช่น การส่งพัลส์ 1 ลูก จากเครื่องส่ง ผ่านสายส่งไปยังเครื่องรับ ถ้าต้องการให้เครื่องรับ รับพัลส์ได้เหมือนเดิมทุกประการ ต้องใช้แบนด์วิดธ์กว้างถึงอินฟินิตี้ เพื่อส่งสเปคตรัมทุกๆฮาโมนิคผ่านสายส่งไปยังเครื่องรับ แต่ในทางปฏิบัติมิได้ส่งสเปคตรัมทุกๆฮาโมนิค แต่จะส่งเพียงฮาโมนิคแรกๆเพียงไม่กี่ฮาโมนิค ซึ่งเป็นสเปคตรัม ที่พลังงานของพัลส์ส่วนใหญ่สะสมอยู่ ทางเครื่องรับสามารถรับพัลส์ได้ มีรูปร่างใกล้เคียงกับพัลส์จริงๆเท่านั้น แต่เพียงพอที่จะแยกแยะได้ว่าเป็นพัลส์หรือไม่ การลดขนาดข้อมูลเกิดขึ้น เนื่องจากการจำกัดสเปคตรัมให้เหลือเพียงพอที่เครื่องรับสามารถแยกแยะได้ นั่นคือ ใช้แบนด์วิดธ์ในการส่งพัลส์ลดลงเทคนิคนี้เหมาะสำหรับการใช้งานกับข้อมูลประเภทภาพและเสียง ซึ่งยอมให้การตัดทอนข้อมูลบางส่วนได้

2 เทคนิคที่สามารถนำข้อมูลกลับคืนมาได้เหมือนเดิมทุกประการ

เราอาจคิดว่า ข้อมูล (data) ประกอบไปด้วยข่าวสาร และส่วนที่ซ้ำ (Redundancy) การลดขนาดข้อมูลทำได้โดยการตัดส่วนที่ซ้ำหรือลดส่วนที่ซ้ำให้เหลือน้อยที่สุด ซึ่งส่วนที่ซ้ำสามารถนำกลับคืนมาได้ นั่นคือข้อมูลสามารถนำกลับคืนมาได้เหมือนเดิมทุกประการ เช่นถ้าต้องการส่งข้อมูล '00000012400001111' ไปให้เครื่องรับ โดยกำหนดให้ส่งข้อมูลครั้งละตัวตามปรกติต้องส่งข้อมูล 17 ครั้ง แต่เราอาจนำข้อมูลส่วนที่ซ้ำมาเข้ารหัส โดยกำหนดข้อมูลพิเศษขึ้นมาตัวหนึ่งเพื่อ เป็นสัญลักษณ์บอกกับเครื่องรับว่า ถ้าได้รับข้อมูลพิเศษตัวนี้ข้อมูลที่ตามมา 2 ตัวคือ จำนวนข้อมูลที่ซ้ำและค่าของข้อมูลตามลำดับ ถ้ากำหนดให้ข้อมูลพิเศษคือ A ข้อมูล '000000'เข้ารหัสเป็น 'A60' ข้อมูล '0000' เข้ารหัสเป็น 'A40' ข้อมูล '1111' เข้ารหัสเป็น 'A41' ข้อมูลทั้งหมดเมื่อผ่านการเข้ารหัสแล้วมีลักษณะดังนี้ 'A60124A40A41' ซึ่งส่งข้อมูลเพียง 12 ครั้ง เครื่องรับก็สามารถ รับข้อมูลได้เหมือนเดิมทุกประการ และประหยัดเวลาที่ใช้ส่งได้ 5 ครั้ง การเข้ารหัสข้อมูลในลักษณะที่เรียกว่าRun Length Encoding

เทคนิคนี้เหมาะสำหรับข้อมูลประเภทฐานข้อมูลและข้อมูลประเภทตัวอักษร เป็นต้น

การลดขนาดข้อมูลสามารถอธิบายได้โดย ทฤษฎีข่าวสาร(Information Theory) มีศัพท์หลายคำที่เกี่ยวข้องกับเทคนิคการลดขนาดข้อมูลเช่น ข่าวสาร ข่าวสารเฉลี่ย (entropy) และส่วนที่ซ้ำ

2.1 ข่าวสาร ปริมาณข่าวสารเฉลี่ย และ ส่วนที่ซ้ำ

ข่าวสาร (Information) ในความหมายของ ทฤษฎีข่าวสาร คือปริมาณความไม่แน่นอนของเหตุการณ์ (measure of uncertainty) เช่น สำหรับเหตุการณ์ (event)ที่เกิดค่อนข้างแน่นอน เหตุการณ์นั้นมีข่าวสารน้อย สำหรับเหตุการที่เกิดไม่มากครั้ง แสดงว่าเหตุการณ์นั้นมีข่าวสารมาก

การวัดปริมาณข่าวสาร ของเหตุการณ์ คำนวณได้จาก

$$I_i = - \log_2 P_i \text{ บิท}$$

โดยที่ P_i คือความน่าจะเกิด(probability) ของเหตุการณ์ i

ปริมาณข่าวสาร ของเหตุการณ์ i บอกเราว่าควรจะใช้เนื้อที่ในการเก็บเหตุการณ์ i เท่าไร ตัวอย่างเช่น เหตุการณ์ x มีความน่าจะเกิดเท่ากับ 1/8 แล้วแสดงว่า เนื้อที่ที่ใช้เก็บเหตุการณ์ x คือ 3 บิท

ปริมาณข่าวสารเฉลี่ยของเหตุการณ์ทั้งหมดหรือ entropy คำนวณได้จาก

$$H = - (P_1 \log_2 P_1 + P_2 \log_2 P_2 + \ldots + P_M \log_2 P_M)$$

โดยที่ M คือเหตุการณ์ที่เป็นไปได้ทั้งหมด

ปริมาณข่าวสารเฉลี่ย บอกเราว่าควรจะเก็บข้อมูลทั้งหมดด้วยเนื้ออย่างน้อยที่สุดเท่าไรเช่นข้อมูล 'e' มีความน่าจะเกิดเท่ากับ 1/16 ดังนั้นข้อมูล 'eeeee' ใช้เนื้อที่ในการเก็บเท่ากับ20 บิท

ข่าวสารเฉลี่ย มีเงื่อนไขที่น่าสนใจ 2 กรณีคือ

1 เมื่อ M = 1 คือมีเหตุการณ์เดียว ดังนั้น $P_i = 1$ ทำให้ H = 0 หมายความว่าไม่มีข่าวสารเฉลี่ยของเหตุการณ์ที่ทราบแน่นอนแล้ว

2 เมื่อ $P_i = 0$ ทำให้ H = 0 หมายความว่าไม่มีข่าวสารเฉลี่ยของเหตุการณ์ที่เป็นไปไม่ได้

นั่นคือ $H = - P \log_2 p = 0$ (เมื่อ $P \to 0$)

นิยามของส่วนที่ซ้ำ คือ ผลต่างระหว่างปริมาณของข้อมูลและข่าวสารเฉลี่ย

$$R = \log_2 M - H$$

ส่วนที่ซ้ำเกิดขึ้น เนื่องจาก การกระจายของข้อมูลไม่เท่ากัน คือความน่าจะเกิดของข้อมูลทุกตัวไม่เท่ากัน ถ้าการกระจายของข้อมูลเท่ากัน แล้ว $H = \log_e M$ ซึ่ง H จะมีค่าสูงสุดดังนั้น $R = 0$ แต่ ข้อมูล โดยทั่วไปมักกระจายไม่เท่ากัน

ดังนั้นเทคนิคการลดขนาดข้อมูลจะใช้ประโยชน์ จากการกระจายไม่เท่ากันของข้อมูลนี้ เพื่อลดขนาดของข้อมูล โดยการเข้ารหัสข้อมูลด้วยรหัสที่เท่ากับ หรือ ต่างจาก entropy น้อยที่สุด ตัวอย่างของรหัสที่ใช้ในการเข้ารหัสข้อมูล คือ ฮัฟแมน(Huffman) หรือ Shannon Fano

2.2 อัลกอริทึมของฮัฟแมน

ผู้ที่คิดวิธีการลดขนาดข้อมูลนี้ก็คือ นาย D.A.Huffman ศิษย์เก่าจาก MIT เขาได้เสนอกระบวนการ "เข้ารหัส" แบบนี้มาตั้งแต่ต้นทศวรรษ 1950 สำหรับหลักการสำคัญของการลดขนาดข้อมูลนี้ก็คือ การลดจำนวนบิตที่ใช้แทนตัวอักษรที่มีการใช้บ่อยๆ และเพิ่มจำนวนบิตที่ใช้แทนตัวอักษรที่มีการใช้น้อยๆ

การทำงานชนิดนี้อยู่บนพื้นฐานความจริงที่ว่า ในประโยคหรือในไฟล์ข้อมูลที่เป็นภาษาอังกฤษหนึ่งๆ จะมีการใช้อักษรภาษาอังกฤษซ้ำไปซ้ำมาเสมอ และบางตัวก็ไม่ได้ใช้บ่อยสักเท่าไร ถ้าภาษาอังกฤษตัวหนึ่งๆถูกแทนด้วยเลขฐานสองจำนวน 8 บิต แล้วเราพบว่าในไฟล์ข้อมูลของเรามีตัวอักษร "a" มากที่สุดและพบว่าตัวอักษร "z" เป็นตัวอักษรที่พบน้อยตามวิธีการของ ฮัฟแมน เราก็อาจจะแทนอักษร "a" ด้วยรหัส 4 บิต และให้อักษร "z" แทนด้วยรหัส 14 บิต เป็นต้น

ปัจจุบันได้มีโปรแกรมแอพพลิเคชันหลายๆชนิดที่ได้ประยุกต์ใช้วิธีการลดขนาดของ ฮัฟแมนนี้ เช่น การลดขนาดแบบ MNP-5 ของโมเด็ม ซึ่งได้ใช้การลดขนาดข้อมูล ฮัฟแมน แบบไดนามิก การเข้ารหัสแบบ Shannon-Fano การเข้ารหัสบางส่วนของโปรแกรม PKZIP การทำงานตอนท้ายของการลดขนาดแบบ JPEG เป็นต้น

การสร้างรหัสของ ฮัฟแมน ก่อนอื่นอ่านข้อมูลหนึ่งรอบ เพื่อหาความถี่ของแต่ละข้อมูลจากนั้นเรียงลำดับของข้อมูลโดยข้อมูล ที่มีความถี่มากอยู่ด้านหนึ่ง และข้อมูลที่มีความถี่ น้อยอยู่ด้านหนึ่ง และกำหนดให้ข้อมูลแต่ละตัวคือโหนด(node)ใดๆของทรี(tree)แล้วนำมาสร้าง ทรี ตามวิธีต่อไปนี้

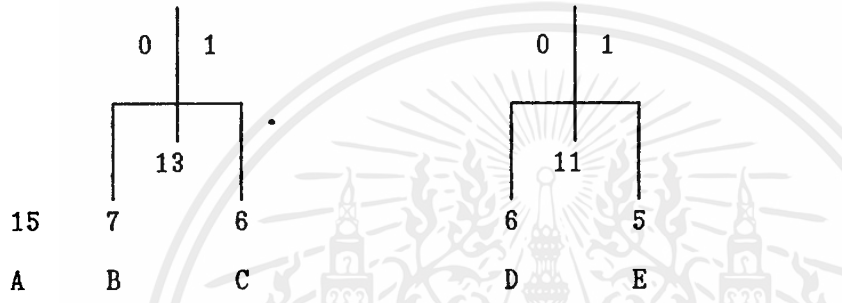
1. นำเอาข้อมูลหรือโหนดที่มีความถี่น้อยที่สุดมาทำเป็น โหนดลูก(child node)ของ โหนดพ่อ (parent node) ซึ่งความถี่ของโหนดพ่อนี้จะเท่ากับผลรวมของความถี่ของโหนดลูกทั้งสอง
2. โหนดพ่อ จะถูกนำไปรวมกับโหนด อื่นๆที่เหลือและโหนดลูกที่ใช้แล้วทั้งสองจะไม่ถูกนำมาใช้อีก
3. โหนดลูก หนึ่งจะถูกกำหนดให้เป็นบิท 0 และอีกโหนดเป็นบิท 1
4. กระทำซ้ำตั้งแต่ 1 จนกระทั่งเหลือ โหนดเดียว เรียกว่า ราก(root) ของ ทรี

ตัวอย่างเช่น ให้ข้อมูลมีลักษณะดังนี้

"AEDBAACBADCABECADACEDADACAEBABABEADABAD"

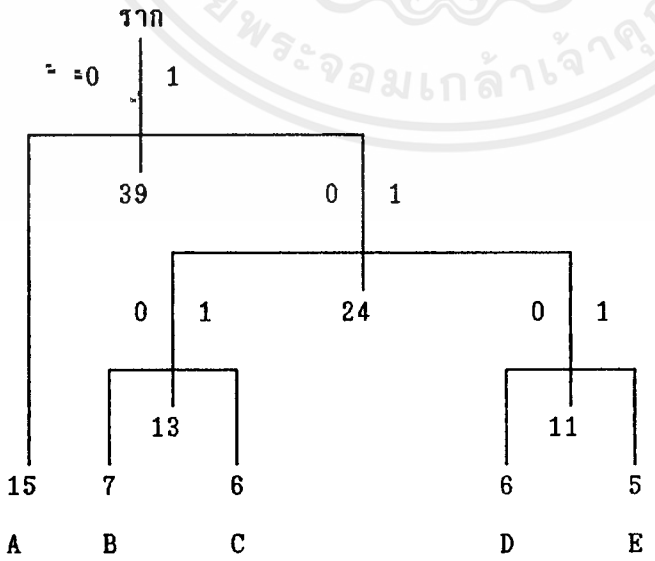
ความถี่	15	7	6	6	5
	A	B	C	D	E

โหนด ที่มีน้ำหนักน้อยที่สุด คือ D และ E ทั้งสอง โหนด จะถูกเชื่อมให้เป็น โหนดพ่อซึ่งมีน้ำหนัก 11 แล้วเรากำหนดให้ โหนด D มีค่าเป็น 0 และโหนด E มีค่าเป็น 1 รอบที่สอง B และ C จะถูกนำมาเชื่อมต่อเป็น โหนดพ่อ ใหม่อีก ซึ่งมีน้ำหนักเป็น 13 ซึ่งจะเป็นดังรูป 2.1



รูปที่ 2.1 ฮัฟแมนทรี เมื่อผ่านไป 2 รอบ

รอบต่อมา 2 โหนด ที่มีน้ำหนักน้อยที่สุด คือ โหนดพ่อ ของ B/C กับ D/E จะถูกเชื่อมเข้าเป็นโหนดพ่อ หลังจากนั้นจะเหลือโหนดเพียง 2 โหนด เมื่อนำมารวมกันจะได้ ราก ของ ทรีซึ่งมีลักษณะดังรูป 2.2



รูปที่ 2.2 ฮัฟแมนทรี

การหารหัสของข้อมูลทำได้โดยการ เดินทางจากรากไปยังข้อมูลตัวนั้นผ่าน โหนดใดให้จำค่าของโหนดนั้นไว้ รหัสของข้อมูลคือ ค่าของโหนดที่จำไว้ตามลำดับ ดังนั้นรหัสของข้อมูลคือ

A = 0

B = 100

C = 101

D = 110

E = 111

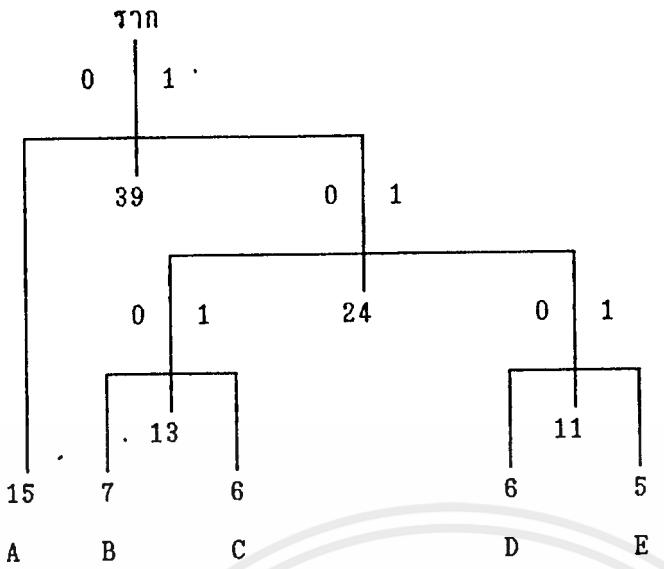
หลังจากได้รหัสของข้อมูลแล้วจึงอ่านข้อมูลอีกรอบเพื่อลดขนาดข้อมูลโดยการเข้ารหัสข้อมูลทั้งหมดโดยการแทนข้อมูลด้วยรหัสของข้อมูลดังนั้นจะได้รหัสของข้อมูลมีลักษณะเรียงเป็นลำดับเช่นเดียวกับลำดับของข้อมูลดังรูป 2.3

"0 111 110 100 0 0 101 ...." รหัสของข้อมูลมีลักษณะเป็นบิท

"A E D B A A C ...." ข้อมูลมีลักษณะเป็นไบท์

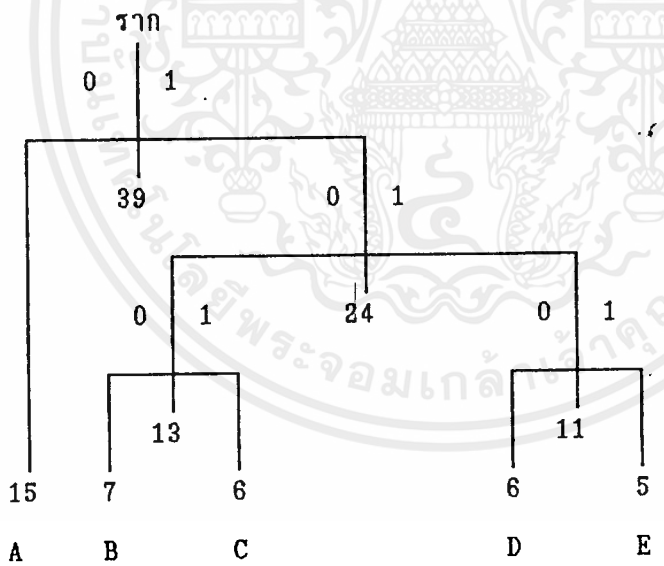
รูปที่ 2.3 ตัวอย่างการเข้ารหัสข้อมูล

สำหรับการขยายขนาดข้อมูลกลับทำได้โดย การสร้างทรีขึ้นมาก่อนโดยทรีนี้คือทรีอันเดียวกับที่ใช้ในการลดขนาดข้อมูล โดยจะอ่านรหัสเข้ามาทีละบิทและเริ่มเดินทางจากรากไปยังกิ่งที่มีค่าเท่ากับรหัส 1 บิทที่อ่านเข้ามา และกระทำเช่นนี้ต่อไปจนกระทั่งไม่มี โหนดย่อยลงไปอีกก็จะได้ข้อมูลกลับคืนมาหนึ่งตัว หลังจากนั้น เริ่มเดินทางจากรากใหม่ จนกระทั่งได้ข้อมูลกลับคืนมาจนครบหมด จากรหัสของข้อมูลข้างต้นบิทตัวแรกคือ 0 ดังนั้นสามารถถอดรหัสออกมาได้เป็น A ดังรูป 2.4



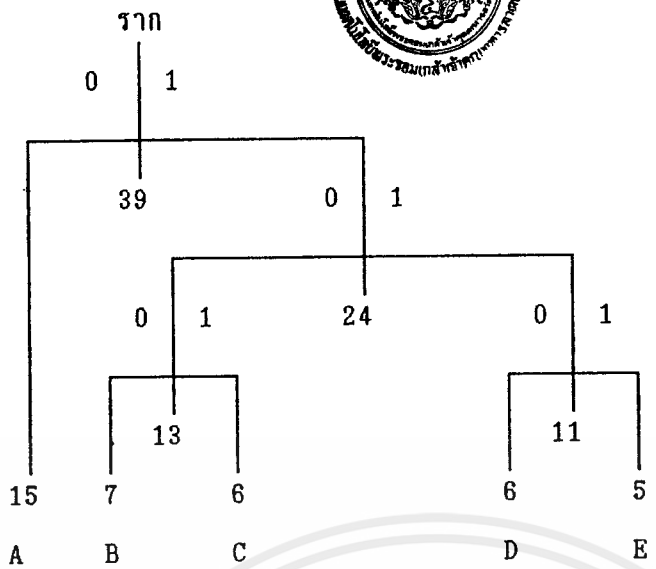
รูปที่ 2.4 แสดงการถอดรหัสเมื่อได้รับรหัส 0

จากนั้นอ่านรหัสข้อมูลบิทที่ 2 คือ 1 เมื่อเดินทางจากรากไปยังทรีจะมีลักษณะดังรูป 2.5



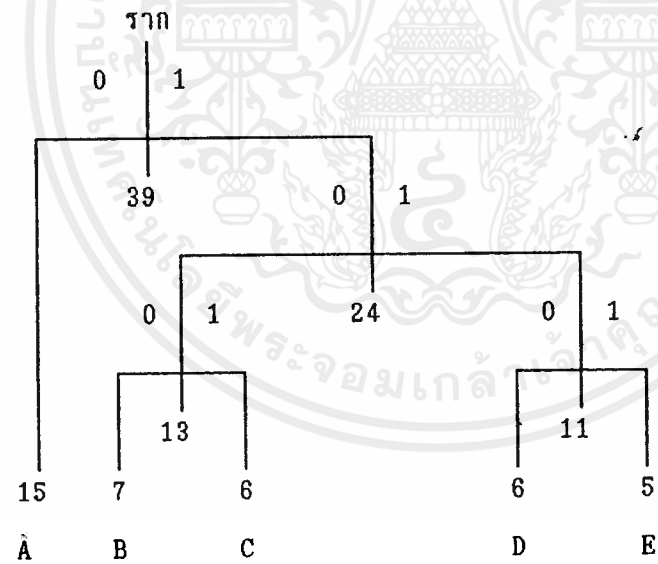
รูปที่ 2.5 แสดงการถอดรหัสเมื่อได้รับรหัส 1

จากนั้นอ่านรหัสข้อมูลบิทที่ 3 คือ 1 เมื่อเดินทางจากรากไปยังทรีจะมีลักษณะดังรูป 2.6



รูปที่ 2.6 แสดงการถอดรหัสเมื่อได้รับรหัส 11

จากนั้นอ่านรหัสข้อมูลบิทที่ 1 คือ 1 เมื่อเดินทางจากรากไปยังทรีจะมีลักษณะดังรูป 2.7



รูปที่ 2.7 แสดงการถอดรหัสเมื่อได้รับรหัส 111

ก็จะสามารถถอดรหัสได้เป็น E และจะกระทำเช่นนี้ต่อไปจนกระทั่งไม่มีรหัสข้อมูลเหลือ แล้วจะได้ข้อมูลกลับมาเหมือนเดิมทุกประการ

2.3 อัลกอริทึมของ LZW

การลดขนาดข้อมูล LZW ซึ่งถูกนำมาเสนอโดยนาย Welch ในปี 1984 และมีคนนิยมนำมาใช้กันในระยะปีสองปีนี้เอง หลักการที่สำคัญของ LZW ก็คือการสร้างรหัสอักษรใหม่นอกเหนือจากที่เคยกำหนดไว้แล้ว เช่น รหัสอักษรใน IBM PC ซึ่งเป็นรหัสอักษรแบบ 8 บิต มีค่าตัวอักษรทั้งหมด 0-255 การเข้ารหัสของ LZW จะใช้การเพิ่มตัวอักษณเป็นตัวที่ 256,257และต่อไปเรื่อยๆโดยรหัสตัวอักษรที่เพิ่มขึ้นมานี้ จะใช้แทน "คู่" ตัวอักษรที่มีการใช้บ่อยๆ

สมมติว่าเรามีข้อความที่ต้องการย่อคือ This_is_a_cat (เครื่องหมาย "_"เพื่อเป็นการเน้นให้รู้ว่านั่นคือ space หรือ ช่องว่าง)เมื่อมีการอ่านข้อมูลเข้ามา เริ่มจากตัวอักษรT ซึ่งก็เป็นตัวอักษรที่มีรหัสตาม ASCII อยู่แล้ว ตัวต่อมาก็จะเป็น h ซึ่งเมื่อร่วมกับอักษร T ก็จะเป็น Th

ตัวอักษร Th นี้เองที่เราจะนิยามให้มันมีรหัสอักษรใหม่เป็น 256 จากนั้นก็อ่านอักษร i เข้ามา และก็มีการรวมอักษรตัวเก่าล่าสุดหรือตัว h มาจับคู่ตัวอักษร i กลายเป็น hi จากนั้นก็นิยามรหัสอักษรให้ใหม่เป็น 257 คู่ต่อไปเป็น is ก็ให้นิยามเป็น 258 ต่อไปก็จะเป็น s+(space)ซึ่งให้เป็นรหัสอักษร 259 คู่ต่อไปก็เป็น (space)+i ซึ่งก็เป็นรหัสอักษรที่ 259

ตลอดระยะเวลาที่เราเริ่มลดขนาดข้อมูลมาตั้งแต่อักษร T ถึงตัว (space) ถ้าสังเกตจากรูปที่ 2.8 แล้ว จะพบว่าเรายังไม่ได้มีการลดขนาดข้อมูลใดๆเลย และแล้วเมื่อเริ่มมีการซ้ำโดยอักษรคู่ต่อไปเป็น is อีกแล้ว ตอนนี้เองที่เราจะเก็บข้อมูล is เป็นรหัสอักษร 258 แทน ทั้งนี้เพราะเราพบว่ามีอักษรคู่ลักษณะนี้อยู่แล้วในพจนานุกรมรหัสอักษรใหม่ ซึ่งการแทนอักษร is ชุดหลังนี้ด้วยรหัส 258 ก็จะเสมือนกับเป็นการลดตัวอักษร 2 ตัว ให้เลือกเหลือเพียงตัวอักษรเดียวนั่นเอง (แต่ถ้าเป็นตัวอักษรชนิด 9 บิต)

input	Compression table	compressed string
T	—	—
h	256 — *T+h*	*T*
i	257 — *h+i*	*h*
s	258 — *i+s*	*i*
(SPACE)	259 — *s+(space)*	*s*
i	260 — *(space)+i*	*(space)*
s	—	—
(space)	261—258+*(space)*	*258*
a	262—*(space)+a*	*(space)*

รูปที่ 2.8 แสดงขั้นตอนการลดขนาดข้อมูลแบบ LZW เมื่อมีการป้อนข้อมูลเข้ามาเป็น This_is_a_cat

ส่วนสำคัญของอัลกอริทึม LZW คือตารางรหัส (String table หรือ Translation Table) รหัสแต่ละตัวในตารางจะประกอบด้วย ส่วนสตริงนำหน้า (Prefix string หรือ w) และส่วนตัวอักษร (suffix character หรือ K)

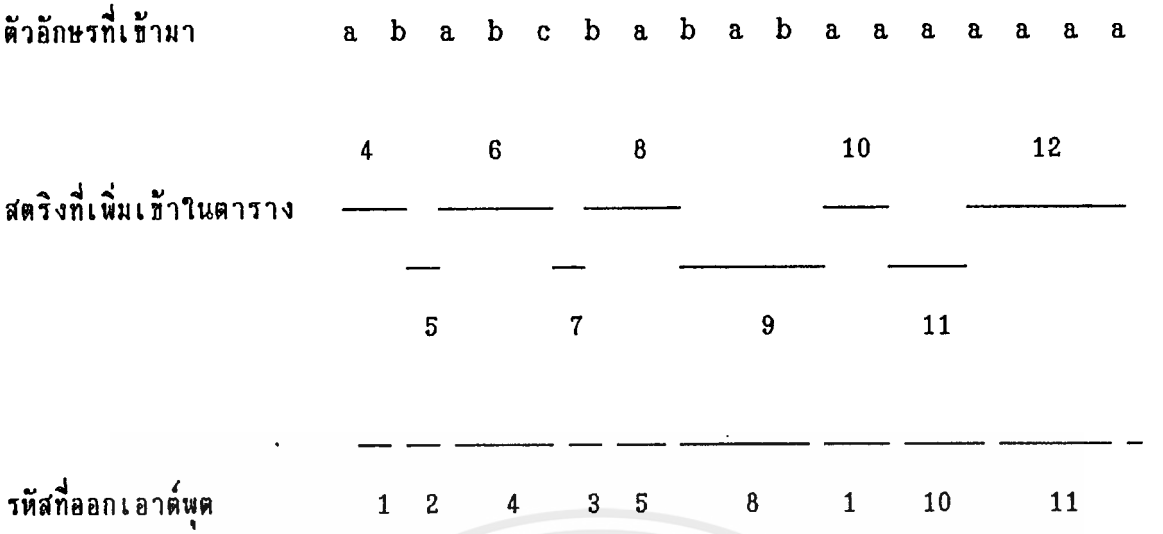
ข้อมูลแต่ละตัวอักษรจะถูกพิจารณาตามลำดับเพื่อหาสตริงที่ยาวที่สุดที่ตรงกับสตริงในตารางจนกระทั่งไม่สามารถหาได้จึงทำการเพิ่มสตริงที่มาใหม่นั้นเข้าไว้ในตาราง แล้วส่งรหัสของสตริงที่ยาวที่สุดนั้นออกเอาต์พุตเมื่อทำซ้ำขั้นตอนที่ว่านั้นจนจบ ก็จะได้ข้อมูลที่ลดขนาดแล้ว

ขั้นตอนที่กล่าวมาข้างต้น สามารถอธิบายได้ดังนี้

```
initialize table                ; เริ่มด้วยตารางที่ว่างเปล่า
read first input character and store in prefix string or w ; อ่านข้อมูล
                                    ตัวแรกเข้ามาเก็บไว้ใน w
step:read next input character K  ; อ่านข้อมูลตัวต่อมาเก็บไว้ที่ K
     If no such K (end of input)  ; แต่ถ้าไม่มีข้อมูลเหลือแล้ว
          output code(w)          ; ให้เอาพุท w
          Exit                    ; และสิ้นสุดการทำงาน
     If wK exists in string table ; ถ้า wK มีในตาราง
          w = wK                  ; ให้ w = wK
     else                         ; ถ้า wK ไม่มีในตาราง
          add wK in string table  ; ให้เพิ่ม wK ในตาราง
          output code(w)          ; ให้เอาพุท w
          w = K                   ; และให้ w = K
     repeat step                  ; กระทำซ้ำจนกว่าไม่มีข้อมูลเหลือ
```

จะเห็นว่าทุกๆรอบการทำงาน สตริงนำหน้า w จะมีการเปลี่ยนแปลง ถ้าสตริงนำหน้ารวมกับตัวอักษรเป็นสตริง wK มีอยู่ในตารางรหัสสตริงนำหน้าก็จะยาวขึ้นอีก 1 ตัวอักษร (เพิ่มอักษรKเข้าไป)แต่ถ้า wK ไม่อยู่ในตาราง ก็จะเพิ่มสตริงใหม่เข้าไว้ในตาราง แล้วส่งรหัสของสตริงนำหน้าออกทางเอาพุต สตริงนำหน้าก็จะเริ่มต้นใหม่มีค่าเป็น K ยาวเพียง 1 ตัวอักษร

ตัวอย่างการทำงานของอัลกอริทึมเป็นดังรูปที่ 2.9 ซึ่งสมมติว่ามีตัวอักษรในชุดรหัสเพียง 3 ตัว คือ a,b,c



รูปที่ 2.9 ตัวอย่างการลดขนาดข้อมูล

ตารางรหัส		ตารางในรูปแบบ (w,K)	
1	a	1	a
2	b	2	b
3	c	3	c
4	ab	4	1b
5	ba	5	2a
6	abc	6	4c
7	cb	7	3b
8	bab	8	5b
9	baba	9	8a
10	aa	10	1a
11	aaa	11	10a
12	aaaa	12	11a

ตารางที่ 2.1 ตารางรหัสที่ได้จากตัวอย่าง

จากรูปที่ 2.9 ตัวอักษรถูกอ่านเข้ามาจากซ้ายไปขวาเริ่มต้นที่การอ่านตัวอักษร "a" เข้ามาแล้ว กำหนดค่าให้แก่ w และตัวถัดมาคือ "b" กำหนดค่าให้แก่ K เนื่องจาก wk ซึ่งเท่ากับ "ab"

ไม่มีในตารางรหัส(ในตอนแรกตารางมีเพียง 3 ตัวคือ a,b,c)จึงเพิ่ม "ab" เข้าไว้ในตารางเป็นรหัสหมายเลข 4 และส่ง w คือรหัสหมายเลข 1 ออกทางเอาต์พุต ต่อจากนั้นจึงกำหนดให้ w มีค่าเป็น K ซึ่งเท่ากับ "b"

w = "a"

K = "b"

ดังนั้น wk = "ab" ซึ่งไม่มีในตาราง ให้กลับไปเพิ่มค่ารหัสในตารางหรือรหัสที่ 4 (ในการลดขนาดข้อมูลจริงๆที่เป็น ตัวอักษร ชนิด 8 บิท ซึ่งมีรหัสอักษร 256 ตัว เราก็จะเพิ่มเข้าไปเป็นรหัสที่ 257 แทนหมายเลข 4)

output "1" ส่งรหัสหมายเลข 1 หรืออักษร "a"นั่นเอง ออกไปยังที่เอาต์พุต

w = K ถึงตรงนี้ตัวแปร w ก็จะเก็บค่า "b" เอาไว้แทนแล้ว

ต่อไปเป็นการอ่านตัวอักษร "a" (ซึ่งอยู่ในตารางที่ 3 จากซ้ายมือ)เข้ามาเก็บไว้ในตัวแปร K ในรอบนี้ wK คือ "ba" ก็ไม่มีในตาราง จึงเพิ่มเข้าไปเป็นรหัสอักษรหมายเลข 5 ส่งรหัสอักษรหมายเลข 2 ออกทางเอาต์พุต กำหนดให้ W มีค่าเท่ากับ "a"

จากนั้นอ่าน "b" (ตัวที่ 4 จากซ้ายมือ)เข้ามากำหนดค่าให้กับ K เนื่องจาก wK ในรอบนี้ คือ "ab" มีอยู่ในตารางเป็นรหัสหมายเลข 4 รอบนี้จึงเพียงกำหนดค่าให้ w เท่ากับ "ab"ไปเลยแล้วส่งรหัสอักษรหมายเลข 4 นี้ออกสู่เอาต์พุต จากนั้นจึงอ่าน "c" เข้ามาเก็บในตัว K และเนื่องจากค่า wK ตัวใหม่นี้ เป็นอักษร "abc" ซึ่งไม่มีอยู่ในตารางดังนั้นจึงทำการเพิ่มรหัสอักษรเข้าไปในตาราง ซึ่งจะเป็นหมายเลข 6 นั่นเอง แล้ววกกลับไปอ่านข้อมูลตัวต่อไป ทำซ้ำนี้ไปเรื่อยๆจนหมดข้อมูล

จะเห็นว่าตัวอัลกอริทึมนั้นประกอบด้วยขั้นตอนเพียงไม่กี่ขั้นตอนเท่านั้น ขั้นตอนที่ยุ่งยากที่สุดน่าจะเป็นการหาว่า wK มีอยู่ในตารางหรือไม่ จากตารางที่ 2.1 จะเห็นว่าเราสามารถจัดรหัสในตารางให้เหมาะสำหรับการค้นหาได้โดยแยกรหัสออกเป็นหมายเลขทรหัสของสตริงนำหน้า และตัวอักษร(w,K) แทนที่จะเก็บรหัสเป็นสตริงที่ขนาดไม่แน่นอน

สำหรับการขยายกลับข้อมูล ข้อดีของ LZW คือ ไม่ต้องมีการเก็บ "ตารางรหัส" ไว้ในข้อมูลเพื่อใช้ในการขยายกลับ ทั้งนี้เพราะตัวขยายกลับ จะสามารถสร้างตารางรหัสได้เอง จากข้อมูลที่มันจะทำการขยายนั่นเอง โดยตารางรหัสนั้นก็จะเป็นตารางเดียวกันกับตอนทำการลดขนาด

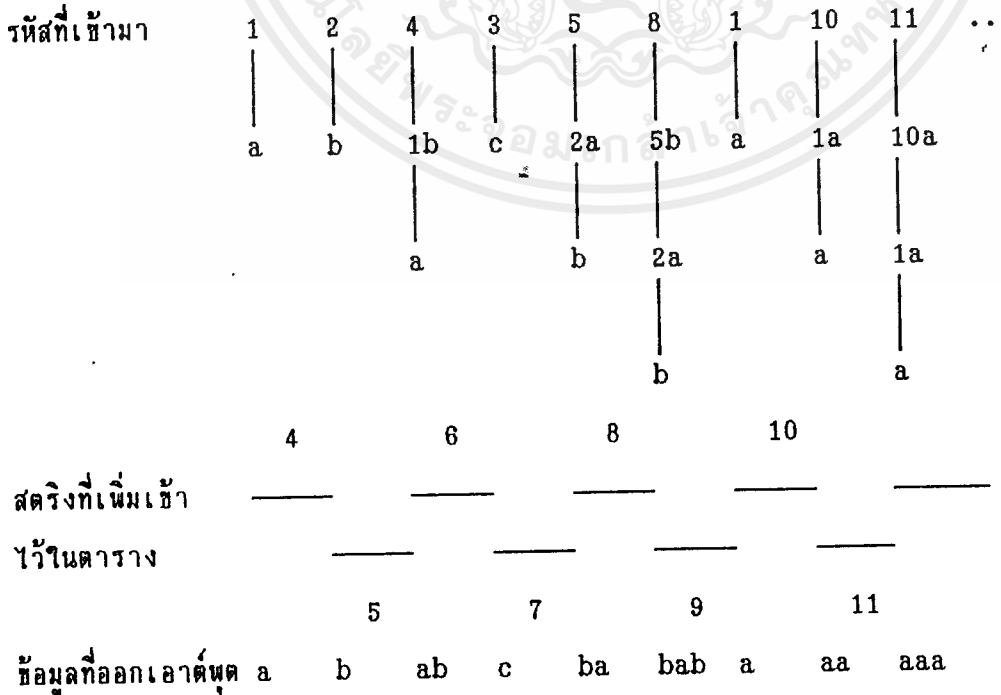
เพื่อถอดรหัสที่เข้ามาจาก wK จนเหลือเพียง KK ที่ได้จะนำไปรวมกับรหัสที่เข้ามาก่อนหน้า (wK) กลายเป็นรหัสตัวใหม่ในตารางรหัส (wk)K

อัลกอริทึมในเบื้องต้นสามารถอธิบายได้ดังนี้

```
read first input code and store in CODE , OLDcode  ; อ่านข้อมูล
                                                   เข้ามาเก็บไว้ที่ CODE, OLDcode
with CODE = code(K), output K                      ; เอาพุท K
```

```
next code:    read next input code and store in CODE , INcode    ; อ่านข้อมูล
                                                                  เข้ามาเก็บไว้ที่ CODE, INcode
              If no new code                                      ; ถ้าไม่มีรหัสใหม่
                   exit                                           ; สิ้นสุดการทำงาน
next symbol:  if CODE = code(wK)                                  ; ถ้า CODE สามารถถอดรหัสได้
                   output K                                       ; เอาพุท K
                   CODE = code(w)                                 ; ให้ CODE = w
                   repeat next symbol                             ; กระทำซ้ำ
              if CODE = code(K)                                   ; ถ้าCODEไม่สามารถถอดรหัสได้
                     output K                                     ; เอาพุท K
                     add code(OLDcode,K) in string table          ; เก็บwKลงในตาราง
                     OLDcode = INcode                             ; ให้ OLDcode = INcode
                     repeat next code                             ; กระทำซ้ำ
```

ตัวอัลกอริทึมข้างต้นนี้มีปัญหาอยู่ 2 ประการคือ การถอดรหัสจะได้ตัวอักษรในลำดับก่อนหลัง เช่น รหัส "cab" (มองในรูป wK คือ "ca" "b") จะถอดได้เป็น "bac" ทั้งนี้เพราะ K ที่ได้ในแต่ละครั้งจะเป็นตัวอักษรท้ายสุดของรหัส การแก้ปัญหาทำได้โดยการใช้ stack เก็บตัวอักษรที่ได้แล้วจึงส่งออกเอาต์พุตในตอนหลัง และปัญหาต่อมาคือ ในบางกรณีการสร้างรหัสจะไม่ทันการอ้างถึงรหัส พิจารณาตัวอย่างในรูปที่ 2.10



รูปที่ 2.10 ตัวอย่างการขยายกลับข้อมูล

ปัญหาจะเกิดขึ้นเมื่อรหัส 8 เข้ามาเพราะในตารางเพิ่งมีถึงรหัส 7 เท่านั้น ปัญหานี้จะเกิดขึ้นเมื่อรหัสเข้ามาเป็นลำดับ KwKwK อย่างไรก็ตาม เราสามารถสร้างรหัส 8 ได้ถ้ารู้ว่า K สุดท้าย (อักษรตัวแรกสุดของรหัส) ที่จะได้จากรหัส 8 นี้คืออะไร ซึ่งจากการพิจารณาจะเห็นว่า ในกรณีที่เกิดปัญหาเช่นนี้ขึ้น K สุดท้ายของรหัสที่ต้องการก็คือ K สุดท้ายของรหัสที่เข้ามาก่อนหน้านั่นเอง ในที่นี้ K สุดท้ายของรหัส 8 จะเท่ากับ K สุดท้ายของรหัส 5 คือ "b" ดังนั้นการแก้ปัญหาจึงทำได้โดยการเก็บ K สุดท้ายของรหัสก่อนหน้าไว้ทุกครั้ง เพื่อนำมาใช้สร้างรหัสเมื่อเกิดกรณีพิเศษดังกล่าวขึ้น

อัลกอริทึมที่ใช้งานได้จริง เป็นดังนี้

```
          read first input code and store in CODE , OLDcode ; อ่านข้อมูล
                                                    เข้ามาเก็บไว้ที่ CODE,OLDcode
          with CODE = code(K)                       ; ซึ่ง CODE คือ K
               output = K                           ; เอาพุท K
               finchar = K                          ; ให้ finchar = K
next code: read next input code and store in CODE , INcode    ; อ่านข้อมูล
                                                    เข้ามาเก็บไว้ที่ CODE,INcode
          if no new code                            ; ถ้าไม่มีรหัสเหลือ
               exit                                 ; สิ้นสุดการทำงาน
          if CODE not defined (special case)        ; ถ้าเกิดกรณีพิเศษ
               output finchar                       ; เอาพุท finchar
               CODE = OLDcode                       ; ให้ CODE = OLDcode
               INcode = code(OLDcode,finchar)       ; INcode = wK
next symbol: if CODE = code(wK)                     ; ถ้าสามารถกระจายรหัสได้
               push K in to stack                   ; เก็บ K ลงในแสตก
               CODE = code(w)                       ; ให้ CODE = w
               repeat next symbol                   ; กระทำซ้ำ
          if CODE = code(k)                         ; ถ้าไม่สามารถกระจายรหัสได้
               output = K                           ; เอาพุท K
               finchar = K                          ; ให้ finchar = K
               do while stack not empty             ; นำเอาข้อมูลที่เก็บลงแสตกออก
                  output pushed data                ; เอาพุท
               add code(OLDcode,K) in string table  ; เก็บ wK เข้าไว้
                                                    ในตาราง
```

```
OLDcode = INcode                ; ให้ OLDcode = INcode
repeat next code                ; กระทำซ้ำ
```

เนื่องจากรหัสที่เข้ามาสามารถนำไปอ้างถึงตารางได้โดยตรง อัลกอริทึมของการขยายกลับจึงมีประสิทธิภาพดีกว่าอัลกอริทึมของการลดขนาดข้อมูล

2.4 อัลกอริทึมของ พจนานุกรมแบบเลื่อนได้ (Sliding Dictionary)

ซึ่งต่อไปจะเรียกอัลกอริทึมของพจนานุกรมแบบเลื่อนได้ว่าอัลกอริทึม

อัลกอริทึมสามารถลดขนาดข้อมูลได้ โดยการหาตำแหน่งของสตริงของข้อมูลที่ได้ผ่านการเข้ารหัสแล้วที่เหมือนกับสตริงของข้อมูลที่กำลังจะเข้ารหัสโดยที่ข้อมูลที่ผ่านการเข้ารหัสแล้วถูกเก็บในส่วนที่เรียกว่า "พจนานุกรม"(Dictionary) และ ข้อมูลที่กำลังจะเข้ารหัสถูกเก็บในส่วนที่เรียกว่า"ส่วนที่จะเข้ารหัส" (Look ahead buffer) เมื่อหาตำแหน่งของสตริงที่เหมือนกันได้แล้ว ก็จะนำมาเข้ารหัสเป็น "ระยะห่างของสตริงทั้งสอง,ความยาวของสตริงที่เหมือนกัน"ซึ่งรหัสนี้จะแทนสตริงที่กำลังจะถูกลดขนาด เช่นถ้าสตริงทั้งสองอยู่ห่างกัน 6 ไบต์ และความยาวของสตริงที่เหมือนกันเท่ากับ 3 ไบต์ แล้วสตริงที่กำลังจะถูกลดขนาด จะนำมาเข้ารหัสเป็น "6,3" จากนั้นนำเอาสตริงที่เข้ารหัสแล้วไปเก็บไว้ในพจนานุกรมและอ่านข้อมูลใหม่เข้ามาเก็บไว้ในส่วนที่จะเข้ารหัส และกระทำการเช่นนี้ต่อไปจนไม่มีข้อมูลใหม่เหลืออีก สำหรับในกรณีที่ไม่มีสตริงที่เหมือนกันเลย ทำให้ไม่สามารถเข้ารหัสได้ จึงแก้ปัญหาโดยการเอาพุทรหัสเป็นสตริงขนาด 1 ไบต์โดดๆ โดยมีการกำหนดบิทพิเศษจำนวน 1 บิท ข้างหน้ารหัสเพื่อบอกว่ารหัสสเอ้าพุทจะเป็นรหัส "ระยะห่างของสตริงทั้งสอง,ความยาวของสตริง" หรือเป็นสตริงขนาด 1 ไบต์

ถ้ากำหนดให้รหัส "ระยะห่างของสตริงทั้งสอง,ความยาวของสตริง" มีบิทพิเศษข้างหน้ารหัสเป็นศูนย์ และกำหนดให้บิทพิเศษข้างหน้ารหัสที่เป็นสตริงขนาด 1 ไบต์ เป็นหนึ่งแล้ว ดังนั้นรหัสของอัลกอริทึมจะมี 2 ลักษณะคือ

1. "0,ระยะห่างของสตริงทั้งสอง,ความยาวของสตริง" เมื่อมีสตริงที่เหมือนกัน
2. "1,สตริงขนาด 1 ไบต์" เมื่อไม่มีสตริงที่เหมือนกันเลย

วิธีการดังกล่าวสามารถอธิบายได้ดังนี้คือ

ขั้นตอนที่

```
1       initialize Dictionary                   ; ทำให้พจนานุกรมขณะเริ่มต้นว่างเปล่า
2       read data to stored in look_ahead_buffer; อ่านข้อมูลเข้ามาเก็บไว้ในส่วน
```

ที่จะเข้ารหัส

```
3 loop:   find  distance and length_of_match_string ; หาระยะห่างของสตริงและ
                                                    ความยาวของสตริงที่เหมือนกันระหว่างพจนานุกรม
                                                    และส่วนที่จะเข้ารหัส
4         if length_of_match_string = 0             ; กรณีที่ไม่มีสตริงที่เหมือนกัน
4.1           outputbit 1                           ; เอาพุทบิทพิเศษเป็น 1
4.2           output first_char_of_look_ahead_buffer ;เอาพุทสตริงขนาด 1 ไบต์
4.3           used_string_length = 1       ; ดังนั้นสตริงที่เข้ารหัสแล้วมีขนาด 1ไบต์
5         else                                      ; กรณีที่มีสตริงเหมือนกัน
5.1           outputbit 0                  ; เอาพุทบิทพิเศษ เป็น 0
5.2           output distance              ; เอาพุทระยะห่างของสตริงทั้งสอง
5.3           output length_of_match_string ; เอาพุทความยาวของสตริงที่เหมือนกัน
5.4           used_string_length = length_of_match_string  ; ดังนั้นสตริงที่เข้า
                                    รหัสแล้วมีขนาดเท่ากับความยาวของสตริงที่เหมือนกัน
6         update Dictionary (used_string_length) ; เก็บสตริงที่เพิ่งจะเข้ารหัสไว้ใน
                                                  พจนานุกรม
7         update look_ahead_buffer (used_string_length) ;  ลบสตริงที่เพิ่งจะเข้า
                                    รหัสออกจากส่วนที่จะเข้ารหัส และอ่านข้อมูลเพิ่มเข้ามาใหม่
8         if look_ahead_buffer is not empty ; ยังมีข้อมูลเหลือในส่วนที่จะเข้ารหัส
9             repeat loop                   ; ดังนั้นจึงวนลูปใหม่
10        else  exit                        ; ข้อมูลทั้งหมดได้เข้ารหัสเรียบร้อยแล้ว
```

สมมติให้ขนาดของพจนานุกรมเท่ากับ 8 ไบต์ ดังนั้นระยะห่างของสตริงทั้งสองจึงมีค่า 8 ค่า เราสามารถแทนระยะห่างของสตริงทั้งสอง ด้วยรหัสขนาด 3 บิทได้ กำหนดให้ขนาดของส่วนที่จะเข้ารหัสเท่ากับ 4 ไบต์ ดังนั้นเราสามารถแทนความยาวของสตริงได้ด้วยรหัสขนาด 2 บิท และกำหนดให้ข้อมูลคือ

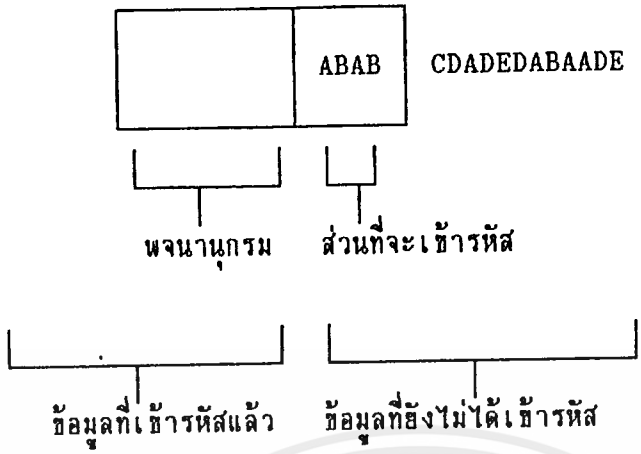
ABABCDADEDABAADE

เมื่อกระทำตามอัลกอริทึมข้างต้นสามารถอธิบายได้ดังนี้

ขั้นตอนที่ 1 เริ่มแรกพจนานุกรมยังไม่มีข้อมูลที่เข้ารหัส ดังนั้น จึงกำหนดให้พจนานุกรมว่างเปล่า

ขั้นตอนที่ 2 อ่านข้อมูลเข้ามาเก็บไว้ในส่วนที่จะเข้ารหัส เป็นจำนวนเท่ากับขนาดของส่วนที่จะเข้ารหัส

ดังนั้น เมื่อจบขั้นตอนที่2 แล้ว โครงสร้างของอัลกอริทึม จะมีลักษณะดังรูปที่ 2.11



รูปที่ 2.11 โครงสร้างของอัลกอริทึมขณะเริ่มต้น

ขั้นตอนที่3 ขณะเริ่มแรกยังไม่สตริงที่เหมือนกันเพราะพจนานุกรมยังว่างอยู่ ดังนั้นระยะห่างของสตริง จึงเท่ากับ 0 จึงกระทำขั้นตอนที่ 5 ต่อไป

ขั้นตอนที่5 เอาพุทบิทพิเศษเป็น 1

ขั้นตอนที่6 เอาพุทสตริงขนาด 1 ไบต์จากส่วนที่จะเข้ารหัส ในที่นี้คือ A

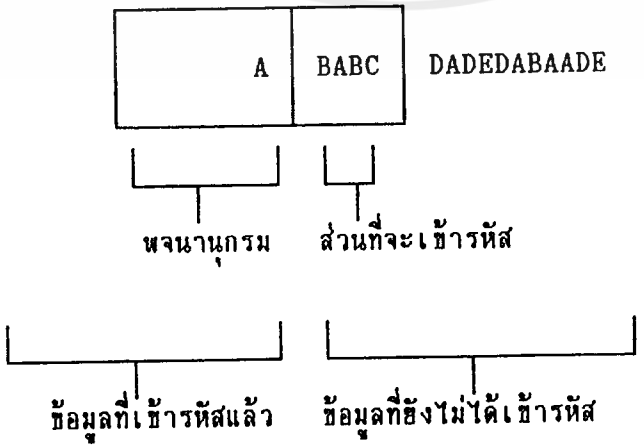
ดังนั้น สตริง A จะถูกเข้ารหัสเป็น "1,A" จากนั้นจึงมากระทำขั้นตอนที่7

ขั้นตอนที่7 ความยาวของสตริงที่เพิ่งจะเข้ารหัสคือ 1 ตัว

ขั้นตอนที่13 เก็บสตริงที่เพิ่งจะเข้ารหัส คือ A ไว้ในพจนานุกรม

ขั้นตอนที่14 ลบสตริงที่เพิ่งจะเข้ารหัสออกจากส่วนที่จะเข้ารหัสและอ่านข้อมูลเพิ่มเข้ามาใหม่อีก

เมื่อจบขั้นตอนที่ 14 แล้ว โครงสร้างของ อัลกอริทึมจะมีลักษณะดังรูปที่ 2.12



รูปที่ 2.12 โครงสร้างอัลกอริทึมเมื่อผ่านขั้นตอนที่ 14

จะเห็นได้ว่าเมื่อเปรียบเทียบโครงสร้างของอัลกอริทึมขณะเริ่มต้นและเมื่อผ่านขั้นตอนที่ 14 มาแล้วเสมือนกับว่าเป็นการเลื่อนพจนานุกรมและส่วนที่จะเข้ารหัสไปทางขวามือ 1 ช่อง

ขั้นตอนที่15 ยังมีข้อมูลเหลือในส่วนที่จะเข้ารหัสอีก ดังนั้นกลับไปกระทำขั้นตอนที่ 3. ซ้ำอีก

ขั้นตอนที่3 ก็ยังไม่มีสตริงในพจนานุกรมที่เหมือนกับสตริงในส่วนที่จะเข้ารหัส

ขั้นตอนที่5 เอาพุทบิทพิเศษเป็น 1

ขั้นตอนที่6 เอาพุทสตริงขนาด 1 ไบต์จากส่วนที่จะเข้ารหัส ในที่นี้คือ B

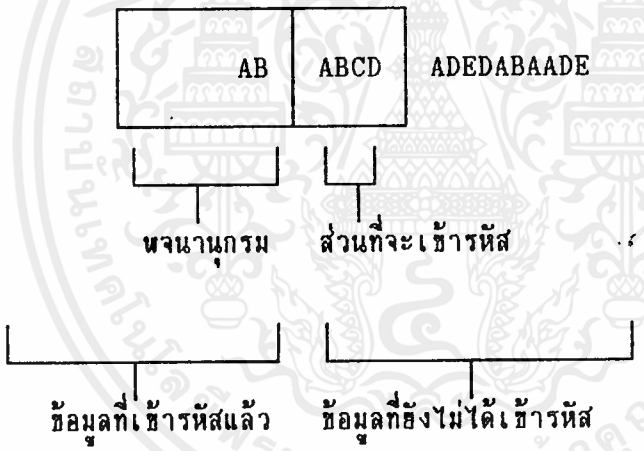
ดังนั้น สตริง B จะถูกเข้ารหัสเป็น "1,B" จากนั้นจึงมากระทำขั้นตอนที่7

ขั้นตอนที่7 ความยาวของสตริงที่เพิ่งจะเข้ารหัสคือ 1 ตัว

ขั้นตอนที่13 เก็บสตริงที่เพิ่งจะเข้ารหัส คือ B เพิ่มไว้ในพจนานุกรม

ขั้นตอนที่14 ลบสตริงที่เพิ่งจะเข้ารหัสออกจากส่วนที่จะเข้ารหัสและอ่านข้อมูลเพิ่มเข้ามาใหม่อีก

เมื่อจบขั้นตอนที่ 14 รอบที่ 2 แล้ว โครงสร้างของ อัลกอริทึมจะมีลักษณะดังรูปที่ 2.13



รูปที่ 2.13 โครงสร้างอัลกอริทึมเมื่อผ่านขั้นตอนที่ 14 รอบที่ 2

ขั้นตอนที่15 ยังมีข้อมูลเหลือในส่วนที่จะเข้ารหัสอีก ดังนั้นกลับไปกระทำขั้นตอนที่3 ซ้ำอีก

ขั้นตอนที่3. มีสตริงในพจนานุกรมที่เหมือนกับสตริงในส่วนที่จะเข้ารหัสคือ AB ดังนั้นระยะห่างของสตริง AB ในพจนานุกรมกับสตริง AB ในส่วนที่จะเข้ารหัสคือ 2 ช่องและความยาวของสตริงที่เหมือนกันคือ 2 ตัว

ขั้นตอนที่4 ความยาวของสตริงที่เหมือนกันคือ 2 ตัวดังนั้นไปกระทำการขั้นตอนที่ 9

ขั้นตอนที่9 เอาพุทบิท 1

ขั้นตอนที่10 เอาพุท ระยะห่างของสตริงทั้งสองคือ 2 ช่อง

ขั้นตอนที่11 เอาพุท ความยาวของสตริงทั้งสองคือ 2 ตัว

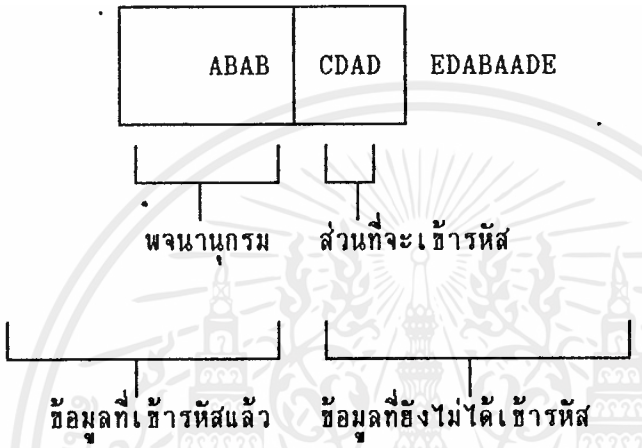
ดังนั้นสตริง AB จะถูกเข้ารหัสเป็น "0,2,2" จากนั้นจึงมากระทำขั้นตอนที่ 12

ขั้นตอนที่12 ความยาวของสตริงที่เข้ารหัสคือ 2 ตัว

ขั้นตอนที่13 เก็บสตริงที่เพิ่งจะเข้ารหัส คือ AB เพิ่มไว้ในพจนานุกรม

ขั้นตอนที่14 ลบสตริงที่เพิ่งจะเข้ารหัสออกจากส่วนที่จะเข้ารหัสและอ่านข้อมูลเพิ่มเข้ามาใหม่อีก

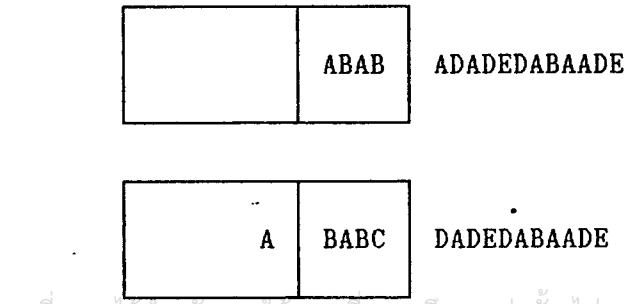
เมื่อจบขั้นตอนที่ 14 รอบที่ 3 แล้ว โครงสร้างของ อัลกอริทึมจะมีลักษณะดังรูปที่ 2.14



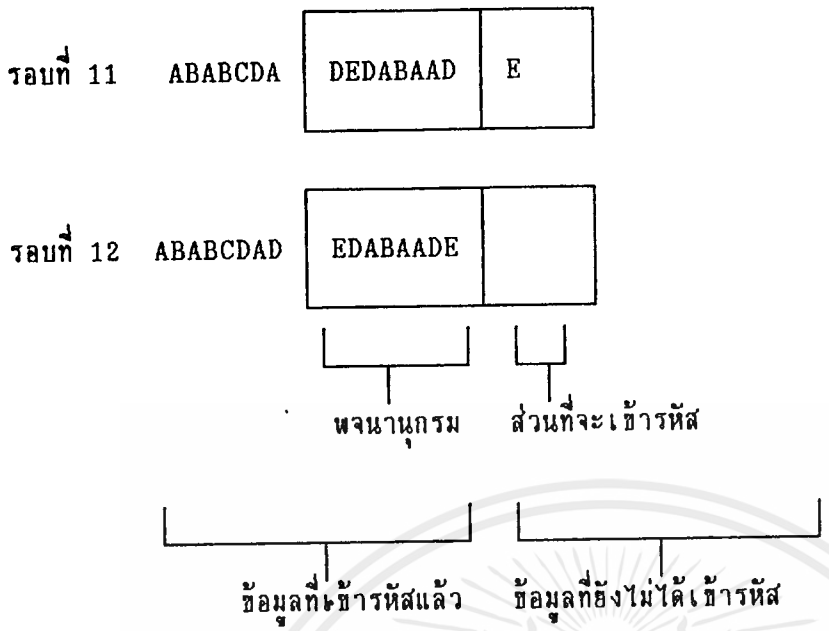
รูปที่ 2.14 โครงสร้างอัลกอริทึมเมื่อผ่านขั้นตอนที่ 14 รอบที่ 3

ขั้นตอนที่15 ยังมีข้อมูลเหลือในส่วนที่จะเข้ารหัสอีก ดังนั้นกลับไปกระทำขั้นตอนที่ 3 ซ้ำอีก

และจะกระทำเช่นนี้ไปเรื่อยๆ จนกว่าจะไม่มีข้อมูลในส่วนที่จะเข้ารหัส นั่นคือข้อมูลได้ถูกเข้ารหัสหมดแล้ว ดังนั้น โครงสร้างของ อัลกอริทึม และรหัสของข้อมูลเมื่อผ่านขั้นตอนที่ 14 รอบต่างๆ จนหมดข้อมูลแล้วสามารถสรุปได้ดังรูปที่ 2.15



รอบที่ 2		AB	ABCD	ADEDABAADE	รหัสเอาพุท "1,B"
รอบที่ 2		ABAB	CDAD	EDABAADE	รหัสเอาพุท "0,2,2"
รอบที่ 3		ABABC	DADE	DABAADE	รหัสเอาพุท "1,C"
รอบที่ 4		ABABCD	ADED	ABAADE	รหัสเอาพุท "1,D"
รอบที่ 5		ABABCDA	DEDA	BAADE	รหัสเอาพุท "0,4,1"
รอบที่ 6		ABABCDAD	EDAB	AADE	รหัสเอาพุท "0,2,1"
รอบที่ 7	A	BABCDADE	DABA	ADE	รหัสเอาพุท "1,E"
รอบที่ 8	ABA	BCDADEDA	BAAD	E	รหัสเอาพุท "0,4,2"
รอบที่ 9	ABAB	CDADEDAB	AADE		รหัสเอาพุท "1,B"
รอบที่ 10	ABABC	DADEDABA	ADE		รหัสเอาพุท "0,2,1"



รูปที่ 2.15 โครงสร้างอัลกอริทึมเมื่อผ่านขั้นตอนที่ 7 รอบต่างๆ

การคำนวณว่า อัลกอริทึมลดขนาดข้อมูลได้อย่างไรนั้นสามารถคำนวณได้คือ ถ้าให้ข้อมูลแต่ละตัวมีขนาด 8 บิทดังนั้นข้อมูลที่ผ่านการลดขนาดข้อมูลแล้วคือ

ABABCDADEDABAADE

ซึ่งมี 16 ไบท์ดังนั้นข้อมูลนี้ใช้จำนวนบิทในการเก็บข้อมูลคือ 8 x 16 = 128 บิท

สำหรับรหัส "0,ระยะห่างของสตริงทั้งสอง,ความยาวของสตริง" ใช้จำนวนบิทในการเก็บรหัสคือ

1 + 3 + 2 = 6 บิท

และรหัส "1,สตริงขนาด 1 ไบต์" ใช้จำนวนบิทในการเก็บรหัสคือ 1 + 8 = 9 บิท

การคำนวณหาขนาดของรหัสรวมทั้งหมดที่ได้จากการลดขนาดข้อมูล

= (6 x 9 บิท) + (7 x 6 บิท)

= 96 บิท

ดังนั้นอัลกอริทึมนี้สามารถลดขนาดข้อมูลลงได้ = 128 - 96 = 32 บิท = 4 ไบท์

การลดขนาดข้อมูลด้วยวิธีพจนานุกรมแบบเลื่อนได้นี้จะสามารถลดขนาดข้อมูลได้มากหรือน้อยขึ้นอยู่กับว่ามีสตริงในส่วนที่จะเข้ารหัส เหมือนกับสตริงในพจนานุกรมมากน้อยเท่าไร ถ้ามีสตริงเหมือนกันมากก็จะสามารถลดขนาดข้อมูลได้มากกว่าข้อมูลที่มีสตริงเหมือนกันน้อย และถ้าสตริงที่เหมือนกันมีขนาดยาวมากก็สามารถลดขนาดข้อมูลได้มากกว่า สตริงที่เหมือนกันแต่มีขนาดสั้นกว่า

สำหรับในกรณีที่ขนาดของเอาพุท(ผลรวมของรหัสทั้งหมด) มีขนาดใหญ่กว่าขนาดของข้อมูลเกิดจากการที่ไม่มีสตริงที่เหมือนกันเลยมากๆทำให้ได้รหัส "1,สตริงขนาด 1 ไบต์" ซึ่งมีขนาด 9 บิทออกเอาพุทมากและ รหัสขนาด 9 บิทนี้ใช้แทนข้อมูลสตริง 1 ไบท์ซึ่งมีขนาด 8 บิท จะเห็นได้ว่ารหัสมีขนาดมาก

กว่าขนาดข้อมูลจริงๆเป็นจำนวน 1 บิท

ดังนั้นข้อมูลที่มีแนวโน้มในลักษณะที่มีสตริงไม่เหมือนกัน ระหว่างพจนานุกรมและส่วนที่จะเข้ารหัสมากกว่าที่จะมีสตริงที่เหมือนกัน ก็อาจทำให้เอาพุทมีขนาดใหญ่กว่าขนาดของข้อมูล

สำหรับวิธีการขยายขนาดของข้อมูลนั้น ทำได้โดยการอ่านบิทพิเศษเข้ามา 1 บิทก่อน ถ้าบิทพิเศษนี้เป็น 1 ก็แสดงว่า รหัสที่ตามมาเป็นสตริงขนาด 1 ไบต์ ให้อ่านสตริงขนาด 1 ไบต์ และนำออกเป็นเอาพุท แต่ถ้าบิทพิเศษนี้เป็น 0 แสดงว่า รหัสที่ตามมาคือ"ระยะห่างของสตริง" และ "ความยาวของสตริงที่เหมือนกัน" ตามลำดับ ก็ให้นำเอาสตริงในพจนานุกรมที่ห่างออกไปเท่ากับ "ระยะห่างของสตริง" และความยาวเท่ากับ "ความยาวของสตริง" ออกเอาพุท หลังจากนำสตริงที่ได้จากการถอดรหัสออกเอาพุทแล้วก็นำเอาสตริงนั้นมาเก็บเพิ่มไว้ในพจนานุกรมด้วย เพื่อนำไปใช้ในการถอดรหัสต่อไป จากนั้นก็จะเริ่มอ่านบิทพิเศษของรหัสตัวต่อไปมาเพื่อถอดรหัสอีก และจะกระทำเช่นนี้ต่อไปจนกระทั่งไม่มีรหัสเหลือ ก็จะได้ข้อมูลกลับมาเหมือนเดิมทุกประการ

สำหรับการขยายขนาดของข้อมูลนั้นสามารถอธิบายได้ดังนี้

```
1         initialize Dictionary              ; ทำให้พจนานุกรมขณะเริ่มต้นว่างเปล่า
2  loop:  read  1 bit to stroed in flag      ; อ่านบิทพิเศษ 1 บิทเข้ามาเพื่อใช้ในการ
                                                พิจารณา
3         if flag = 1                        ; ถ้าบิทพิเศษเป็น 1
3.1           read string 1 byte             ; ให้อ่านสตริงขนาด 1 ไบต์เข้ามา
3.2           output string 1 byte           ; และเอาพุทสตริงขนาด 1 ไบต์นี้
3.3           decoded_string_length = 1      ; ดังนั้นสตริงที่ถอดรหัสได้มีขนาด 1 ไบต์
4         else                               ; หรือถ้าบิทพิเศษเป็น 0
4.1           read distance of_string        ; อ่าน"ระยะห่างของสตริง"
4.2           read length_of_string          ; อ่าน"ความยาวของสตริงที่เหมือนกัน"
4.3           copy  string  in  Dictionary to output; นำเอาสตริงในพจนานุกรม
                    เริ่มจากจุดที่ห่างออกไปเท่ากับ"ระยะห่างของสตริง" และยาวเท่ากับ
                    "ความยาวของสตริงที่เหมือนกัน" ออกเอาพุท
4.4           decoded_string_length  =  length_of_string; ดังนั้นสตริงที่ถอดรหัส
                                        ได้มีขนาดเท่ากับ "ความยาวของสตริงที่เหมือนกัน"
5         update Dictionary (decode_string_length)   ; เก็บสตริงที่ถอดรหัสได้แล้ว
                                                       ไว้ในพจนานุกรม
6         If not end_of_file                 ; ยังมีรหัสเหลืออีกหรือไม่
```

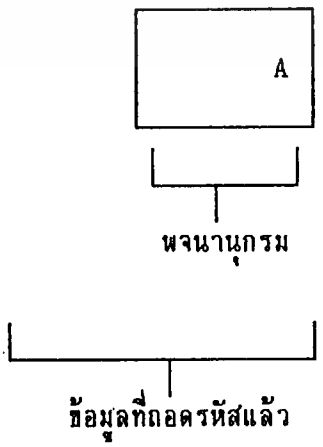
```
7              Repeat  loop              ; ถ้ายังมีเหลือจะไปกระทำการซ้ำ
8         else    Exit                   ; ถ้าไม่มีรหัสเหลืออีกแสดงว่ารหัสถูกถอด
                                           ออกมาเป็นข้อมูลหมดแล้ว
```

ตัวอย่างการถอดรหัสข้อมูลที่ได้ลดขนาดแล้วโดยมีลำดับของรหัสเป็นดังนี้คือ
"1,A","1,B","0,2,2","1,C","1,D","0,4,1","0,2,1","1,E","0,4,2","1,B",
"0,2,1","0,7,2","0,7,1"

และจากที่ได้กำหนดขนาดของพจนานุกรมไว้แล้วคือ 8 ไบต์
เมื่อกระทำตามอัลกอริทึมของการขยายข้อมูลสามารถอธิบายได้ดังนี้

เริ่มต้นด้วยการถอดรหัส "1,A"
ขั้นตอนที่ 1 เริ่มต้นด้วยพจนานุกรมว่างเปล่า
ขั้นตอนที่ 2 อ่านบิทพิเศษเข้ามา 1 บิท คือ 1
ขั้นตอนที่ 3 บิทพิเศษที่อ่านเข้ามาเป็น 1 แสดงว่า รหัสที่ตามเข้ามาคือสตริงขนาด 1 ไบต์
ขั้นตอนที่ 3.1 อ่านสตริงขนาด 1 ไบต์คือ A
ขั้นตอนที่ 3.2 เอาพุทสตริง A
ขั้นตอนที่ 3.2 ดังนั้นสตริงที่ถอดรหัสแล้วมีขนาด = 1 ไบต์
ขั้นตอนที่ 5 เก็บสตริง A ไว้ในพจนานุกรม

เมื่อผ่านขั้นตอนที่ 5 รอบแรกมาแล้วนั้นพจนานุกรมจะมีลักษณะเป็นดังรูปที่ 2.16



รูปที่ 2.16 พจนานุกรมเมื่อผ่านขั้นตอนที่ 5 รอบแรกมาแล้ว

ขั้นตอนที่ 6 ยังมีรหัสเหลืออีกดังนั้น กระทำซ้ำ ขั้นตอนที่2

ต่อไปคือ การถอดรหัส "1,B"

ขั้นตอนที่ 2 อ่านบิทพิเศษเข้ามา คือ 1

ขั้นตอนที่ 3 บิทพิเศษที่อ่านเข้ามาเป็น 1 แสดงว่า รหัสที่ตามเข้ามาคือสตริงขนาด 1 ไบต์

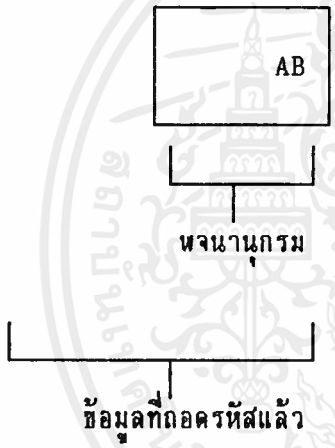
ขั้นตอนที่ 3.1 อ่านสตริงขนาด 1 ไบต์คือ B

ขั้นตอนที่ 3.2 เอาพุทสตริง B

ขั้นตอนที่ 3.3 ดังนั้นสตริงที่ถอดรหัสแล้วมีขนาด = 1 ไบต์

ขั้นตอนที่ 5 เก็บสตริง B ไว้ในพจนานุกรม

เมื่อผ่านขั้นตอนที่ 5 รอบที่ 2 แล้วนั้นพจนานุกรมจะมีลักษณะเป็นดังรูปที่ 2.17



รูปที่ 2.17 พจนานุกรมเมื่อผ่านขั้นตอนที่ 5 รอบ 2 มาแล้ว

ขั้นตอนที่ 6 ยังมีรหัสเหลืออีกดังนั้น กระทำซ้ำ ขั้นตอนที่2

ต่อไปคือ การถอดรหัส "0,2,2"

ขั้นตอนที่ 2 อ่านบิทพิเศษเข้ามา คือ 0

ขั้นตอนที่ 3 บิทพิเศษเข้ามาเป็น 0 แสดงว่ารหัสที่ตามมาคือ"ระยะห่างของสตริง" และ "ความยาวของสตริง"ตามลำดับ

ขั้นตอนที่ 4.1 อ่าน"ระยะห่างของสตริง"เข้ามา คือ 2

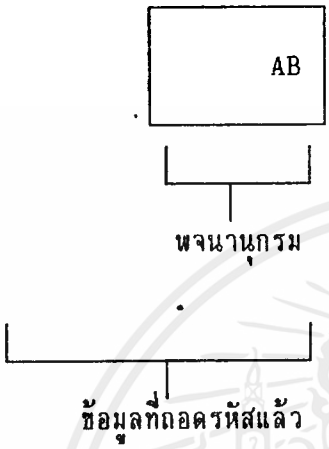
ขั้นตอนที่ 4.2 อ่าน"ความยาวของสตริง"เข้ามา คือ 2

ขั้นตอนที่ 4.3 นำเอาสตริงในพจนานุกรมที่ห่างออกไป 2 ช่อง ยาว 2 ตัว คือ AB ออกเอาพุท

ขั้นตอนที่ 4.4 ดังนั้นสตริงที่ถอดรหัสแล้วจะมีขนาด 2 ไบท์

ขั้นตอนที่ 5 เก็บสตริงที่ถอดรหัสแล้วไว้ในพจนานุกรม

เมื่อผ่านขั้นตอนที่ 5 รอบที่ 3 แล้วนั้นพจนานุกรมจะมีลักษณะเป็นดังรูปที่ 2.18

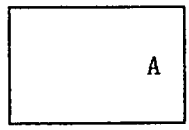


รูปที่ 2.18 พจนานุกรมเมื่อผ่านขั้นตอนที่ 5 รอบ 3 มาแล้ว

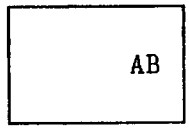
ขั้นตอนที่ 6 ยังมีรหัสข้อมูลเหลืออีกดังนั้นอ่านบิทพิเศษเข้ามาเพื่อถอดรหัสต่อไป และจะกระทำซ้ำเช่นนี้ต่อไปจนกระทั่งไม่มีข้อมูลเหลืออีก ก็จะได้ข้อมูลกลับคืนมาเหมือนเดิมทุกประการ

ดังนั้นผลของการถอดรหัสทั้งหมดสามารถเขียนสรุปได้ดังนี้รูปที่ 2.19

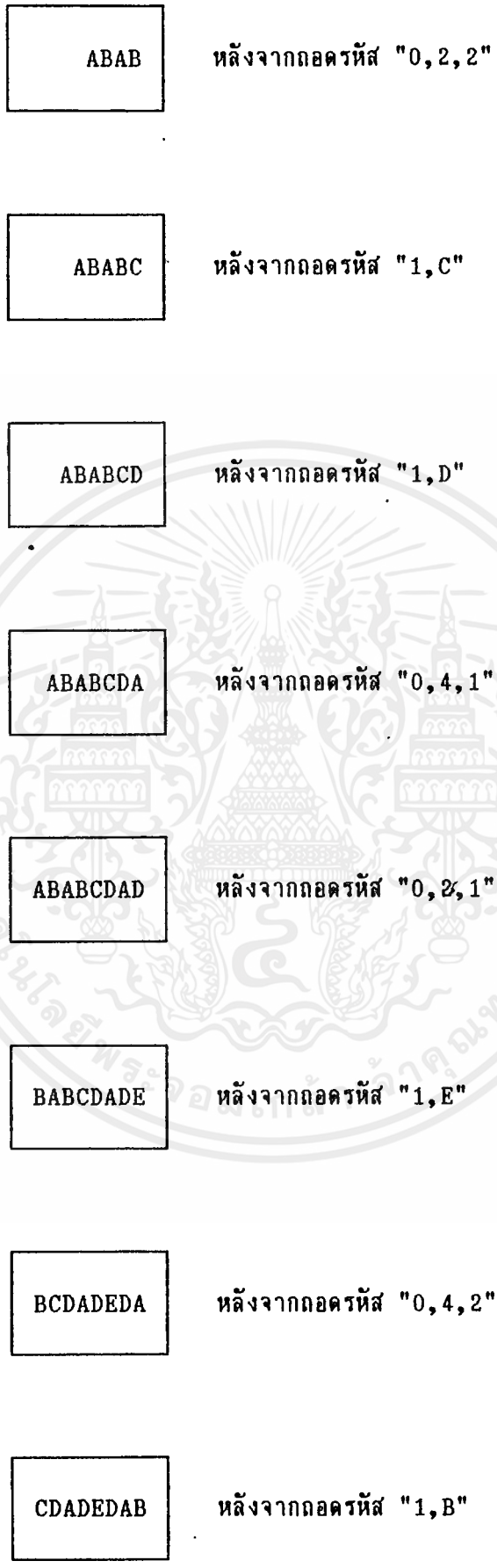

ขณะเริ่มต้น

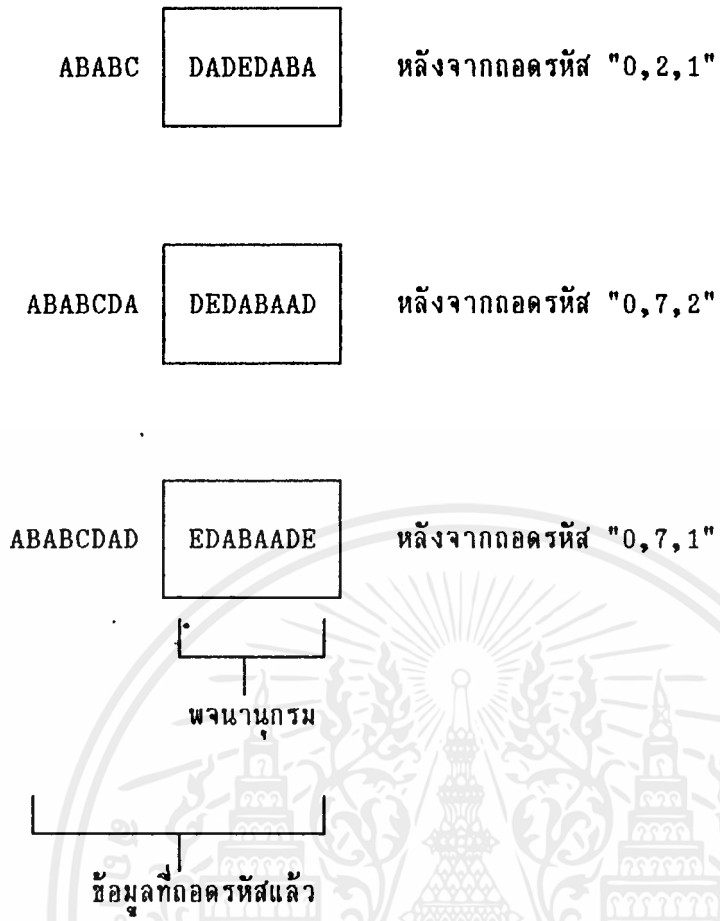


หลังจากถอดรหัส "1,A"



หลังจากถอดรหัส "1,B"





รูปที่ 2.19 แสดงการถอดรหัสข้อมูล

บทที่ 3

ผลการศึกษาเทคนิคการลดขนาดข้อมูลของ PKZIP 1.0

ผลการศึกษาเทคนิคการลดขนาดข้อมูลของ PKZIP 1.0 แบ่งออกเป็น 4 ส่วนคือ

1 ส่วนของโครงสร้าง Zipfile เป็นส่วนที่บรรยายโครงสร้างของแฟ้มข้อมูลที่ถูกลดขนาดแล้ว

2 ส่วนของเทคนิคการเลือกวิธีลดขนาดข้อมูล

3 ส่วนของอัลกอริทึม Shrinking

4 ส่วนของอัลกอริทึม Imploding

ดังมีรายละเอียดต่อไปนี้

3.1 โครงสร้างของ Zipfile

Zipfile คือแฟ้มข้อมูลที่ได้จากการลดขนาดด้วย PKZIP

โครงสร้างของ Zipfile โดยทั่วไป มีลักษณะเป็นดังนี้ คือ

[local file header + data file]...[local file header + data file][central directory] end of central directory record

3.1.1 Local file header

แฟ้มข้อมูลที่ ถูกลดขนาดข้อมูล จะถูกเก็บในรูปของ [local file header + data file] เรียงกันไป ซึ่ง local file header คือส่วนที่บอกลักษณะของแฟ้มข้อมูลที่ถูกลดขนาดข้อมูล data file คือส่วนที่เก็บรหัสของข้อมูล data file นี้เป็นส่วนที่เก็บรหัสของแฟ้มข้อมูลรายละเอียดของ local file header มีดังนี้

local file header:

00-03	local file header signature	4 bytes (0x04034B50)
04-05	version needed to extract	2 bytes
06-07	general purpose bit flag	2 bytes
08-09	compression method	2 bytes
0A-0B	last mod file time	2 bytes
0C-0D	last mod file date	2 bytes
0E-11	crc-32	4 bytes
12-15	compressed size	4 bytes
16-19	uncompressed size	4 bytes

1A-1B	filename length	2 bytes
1C-1D	extra field length	2 bytes
1E-..	filename (variable size)	
	extra field size (variable size)	

3.1.2 Central Directory Structure

Central Directory Structure เป็นส่วนของการเก็บข้อมูลลักษณะของทุกๆแฟ้มข้อมูลที่ถูกลดขนาดแล้วซึ่งถูกเก็บเป็นลำดับต่อเนื่องหลังจาก [local file header + data] ของทุกแฟ้มข้อมูลเก็บใน Zipfile ครบทุกๆแฟ้มข้อมูลแล้ว มีโครงสร้างดังนี้

:

Central directory structure:

[file header]...[file header] end of central dir recored

File header:

00-03	central file header signature	4 bytes (0x02014B50)
04-05	version made by	2 bytes
06-07	version needed to extract	2 bytes
08-09	general purpose bit flag	2 bytes
0A-0B	compression method	2 bytes
0C-0D	last mod file time	2 bytes
0E-0F	last mod file date	2 bytes
10-13	crc-32	4 bytes
14-15	compressed size	2 bytes
16-17	uncompressed size	2 bytes
18-19	filename length	2 bytes
1A-1B	extra field length	2 bytes
1C-1D	file comment length	2 bytes
1E-1F	disk number start	2 bytes
20-21	internal file attributes	2 bytes
22-23	external file attributes	2 bytes
24-27	relative offset of local header	4 bytes

28-..	filename (variable size)	
	extra field (variable size)	
	file comment (variable size)	

End of central dir record:

00-03	end of central dir signature	4 bytes
04-05	number of this disk	2 bytes
	number of the disk with the	
06-07	start of the central directory	2 bytes
	total number of entries in	
08-09	the central dir	2 bytes
0A-0D	size of central directory	4 bytes
	offset of start of central	
	directory with respect to	
0E-11	the starting disk number	4 bytes
12-13	zipfile comment length	2 bytes
14-..	zipfile comment (variable size)	

ฟิลด์ที่เกี่ยวข้องกับการศึกษาเทคนิคการลดขนาดข้อมูลคือ

general purpose bit flag :

บิท 1 : ถ้าใช้วิธีลดขนาดแบบ Imploding
- ถ้าเซทแสดงว่าใช้ พจนานุกรม ขนาด 8 กิโลไบต์
- ถ้าเคลียร์ แสดงว่าใช้ พจนานุกรม ขนาด 4 กิโลไบต์

บิท 2 : ถ้าใช้วิธีลดขนาดแบบ Imploding
- ถ้าเซตแสดงว่าใช้ 3 SF ทรี (Shannon-Fano tree) ในการลดขนาดข้อมูล
- ถ้าเคลียร์ แสดงว่าใช้ 2 SF ทรี

compression method :

0 ไฟล์ถูกเก็บโดยไม่มีการลดขนาด (stored)

1 ไฟล์ถูกลดขนาดโดยวิธีการ Shrinking

6 ไฟล์ถูกลดขนาดโดยวิธี Imploding

crc-32 :

เป็นค่าที่ได้ระหว่างการประมวลผลข้อมูล เพื่อใช้ในการตรวจสอบความผิดพลาดของข้อมูล ตัวอย่างของโครงสร้างของ Zipfile เมื่อมีการลดขนาด 3 แฟ้มข้อมูลมีดังรูป 3.1

```
0000: 50 4B 03 04 0A 00 00 00 01 00 8B 16 90 1A B6 FB
0010: DD 6B 21 00 00 00 22 00 00 00 09 00 00 00 44 41
0020: 54 41 31 2E 44 41 54 61 C4 60 41 43 66 CD 9C 81
0030: 64 CE 24 44 63 06 8D 1C 35 66 EA A0 59 13 66 8E
0040: 9A 81 65 12 96 C9 93 10 50 4B 03 04 0A 00 00 00
0050: 01 00 A5 16 90 1A 6A 5C 8E F1 30 00 00 00 32 00
0060: 00 00 09 00 00 00 44 41 54 41 32 2E 44 41 54 61
0070: D4 98 51 C3 86 4C 1A 3A 64 DE 08 24 43 66 E0 19
0080: 35 67 76 98 99 C3 E6 CC 1C 32 3B 2C BE 29 03 51
0090: 0D 99 39 6A 20 7E 64 23 E7 4D C3 35 6C 20 02 50
00A0: 4B 03 04 0A 00 00 00 01 00 AF 16 90 1A 93 DD FC
00B0: FE 28 00 00 00 2D 00 00 00 09 00 00 00 44 41 54
00C0: 41 33 2E 44 41 54 66 CE A0 11 98 E6 8D C0 33 73
00D0: CE A8 31 43 86 CC 9A 33 64 12 AE 99 C3 46 CD 0B
00E0: 84 01 D7 30 54 03 F1 4C 9A 39 1C 13 82 04 50 4B
00F0: 01 02 0A 00 0A 00 00 00 01 00 8B 16 90 1A B6 FB
0100: DD 6B 21 00 00 00 22 00 00 00 09 00 00 00 00 00
0110: 00 00 00 00 20 00 00 00 00 00 00 00 44 41 54 41
0120: 31 2E 44 41 54 50 4B 01 02 0A 00 0A 00 00 00 01
0130: 00 A5 16 90 1A 6A 5C 8E F1 30 00 00 00 32 00 00
0140: 00 09 00 00 00 00 00 00 00 00 00 20 00 00 00 48
0150: 00 00 00 44 41 54 41 32 2E 44 41 54 50 4B 01 02
0160: 0A 00 0A 00 00 00 01 00 AF 16 90 1A 93 DD FC FE
0170: 28 00 00 00 2D 00 00 00 09 00 00 00 00 00 00 00
0180: 00 00 20 00 00 00 9F 00 00 00 44 41 54 41 33 2E
0190: 44 41 54 50 4B 05 06 00 00 00 00 03 00 03 00 5B
01A0: FF FF FF EE 00 00 00 00 00
```

รูปที่ 3.1 ตัวอย่างของโครงสร้างของ Zipfile

local file header ของแฟ้มข้อมูล DATA1.DAT มีตำแหน่งดังนี้

local file header signature	4 bytes	อยู่ที่ตำแหน่ง 00-03
version needed to extract	2 bytes	อยู่ที่ตำแหน่ง 04-05
general purpose bit flag	2 bytes	อยู่ที่ตำแหน่ง 06-07
compression method	2 bytes	อยู่ที่ตำแหน่ง 08-09
last mod file time	2 bytes	อยู่ที่ตำแหน่ง 0A-0B
last mod file date	2 bytes	อยู่ที่ตำแหน่ง 0C-0D
crc-32	4 bytes	อยู่ที่ตำแหน่ง 0E-11
compressed size	4 bytes	อยู่ที่ตำแหน่ง 12-15
uncompressed size	4 bytes	อยู่ที่ตำแหน่ง 16-19
filename length	2 bytes	อยู่ที่ตำแหน่ง 1A-1B
extra field length	2 bytes	อยู่ที่ตำแหน่ง 1C-1D

filename (variable size) อยู่ที่ตำแหน่ง 1E-26 มีขนาดเท่ากับ filename length

extra field size (variable size) ไม่มี

ดังนั้นตำแหน่งของรหัสข้อมูล คือ 27 ไปจนถึง ตำแหน่ง 27 + uncompressed size - 1 คือ 47 ข้อมูลต่อจากนี้ก็จะเป็น local file header ของแฟ้มข้อมูลต่อไป จะเริ่มต้นที่ตำแหน่ง 48 และ local file header ของแฟ้มข้อมูลสุดท้าย จะเริ่มต้นที่ตำแหน่ง 9F หลังจากนั้นก็จะเป็นการเก็บ central directory structure โดยเริ่มเก็บ file header ของแฟ้มข้อมูล DATA1.DAT ที่ตำแหน่ง 0EE แฟ้มข้อมูลถัดมาเริ่มที่ตำแหน่ง 0125 และแฟ้มข้อมูลสุดท้าย ที่ตำแหน่ง 015C จากนั้นก็เก็บข้อมูลของ End of dir record ซึ่งเริ่มที่ ตำแหน่ง 193 จนกระทั่งถึงจุดสิ้นสุดของแฟ้มข้อมูล

3.2. เทคนิคการเลือกวิธีลดขนาดข้อมูล

ในการเลือกวิธีการลดขนาดข้อมูล PKZIP ใช้วิธีการลดข้อมูลต่างๆ กันตามความเหมาะสมของข้อมูล เช่น ลักษณะของข้อมูลเป็นอย่างไร ขนาดของข้อมูลเป็นเท่าไร

นอกจากนี้ยังมีตัวเลือก (Option) สำหรับให้ผู้ใช้ เลือกวิธีการลดขนาดข้อมูลเองได้ด้วยตนเอง ดังนั้นเงื่อนไขที่ใช้ในการเลือกวิธีการลดขนาดข้อมูล คือ

1. ตัวเลือกและชนิดของข้อมูล
2. ขนาดของแฟ้มข้อมูล

1. ตัวเลือกและชนิดของข้อมูล

ผู้ใช้สามารถเลือกวิธีการลดขนาดข้อมูลโดยใช้ตัวเลือกดังนี้

-ex ใช้วิธี Implode สำหรับทุกแฟ้มข้อมูล ใช้ลดขนาดข้อมูลให้ได้มากที่สุด

-es ใช้วิธี Shrink สำหรับทุกแฟ้มข้อมูล ใช้เวลาในการลดขนาดข้อมูลน้อยที่สุด

-ea ใช้วิธี Shrink สำหรับแฟ้มข้อมูลชนิด ไบนารี (binary file) ทุกแฟ้มข้อมูล
ใช้วิธี Implode สำหรับแฟ้มข้อมูลชนิด ตัวอักษร (text file) ทุกแฟ้มข้อมูล

-eb ใช้วิธี Shrink สำหรับแฟ้มข้อมูลชนิด ตัวอักษร ทุกแฟ้มข้อมูล
ใช้วิธี Implode สำหรับแฟ้มข้อมูลชนิด ไบนารี ทุกแฟ้มข้อมูล

โดยปรกติ PKZIP จะใช้เป็นตัวเลือกที่ -ex ถ้าหากไม่ระบุตัวเลือก

2. ขนาดของแฟ้มข้อมูล

ถ้าใช้วิธีการลดขนาดข้อมูล Imploding และขนาดของแฟ้มข้อมูลน้อยกว่า 320 ไบท์แล้วจะเปลี่ยนมาใช้วิธี Shrinking แทนเพราะว่าถ้าใช้ว่าวิธี Imploding ขนาดของเอาพุทอาจจะมีขนาดใหญ่กว่าแฟ้มข้อมูลอินพุท เพราะวิธี Imploding จะมีส่วนหัว(header)ของรหัสรวมอยู่ด้วย ซึ่งใช้เก็บข้อมูลสำหรับใช้ในการลดขนาดข้อมูลไว้ แต่วิธี Shrinking จะไม่มีส่วนหัวรวมอยู่ด้วย ดังนั้นขนาดของเอาพุทมักจะมีขนาดเล็กกว่าอินพุทเสมอ

โฟลว์ชาร์ทของเทคนิคการเลือกวิธีการลดขนาดข้อมูลดังรูป 3.6-3.7 สามารถอธิบายได้คือเริ่มด้วยการกำหนดค่าในกรณีที่ไม่มีพารามิเตอร์ส่งมา (ค่า default)ให้เป็นดังนี้คือ ให้วิธีการลดขนาดข้อมูลสำหรับแฟ้มข้อมูลประเภทตัวอักษร และวิธีการลดขนาดข้อมูลสำหรับแฟ้มข้อมูลประเภทไบนารีให้เป็น Imploding จากนั้นมาพิจารณาว่ามีพารามิเตอร์ส่งมาหรือไม่ ถ้าไม่มีให้ไปกระทำคำสั่ง ณ จุดเชื่อมต่อที่ 1 (connector 1) ต่อไป แต่ถ้ามีพารามิเตอร์ส่งมา ให้พิจารณาว่าเป็นพารามิเตอร์ที่ใช้เลือกวิธีลดขนาดข้อมูลหรือไม่ พารามิเตอร์ที่ใช้ในการเลือกวิธีการลดขนาดข้อมูล จะนำไปใช้ในการพิจารณาการเลือกวิธีการลดขนาดข้อมูล คือ

ถ้าพารามิเตอร์ คือ "-es" จะเป็นการกำหนดให้วิธีการลดขนาดข้อมูลประเภทตัวอักษร และวิธีการลดขนาดข้อมูลประเภทไบนารี คือ Shrinking

ถ้าพารามิเตอร์ คือ "-ex" จะเป็นการกำหนดให้วิธีการลดขนาดข้อมูลประเภทตัวอักษร และวิธีการลดขนาดข้อมูลประเภทไบนารี คือ Imploding

ถ้าพารามิเตอร์ คือ "-ea" จะเป็นการกำหนดให้วิธีการลดขนาดข้อมูลประเภทตัวอักษร คือ Imploding และวิธีการลดขนาดข้อมูลประเภทไบนารี คือ Shrinking

ถ้าพารามิเตอร์ คือ "-eb" จะเป็นการกำหนดให้วิธีการลดขนาดข้อมูลประเภทตัวอักษร คือ Shrinking และวิธีการลดขนาดข้อมูลประเภทไบนารี คือ Imploding

หลังจากกำหนดวิธีการลดขนาดข้อมูลได้แล้วให้ไปกระทำคำสั่ง ณ จุดเชื่อมต่อที่ 1 ที่จุดเชื่อมต่อที่ 1 จะพิจารณาเลือกวิธีการลดขนาดข้อมูล โดยพิจารณาจากประเภทของแฟ้มข้อมูลที่วิเคราะห์ได้ ซึ่งเทคนิคการวิเคราะห์แฟ้มข้อมูลจะได้กล่าวถึงในรายละเอียดต่อไป ถ้าแฟ้มข้อมูลคือแฟ้มข้อมูลประเภทตัวอักษรก็จะเลือกใช้วิธีลดข้อมูล ประเภทตัวอักษรที่ได้จากการกำหนดไว้แล้ว จากการระบุพารามิเตอร์สำหรับแฟ้มข้อมูลประเภทไบนารี ก็จะเลือกใช้วิธีที่ได้กำหนดไว้แล้วเช่นกัน หลังจากนั้นมาพิจารณาขนาดของแฟ้มข้อมูล ในกรณีสำหรับวิธีการลดขนาดข้อมูล Imploding ถ้าหากขนาดของแฟ้มข้อมูลน้อยกว่า 320 ไบท์ แล้วให้เปลี่ยนไปใช้วิธีการลดขนาดข้อมูล Shrinking แทน หลังจากนั้นจะเริ่มเข้าสู่ขบวนการลดขนาดข้อมูลตามวิธีการที่เลือกได้

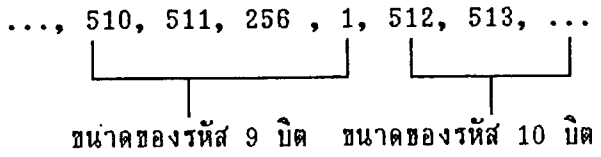
เทคนิคการวิเคราะห์ของแฟ้มข้อมูลสามารถอธิบายได้ดังนี้คือ เริ่มด้วยการวิเคราะห์ว่าชนิดของแฟ้มข้อมูลเป็นเท่าไร ถ้าขนาดของแฟ้มข้อมูลมีขนาดน้อยกว่า 3 กิโลไบท์ ให้จุดเริ่มของการวิเคราะห์เริ่มที่ไบท์ที่ 0. และขนาดของข้อมูลที่จะวิเคราะห์ คือขนาดของแฟ้มข้อมูลนั้น ถ้าขนาดของแฟ้มข้อมูลมีขนาดมากกว่าหรือเท่ากับ 3 กิโลไบท์ แต่ไม่เกิน 9 กิโลไบท์ ให้จุดเริ่มต้นของการวิเคราะห์เริ่มที่ไบท์ที่ 0 และขนาดของข้อมูลที่จะวิเคราะห์คือ 3 กิโลไบท์ ถ้าขนาดของแฟ้มข้อมูลมีขนาดมากกว่าหรือเท่ากับ 9 กิโลไบท์ ให้จุดเริ่มต้นของการวิเคราะห์เริ่มที่ไบท์ที่ 6144 (6 กิโลไบท์)และขนาดของข้อมูลที่จะวิเคราะห์คือ 3 กิโลไบท์ จากนั้นจึงมีการคำนวณ น้ำหนักเฉลี่ยของข้อมูลที่จะวิเคราะห์ โดยที่น้ำหนักของข้อมูลแต่ละตัวได้มาจากการกำหนดขึ้นเอง ถ้าน้ำหนักเฉลี่ยของข้อมูลที่วิเคราะห์น้อยกว่าค่ามาตราฐานคือ 8 แสดงว่าแฟ้มข้อมูลนั้น เป็นแฟ้มข้อมูลประเภทตัวอักษร ถ้าน้ำหนักเฉลี่ยของข้อมูลที่วิเคราะห์ได้มากกว่าหรือเท่ากับ 8 แสดงว่าแฟ้มข้อมูลนั้นเป็นแฟ้มข้อมูลประเภทไบนารี ดังโฟลว์ชาร์ทรูปที่ 3.8-3.9

3.3. เทคนิคการลดขนาดข้อมูลชนิด Shrinking

Shrinking เป็นวิธีการลดขนาดโดยใช้อัลกอริทึมของ LZWโดยมีการเพิ่มเทคนิคพิเศษคือเคลียร์ข้อมูลบางส่วนเมื่อตารางเต็ม โดยที่ขนาดของรหัสเริ่มจาก 9 บิตและสูงสุด 13 บิต Shrinking ต่างจาก LZW ทั่ว ๆไป คือ

1.เมื่อขนาดของรหัสที่ใช้มากกว่าขนาดของรหัสที่กำหนดไว้(Current Code Size) โปรแกรมลดขนาดข้อมูลจะส่งรหัสควบคุม "256,1" เป็นการระบุว่าต่อไปจะส่งรหัสที่มีขนาดมากขึ้นกว่าเดิม เมื่อทางด้านโปรแกรมขยายขนาดข้อมูลได้รับรหัส "256,1"ก็จะรับทราบการเพิ่มขนาดของรหัสที่ส่งมา เช่น ถ้ากำหนดให้ใช้รหัสขนาด 9 บิต ค่าของรหัสที่มีค่าสูงสุดคือ 511 (ฐานสิบ) หรือ 1FF (ฐานสิบหก) เมื่อรหัส 511 ถูกใช้ไปเรียบร้อยแล้ว รหัสที่ใช้ต่อไปคือ 512 ซึ่งมากกว่าค่าของรหัสสูงสุดของรหัสที่กำหนดไว้ ดังนั้น ก่อนที่จะส่งรหัส 512 นั้น ต้องมีการส่งรหัส "256,1" ไปก่อน

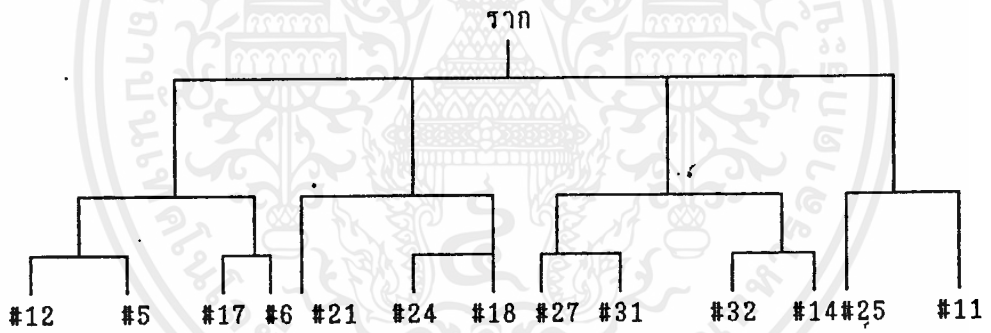
ตัวอย่าง ลำดับของรหัสเมื่อมีการเพิ่มขนาดของรหัสจะมีลักษณะดังรูป 3.2



รูปที่ 3.2 แสดงการเพิ่มขนาดของรหัสด้วยรหัสควบคุม"256,1"

2.เมื่อตารางที่ใช้เต็ม Shrinking จะไม่เคลียร์ตารางข้อมูลเก่าทั้งหมด แต่จะเคลียร์เป็นบางส่วน โดยส่งรหัสควบคุม "256,2" เป็นสัญญาณบอกการเคลียร์ตารางบางส่วน โดยการเคลียร์ทุก ๆ ใบ (leaf node)ของ LZ ทรี และใช้ขนาดของรหัสที่กำหนดไว้ต่อไป โดยที่ใบของLZ ทรีจะถูกนำกลับมาใช้ใหม่ (reused) โดยที่ใบที่มีค่าของรหัสต่ำที่สุดถูกใช้ก่อน ส่วนใบที่มีค่ารหัสสูงสุดจะถูกใช้หลังสุด

ตัวอย่าง การเคลียร์ใบของ LZ ทรีจะมีลักษณะดังรูป 3.3



รูปที่ 3.3 แสดงการเคลียร์ใบของ LZ Tree

โหนดที่มีเครื่องหมาย # นำหน้าทั้งหมดคือใบ เมื่อใบถูกเคลียร์แล้วลำดับของโหนดที่จะถูกนำกลับไปใช้ใหม่ตามลำดับคือ #5,#6,#11,#12,#14,#17,#18,#21,#24,#25,#27,#31และ#32

อัลกอริทึมของ Shrinking สามารถอธิบายได้ดังนี้คือ

```
initialize table                          ; เริ่มด้วยการเคลียร์ตาราง
read first data and store in lastcode     ; อ่านข้อมูลตัวแรกเข้ามาให้เป็น w
                                            (lastcode)
code = 257                                ; กำหนดให้ค่าตัวชี้เริ่มต้น(index)คือ 257
```

```
     codesize = 9                          ; กำหนดให้ขนาดของรหัสเริ่มต้น คือ 9
     maxcode = (1 << codesize) - 1         ; และค่าตัวชี้สูงสุดสำหรับขนาดเริ่มต้นคือ 2°-1
loop:if end of file                        ; ถ้าข้อมูลหมดแล้ว
          exit                             ; ให้สิ้นสุดการทำงาน
     else                                  ; ถ้าข้อมูลยังไม่หมด
          read data and stored in suffix ; อ่านข้อมูลเข้ามาและกำหนดให้เป็น K
                                              (suffix)
          if table full                 ; ถ้าตารางเต็ม
               output lastcode          ; ให้เอาพุท w
               output "256,2"           ; ให้เอาพุทรหัสควบคุม "256,2"
               call partial clear       ; เรียกโปรแกรมย่อยซึ่งทำหน้าที่เคลียร์ใบของทรี
               lastcode = suffix        ; ให้ w = K
          else                          ; ถ้าตารางยังไม่เต็ม
               if lastcode+suffix is in table  ; ถ้ามี w+K ในตารางแล้ว
                    lastcode = index of lastcode+suffix ; ให้ w = ตัวชี้ของw+K
               else                     ; ถ้าไม่มี w+K ในตาราง
                    output lastcode     ; เอาพุท w
                    index of lastcode+suffix = code ; ให้นำเอา w+K เก็บลงใน
                                                      ตารางโดยเก็บตัวชี้ของ w+K ไว้ในตาราง
                    code = code + 1     ; เพิ่มค่าตัวชี้อีกหนึ่ง
                    lastcode = suffix ; ให้ w = K
                    if code > maxcode and codesize < 13 ; ถ้าค่าของตัวชี้มากกว่า
                                                          ค่าตัวชี้สูงสุดสำหรับรหัสปัจจุบันและขนาดของรหัส
                                                          ไม่เกิน 13
                         output "256,1" ; เอาพุทรหัสควบคุม "256,1"
                         codesize = codesize + 1 ; เพิ่มขนาดของรหัส
                         maxcode = ( 1 << codesize ) -1  ; ปรับค่าสูงสุดสำหรับ
                                                           ขนาดของรหัสใหม่
     repeat loop          ; กระทำซ้ำจนกว่าข้อมูลจะหมด
```

จะเห็นได้ว่า อัลกอริทึมของ Shrinking มีลักษณะคล้าย LZW มากมีส่วนที่เพิ่มเข้ามาคือ

1 มีการใช้รหัสควบคุม "256,1" สำหรับควบคุมการเพิ่มขนาดของรหัส และรหัสควบคุม

"256,2" สำหรับควบคุมการเคลียร์ใบของทรี (เคลียร์ข้อมูลบางส่วน)

2 โปรแกรมย่อยที่ทำหน้าที่เคลียร์ใบของทรีดังที่อธิบายไปแล้วข้างต้น

อัลกอริทึมของการขยายขนาดข้อมูล Unshrinking สามารถอธิบายได้ดังนี้

```
    initialize table          ; เริ่มต้นด้วยการเคลียร์ตาราง
    index = 257               ; กำหนดค่าตัวชี้เริ่มต้น คือ 257
    codesize = 9              ; กำหนดขนาดรหัสเริ่มต้น คือ 9
    maxcode = ( 1 << codesize ) -1 ; กำหนดค่าตัวชี้สูงสุดสำหรับขนาดรหัสเริ่มต้น=2^-1
    read first code and store in oldcode ;อ่านข้อมูลตัวแรกเข้ามาเก็บไว้ในoldcode
    finchar = oldcode         ; ให้ finchar = oldcode
loop1:
    if end of code            ; ถ้าไม่มีรหัสเหลือ
        exit                  ; ให้สิ้นสุดการทำงาน
    else                      ; ถ้ามีรหัสเหลือ
        read code and store in code   ; อ่านรหัสเข้ามาเก็บไว้ที่ code
        if code = 256                 ; ถ้า code เป็นรหัสควบคุม
            read code and store in code   ; อ่านรหัสควบคุมตัวถัดมาเก็บไว้ที่code
            if code = 1                   ; ถ้าเป็น 1
                codesize = codesize + 1   ; ให้เพิ่มขนาดของรหัสที่ใช้อยู่
            else                          ; ถ้าเป็น 2
                call partial clear        ; เรียกโปรแกรมย่อยซึ่งทำหน้าที่เคลียร์
                                            ใบของทรี
        else                              ; ถ้า code ไม่เป็นรหัสควบคุม
            incode = code                 ; ให้ incode = code
            if code is not in table       ; ถ้า incode ไม่มีในตาราง แสดงว่า
                                            เกิดกรณีพิเศษขึ้น
                push finchar in to stack ; ให้เก็บ finchar ลงในแสตก
                code = oldcode           ; ให้ code = oldcode
loop2:      if code > 257                ; ถ้า code ยังอยู่ในรูป w+K
                push suffix of code in to stack    ; เก็บ K ลงในแสตก
                code = prefix of code    ; ให้ code = w
                repeat loop2             ; กระจาย w ลงมาให้หมด
```

```
            finchar = suffix of code       ; ให้ finchar = K
            pop all pushed data from stack ; นำข้อมูลที่เก็บลงแสตกออกเอาพุท
                 to output
            add oldcode+finchar in table   ; แล้วเพิ่ม w+K ลงในตาราง
            oldcode = incode               ; ให้ oldcode = incode
    repeat loop1                           ; กระทำซ้ำ loop1
```

จะเห็นว่าอัลกอริทึมของ Unshrinking คล้ายกับ อัลกอริทึมของ LZW การขยายขนาดข้อมูลกลับมาก โฟลว์ชาร์ท แสดงการทำงานของโปรแกรม Shrinking และ Unshrinking ที่จำลองการทำงานมาจาก โปรมแกรม PKZIP มีดังรูป 3.10 - 3.13

3.4. การลดขนาดข้อมูลชนิด Imploding

Imploding ใช้อัลกอริทึมการลดขนาดข้อมูล 2 ชนิดต่อกันเป็น 2 ตอน โดย ตอนแรกใช้อัลกอริทึมของพจนานุกรมแบบเลื่อนได้ ลดจำนวนของสตริงที่ซ้ำกัน ส่วนตอนที่ 2 ใช้ SF ทรี (multiple SF tree) เข้ารหัสของรหัสเอาท์พุตของอัลกอริทึมพจนานุกรมแบบเลื่อนได้ ดังรูป 3.4

รูปที่ 3.4 เทคนิคการลดขนาดข้อมูลด้วยวิธี Imploding

ขนาดของ พจนานุกรม ที่ใช้คือ 4 กิโลไบท์ หรือ 8 กิโลไบท์ และจำนวนของ SF ทรี ที่ใช้ คือ 2 หรือ 3 ทรี Implodingเลือกใช้ขนาดของพจนานุกรมและจำนวนของ SF ทรีจากการวิเคราะห์ข้อมูลของแฟ้มที่จะถูกลดขนาด คือ

1. ถ้าข้อมูลเป็นประเภทตัวอักษรและมีขนาดอย่างน้อย 5632 ไบท์จะเลือกใช้พจนานุกรมขนาด8 กิโลไบท์และใช้ SF ทรี 3 ทรี

2. ถ้าข้อมูลเป็นประเภทไบนารีหรือตัวอักษรที่มีขนาดน้อยกว่า 5632 ไบต์จะเลือกใช้พจนานุกรม ขนาด 4 กิโลไบท์ และใช้ SF ทรี 2 ทรี

การลดขนาดข้อมูลในตอนแรกมีลักษณะเหมือนกับการลดขนาดข้อมูลด้วยวิธีพจนานุกรมแบบเลื่อนได้ ทุกประการ ดังนั้นจะไม่กล่าวซ้ำอีกแต่จะกล่าวถึงอัลกอริทึมในตอนที่ 2 คือเข้ารหัสด้วย SF ทรี ซึ่งมีวิธีการเหมือนกับการเข้ารหัสด้วยวิธี ฮัฟแมน แต่รายละเอียดต่างไป

3.4.1 การลดขนาดข้อมูลโดยใช้ SF ทรี

SF ทรีมี 3 ชนิด คือ

1. Literal SF ทรี ใช้ในการเข้ารหัสข้อมูลทั้งหมด 256 ตัว

2. Length SF ทรี ใช้ในการเข้ารหัส ส่วนของ"ความยาวของสตริง"ของรหัส"0,ระยะห่างของสตริง,ความยาวของสตริง" ที่ได้จากอัลกอริทึมพจนานุกรมแบบเลื่อนได้ มี 3 แบบ คือ แบบ A แบบ B และแบบ C

3. Distance SFทรีใช้ในการเข้ารหัสส่วนของ"ระยะห่างของสตริง"ของรหัส"0,ระยะห่างของสตริง,ความยาวของสตริง" ที่ได้จากอัลกอริทึมพจนานุกรมแบบเลื่อนได้ มี 3 แบบ คือ แบบ X แบบ Y และ แบบ Z

ถ้า Imploding เลือกใช้ 2 ทรี ทรีที่ถูกใช้ในการเข้ารหัสคือ Length SF ทรี และ Distance SF ทรี

SF ทรี ที่ถูกเก็บในรูปของรหัสข้อมูล ก่อนนำออกมาใช้ต้องมีการถอดรหัสออกมา เพื่อนำไปสร้างเป็น SF ทรี ซึ่งSF ทรีนี้ใช้ทั้งการลดและขยายขนาดของข้อมูล ดังนั้น รหัสข้อมูลของ SF ทรีจึงถูกส่งไปในเอาท์พุตไฟล์ด้วย

การถอดรหัสข้อมูลของ SF ทรี มีวิธีการดังต่อไปนี้คือ

1. รหัสข้อมูลของ ทรี แต่ละไบต์จะถูกถอดรหัสออกมาเป็น 4 บิตล่างคือ จำนวนบิตที่ใช้แทนสำหรับแต่ละข้อมูล +1 (1-16) 4 บิตบน คือ จำนวนของข้อมูลที่มีจำนวนบิตนี้ +1 (1-16)

2. เรียงลำดับข้อมูลตามความยาวของบิทโดยให้ ข้อมูลที่มีความยาวบิทสั้นที่สุดอยู่บนสุด และ ให้ข้อมูลที่มีความยาวบิทที่ยาวที่สุดอยู่ล่างสุด

3. ใช้อัลกอริทึม decode เพื่อสร้างรหัส SF

4. สลับตำแหน่งของบิททุกบิทใน รหัส SF จากบิทที่มีนัยสำคัญสูงสุด มาเป็นบิทที่มีนัยสำคัญต่ำสุด และบิทที่มีนัยสำคัญต่ำสุดมาเป็นบิทที่มีนัยสำคัญสูงสุด เช่นถ้า รหัส SF ที่ได้จากการใช้อัลกอริทึม decode คือ 2C (00101100) เมื่นำมาสลับตำแหน่งจะได้เป็น 34 (00110100)

5. จัดลำดับของ รหัส SF ให้อยู่เหมือนตำแหน่งเดิม ก่อนที่มีการเรียงลำดับความยาวบิทก็จะได้รหัสที่ใช้ในการลดขนาดข้อมูลของ SF ทรี

อัลกอริทึม decode สามารถอธิบายได้ดังนี้

```
     code = 0
     code_increment = 0          ; กำหนดค่าเริ่มต้นที่ใช้ในการสร้างรหัส SF
     last_bit_length = 0
     i = number of SF tree - 1   ; ให้ i = จำนวนของ SF ทรี - 1
loop:if i >= 0                   ;    ถ้าสร้างรหัสยังไม่ครบ
          code = code + code_increment  ; รหัสสำหรับค่า i ใดๆ
          if bit_lenght(i) != last_bit_length   ; ถ้าความยาวของรหัสไม่เท่ากับ
                                                  ความยาวของรหัสปัจจุบัน
               last_bit_length = bit_length(i)  ; ให้ปรับความยาวบิทปัจจุบัน
               code_increment = 1 << (16 - last_bit_length) ; และปรับการเพิ่ม
                                                              ค่าของรหัส SF ทรี
          SF_code(i) = code    ; ดังนั้นจะได้รหัสของ SF ทรีสำหรับค่า i ใดๆ
          i = i + 1            ; ลดจำนวนของรหัส SF ทรีที่ต้องการถอดรหัส
     else exit                 ; แสดงว่าถอดรหัสหมดแล้ว
     repeat loop               ; กระทำซ้ำจนกว่าจะถอดรหัสหมด
```

อัลกอริทึมมีความซับซ้อนมากกว่าปกติ แต่ทำให้ใช้พื้นที่ในการเก็บข้อมูลน้อยกว่าปกติอยู่มาก ตามปกติการเก็บข้อมูลของ ทรี จะใช้ 256 ไบท์ แต่ละไบท์เก็บความถี่ของตัวอักษรแต่ละตัว สำหรับทรี ซึ่งอยู่ในรูปของการเข้ารหัสเดียวกันนี้ใช้จำนวนข้อมูลเพียง 152 ไบต์เท่านั้น

ตัวอย่างแสดงการถอดรหัสของรหัส SF จำนวน 8 ค่า

ให้ข้อมูลของรหัส SF คือ 0X42 , 0X01 , 0X13
0X42 ถอดรหัสได้เป็น รหัส 5 ค่า ซึ่งมีความยาวบิทเท่ากับ 3 บิท
0X01 ถอดรหัสได้เป็น รหัส 1 ค่า ซึ่งมีความยาวบิทเท่ากับ 2 บิท
0X13 ถอดรหัสได้เป็น รหัส 2 ค่า ซึ่งมีความยาวบิทเท่ากับ 4 บิท

สามารถตีความได้ว่ามีข้อมูล 8 ตัว คือ 0-7 มีความยาวของรหัสของข้อมูลเป็น 3,3,3,3, 2,4,4 บิท ตามลำดับ หลังจาก ดำเนินการตามอัลกอริทึม decode ดังกล่าวแล้วได้ผลลัพธ์ออกมาเป็น ดังรูป 3.5

Val	Sorted	Constructed Code	Reversed Value	Order Restored	Original Length
0:	2	1100000000000000	11	101	3
1:	3	1010000000000000	101	001	3
2:	3	1000000000000000	001	110	3
3:	3	0110000000000000	110	010	3
4:	3	0100000000000000	010	100	3
5:	3	0010000000000000	100	11	2
6:	4	0001000000000000	1000	1000	4
7:	4	0000800000000000	0000	0000	4

รูปที่ 3.5 ตัวอย่างแสดงการถอดรหัสของรหัส SF จำนวน 8 ค่า

หลังจากสร้าง SF ทรีตามต้องการได้แล้ว จึงนำเอารหัสที่ได้จากอัลกอริทึมพจนานุกรมแบบเลื่อนได้ มาเข้ารหัสอีกครั้งโดยใช้รหัส SF อัลกอริทึม แสดงการเข้ารหัส โดยใช้ SF ทรีสามารถอธิบายได้ดังนี้คือ

```
        initialize Dictionary              ; ทำให้พจนานุกรมขณะเริ่มต้นว่างเปล่า
        build SF_code                      ; เรียกโปรแกรมย่อยเพื่อสร้างรหัส SF
        read data to stored in look_ahead_buffer; อ่านข้อมูลเข้ามาเก็บไว้ในส่วน
                                                  ที่จะเข้ารหัส
loop:   find  distance and length_of_match_string ; หาระยะห่างของสตริงและ
                                                  ความยาวของสตริงที่เหมือนกันระหว่างพจนานุกรม
                                                  และส่วนที่จะเข้ารหัส
        if length_of_match_string < min_match  ; กรณีที่มีสตริงที่เหมือนกันน้อยกว่า
                                                  ค่าที่กำหนดไว้
            outputbit 1                          ; เอาพุทบิทพิเศษเป็น 1
            if number_of_SF_tree = 2       ; ถ้าจำนวนของทรีที่ใช้คือ 2
              output first_char_of_look_ahead_buffer ;เอาพุทสตริงขนาด1ไบต์
            else                           ; ถ้าจำนวนของทรีที่ใช้คือ 3
              output Literal_code of first_char_of_look_ahead_buffer
```

```
                                    ; เอาพุทรหัสของสตริงขนาด 1 ไบท์
      used_string_length = 1        ; ดังนั้นสตริงที่เข้ารหัสแล้วมีขนาด1ไบต์
else                              ; กรณีที่มีสตริงที่เหมือนกันไม่น้อยกว่าค่าที่กำหนดไว้
      outputbit 0                   ; เอาพุทบิทพิเศษ เป็น 0
      if size_of_dictionary = 8K    ; ถ้าขนาดของพจนานุกรมคือ8กิโลไบท์
        output 7 lower bit of distance ; เอาพุท 7 บิทล่างของระยะห่าง
                                       ของสตริงทั้งสอง
      else                          ; ถ้าขนาดของพจนานุกรมคือ4กิโลไบท์
        output 6 lower bit of distance ; เอาพุท 6 บิทล่างของระยะห่าง
                                       ของสตริงทั้งสอง
      output Length_code of 6 upper bit of distance
                                    ; เอาพุท 6 บิทบนของระยะห่าง
                                      ของสตริงทั้งสอง
      if length_of_match_string >= 63 + min_match ;ถ้าสตริงที่เหมือนกัน
                                                   มากกว่า 63 + ค่าที่กำหนดไว้
        output Length_code of 63    ; เอาพุทรหัสของ 63
        output length_of_match_string - 63 - min_match ;เอาพุทความ
                                                         ยาวของสตริงที่เหลือ
      else
        output length_of_match_string;เอาพุทความยาวของสตริงที่เหมือนกัน
      used_string_length = length_of_match_string  ; ดังนั้นสตริงที่เข้า
                                                     รหัสแล้วมีขนาดเท่ากับความยาวของสตริงที่เหมือนกัน
update Dictionary (used_string_length) ; เก็บสตริงที่เพิ่งจะเข้ารหัสไว้ใน
                                         พจนานุกรม
update look_ahead_buffer (used_string_length) ;  ลบสตริงที่เพิ่งจะเข้า
                                    รหัสออกจากส่วนที่จะเข้ารหัส และอ่านข้อมูลเพิ่มเข้ามาใหม่
if look_ahead_buffer is not empty ; ยังมีข้อมูลเหลือในส่วนที่จะเข้ารหัส
      repeat loop                   ; ดังนั้นจึงวนลูปใหม่
else  exit                          ; ข้อมูลทั้งหมดได้เข้ารหัสเรียบร้อยแล้ว
```

จะเห็นได้ว่า Imploding ลดขนาดข้อมูลโดยใช้พจนานุกรม 4 กิโลไบท์ หรือ 8 กิโลไบท์และใช้ SF ทรี 2 หรือ 3 ทรี และถ้าใช้ 3 ทรี กำหนดให้ความยาวของสตริงที่สามารถเข้ารหัสได้คือ 3

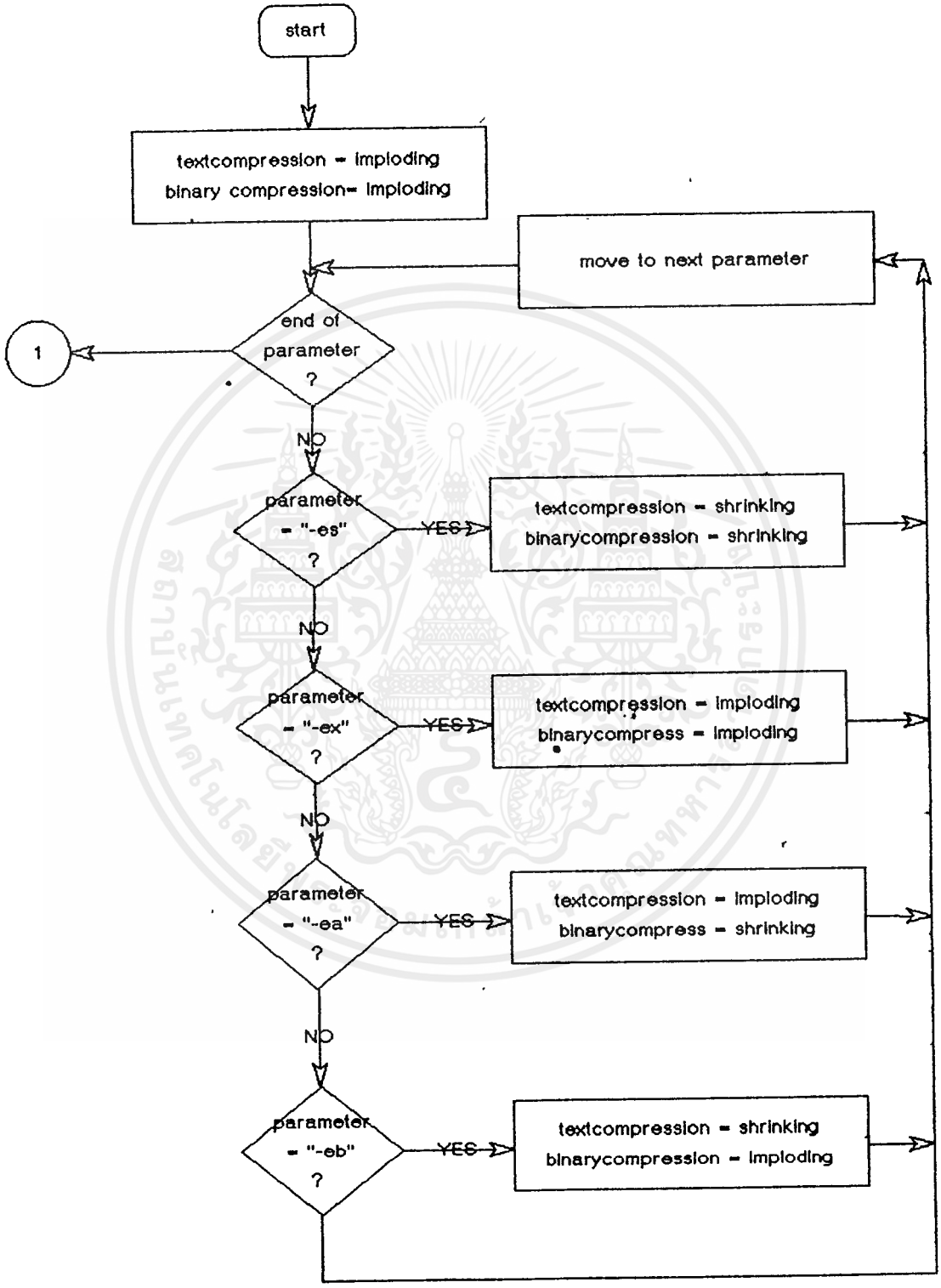
(min_match = 3) แต่ถ้าใช้ 2 ทรี กำหนดให้ความยาวของสตริงที่สามารถเข้ารหัสได้คือ 2 (min_match = 2) ซึ่ง Imploding สามารถลดขนาดข้อมูลได้มากหรือน้อยขึ้นอยู่กับเงื่อนไขอันเดียวกับอัลกอริทึมของ พจนานุกรมแบบเลื่อนได้ คือจะขึ้นอยู่กับจำนวนสตริงที่เหมือนกัน และความยาวของสตริงที่เหมือนกัน มีมากหรือมีน้อย ถ้ามีมากก็จะทำให้สามารถลดขนาดข้อมูลลงได้มาก ถ้ามีน้อยก็จะทำให้สามารถลดขนาดข้อมูลลงได้น้อย นอกจากนี้ยังมีการลดขนาดข้อมูลอีก ตอนหนึ่งโดยใช้ SF ทรี ซึ่งมีลักษณะเหมือน อัลกอริทึมของฮัฟแมนมากแต่ในส่วนนี้ จะทำให้เสียเวลาในการทำงานเพิ่มขึ้น

ในส่วนของการขยายข้อมูลกลับ มีลักษณะตรงข้ามกับการลดขนาดข้อมูล อัลกอริทึมของการขยายขนาดข้อมูลของ Imploding เรียกว่า Exploding ซึ่งอธิบายได้ดังนี้

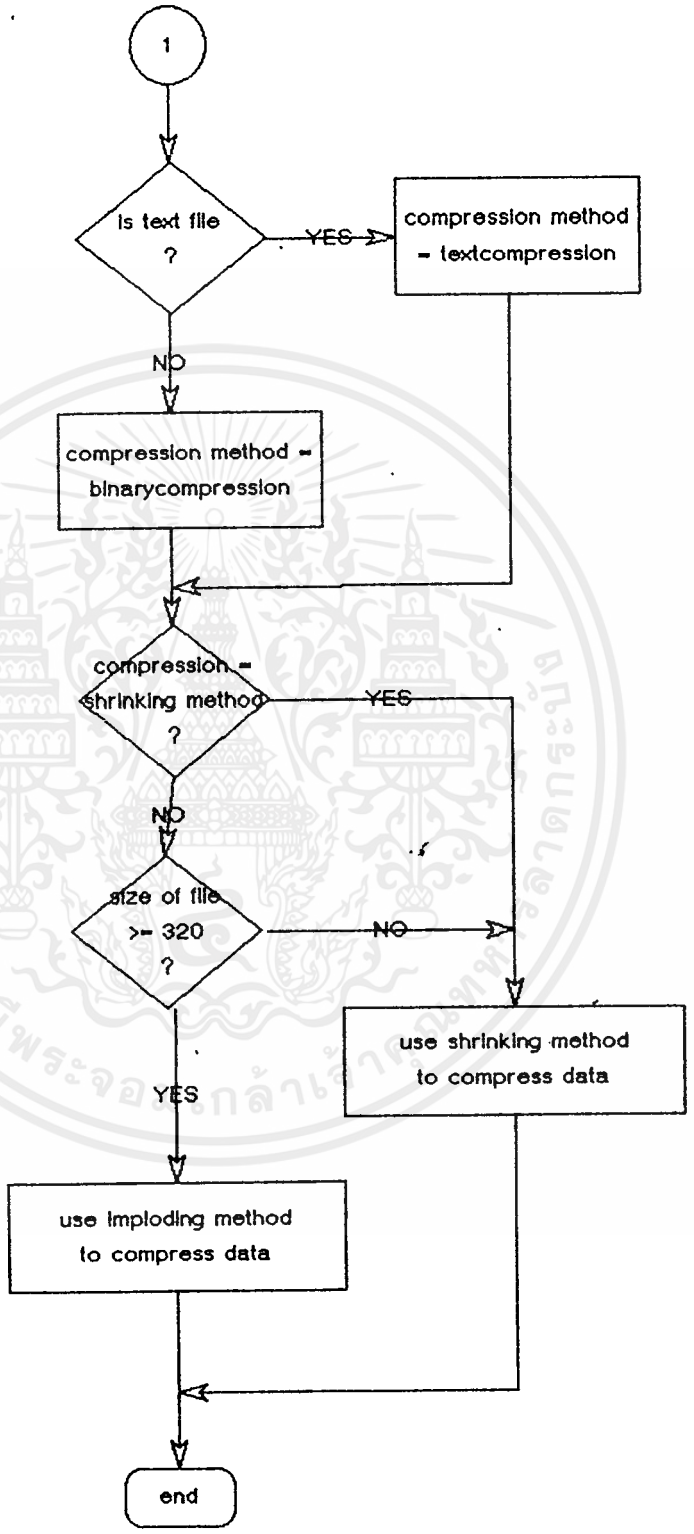
```
        initialize dictionary              ; ทำให้พจนานุกรมขณะเริ่มต้นว่างเปล่า
        build SF_code                      ; เรียกโปรแกรมย่อยเพื่อสร้างรหัส SF
loop:   read  1 bit to stroed in flag      ; อ่านบิทพิเศษ 1 บิทเข้ามาเพื่อใช้ในการ
                                             พิจารณา
        if flag = 1                        ; ถ้าบิทพิเศษเป็น 1
          if number_of_tree = 2            ; ถ้าจำนวนของทรีที่ใช้คือ 2
            read string 1 byte             ; ให้อ่านสตริงขนาด 1 ไบต์เข้ามา
            output string 1 byte           ; และเอาพุทสตริงขนาด 1 ไบต์นี้
          else                             ; ถ้าจำนวนของทรีที่ใช้คือ 3
            decode string 1 byte using Literal_SF_tree ; ถอดรหัสของสตริง
                                                         ขนาด 1 ไบท์จาก SF ทรี
            output string 1 byte           ; และเอาพุทสตริงขนาด 1 ไบต์นี้

          decoded_string_length = 1 ; ดังนั้นสตริงที่ถอดรหัสได้มีขนาด 1 ไบต์
        else                               ; หรือถ้าบิทพิเศษเป็น 0
          if size_of_dictionary = 8K       ; ถ้าขนาดของพจนานุกรมที่ใช้คือ 8 กิโลไบท์
            read 7 lower bit of distance   ; อ่าน 7 บิทล่างของรหัส
                                             "ระยะห่างของสตริง"
          else                             ; ถ้าขนาดของพจนานุกรมที่ใช้คือ 4 กิโลไบท์
            read 6 lower bit of distance   ; อ่าน 6 บิทล่างของรหัส
                                             "ระยะห่างของสตริง"
          decode 6 upper bit of distance using Distance_SF_tree
                                           ; ถอดรหัสของ 6 บิทบนของรหัส
```

```
                                        "ระยะห่างของสตริง"โดยใช้ SFทรี
decode match_string_length using Length_SF_tree
                                        ; ถอดรหัส"ความยาวของสตริง"โดย
                                          ใช้ SF ทรี
if match_string_length = 63             ; ถ้า"ความยาวของสตริง" = 63
   read more data any add to match_string_length
                                        ; อ่านจำนวนข้อมูลที่เกินจากนี้มาบวกเพิ่มอีก
copy  string  in  dictionary to output; นำเอาสตริงในพจนานุกรม
          เริ่มจากจุดที่ห่างออกไปเท่ากับ"ระยะห่างของสตริง" และยาวเท่ากับ
          "ความยาวของสตริงที่เหมือนกัน" ออกเอาพุท
decoded_string_length  =  length_of_string; ดังนั้นสตริงที่ถอดรหัส
                          ได้มีขนาดเท่ากับ "ความยาวของสตริง"
updatedictionary (decode_string_length)  ; เก็บสตริงที่ถอดรหัสได้แล้ว
                                           ไว้ในพจนานุกรม
If not end_of_file                      ; ยังมีรหัสเหลืออีกหรือไม่
   repeat  loop                         ; ถ้ายังมีเหลือจะไปกระทำการซ้ำ
else    exit                            ; ถ้าไม่มีรหัสเหลืออีกแสดงว่ารหัสถูกถอด
                                          รหัสออกมาเป็นข้อมูลหมดแล้ว
```

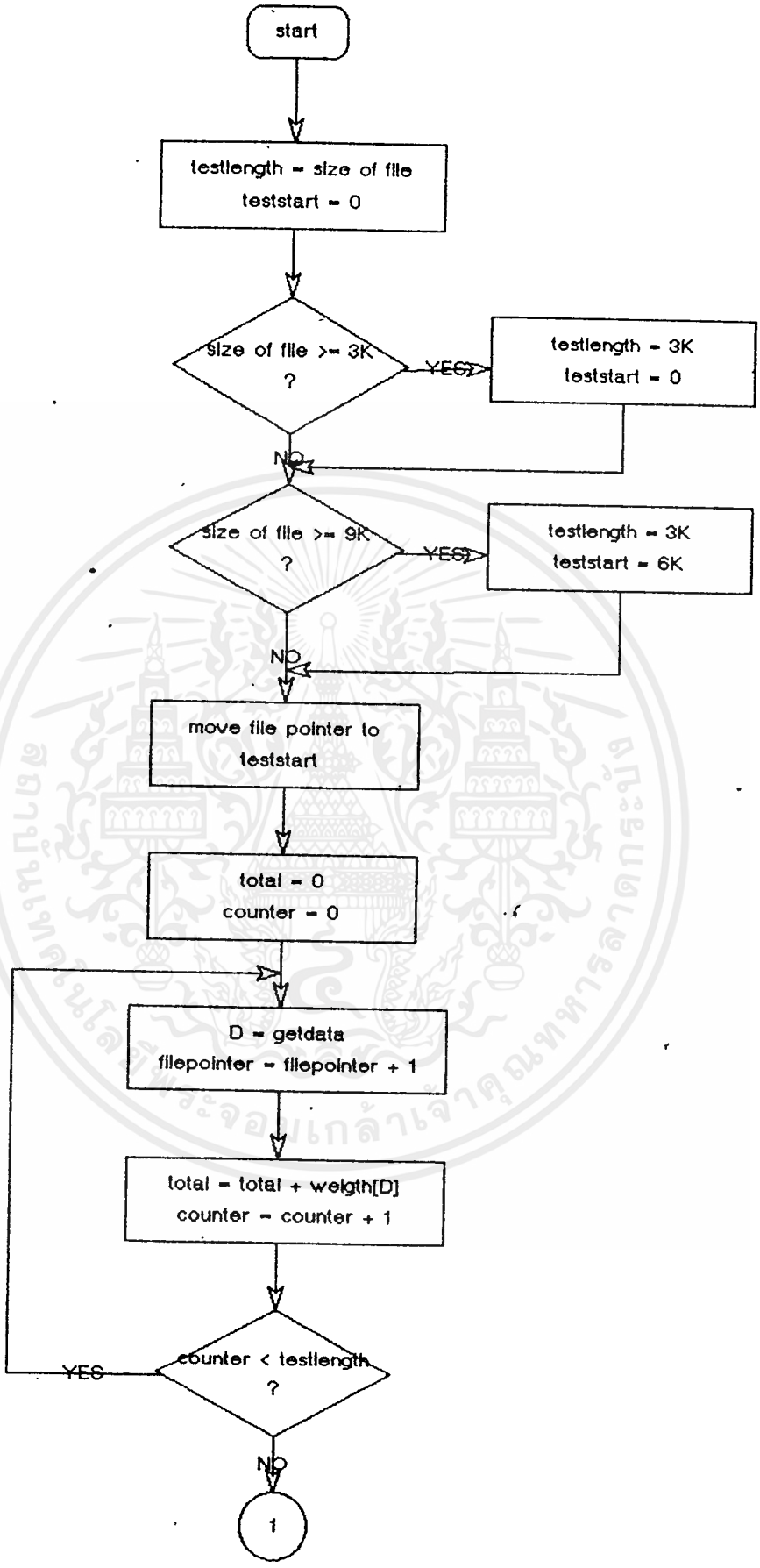
โฟลว์ชาร์ท แสดงการทำงานของโปรแกรม Imploding และ Exploding ที่จำลองการทำงานมาจาก โปรมแกรม PKZIP มีดังรูป 3.15 - 3.18



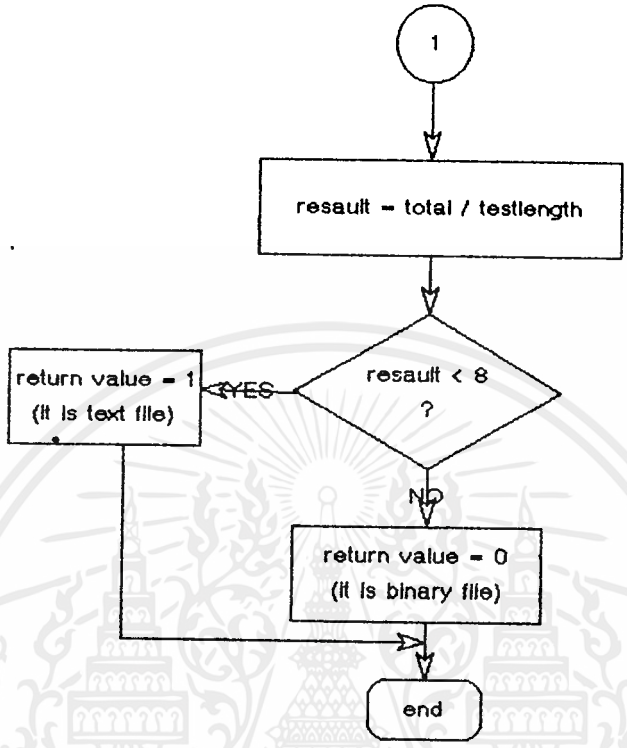
รูปที่ 3.6 โฟลว์ชาร์ทแสดงการเลือกวิธีการลดขนาดข้อมูล



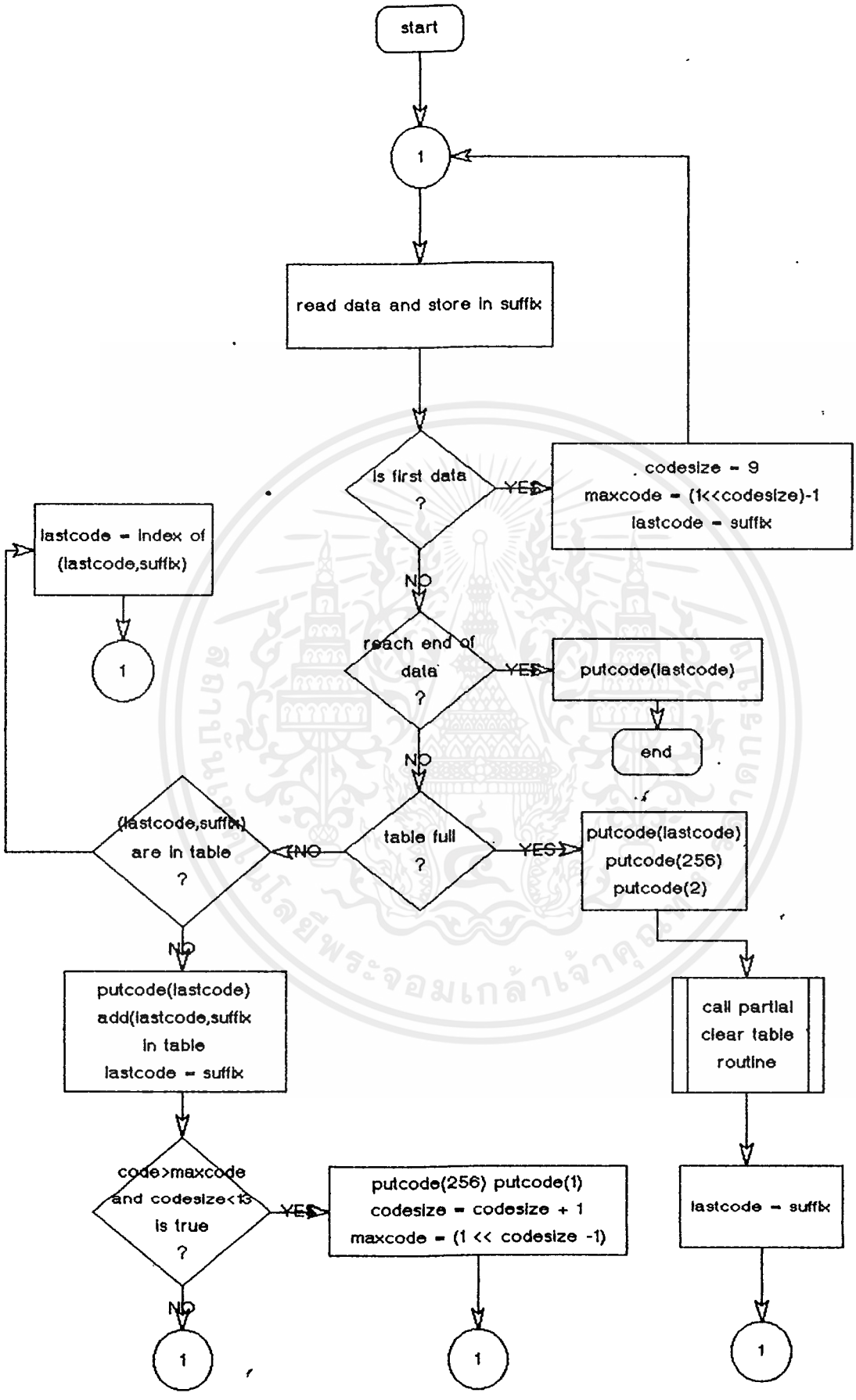
รูปที่ 3.7 โฟลว์ชาร์ทแสดงการเลือกวิธีการลดขนาดข้อมูล



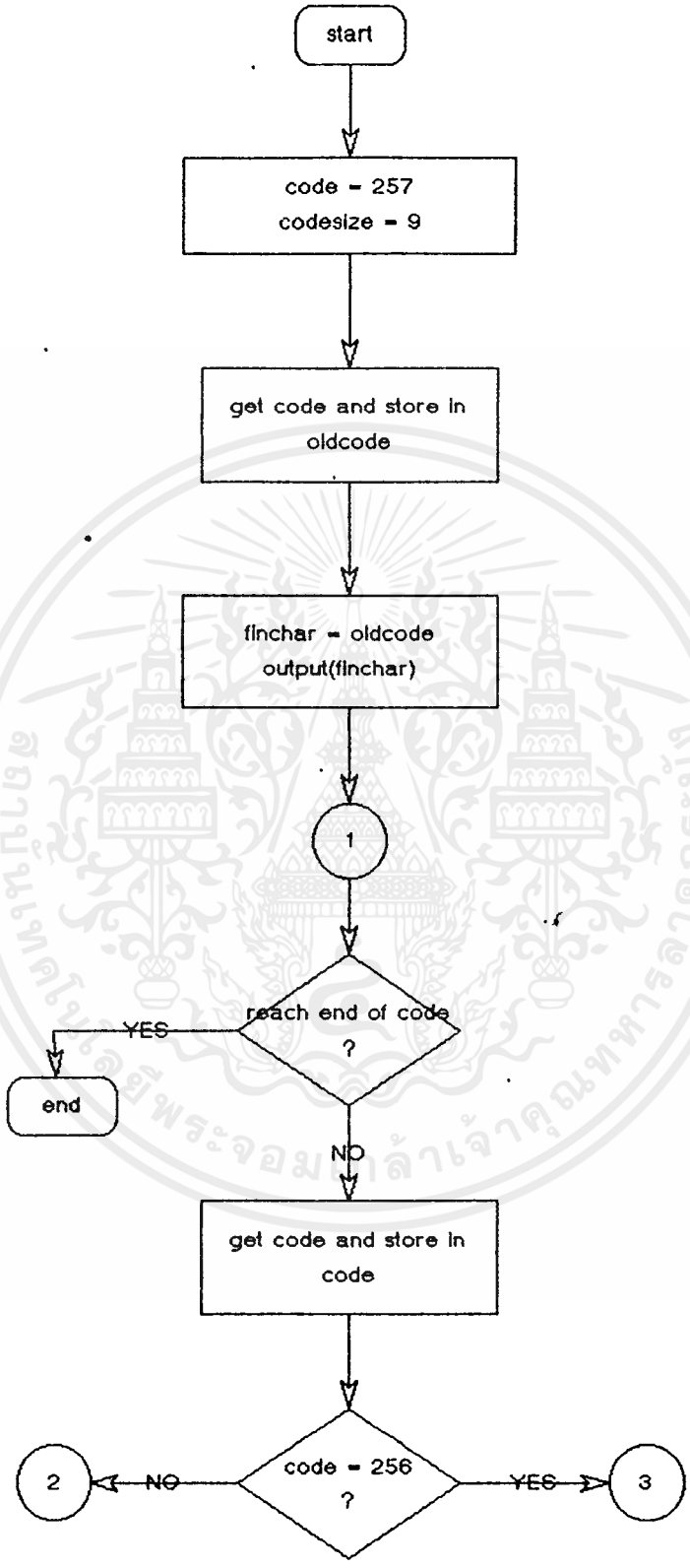
รูปที่ 3.8 โฟลว์ชาร์ทแสดงการวิเคราะห์ชนิดของแฟ้มข้อมูล



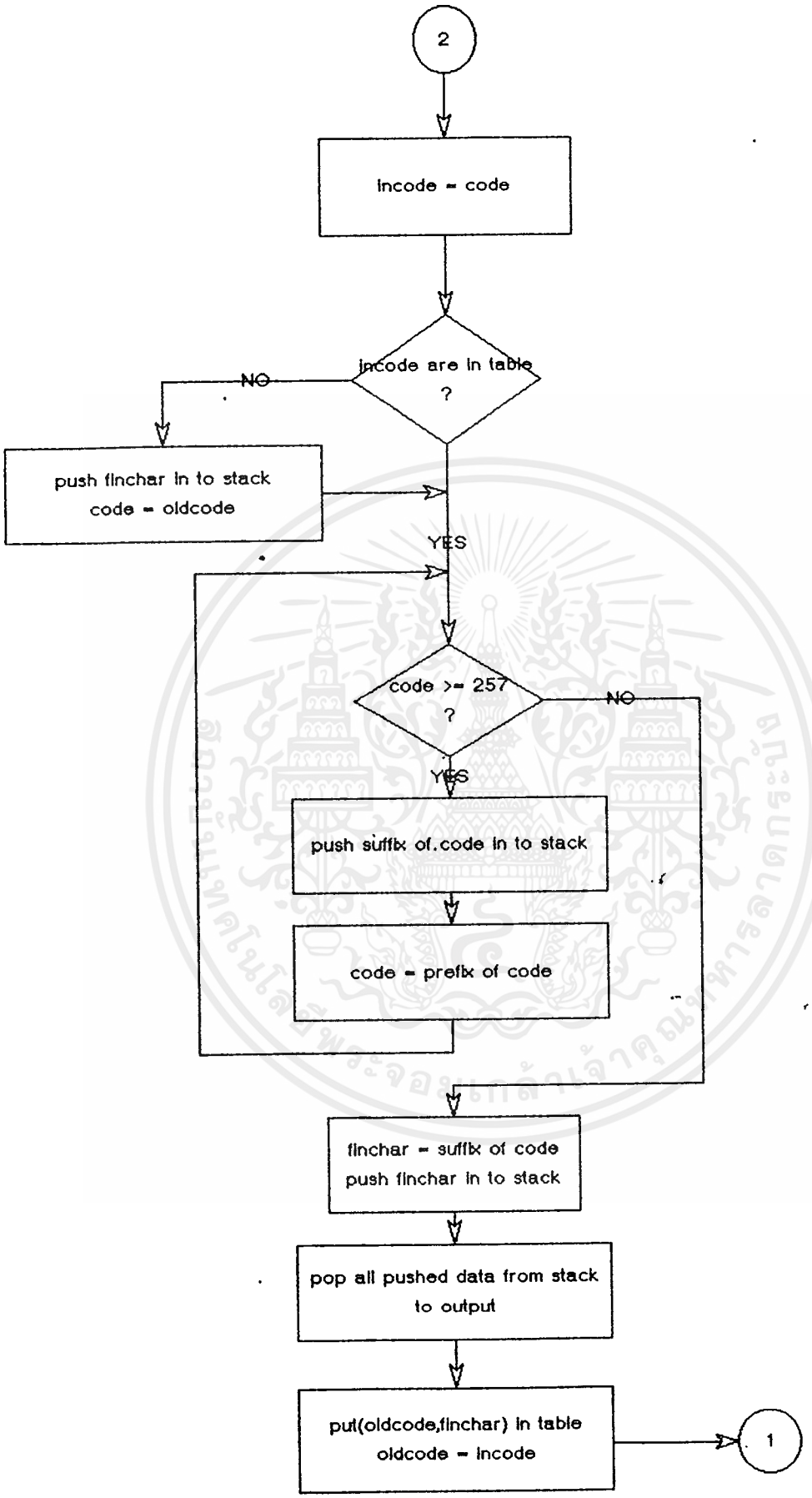
รูปที่ 3.9 โฟลว์ชาร์ทแสดงการวิเคราะห์ชนิดของแฟ้มข้อมูล



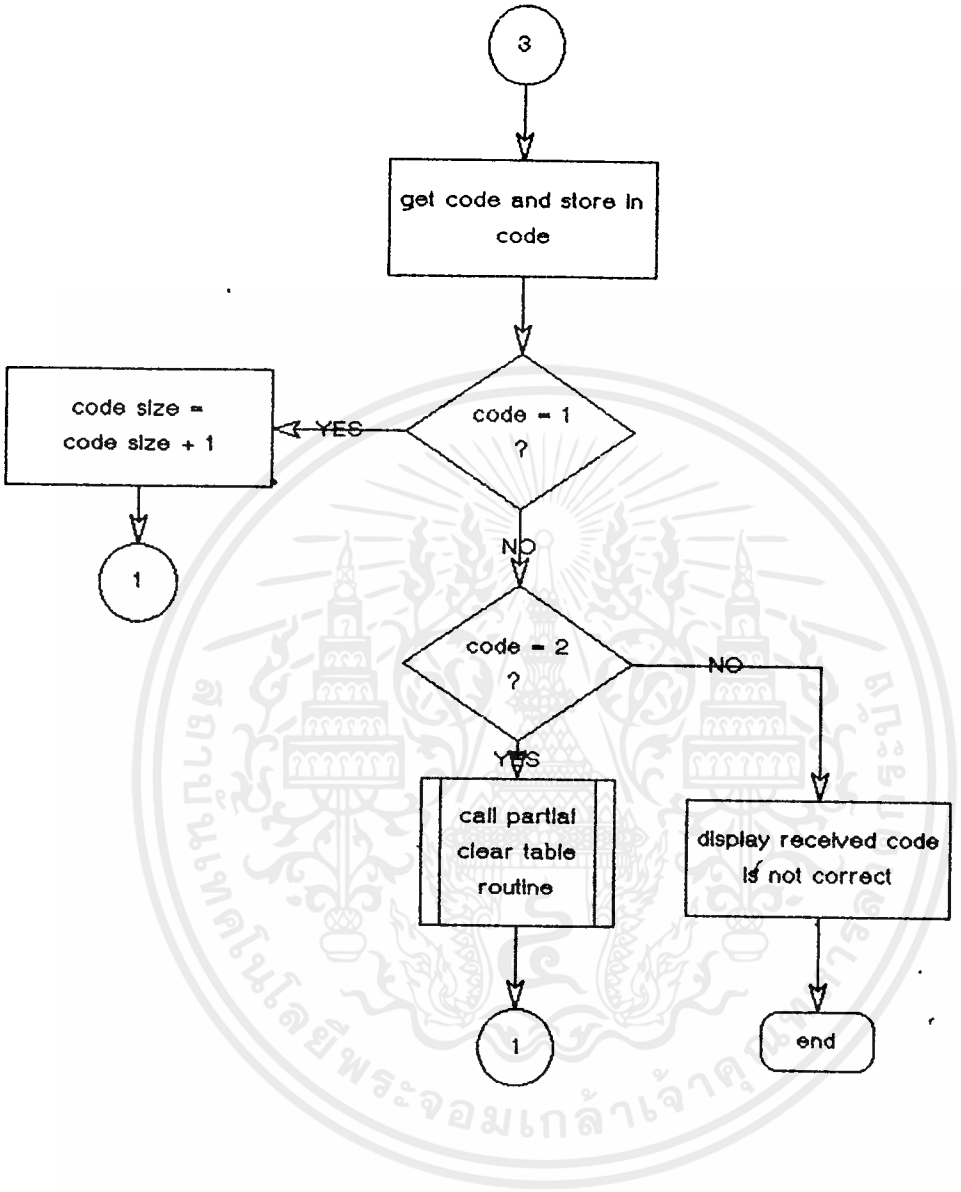
รูปที่ 3.10 โฟลว์ชาร์ทแสดงอัลกอริทึม Shrinking



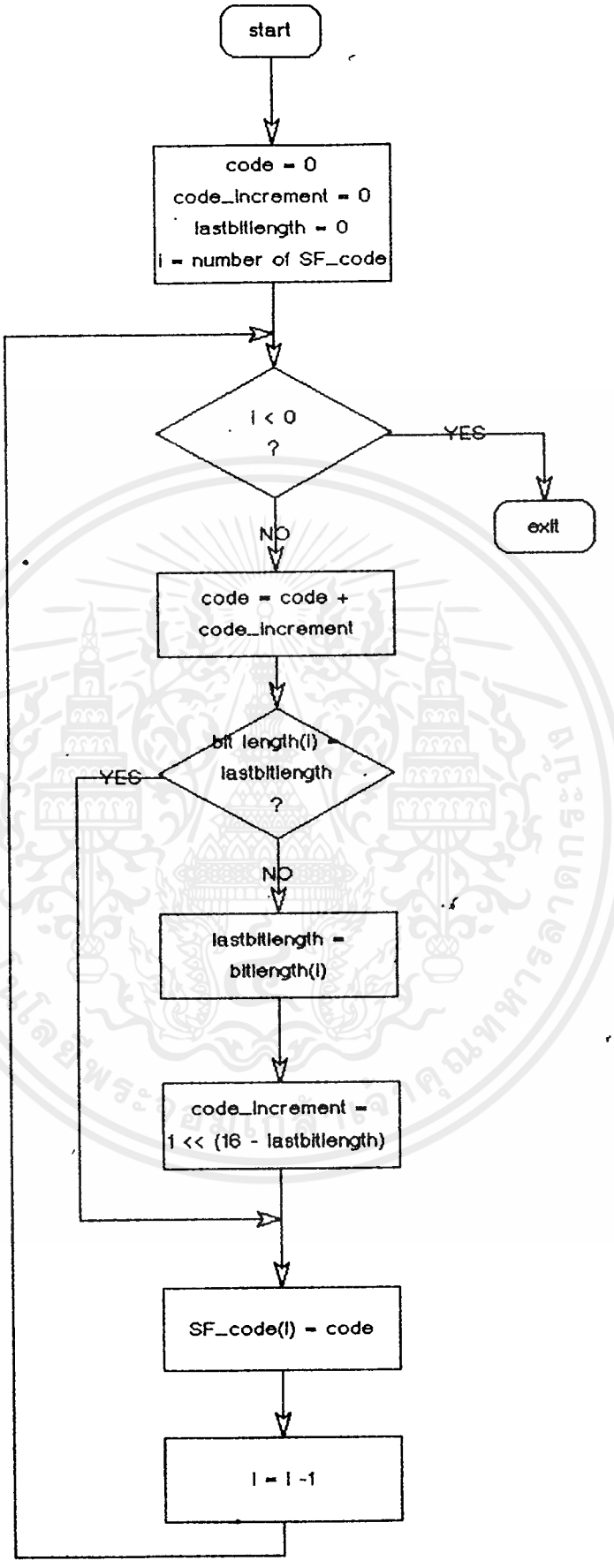
รูปที่ 3.11 โฟลว์ชาร์ทแสดงอัลกอริทึม Unshrinking



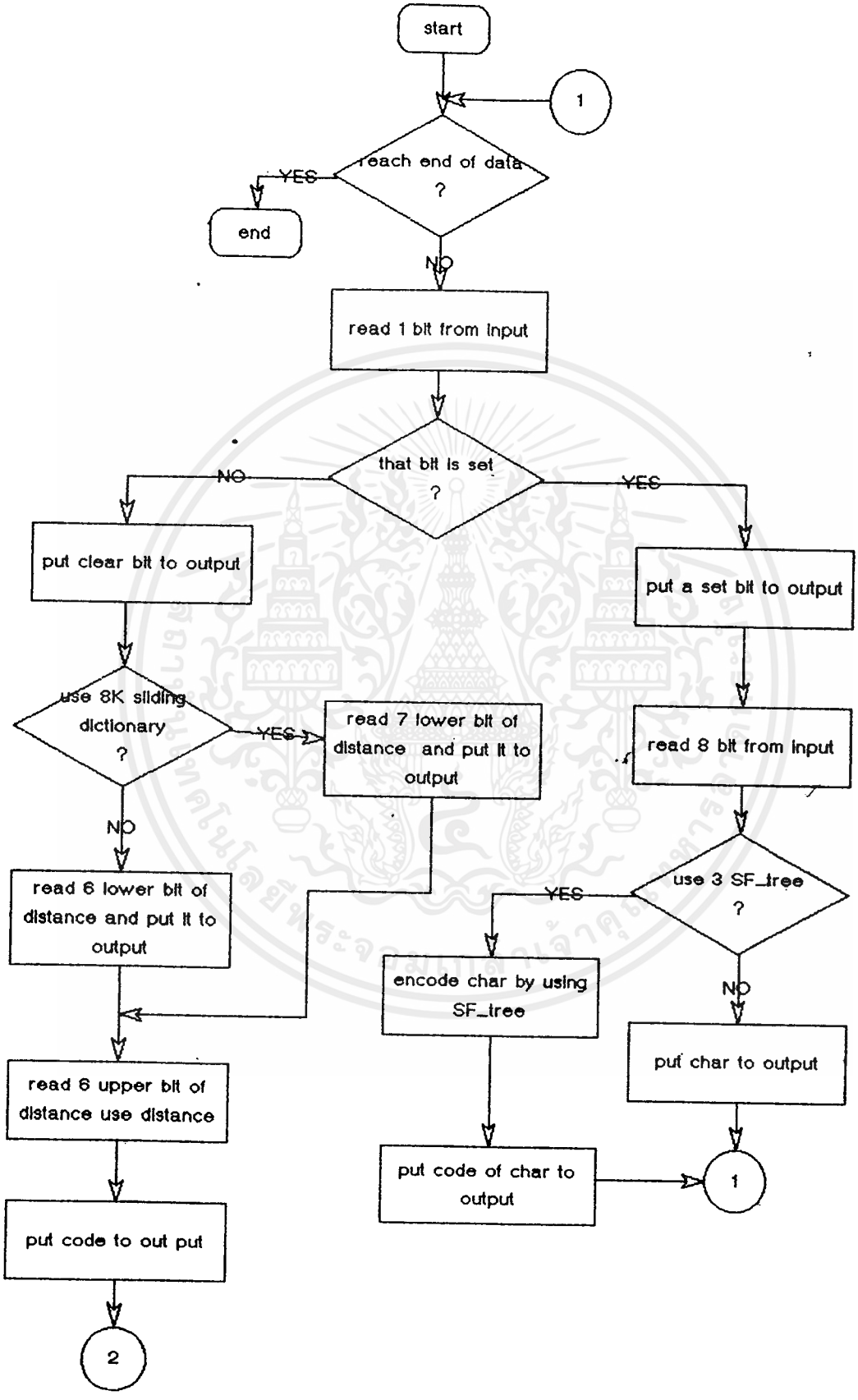
รูปที่ 3.12 โฟลว์ชาร์ทแสดงอัลกอริทึม Unshrinking



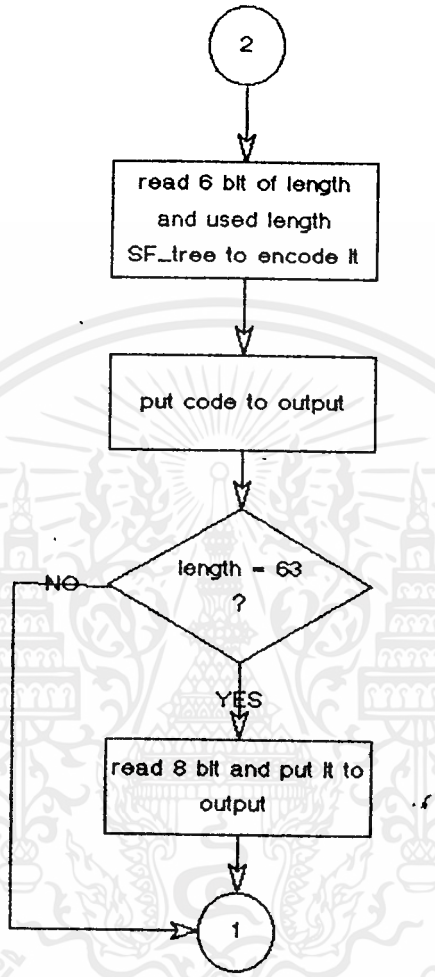
รูปที่ 3.13 โฟลว์ชาร์ทแสดงอัลกอริทึม Unshrinking



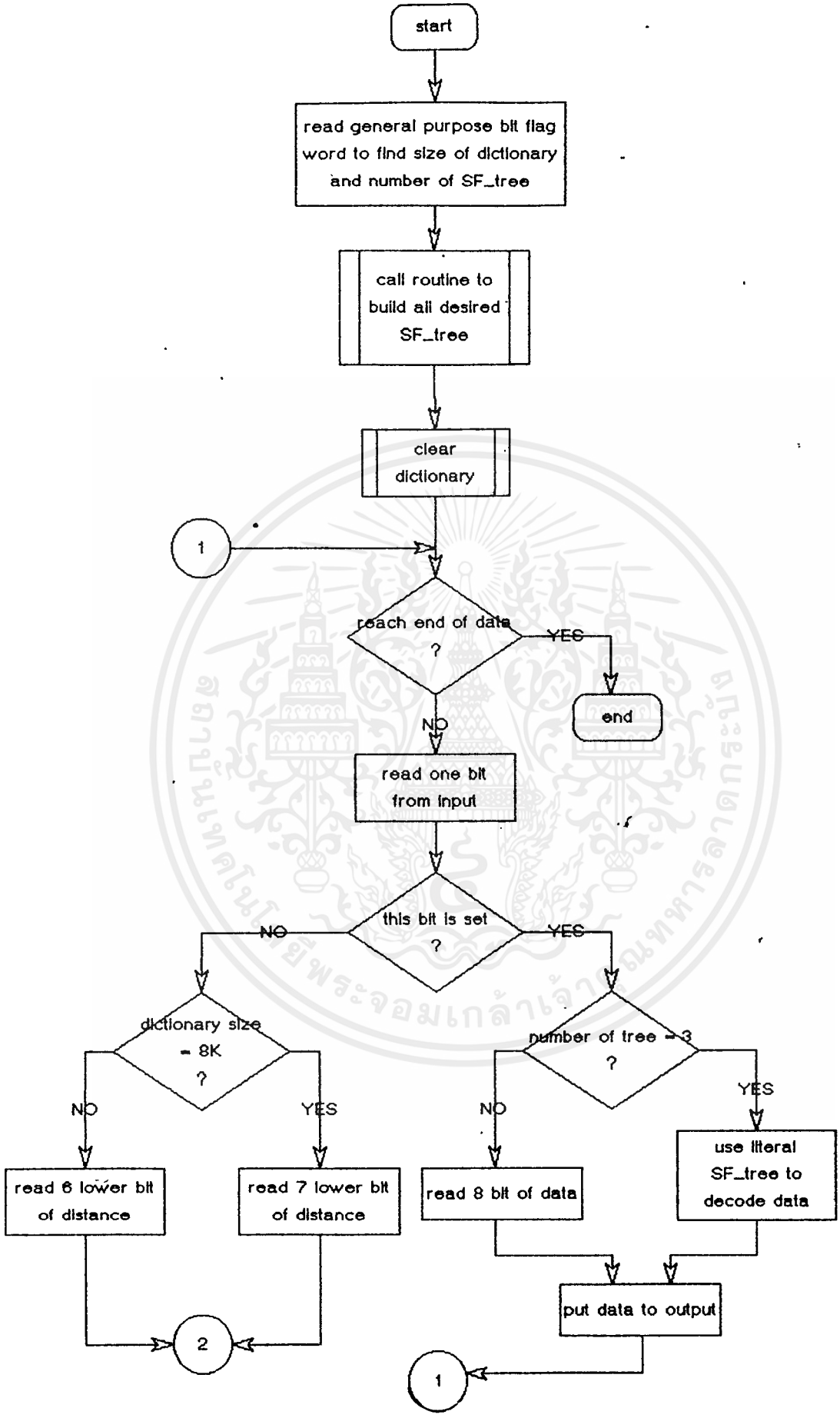
รูปที่ 3.14 โฟลว์ชาร์ทแสดงการถอดรหัสของ SF ทรี



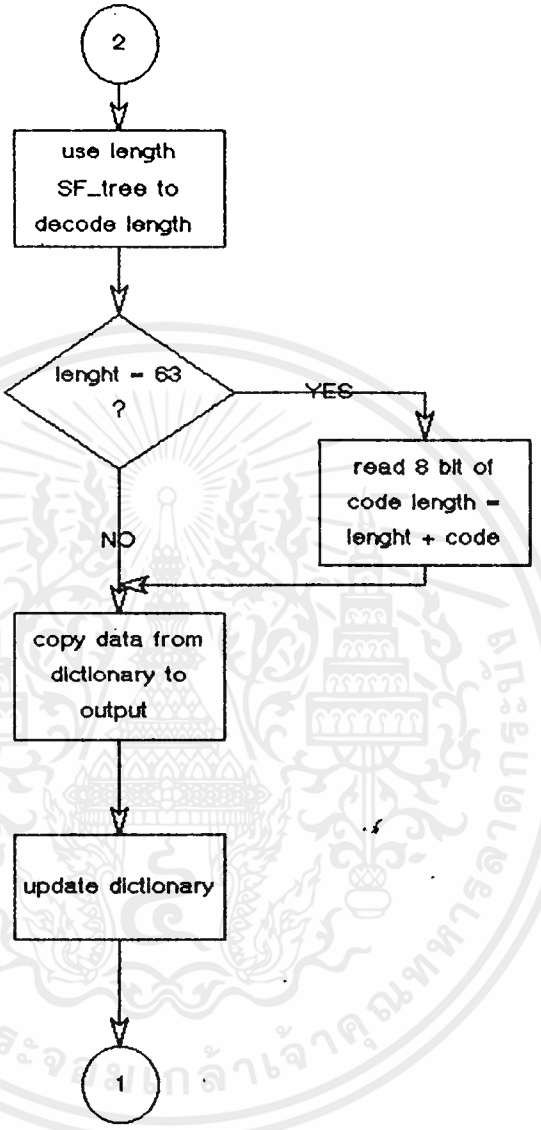
รูปที่ 3.15 โฟลว์ชาร์ทแสดงอัลกอริทึม Imploding



รูปที่ 3.16 โฟลว์ชาร์ทแสดงอัลกอริทึม Imploding



รูปที่ 3.17 โฟลว์ชาร์ทแสดงอัลกอริทึม Exploding



รูปที่ 3.18 โฟลว์ชาร์ทแสดงอัลกอริทึม Exploding

บทที่ 4 สรุปผลการศึกษา

ผลการทดสอบโปรแกรม

การทดสอบโปรแกรมซึ่งจำลองมาจากอัลกอริทึมของโปรแกรมPKZIP ใช้อุปกรณ์ในการทดสอบคือ เครื่องคอมพิวเตอร์ IBM พีซี ซึ่งใช้หน่วยประมวลผล 486 ความเร็ว 33 เมกะเฮอร์ซ หน่วยความจำแคช (cache memory) ขนาด 64 กิโลไบท์ หน่วยความจำพื้นฐานขนาด 4 เมกะไบต์ ฮาร์ดดิสขนาด 80 เมกะไบต์ และติดตั้งโปรแกรม stacker บนฮาร์ดดิสด้วย แฟ้มข้อมูลที่ใช้ทดสอบมี 2 ชนิด คือ แฟ้มข้อมูลชนิด เอกซีคิวเอเบิล (exe file) และข้อมูลชนิดตัวอักษร (text file) ตามปกติแล้ว แฟ้มข้อมูลที่ใช้ทดสอบกันโดยทั่วไปจะมีแฟ้มข้อมูลชนิด กราฟิกด้วย แต่ในกรณีนี้โปรแกรม PKZIP ถือว่า แฟ้มข้อมูล ชนิดกราฟิกมีความหมายเท่ากับแฟ้มข้อมูลชนิด เอกซีคิวเอเบิล แฟ้มข้อมูลชนิดตัวอักษรที่ใช้ทดสอบจะมีขนาด 449,101 ไบต์ ส่วนแฟ้มข้อมูลชนิดเอกซีคิวเอเบิล ที่ใช้ทดสอบมีขนาด 290,249 ไบต์ วิธีทดสอบทำได้โดย จับเวลาที่ต้องใช้ในการคำนวณ 2 ค่าคือเวลาที่ใช้ในการลดขนาดข้อมูล และเวลาที่ใช้ในการขยายขนาดข้อมูล และบันทึกขนาดของแฟ้มข้อมูลที่ถูกลดขนาดแล้วไว้เพื่อคำนวณหา ค่าต่างๆที่ต้องการ ค่าต่างๆที่ต้องการหาได้ดังนี้คือ

$$\text{อัตราการลดขนาดข้อมูล} = 1 - \frac{\text{ขนาดของแฟ้มข้อมูลที่ยังไม่ได้ลดขนาด}}{\text{ขนาดของแฟ้มข้อมูลที่ลดขนาดแล้ว}} \times 100 \text{ หน่วย \%}$$

$$\text{อัตราเร็วของการลดขนาดข้อมูล} = \frac{\text{ขนาดของแฟ้มข้อมูลที่ยังไม่ได้ลดขนาด}}{\text{เวลาที่ใช้ในการลดขนาดข้อมูล}} \text{ หน่วยไบต์ต่อวินาที}$$

$$\text{อัตราเร็วของการขยายขนาดข้อมูล} = \frac{\text{ขนาดของแฟ้มข้อมูลที่ยังไม่ได้ลดขนาด}}{\text{เวลาที่ใช้ในการขยายขนาดข้อมูล}} \text{ หน่วยไบต์ต่อวินาที}$$

ผลของการทดสอบโปรแกรมได้ผลดังตารางที่ 4.1

	compression ratio (%)		compression speed (bytes per second)		expansion speed (bytes per second)	
	text file	exe file	text file	exe file	text file	exe file
Shrink	63	27	8982	5092	29940	18142
Implode	71	42	454	208	21386	16027
Shrink (PKZIP)	63	27	89820	41464	89820	48375
Implode (PKZIP)	71	42	32079	24187	74850	48375

ตารางที่ 4.1 ผลของการทดสอบโปรแกรม

วิเคราะห์ผลการทดสอบโปรแกรม

จะเห็นได้ว่า อัตราการลดขนาดของโปรแกรมที่เขียนจำลองขึ้น เปรียบเทียบกับโปรแกรม PKZIPแล้ว จะมีอัตราที่ใกล้เคียงกันมาก โดยที่โปรแกรม PKZIP สามารถลดข้อมูลได้ดีกว่าเล็กน้อย เนื่องจากว่าโปรแกรม PKZIP ได้เพิ่มเทคนิคพิเศษบางประการ ในการลดขนาดข้อมูลเข้าไปอีก คือในกรณีของอัลกอริทึม shrink ตามปกติ จะเคลียร์ใบของทรี เมื่อใบถูกใช้หมดทุกใบแล้ว แต่โปรแกรม PKZIPจะวัดอัตราการลดขนาดข้อมูลไว้ตลอดเวลา และเมื่อจะเคลียร์ใบของทรีก็จะพิจารณาอัตราการลดขนาดข้อมูลที่วัดได้ ประกอบการตัดสินใจว่าจะเคลียร์ใบของทรีหรือไม่ ถ้าอัตราการลดขนาดข้อมูลที่วัดได้เริ่มตกลงก็จะเคลียร์ใบของทรี แต่ถ้าอัตราการลดขนาดข้อมูลที่วัดได้ยังไม่ตกลงก็จะไม่เคลียร์ใบของทรี ถึงแม้ว่าจะใช้ทุกใบแล้วก็ตาม จะทำให้ได้ผลการลดขนาดข้อมูลได้ดีกว่าปกติ สำหรับในกรณีของ อัลกอริทึม Implode ตามปกติขนาดสตริงที่เหมือนกันจะกำหนดไว้ไม่เกิน 64 ตัวแต่ PKZIP สามารถจะกำหนดขนาดของสตริงได้มากถึง 320 ตัว ทำให้ PKZIP สามารถเข้ารหัสสตริงที่ยาวเกิน 64 ตัวได้ จึงทำให้สามารถลดขนาดข้อมูลได้ดีกว่า

ในด้านอัตราเร็วของการลดขนาดข้อมูลและอัตราเร็วของการขยายขนาดข้อมูล เมื่อเปรียบเทียบกันแล้วจะเห็นว่าโปรแกรมที่จำลองขึ้นมีอัตราเร็วที่ด้อยกว่า เนื่องจากโปรแกรมที่จำลองขึ้นเขียนโดยภาษา C ซึ่งให้ขนาดของโปรแกรม EXE ที่ใหญ่กว่า โปรแกรม PKZIP ซึ่งเขียนด้วยภาษา ASSEMBLY ทำให้เวลาที่ใช้ในการทำงานของโปรแกรมมากขึ้น เนื่องจากว่าโปรแกรมแปลภาษา C

(C compiler) สร้างรหัสคำสั่งได้ไม่กระชับ เท่ากับโปรแกรมที่เขียนด้วยภาษา ASSEMBLY นอกจากนี้วิธีการเขียนโปรแกรม ก็มีผลต่ออัตราการเร็วในการทำงานของโปรแกรมด้วย ถ้าใช้การเขียนโปรแกรมไม่เหมาะสมก็อาจทำให้ใช้เวลาในการทำงานของโปรแกรมมากขึ้นด้วย

เอกสารอ้างอิง

[1] M. Nelson, "The Data Compression Book," Reading, MA:M&T Books, 1991.

[2] D.A. Huffman, "A method for the construction of minimum-redundancy codes," Proc. IRE, vol. 40, pp. 1098-1101, 1952.

[3] T.A. Welch, "A Technique for high-performance data compression," IEEE Computer, vol. 17, pp. 8-19, June 1984.

[4] J. Ziv and A. Lempel, "A universal algorithm for sequential data compression," IEEE Trans. Inform. Theory, vol. IT-23, pp. 337-343, May 1977.

[5] J. Ziv and A. Lempel, "Comparison of individual sequences via variable-rate coding," IEEE Trans. Inform. Theory, vol. IT-24, no. 5, pp. 530-536, 1978.

[6] วิโรจน์ อัศวรังสี และ บุญชู งามไพโรจน์พิบูลย์ "ลด...ลดขนาดข้อมูล" คอมพิวเตอร์รีวิว. ฉบับที่ 92 หน้าที่ 164-170 เมษายน 2535

[7] สันติ หวังเกื้อกูล "การลดขนาดข้อมูลด้วยวิธี LZW" คอมพิวเตอร์รีวิว ฉบับที่ 92 หน้าที่ 171-176 เมษายน 2535

กิตติกรรมประกาศ

ขอบคุณท่านอาจารย์ เพื่อน ๆ และน้อง ๆ ทุกคน ที่มีส่วนช่วยให้วิทยานิพนธ์นี้สำเร็จลงได้