บทที่ 5

โครงสร้างข้อมูลแบบดี-เอส ทรีและการลดขนาดข้อมูลโดยวิธีแอลแซดดับบลิว

จากบทที่ 3 ซึ่งได้กล่าวถึงโครงสร้างการจัดเก็บข้อมูลและวิธีการสืบค้นข้อมูลแบบต่างๆ ซึ่งจะเห็นว่าแต่ละวิธีมีข้อดีข้อด้อยแตกต่างกันไป ผู้วิจัยได้พิจารณาว่าสำหรับการพัฒนาโปรแกรมพจนุกรมอิเล็กทรอนิกส์ ควรใช้โครงสร้างข้อมูลแบบดี-เอส ทรี ซึ่งมีข้อดี คือ ใช้พื้นที่ในการจัดเก็บข้อมูลน้อยและสามารถสืบค้นได้อย่างรวดเร็ว

สำหรับการลดขนาดข้อมูลนั้นในบทที่ 4 ได้กล่าวถึงวิธีการบีบอัดข้อมูลแบบต่างๆ ซึ่งผู้วิจัยได้เลือกวิธีแอลแซดดับบลิว มาใช้สำหรับการลดขนาดข้อมูลพจนานุกรม ซึ่งมีข้อดีคือ ในตอนเริ่มต้นจะนำรหัสแอสกี 0-255 มาใช้เป็นรหัสเริ่มต้นฉะนั้นจึงสามารถเข้ารหัสเป็นวลีได้ และอีกประการในขั้นตอนการลดขนาดข้อมูลไม่ต้องเก็บตารางรหัสไว้ใช้สำหรับการขยายข้อมูล ทำให้สามารถประหยัดพื้นที่ในการจัดเก็บได้มาก

ในบทนี้จะกล่าวถึงรายละเอียดโครงสร้างข้อมูลแบบดี-เอส ทรี และวิธีการลดขนาดข้อมูลแอลแซดดับบลิว

โครงสร้างข้อมูลแบบดี-เอส ทรี

ดิจิตอลเสิช-ทรี (Aoe, 1987) หมายถึง โครงสร้างข้อมูลแบบต้นไม้ที่มีการสืบค้นแบบดิจิตอล สามารถเรียกโดยย่อว่า ดี-เอส ทรี โครงสร้างแบบดี-เอส ทรี ปันโครงสร้างที่ประกอบด้วยโหนดต่างๆ และเส้นทางเดินระหว่างโหนด โหนดเริ่มต้นเรียกว่า โหนดราก ส่วนโหนดอื่นๆ ภายในโครงสร้างเรียกว่าโหนดใบ แตกต่างจากโครงสร้างข้อมูลแบบต้นไม้ตรงที่เส้นทางเดินระหว่างโหนดของโครงสร้างข้อมูลแบบต้นไม้ เป็นการแทนค่าอักษรแต่ละตัวที่ประกอบเป็นคำศัพท์ ส่วนเส้นทางเดินระหว่างโหนดของดี-เอส ทรี เป็นผลลัพธ์ที่ได้มาจากการคำนวณระหว่างตัวเลขที่ได้จากการแปลงค่าตัวอักษรที่ประกอบกันเป็นคำศัพท์หรือเฉพาะส่วนต้นของคำศัพท์ กับค่าที่เก็บในโหนดก่อนหน้า สำหรับขั้นตอนการเพิ่มหรือการสืบค้นคำศัพท์ของโครงสร้างแบบนี้ได้มาจากการสร้างเส้นทางที่เริ่มต้นจากโหนดรากจนกระทั่ง ตำแหน่งสุดท้ายของศัพท์ทุกคำโดยจะใช้สัญญลักษณ์ "#" เพื่อเป็นตัวบอกสิ้นสุดคำศัพท์

เนื่องจากในการแสดงดี-เอส ทรีในเชิงโปรแกรม จะใช้โครงสร้างข้อมูลแบบดับเบิลอะเรย์ทรัย ซึ่งการจัดโครงสร้างแบบนี้ (Aoe, 1987) นักวิจัยชาวญี่ปุ่นได้คิดค้นขึ้นโดยนำหลักการมาจากโครงสร้างข้อมูลแบบทริปเปิ้ลอะเรย์ที่ใช้งานกับทรานซิชั่นเทเบิ้ลสำหรับแยกซ์

การสืบค้นข้อมูลโดยวิธีดี-เอส ทรี โดยการจัดโครงสร้างแบบดับเบิลอะเรย์ การค้นหาเส้นทางเดินในดี-เอส ทรี สามารถคำนวณจากข้อมูลในดับเบิลอะเรย์โดยใช้เวลา O(1) และอย่างช้าที่สุดใช้เวลา O(k) โดยที่ k เป็นความยาวของคำหลัก การจัดเก็บข้อมูลโดยใช้ดับเบิลอะเรย์นี้จะใช้เนื้อที่ในการจัดเก็บน้อยและสามารถสืบค้นภายในเวลาอันรวดเร็ว

ให้ขนาดของดับเบิลอะเรย์ = $n + cm$

โดยที่ n = จำนวนโหนดในดี-เอส ทรี

m = จำนวนของตัวอักษรที่ประกอบเป็นคำศัพท์ที่อ่านเข้ามา

c = ค่าคงที่ซึ่งขึ้นกับแต่ละดับเบิลอะเรย์

จากการทดสอบพบว่าใช้เวลามากที่สุดในการ

ลบข้อมูล = cm

เพิ่มข้อมูล = cm^2

โดยที่ c มีค่าอยู่ระหว่าง 0.17 - 1.13 ซึ่งมีค่าน้อยมาก ฉะนั้นเวลาที่ใช้ในการสืบค้นข้อมูลจึงขึ้นกับความยาวของคำศัพท์ที่อ่านเข้ามา

<u>การจัดเก็บข้อมูลด้วยโครงสร้างข้อมูลแบบดับเบิลอะเรย์</u>

การจัดเก็บและสืบค้นข้อมูลโดยวิธีดี-เอสทรีนั้นมีการใช้สัญญลักษณ์ต่างๆ ดังรายละเอียดต่อไปนี้

1. S แทนกลุ่มของโหนด
2. I แทนกลุ่มของอักขระอ่านเข้ามา
3. g เรียกว่า goto function แทนฟังก์ชันจาก S X I ถึง S U {fail}
4. s_1 แทนโหนดเริ่มต้นหรือโหนดรากใน S
5. A แทนโหนดของคำศัพท์ที่สนใจ
6. s_r เป็นโหนดที่เป็นสมาชิกของ A เมื่อเส้นทางเดินของโหนด s_1 ถึงโหนด s_r ให้ผลลัพธ์เป็นคำศัพท์(x#) คำหนึ่งในกลุ่มข้อมูล K
7. ถ้าให้อาร์ค a ในกลุ่มอักขระ I เป็นเส้นทางเดินจากโหนด s_i ไปยัง s_j ซึ่งแสดงโดยฟังก์ชัน $g(s_i, a) = s_j$ ถ้าไม่สามารถสร้างอาร์คจากฟังก์ชัน goto ได้ถือว่าไม่สำเร็จ
8. กำหนดให้ a, b, c, d, e $\in$ I U {#}; x, y, z $\in$ (I U {#})*

โดย $\in$ เป็นสายอักขระว่างและฟังก์ชัน goto สามารถแสดงสายอักขระในกลุ่มข้อมูล K ได้ตามสมการ $g(s_r, \in) = s_r$, $g(s_r, ax) = g(g(s_r, a), x)$

โครงสร้างการจัดเก็บข้อมูลแบบดับเบิลอะเรย์ประกอบด้วยส่วนต่างๆ ดังนี้

1. ประเภทของโหนด

โครงสร้างข้อมูลแบบดับเบิลอะเรย์ประกอบด้วยโหนดต่างๆหลายประเภทด้วยกัน ดังนี้

1.1 เซพพาเรทโหนด (Separate node) แทนด้วยสัญญลักษณ์ S_p หมายถึงโหนดที่ทำให้ทราบว่าคำศัพท์คำหนึ่งแตกต่างจากคำศัพท์คำอื่นในกลุ่มข้อมูล K ถ้ากำหนดให้คำหลัก xay ในดี-เอส ทรี K โหนด S_r จะเป็นเซพพาเรทโหนดก็ต่อเมื่อ $g(s_1, xa) = s_r$ ถ้า a เป็นส่วนที่ทำให้คำหลัก xay แตกต่างจากคำอื่นใน K

1.2 มัลติโหนด (Multinode) แทนด้วยสัญญลักษณ์ S_M หมายถึงโหนดทุกๆ โหนดที่อยู่ในเส้นทางจากโหนดรากถึงเซพพาเรทโหนด

1.3 ซิงเกิลโหนด (Single - node) แทนด้วยสัญญลักษณ์ S_l หมายถึงโหนดทุกๆโหนดที่อยู่ระหว่างเส้นทางจากเซพพาเรทโหนดมายังโหนดสุดท้ายของคำศัพท์

จากโหนดทั้ง 3 ประเภท สามารถสรุปได้ว่าเซพพาเรทโหนด เป็นโหนดร่วมกันระหว่างมัลติโหนดกับซิงเกิลโหนด ($S_p = S_M \cap S_l$)

2. โครงสร้างการจัดเก็บข้อมูลแบบดับเบิลอะเรย์ทรัย

การสืบค้นแบบดิจิตอล-เสิชทรี ซึ่งเป็นโครงสร้างที่ประกอบด้วยมัลติโหนด เซพพาเรทโหนด ซิงเกิลโหนดและเส้นทางเดินระหว่างโหนด โดยที่เส้นทางเดินระหว่างโหนดของมัลติโหนดหรือเซพพาเรทโหนดจะเก็บในอะเรย์เบสและอะเรย์เช็ค ซิงเกิลสตริงจะจัดเก็บไว้ในอะเรย์เทล ซึ่งอะเรย์แต่ละชนิดมีรายละเอียด ดังนี้

2.1 อะเรย์เบส ประกอบด้วย 2 ส่วน คือ ดรรชนีและค่าในอะเรย์ ดรรชนีใช้แทนหมายเลขมัลติโหนดหรือเซพพาเรทโหนด ส่วนค่าในอะเรย์มี 2 ประเภท คือ

ถ้ามีค่าเป็นบวก หมายถึง ตัวเลขที่ใช้สำหรับคำนวณเส้นทางเดินของโหนดหนึ่งไปยังอีกโหนดหนึ่ง

ถ้ามีค่าเป็นลบ หมายถึง ตำแหน่งของซิงเกิลสตริงในอะเรย์เทล

2.2 อะเรย์เช็ค ประกอบด้วย 2 ส่วนคือ ดรรชนีและค่าในอะเรย์

ดรรชนีใช้แทนหมายเลขโหนด ส่วนค่าในอะเรย์ คือ หมายเลขพาเรนโหนด เช่น โหนดหมายเลข 5 สร้างเส้นทางเดินไปยังโหนดที่ 12 เพราะฉะนั้นค่าของอะเรย์เช็คที่ 12 มีค่า

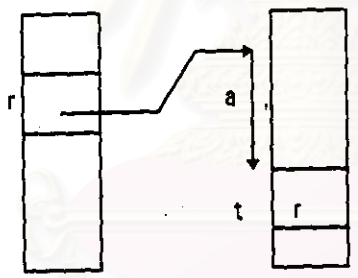
เป็น 5 ค่าที่เก็บในอะเรย์ที่ 1 คือดรรชนีสูงสุดของอะเรย์เบสและเช็ค

2.3 อะเรย์เทล ใช้เก็บชิงเกิลสตริง ประกอบด้วยตัวอักษร สัญญลักษณ์สิ้นสุดคำ "#" สัญญลักษณ์แบ่งชิงเกิลสตริง "$" และสัญญลักษณ์ระบุตำแหน่งที่ไม่ได้ใช้งาน (สัญญลักษณ์การไม่ได้ใช้งาน) "?"

นิยาม กำหนดให้สายอักขระ x ซึ่งสอดคล้องกับฟังก์ชัน $g(s_i, x) = s_t$ โดยที่โหนด s_i เป็นเซพพาเรทโหนดและโหนด s_t เป็นสมาชิกของแอคเซพติงโหนด ฉะนั้น x จะเป็นชิงเกิลสตริงของเซพพาเรทโหนด S_i ซึ่งสามารถแทนด้วย สัญญลักษณ์ STR(s_i)

จากนิยามข้างต้นพบว่า S_M X (I U (#)) ถึง S_M จะเก็บในดับเบิลอะเรย์ (อะเรย์เบสและอะเรย์เช็ค) และ S_i X (I U (#)) ถึง S_t เป็นชิงเกิลสตริงและเก็บในอะเรย์เทล ฉะนั้นการจัดเก็บสายอักขระ จะมี 2 กรณี คือ

1. นำสายอักขระมาสร้างมัลติโหนดและเซพพาเรทโหนด และจัดเก็บข้อมูลเกี่ยวกับเส้นทางเดินของโหนดในอะเรย์เบสและอะเรย์เช็ค
2. เมื่อไม่สามารถสร้างเซพพาเรทโหนดได้แล้ว จะสร้างชิงเกิลโหนดและจัดเก็บชิงเกิลสตริงในอะเรย์เทล



รูปที่ 5-1 โครงสร้างข้อมูลดับเบิลอะเรย์ทรัย สำหรับ $g(s_i, a) = s_t$

3. ความสัมพันธ์ภายในโครงสร้างแบบดับเบิลอะเรย์

โครงสร้างข้อมูลแบบดับเบิลอะเรย์ประกอบด้วยอะเรย์ขนาด 1 มิติจำนวน 2 อะเรย์ คือ อะเรย์เบส และอะเรย์เช็ค มีหมายเลขประจำโหนดเป็นดรรชนีเพื่อแสดงความสัมพันธ์ระหว่างเบสและเช็ค

ข้อกำหนดของดับเบิลอะเรย์ และเทล ของดี-เอส ทรี มีเงื่อนไขดังนี้

3.1 สำหรับ arc $r \xrightarrow{a} t$ ให้

BASE [r] + a = t

และ CHECK [t] = r

3.2 สำหรับเซพพาเรทโหนด โหนด r ให้

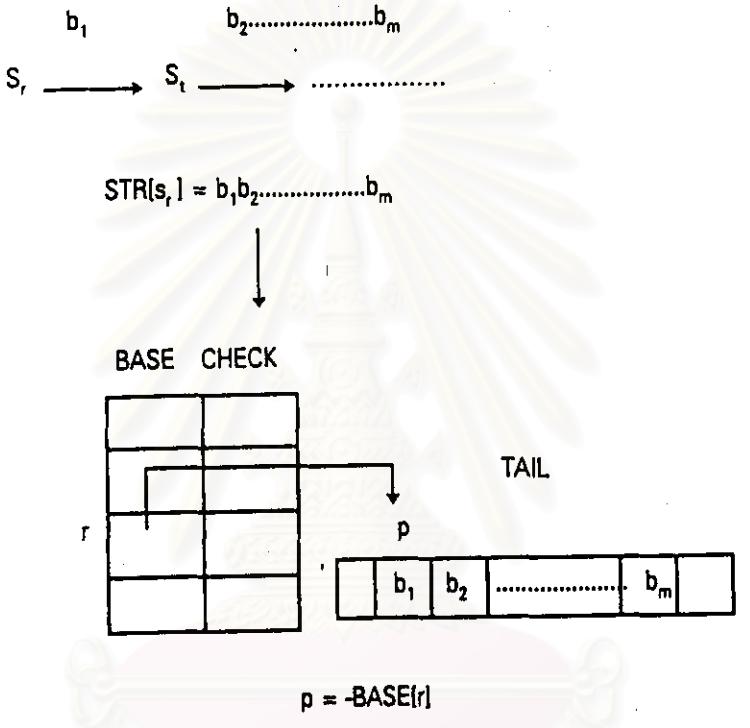
3.2.1 BASE[r] < 0

3.2.2 ถ้า STR[s_r] = b_1b_2.....b_m โดยที่ m >= 0

เก็บ STR[s_r] ใน TAIL ที่ตำแหน่ง p โดย p = -BASE [r]

TAIL[p] = b_1, TAIL[p+1] = b_2, ...TAIL[p+m-1] = b_m

3.1 คือเงื่อนไขของดับเบิลอะเรย์ และ 3.2 คือเงื่อนไขของเซพพาเรทโหนด



รูปที่ 5-2 แสดงความสัมพันธ์ของดับเบิลอะเรย์และเทล

การสืบค้นข้อมูลของโครงสร้างแบบดี-เอส ทรีนี้ จะใช้ค่าตัวเลขของตัวอักษร a แทนตัวอักษร a ในการตรวจสอบ ว่ามี r–^{a}–t อยู่หรือไม่ ให้เปรียบเทียบว่า CHECK [BASE [r] + a] = r หรือไม่ หากเท่ากันแสดงว่ามี r–^{a}–t อยู่จริง ฉะนั้นในการสืบค้น arc หนึ่งๆจะใช้เวลามากที่สุดเป็น O(1)

จากรูปที่ 5-2 โหนด S_r และโหนด S_t ใดๆซึ่งเป็นสมาชิกของ S (กลุ่มของโหนดในดีเอส-ทรี) ได้มาจากฟังก์ชัน $g(s_r, a) = s_t$ ก็ต่อเมื่อดับเบิลอะเรย์ที่แสดงกลุ่มข้อมูล K หรือของคำศัพท์นั้นเป็นไปตามสมการ BASE[r] + a = t และ CHECK[t] = r ซึ่งหมายถึงว่าโหนด s_t เป็นสมาชิกของ S และมีความสัมพันธ์โดยตรงกับค่าตรรชนี r หรือหมายเลขโหนดของดับเบิลอะเรย์ ซึ่งหมายเลขโหนดของโหนดถัดไปและเส้นทางระหว่างโหนดของโครงสร้างข้อมูลแบบดับเบิลอะเรย์นั้นได้มาจากการคำนวณค่าของ BASE[r] กับค่าตัวเลขของตัวอักษรที่อ่านเข้ามาตำแหน่งปัจจุบัน

กำหนดให้

x แทนคำศัพท์หนึ่งคำ โดยที่ $x = a_1 a_2 \ldots\ldots\ldots a_n a_{n+1}$, $a_{n+1} = \#$

r แทนหมายเลขโหนดปัจจุบัน

t แทนหมายเลขโหนดถัดไป

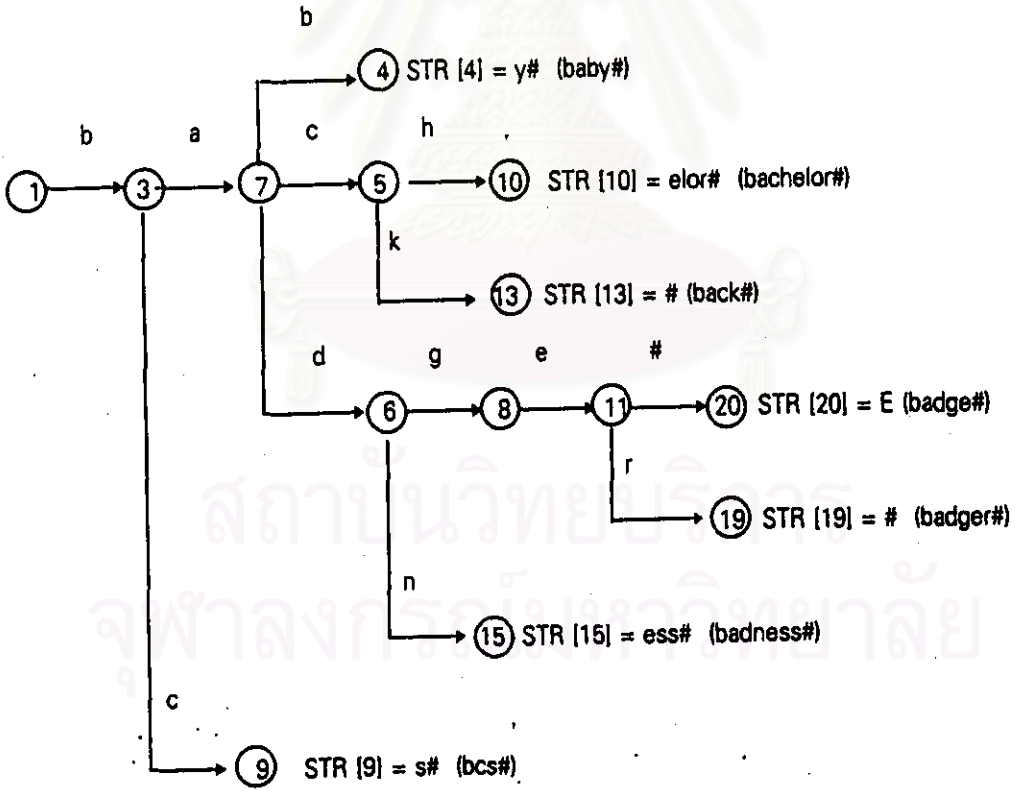
h แทนตำแหน่งของตัวอักษรในคำศัพท์

a_n แทนตัวอักษรใดๆในคำศัพท์ เราจะแทนค่าตัวเลขของตัวอักษรนั้น

DA-SIZE แทนขนาดสูงสุดของอะเรย์เบสและเช็ค

ตัวอย่างโครงสร้างการจัดเก็บข้อมูลแบบดี-เอส ทรี

กำหนด K1 = { baby#, bachelor#, back#, badge#, badger#, badness#, bcs# } แสดงด้วยดี-เอส ทรี (# เป็นตัวบอกจุดสิ้นสุดของแต่ละคำหลัก)



รูปที่ 5-3 แสดงดี-เอส ทรี ของ K

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
BASE	1	0	6	-17	2	1	2	6	-10	-1	1	0	-20	0	-24	0	0	0	-22	-13
CHECK	20	0	1	7	7	7	3	6	3	5	8	0	5	0	6	0	0	0	11	11

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26
TAIL	e	l	0	r	#	$	?	?	?	s	#	$	$	?	?	?	y	#	$	#	$	#	$	e	s	s

	27	28
TAIL	#	$

รูปที่ 5-4 แสดงดับเบิลอะเรย์ทรัยของ K

อัลกอริทึมการสืบค้นข้อมูลประกอบด้วย

1. การสืบค้นคำศัพท์
2. การเพิ่มคำศัพท์
3. การลบคำศัพท์

1. การสืบค้นคำศัพท์

1. กำหนดค่าเริ่มต้น r = 1 และ h = 0
   สร้างโหนดถัดไปจากโหนดที่ 1 และตัวอักษรตัวแรกของคำศัพท์ตามสมการ
   $t = BASE[r] + a_h$
2. ตรวจสอบว่าโหนดที่สร้างขึ้นมีค่าเกินขนาดของอะเรย์หรือไม่ ถ้ามากกว่าขนาดของอะเรย์ให้ระบุว่าไม่พบคำศัพท์และออกจากการสืบค้น
   ถ้า t > DA-SIZE ออกจากการสืบค้น
3. ตรวจสอบว่าโหนดใหม่เป็นโหนดที่สร้างจากโหนดแรกหรือไม่
   ถ้า CHECK[t] = r แสดงโหนดใหม่สร้างจากโหนดแรก
   ถ้า CHECK[t] != r แสดงว่าสร้างมัลติโหนดไม่สำเร็จ ระบุว่าไม่พบคำศัพท์ออกจากการสืบค้น
4. ตรวจสอบว่าโหนดที่สร้างใหม่เป็นโหนดประเภทใดโดยพิจารณาจาก ค่า BASE[r] ดังนี้
   BASE[r] = 0 แสดงว่าเป็นโหนดที่ยังไม่ได้ใช้งาน ระบุว่าไม่พบคำศัพท์ออกจากการสืบค้น
   BASE[r] > 0 แสดงว่าเป็นมัลติโหนด

BASE[r] < 0 แสดงว่าเป็นเซพพาเรทโหนด ตรวจสอบดังนี้

ถ้า h = n+1 แสดงว่าพบคำศัพท์

ถ้า h != n+1

ตำแหน่งของซิงเกิลสตริงในอะเรย์เทล = ค่าสัมบูรณ์ของ BASE[r]

ใช้ฟังก์ชัน Fetch_Str(-BASE[r]) ดึงซิงเกิลสตริงออกจากอะเรย์เทลและเก็บค่าไว้ใน S_Temp

ใช้ฟังก์ชัน Str_Cmp(a_{h+1} .....$a_n a_{n+1}$, S_Temp) เปรียบเทียบซิงเกิลสตริงกับคำศัพท์ส่วนที่เหลือจากการสร้างมัลติโหนด

โดย a_{h+1} .....$a_n a_{n+1}$ คือ คำศัพท์ส่วนที่เหลือจากการสร้างมัลติโหนด

S_Temp คือ ซิงเกิลสตริง

ถ้าเหมือนกันจะส่งค่า -1 กลับมา

ถ้าไม่เหมือนกันจะส่งค่าจำนวนตัวอักษรที่เหมือนกันกลับมา

5. กรณีที่เป็นมัลติโหนด จากข้อ 4 คือ BASE[r] > 0 สามารถสร้างโหนดต่อไปได้ โดยใช้โหนดที่สร้างใหม่แทนโหนดปัจจุบัน คือ กำหนด r = t และนำค่าตัวอักษรตัวถัดไปมาใช้ โดยเพิ่มค่า h เป็น h+1 ทำตามข้อที่ 1 ถึง 4 จนกระทั่งตัวอักษรในคำศัพท์หมดหรือออกจากการสืบค้นเนื่องจากไม่เป็นไปตามเงื่อนไขที่กำหนดในข้อ 1 - 4

<u>ในการสืบค้นคำศัพท์จะใช้ฟังก์ชันสนับสนุนอีก 2 ฟังก์ชัน คือ</u>

1. ฟังก์ชัน Fetch_Str(p) มีหน้าที่ในการดึงซิงเกิลสตริงมาจากอะเรย์เทล เริ่มจากตำแหน่ง p จนถึงตำแหน่ง p + k โดยที่

   p คือ ค่าสัมบูรณ์ของอะเรย์เบสในกรณีที่ค่า BASE[r] < 0

   k คือ ตำแหน่งสุดท้ายของซิงเกิลสตริง,k >= 0

   TAIL[p + k] = #

2. ฟังก์ชัน Str_Cmp(x,y) มีหน้าที่ในการใช้เปรียบเทียบสายอักขระ x กับสายอักขระ y ว่าเหมือนหรือแตกต่างกัน

   ถ้า x = y จะคืนค่า -1

   ถ้า x ไม่เหมือนกับ y จะคืนค่าจำนวนตัวอักษรนับจากส่วนต้นของคำที่ x เหมือนกับ y

2. การเพิ่มคำศัพท์

ก่อนที่จะเพิ่มคำศัพท์ในพจนานุกรมอิเล็กทรอนิกส์จะต้องทำการสืบค้นคำศัพท์ก่อน เมื่อไม่พบคำศัพท์จึงทำการเพิ่มคำศัพท์ ซึ่งการสืบค้นข้อมูลแล้วไม่พบคำศัพท์มีได้ 2 กรณี คือ

1. ทำการสืบค้นคำศัพท์แล้วไม่พบคำศัพท์ เนื่องจากการสร้างมัลติโหนดไม่สำเร็จ
2. ทำการสืบค้นคำศัพท์แล้วไม่พบคำศัพท์ เนื่องจากเปรียบเทียบชิงเกิลสตริงกับคำศัพท์ส่วนที่เหลือจากการสร้างมัลติโหนดแล้วไม่เหมือนกัน

ในการเพิ่มคำศัพท์จะแบ่งออกเป็น 2 กรณี คือ

1. ฟังก์ชันการเพิ่มข้อมูลแบบที่ 1 A_Insert(r, $a_h a_{h+1}$ .....$a_n a_{n+1}$)

เป็นขั้นตอนการเพิ่มคำศัพท์หลังจากที่สืบค้นคำศัพท์แล้วไม่พบคำศัพท์ เนื่องจากการสร้างมัลติโหนดไม่สำเร็จ

r แทนหมายเลขโหนดปัจจุบัน

$a_h a_{h+1}$ .....$a_n a_{n+1}$ แทนคำศัพท์ส่วนที่เหลือจากการสร้างมัลติโหนด

<u>การทำงานของฟังก์ชัน A_Insert ประกอบด้วย</u>

1. การสร้างมัลติโหนด s_t จากฟังก์ชัน $g(s_r, a_h) = s_t$
2. เก็บชิงเกิลสตริง STR[s_t] (a_{h+1} .....$a_n a_{n+1}$) ไว้ในอะเรย์เทล

<u>ขั้นตอนการเพิ่มข้อมูลของฟังก์ชัน A_Insert</u> (r, $a_h a_{h+1}$ .....$a_n a_{n+1}$)

1. กำหนดค่าของหมายเลขโหนดถัดไป

   t = BASE[r] + a_h
2. ตรวจสอบค่าของ CHECK[t] = 0 ทำงานในข้อ 4

   ถ้าค่าของ CHECK[t] ไม่เท่ากับ 0 ทำงานในข้อ 3
3. เปลี่ยนค่าของ BASE[r] และ BASE[t] ซึ่ง k = CHECK[t] ตามขั้นตอนต่อไปนี้

   3.1 ใช้ฟังก์ชัน SetList(r) เพื่อหาค่า Rlist และ ใช้ฟังก์ชัน SetList(k) เพื่อหาค่า Klist

   3.2 ตรวจสอบจำนวนสมาชิกของ Rlist + 1 ว่ามากกว่า จำนวนสมาชิกของ Klist หรือไม่

   ถ้าเงื่อนไขเป็นจริง ทำงานในข้อ 3.3

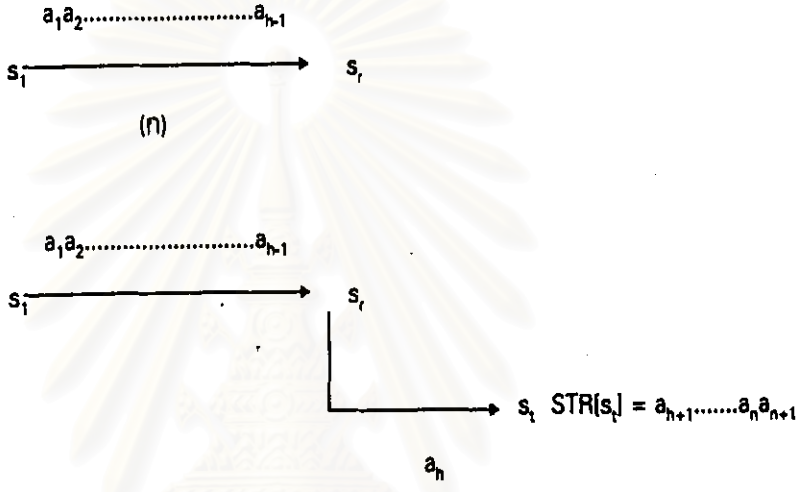
   ถ้าเงื่อนไขเป็นเท็จ ทำงานในข้อ 3.4

   3.3 ใช้ฟังก์ชัน Modify(r, r, (a_h) , Rlist) เพื่อหาค่า BASE[k] โดยที่ต้องอยู่ภายใต้เงื่อนไข

CHECK[BASE[r]+b] = 0 และ b เป็นสมาชิกใน Rlist U (a_h) ต่อไปทำงานในข้อ 4

3.4 ใช้ฟังก์ชัน Modify(r, k, , Klist) เพื่อหาค่า BAS[k] ซึ่งอยู่ภายใต้เงื่อนไข CHECK[BASE[r] + a_h] ไม่เท่ากับ CHECK[BASE[k] + b] และ b เป็นสมาชิกใน Klist ต่อไปทำงานในข้อ 4

4. ใช้ฟังก์ชัน Ins_Str(r, $a_h a_{h+1}$ .....a_{n+1}, POS) เพื่อสร้างโหนดในดับเบิลอะเรย์ตามสมการ $g(s_r, a_h)$ และจัดเก็บ a_{h+1}.....a_{n+1} ในอะเรย์เทล



รูปที่ 5-5 แสดงการทำงานของฟังก์ชัน A_Insert

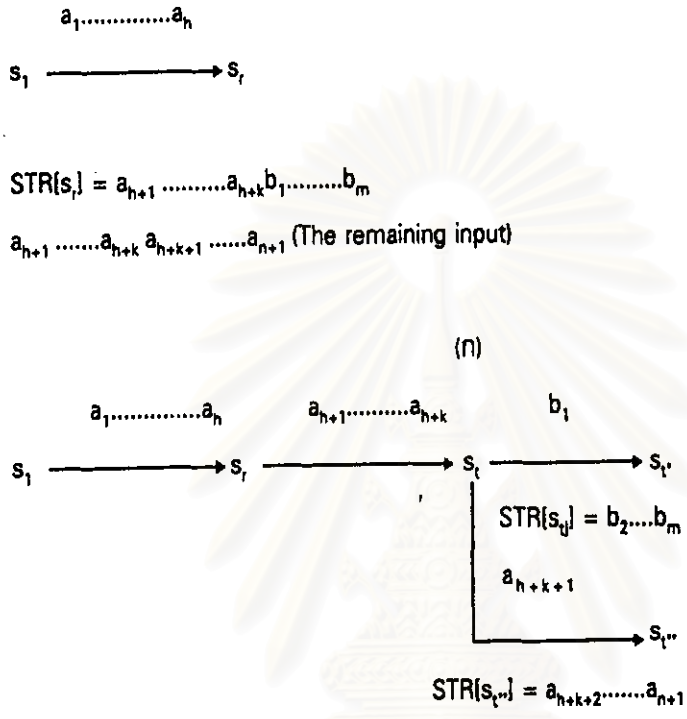
(ก) แสดงดี-เอส ทรี ก่อน A_Insert (ข) แสดงดี-เอส ทรี หลัง A_Insert

ฟังก์ชัน B_Insert(r, a_{h+1} ....a_{h+k} , a_{h+k+1} .....a_n a_{n+1} , b_1.....b_m)

1. กำหนดค่า old_pos = -BASE[r]
2. สร้างเส้นทางเดินของโหนดโดยใช้ตัวอักษรจากสายอักขระที่อ่านเข้ามา ตามขั้นตอนต่อไปนี้
   2.1 หาค่า BASE[r] จากฟังก์ชัน X_Check(a_{h+i}) โดยที่ 1<= i <= k
   2.2 กำหนดค่า CHECK[BASE[r] + a_{h+i}] = r
   2.3 คำนวณค่าโหนด r ใหม่จาก BASE[r] + a_{h+i} ให้สร้างเส้นทางเดินของโหนดจนกระทั่ง i = k
3. หาค่า BASE[r] ใหม่จากฟังก์ชัน X_Check(a_{h+k+1} , b_1)
4. นำสายอักขระส่วนที่ได้จากอะเรย์เทลเก็บในอะเรย์เทลที่ตำแหน่งเดิม โดยใช้ฟังก์ชัน InsStr(r, b_2 .....b_m , Old_Pos)

5. เก็บคำศัพท์ส่วนที่เหลือจากการสร้างมัลติโหนดและซิงเกิลสตริงในอะเรย์เทล ที่ตำแหน่งใหม่โดยใช้ฟังก์ชัน InsStr(r, a_{h+k+2} .....a_n a_{n+1}, POS)

ในตอนเริ่มต้นทั้งอะเรย์เบสและอะเรย์เช็คจะมีเพียงรายการเดียวเท่านั้น คือ BASE[1] = 1 และ CHECK[1] = 1 เมื่อมีการเพิ่มคำศัพท์เข้าไปโครงสร้างของข้อมูลจะเพิ่มขนาดขึ้นเรื่อยๆ



รูปที่ 5-6 แสดงการทำงานของฟังก์ชัน B_Insert

(ก) ดี-เอส ทรีก่อน B_Insert (ข) ดี-เอส ทรีหลัง B_Insert

ฟังก์ชัน Modify(Current_s , h , ADD, ORG)

เป็นฟังก์ชันที่ใช้สำหรับการหาค่าดรรชนีของอะเรย์เบสค่าใหม่ เนื่องจากมีการใช้อะเรย์เบสซ้ำ และทำการปรับค่าของอะเรย์เบสและอะเรย์เช็คใหม่ ซึ่งมีขั้นตอนดังนี้

1. กำหนดค่า old_base = BASE[h]

   หาค่า BASE[h] ใหม่ โดยใช้ฟังก์ชัน X_Check(ADD U ORG)
2. ทำงานซ้ำตั้งแต่ข้อ 3 ถึงข้อ 6 จนกระทั่งค่า c ใน ORG หมด
3. กำหนดค่า t = old_base + c

กำหนดค่า t' = BASE[h] + c

กำหนดค่า BASE[t'] = BASE[t] และ CHECK[t'] = h

$g(s_h, c) = s_t$ ถูกกำหนดใหม่เป็น $g(s_h, c) = s_{t'}$

4. ตรวจสอบว่า BASE[t] มากกว่า 0 หรือไม่

   ถ้าเงื่อนไขเป็นจริง ทำงานในข้อ 5

   ถ้าเงื่อนไขเป็นเท็จ ทำงานในข้อ 6

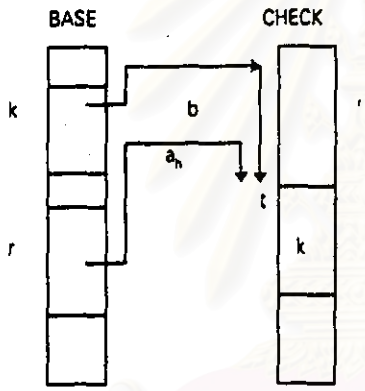
5. กำหนดค่าของ CHECK[q] = t'

   โดยที่ q เป็นไปตามเงื่อนไข CHECK[BASE[t] + b] = t และ q = BASE[t] + b

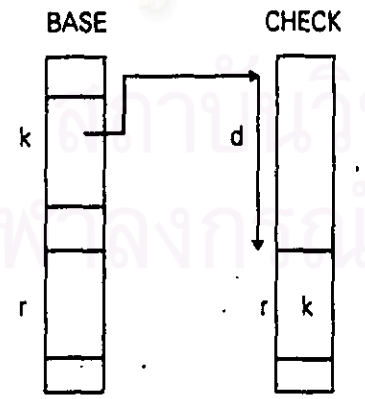
   และกำหนดค่าของ t' = current_s (ถ้า t = current_s)

6. ให้ค่า BASE[t] = 0 และ CHECK[t] = 0

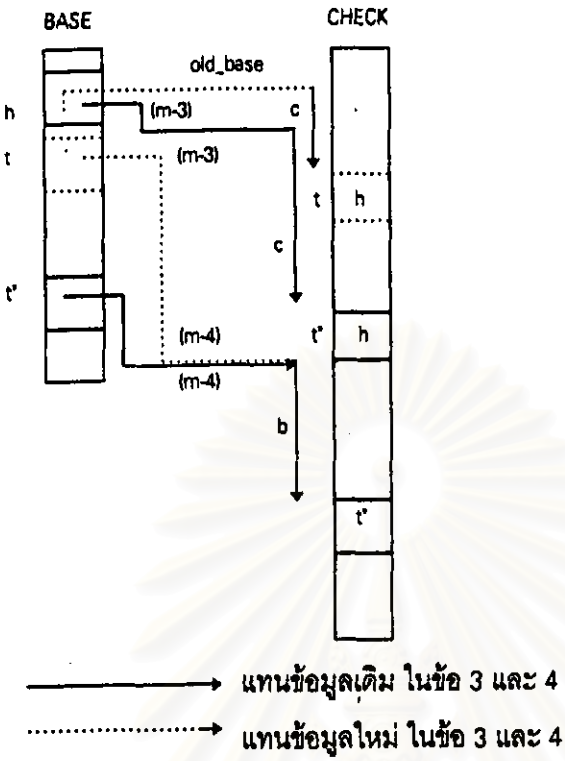
7. ออกจากฟังก์ชันและส่งค่า Current_s กลับ



รูปที่ 5-7 แสดงดับเบิลอะเรย์สำหรับฟังก์ชัน $g(s_k, b) = s_t$ และฟังก์ชัน $g(s_r, a_h) = s_t$



รูปที่ 5-8 แสดงดับเบิลอะเรย์ซึ่งสอดคล้องกับสมการ BASE[k] + d = r



รูปที่ 5-9 แสดงการทำงานของฟังก์ชัน Modify

ฟังก์ชัน Ins_Str(h, e_1e_2 .....e_q, d_pos)

เป็นฟังก์ชันที่ใช้เก็บคำศัพท์ในอะเรย์เทล ณ ตำแหน่งที่ระบุ (d_pos) และหากตำแหน่งนั้นมีข้อมูลอยู่แล้ว จะส่งตำแหน่งสุดท้ายของอะเรย์เทลตำแหน่งใหม่กลับมา ขั้นตอนการทำงานมีดังนี้

1. กำหนดค่าโหนดถัดไป

   t = BASE[h] + e_1

2. กำหนดค่า BASE[t] = -dpos และ CHECK[t] = h
3. ใช้ฟังก์ชัน Str_Tail(d_pos, e_2e_3 .....$e_q$$) เพื่อเก็บสายอักขระ e_2e_3 .....e_q ในอะเรย์เทล ซึ่งฟังก์ชันดังกล่าวจะคืนค่า POS ซึ่งเป็นดรรชนีสูงสุดของอะเรย์เทล หลังจากที่เพิ่มสายอักขระแล้วกลับมา

นอกจากฟังก์ชันหลักๆที่ได้กล่าวไปแล้วนั้นยังมีฟังก์ชันที่ช่วยสนับสนุนการทำงานอีก ดังนี้

ฟังก์ชัน SetList(r)

เป็นฟังก์ชันที่ใช้สำหรับหาชุดของอักขระ a_i ใดๆ ที่สอดคล้องกับสมการ CHECK[BASE[r] + a_i] = r

ฟังก์ชัน N(List)

เป็นฟังก์ชันที่ใช้นับจำนวนข้อมูลใน List พารามิเตอร์ที่เกี่ยวข้องมี ดังนี้

Rlist, Klist, List, ADD, ORG เป็นสัญญลักษณ์ที่ใช้แทนกลุ่มข้อมูลที่เกิดมาจาก I U {#}

POS เป็นตัวแปรชนิดสาธารณะที่ใช้แสดงค่าดรรชนีต่ำสุดของแต่ละรายการในอะเรย์เทล โดยกำหนดค่าเริ่มต้นเป็น 1

ฟังก์ชัน X_Check(List)

เป็นฟังก์ชันที่ใช้หาค่า q ซึ่ง q เป็นค่าของดรรชนีของโหนดที่มีค่าน้อยที่สุดที่สร้างมาจากสมการ CHECK[q +c] = 0 และ c เป็นตัวอักษรที่อยู่ในกลุ่มข้อมูล List

ฟังก์ชัน Str_Tail(p, y)

เป็นฟังก์ชันที่ใช้สำหรับจัดเก็บสายอักขระ y ในตำแหน่ง p ของอะเรย์เทล เมื่อจัดเก็บสายอักขระแล้วจะให้ค่า ดังนี้

ถ้า p = POS จะส่งค่า POS + ความยาวของสายอักขระ y กลับ

ถ้า p < POS จะส่งค่า POS ค่าเดิมกลับ

3. การลบข้อมูลของโครงสร้างแบบดับเบิลอะเรย์

การลบข้อมูลภายในโครงสร้างแบบดับเบิลอะเรย์นั้นจะกระทำได้ต่อเมื่อมีข้อมูลนั้นๆในโครงสร้าง เพราะฉะนั้นในขั้นแรกจะต้องสืบค้นข้อมูลก่อน เมื่อพบแล้วจึงลบคำศัพท์คำนั้นออกไป ฉะนั้นขั้นตอนการลบคำศัพท์สามารถพัฒนามาจากการสืบค้นคำศัพท์เมื่อพบแล้ว จึงลบคำศัพท์นั้นออกไป ซึ่งมีขั้นตอนดังนี้

3.1 ฟังก์ชันการลบคำศัพท์

1. สืบค้นคำศัพท์โดยใช้ฟังก์ชันสืบค้นคำศัพท์ ถ้าพบคำศัพท์ไปทำงานข้อ 2 ถ้าไม่พบคำศัพท์ลบคำศัพท์ไม่ได้ออกจากฟังก์ชันโดยส่งค่าเท็จ
2. ใส่ค่า ? ในอะเรย์เทล ที่ตำแหน่ง -BASE[r] ซึ่งค่าต่างๆได้มาจากฟังก์ชันการสืบค้นนั่นเอง
3. กำหนดค่า BASE[r] = 0 และ CHECK[r] = 0
4. ออกจากฟังก์ชันโดยส่งค่าจริงกลับ

โดยสรุปแล้วขั้นตอนการลบข้อมูลสามารถกระทำได้โดยตัดความสัมพันธ์ของดับเบิลอะเรย์กับอะเรย์เทล โดยแทนค่าของอะเรย์ที่ตำแหน่งของเซพพาเรทโหนดด้วยศูนย์ และแทนที่ส่วนของซิงเกิลสตริงหรือ STR[r] ในอะเรย์เทลด้วยการ์เบจซิมโบล หรือ สัญญลักษณ์ "?"

<u>ตัวอย่างการสืบค้นและเพิ่มคำศัพท์ในดี-เอส ทรี</u>

กำหนดคำศัพท์ที่ต้องการเพิ่มในดี-เอส ทรี ดังนี้ คือ bachelor#, bcs#, badge#, baby#

1. การเพิ่มคำศัพท์ bachelor#
   1.1 หาค่า t = BASE[1] + b = 3
   และ CHECK[3] = 0 ซึ่งไม่เท่ากับ 1 แสดงว่าสร้างมัลติโหนดไม่สำเร็จ
   1.2 เพิ่มคำศัพท์ bachelor# โดยใช้ฟังก์ชัน A_Insert(1, bachelor#) ซึ่งมีขั้นตอนการทำงาน ดังนี้
      1.2.1 หาค่า t = BASE[1] + b = 3 และ CHECK[3] = 0
      1.2.2 ใช้ฟังก์ชัน Ins_Str(1, bachelor#, 1)
         หาค่า CHECK[3] = 1 และ BASE[3] = -1
         เพิ่มซิงเกิลสตริงลงใน TAIL โดย POS = Str_Tail(1, achelor#$) = 10
2. การเพิ่มคำศัพท์ bcs#
   2.1 ทำการสืบค้นคำศัพท์โดย
   หาค่า t = BASE[1] + b = 3 และ CHECK[3] = 1
      BASE[3] = -1 < 0
      h = 1 ไม่เท่ากับ n + 1 = 3
   2.2 หาค่า s_temp = Fetch_str(1) = achelor#
   2.3 ใช้ฟังก์ชัน str_cmp (cs#, achelor#) ซึ่งคืนค่า 0 แสดงว่าสืบค้นคำศัพท์ไม่สำเร็จเนื่องจากเปรียบเทียบซิงเกิลสตริงกับคำศัพท์ส่วนที่เหลือแล้วไม่เหมือนกัน

2.4 เพิ่มสายอักขระส่วนที่เหลือคือ achelor# ลงในทรี โดยใช้ฟังก์ชัน B_insert(3, Ɛ, cs#, achelor#)

ซึ่งมีขั้นตอนการทำงาน ดังนี้

2.4.1 หาค่า old_pos = - BASE[3] = 1

2.4.2 หาค่า BASE[3] = X_Check({c, a}) = 1

2.4.3 ใช้ฟังก์ชัน Ins_Str(3, achelor#, 1)

หาค่า CHECK[2] = 3 และ BASE[2] = -1

เพิ่มซิงเกิลสตริงลงใน TAIL โดย POS = Str_Tail(1, chelor#$) = -10

2.4.4 ใช้ฟังก์ชัน Ins_Str(3, cs#, 10)

หาค่า CHECK[4] = 3 และ BASE[4] = -10

เพิ่มซิงเกิลสตริงลงใน TAIL โดย POS = Str_Tail(10, s#$) = 13

3. การเพิ่มคำศัพท์ badge#

3.1 สืบค้นคำศัพท์โดย

หาค่า t = BASE[1] + b = 3 และ CHECK[3] = 1

หาค่า t = BASE[3] + a = 2 และ CHECK[2] = 3

หาค่า BASE[2] = -1 < 0 และ h = 2 ไม่เท่ากับ n + 1 = 3

3.2 หาค่า s_temp = Fetch_str(1) = chelor#

3.3 ใช้ฟังก์ชัน str_cmp (dge#, chelor#) = 0 แสดงว่าสืบค้นคำศัพท์ไม่สำเร็จเนื่องจากเปรียบเทียบซิงเกิลสตริงกับคำศัพท์ส่วนที่เหลือแล้วไม่เหมือนกัน

3.4 เพิ่มคำศัพท์ส่วนที่เหลือ คือ chelor# โดยใช้ฟังก์ชัน B_Insert (2, Ɛ, dge#, chelor#) = 0

ซึ่งมีขั้นตอนการทำงาน ดังนี้

3.4.1 หาค่า old_pos = -BASE[2] = 1

3.4.2 หาค่า BASE[2] = X_Check({d, c}) = 2

3.4.3 ใช้ฟังก์ชัน Ins_Str(2, chelor#, 1)

หาค่า CHECK[5] = 2 และ BASE[5] = -1

หาค่า POS = Str_Tail(1, helor#$) = 13

3.4.4 ใช้ฟังก์ชัน Ins_Str(2, dge#, 13)

หาค่า CHECK[6] = 2 และ BASE[6] = -13

เพิ่มซิงเกิลสตริงลงใน TAIL โดย POS = Str_Tail(13, ge#$) = 17

4. การเพิ่มคำศัพท์ baby#

4.1 ทำการสืบค้นข้อมูลโดย

หาค่า t = BASE[1] + b = 3 และ CHECK[3] = 1

หาค่า t = BASE[3] + a = 2 และ CHECK[2] = 1

หาค่า t = BASE[2] + b = 4 และ CHECK[4] = 3 ซึ่ง ไม่เท่ากับ 2

แสดงว่าสืบค้นคำศัพท์ไม่พบเนื่องจากสร้างมัลติโหนดไม่สำเร็จ

4.2 เพิ่มคำศัพท์โดยใช้ฟังก์ชัน A_Insert(2, by#) ซึ่งมีขั้นตอนการทำงาน ดังนี้

4.2.1 หาค่า t = BASE[2] + b = 4 และ CHECK[4] = 3 ซึ่งไม่เท่ากับ 0

4.2.2 หาค่า R_LIST = SET_LIST(2) = {c, d}

หาค่า K_LIST = SET_LIST(3) = {a,c}

หาค่า N(R_LIST) + 1 = 3 > N(K_LIST) = 2

4.2.3 ใช้ฟังก์ชัน Modify (2, 3, ϕ, {a, c}) ซึ่งมีขั้นตอนการทำงาน ดังนี้

หาค่า old_base = BASE[3] = 1

หาค่า BASE[3] = X_Check({a, c}) = 6

4.2.4 กรณีที่ a อยู่ใน ORG

หาค่า t = old_base + a = 2

หาค่า t = BASE[3] + a = 7

หาค่า CHECK[7] = 3 และ BASE[7] = BASE[2] = 2

4.2.5 หาค่า BASE[t] = BASE[2] > 0

หาค่า CHECK[5] = CHECK[6] = 7

เนื่องจาก CHECK[5] = CHECK[6] = 2

ฉะนั้น current_s = 2 (เพราะ t = 7 และ current_s = t = 2)

4.2.6 หาค่า BASE[2] = 0 และ CHECK[2] = 0

4.2.7 กรณีที่ c อยู่ใน LIST

หาค่า t = old_base + c = 4 และ t = BASE[3] + c = 9

หาค่า CHECK[9] = 3 และ BASE[9] = BASE[4] = -10

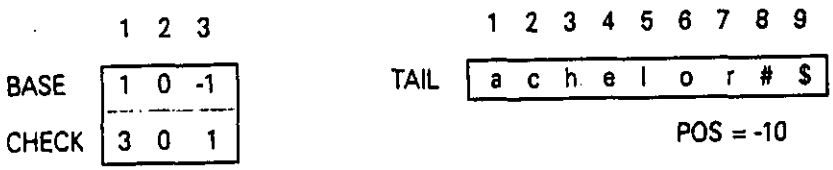
4.2.8 หาค่า BASE[4] = -5 < 0

4.2.9 แทนค่า BASE[4] = 0 และ CHECK[4] = 0

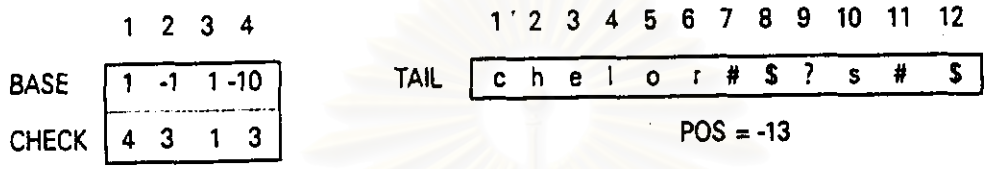
4.2.10 คืนค่า current_s = 7 ซึ่งจะเป็นโหนดใหม่

4.2.11 ใช้ฟังก์ชัน Ins_Str(7, by#, -17)

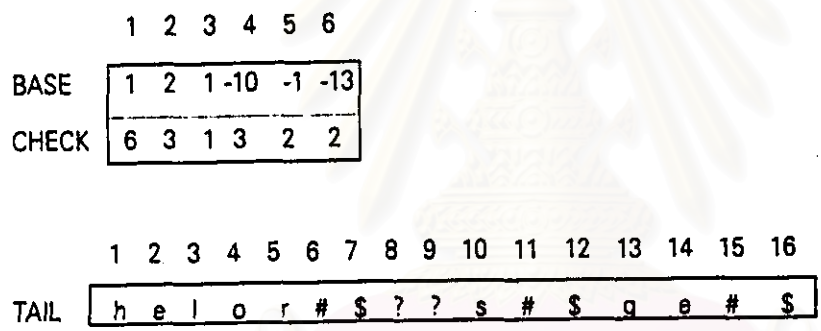
4.2.12 เพิ่มซิงเกิลสตริงลงใน TAIL โดย Str_Tail (17, y#$) = 20



รูปที่ 5-10 แสดงดับเบิลอะเรย์หลังจากเพิ่มคำศัพท์ bachelor# โดยฟังก์ชัน A_Insert(1, bachelor#)



รูปที่ 5-11 แสดงดับเบิลอะเรย์หลังจากเพิ่มคำศัพท์ bcs# โดยฟังก์ชัน B_Insert(3, ε, cs# , achelor#)



รูปที่ 5-12 แสดงดับเบิลอะเรย์หลังจากเพิ่มคำศัพท์ badge#

โดยฟังก์ชัน B_Insert(2, ε, dge# , chelor#)

	1	2	3	4	5	6	7	8	9
BASE	1	0	6	-17	-1	-13	2	0	-10
CHECK	9	0	1	7	7	7	3	0	3

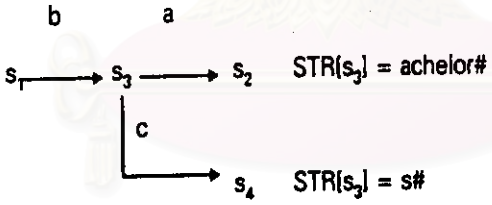
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
TAIL	h	e	l	o	r	#	$	?	?	s	#	$	g	e	#	$	y	#	$

POS = 20

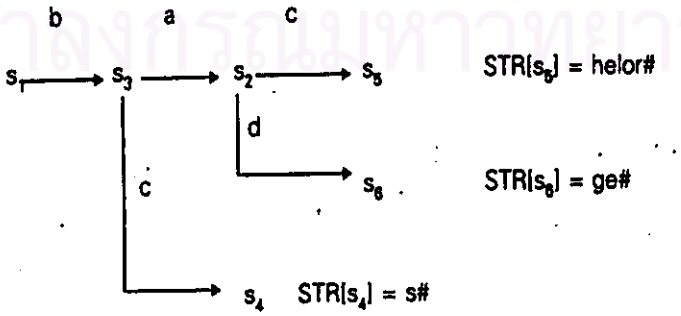
รูปที่ 5-13 แสดงดับเบิลอะเรย์หลังจากเพิ่มคำศัพท์ baby#

โดยฟังก์ชัน A_Insert(2, by#)

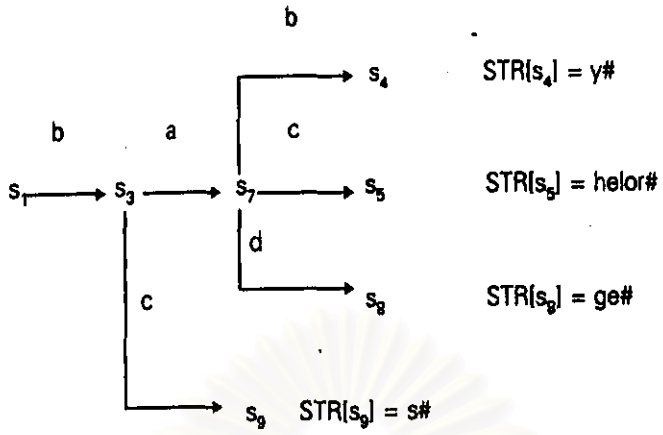
รูปที่ 5-14 แสดงดี-เอส ทรี ของ bachelor#



รูปที่ 5-15 แสดงดี-เอส ทรี ของ bachelor# และ bcs#



รูปที่ 5-16 แสดงดี-เอส ทรี ของ bachelor#, bcs# และ badge#



รูปที่ 5-17 แสดงดี-เอส ทรี ของ bachelor#, bcs#, badge# และ baby#

อัลกอริทึมการลดขนาดข้อมูลข้อมูลโดยวิธีแอลแซดดับบลิว

การลดขนาดข้อมูล ประกอบด้วยโครงสร้าง ดังนี้

```
struct dictionary {
        int code_value;
        int parent_code;
        char character;
} dict[ TABLE_SIZE ];
```

code_value คือ รหัสของวลีปัจจุบัน

parent_code คือ รหัสของวลีก่อนหน้า

character คือ พยัญชนะของโหนดปัจจุบัน

ตัวอย่างเช่น parent_code เก็บรหัสของ GREENLEA character เก็บตัวอักษร F ส่วน code_value เก็บรหัสของ GREENLEAF

จาก struct dict จะเห็นว่าไม่มีพอนย์เตอร์ชี้ไปยังโหนดลูก การหาโหนดลูกจึงใช้แฮชชิงฟังก์ชันเพื่อหาโหนดลูกที่ต้องการ วิธีการ คือ นำ parent_code รวมกับ character เพื่อสร้างเป็น 16 บิตออฟเซ็ท เพื่อหาโหนดที่ต้องการ นอกจากนี้แล้วยังต้องตรวจสอบด้วยว่าโหนดที่พบนั้นถูกใช้โดยโหนดอื่นหรือไม่ การป้องกันการซ้ำกันของโหนดลูกที่ได้จากแฮชชิงฟังก์ชัน วิธีการหนึ่งคือ กำหนดขนาด TABLE_SIZE เป็นตัวเลขไพรม์ (prime number)

ได้กำหนดให้ 255 โหนดแรกใช้สำหรับเก็บพยัญชนะภาษาอังกฤษ เช่น character เก็บตัวอักษร A , code_value เก็บรหัส 65 ส่วน parent_code ไม่มี ฉะนั้นสำหรับโหนดที่สร้างใหม่จะเริ่มตั้งแต่ โหนดที่ 257 เป็นต้นไป ส่วนรหัส 256 จะใช้เป็นตัวบอกสิ้นสุดแฟ้มข้อมูล

การบีบอัดข้อมูลเป็นการอ่านตัวอักษรจากแฟ้มข้อมูลเข้ามาและแทนค่าออกมาเป็นรหัส โดยในขั้นแรกจะกำหนดค่าเริ่มต้นให้กับอะเรย์เป็น -1 โดยที่ 256 โหนดแรกนั้นเป็นโหนดพิเศษ จะเริ่มใช้ตั้งแต่โหนดที่ 257 เป็นต้นไป วิธีการ คือ อ่านตัวอักษรเข้ามาทีละตัว และเก็บเป็นค่า string_code หลังจากนั้นเข้าแฮชชิงฟังก์ชันเพื่อตรวจสอบว่า string_code นี้มีโหนดลูกที่สัมพันธ์กันหรือไม่ ถ้าพบจะนำรหัสของโหนดลูกมาเป็น string_code แต่หากไม่พบโหนดลูกที่ต้องการ ก็จะส่งค่าของโหนดปัจจุบันออกมา และเริ่มต้นกับ string_code ใหม่ โดยนำ string_code ใหม่ รวมกับ character กำหนดเป็นรหัสใหม่ เพื่อว่าหากพบวลีเช่นนี้อีกจะได้แทนค่าด้วยรหัสนี้ได้เลย ต่อไปทำเช่นเดิมจนกระทั่งพบตัวชี้สิ้นสุดแฟ้มข้อมูล

สำหรับการขยายข้อมูล ไม่จำเป็นต้องใช้แฮชชิงฟังก์ชันแต่การขยายข้อมูลเป็นการกระทำที่ย้อนกลับกับการบีบอัดข้อมูล ฉะนั้นจึงต้องนำรหัสบันทึกลงแสตก แล้วจึงปอป (pop) ออกมา และบันทึกลงเอาต์พุตไฟล์ การขยายข้อมูลจะอ่านรหัสจากแฟ้มข้อมูลเข้ามาแทนค่าด้วยสายอักขระและส่งออกเป็น

เอาต์พุต ต่อไปสร้างสายอักขระใหม่เพื่อเก็บไว้ใช้ในการถอดรหัสต่อไป โดยนำ old_code มารวมกับ ตัวอักษรแรกของสายอักขระปัจจุบัน และอ่านรหัสใหม่จากแฟ้มข้อมูลเข้ามาทำเช่นเดิมจนกระทั่งพบตัวชี้สิ้นสุดแฟ้มข้อมูล