บทที่ 2

โครงสร้างข้อมูลทรัยแถวคู่

หากแบ่งการสืบค้นคีย์ตามลักษณะการใช้ จะแบ่งเป็นการสืบค้นแบบจลน์สำหรับเซ็ตของคีย์ที่มีลักษณะจลน์คือมีการเปลี่ยนแปลงคีย์บ่อยๆ และการสืบค้นแบบสถิตย์สำหรับเซ็ตของคีย์ที่มีลักษณะสถิตย์คือไม่มีการเปลี่ยนแปลงคีย์เลย ในการสืบค้นแบบจลน์จะสนใจในเรื่องการเปลี่ยนแปลงคีย์เป็นหลัก ประสิทธิภาพในการใช้เนื้อที่และความเร็วในการสืบค้นจึงค่อนข้างต่ำ

แม้ว่าในการประยุกต์ใช้งานจริง เช่นในการประมวลผลภาษาธรรมชาติจะทราบถึงคีย์(คำศัพท์)ส่วนใหญ่ที่จะใช้อยู่แล้ว ต่อมาในภายหลังจึงเพิ่มคำศัพท์เฉพาะเข้าไปในเซ็ตของคำศัพท์เดิม วิธีการสืบค้นที่ใช้กับเซ็ตของคำศัพท์เหล่านี้เป็นวิธีสืบค้นแบบสถิตย์ซึ่งมีความเร็ว และประหยัดเนื้อที่ในการเก็บ และจะมีประสิทธิภาพมากที่สุดหากสามารถเปลี่ยนแปลงเพิ่มเติมคำศัพท์ได้ แต่ว่าการเพิ่มฟังก์ชั่นในการเพิ่มคีย์สำหรับวิธีการสืบค้นแบบสถิตย์นี้ทำได้ยาก

Aoe[6][7][8][9] ได้เสนอโครงสร้างข้อมูลทรัยแถวคู่เพื่อใช้ทำการเปรียบแบบ (Pattern Matching) ที่มีความเร็วสำหรับคีย์จำนวนมาก เนื่องจากเวลาที่ช้าสุดในการเปรียบ (Matching) ของทรัยแถวคู่นี้เป็นลำดับของความยาวของคีย์ หากนำมาใช้กับการสืบค้นคีย์แล้วจะเห็นถึงประสิทธิภาพของการสืบค้นแบบเชิงเลข

การเปรียบแบบ (Pattern Matching) ของทรัยแถวคู่

เมื่อให้ K เป็นเซ็ตจำกัดของคีย์ ให้เครื่องเปรียบแบบ (Pattern Matching Machine) คือการกำหนดว่าสตริงอินพุต x มีอยู่ในเซ็ต K หรือไม่ ให้ M เป็นเครื่องสถานะจำกัด (Finite State Machine) ที่ใช้กับเซ็ต K แสดงได้ดังนี้

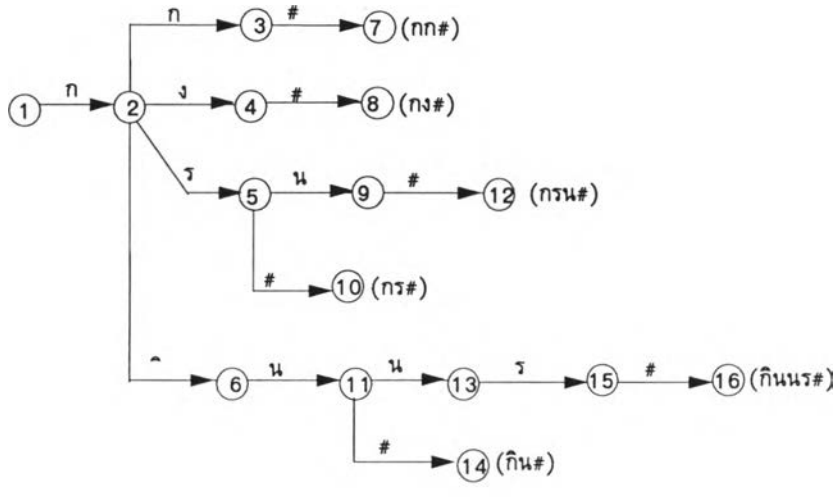
$$M = (S,I,g,s_1,U)$$

โดย S เป็นเซ็ตจำกัดของสถานะ; I เป็นเซ็ตจำกัดของตัวอักษรและสัญญลักษณ์ที่เป็นอินพุต; s_1 เป็นสถานะเริ่มต้น (Start State) ; g เป็นฟังก์ชั่นการผ่าน (Transition Function) เรียกว่าฟังก์ชั่น goto

; U (<= S) เป็นเซ็ตจำกัดของสถานะออก (Output State); และจากนี้จะแสดงสถานะที่ r (โดย r เป็นดรรชนีแสดงลำดับที่ของสถานะ) ด้วย s_r

หากกำหนดให้ฟังก์ชั่น g เป็นฟังก์ชั่นจาก S × I ไปยัง S ∪ {fail} และการผ่านจากสถานะ s_r ไปยังสถานะ s_t คือสัญญลักษณ์ a แล้ว $g(s_r,a) = s_t$ และหากไม่มีการผ่าน a จากสถานะ s_r แล้ว $g(s_r,a)$ = {fail} จากนี้ไปจะแสดงจำนวนของการผ่านที่เข้าสู่ s_r ด้วย indegree(s_r) และแสดงจำนวนของการผ่านที่ออกจาก s_r ด้วย outdegree(s_r) เรียก s_r ที่มี outdegree(s_r)=0 ว่าสถานะสุดท้าย (Final State) ให้ F เป็นเซ็ตของสถานะสุดท้าย (Final State)

กำหนดให้อักษรไทย ก,ข,ค,...,ฮ,ั,ิ,...,ะ,า ∈ I ; K1 = { กก#, กง#, กง#, กร#, กรน#, กิน#, กินนร# } แสดงเครื่องเปรียบแบบ (Pattern Matching Machine) ของ K1 ได้ดังรูปที่ 2.1 # เป็น Endmarker บอกจุดสิ้นสุดของแต่ละคีย์

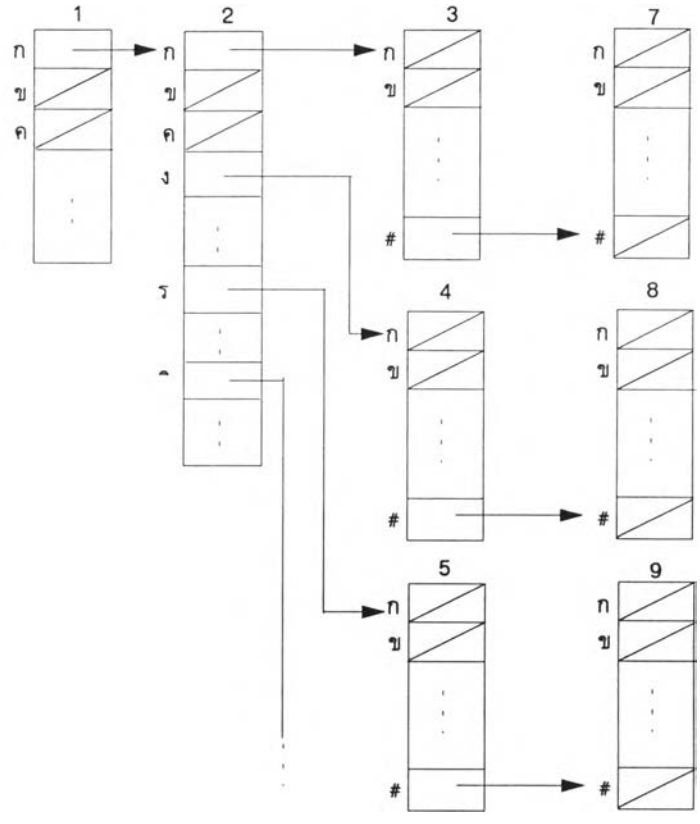


รูปที่ 2.1 เครื่องเปรียบแบบ (Pattern Matching Machine) ของ K1

พิจารณาการสืบค้น กิน# เริ่มต้นที่สถานะที่ 1 ตัวอักษร ก ทำให้วิ่งจาก 1 ไป 2 (g(1,b) = 2) ต่อไป คือ 2—ิ—6 (g(2,ิ)=6) , 6—น—11 (g(6,ิ)=11) , 11—#—14 (g(11,#)=14) พบว่าคีย์ กิน# อยู่ใน K1 หากพิจารณา กรอก# พบว่า เมื่อถึงสถานะที่ 5 แล้วไม่สามารถไปต่อไปได้ (g(5,อ) = {fail}) แสดงว่า กรอก# ไม่อยู่ใน K1

หากแสดงโครงสร้างข้อมูลทรัยของเครื่องสถานะจำกัด (Finite State Machine) ของเซ็ต K1 ในรูปที่ 2.1 บางส่วน จะเป็นดังในรูปที่ 2.2

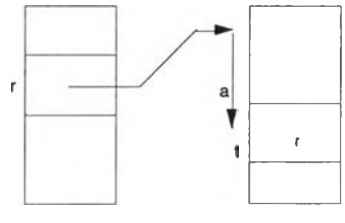
จากรูปที่ 2.2 แสดงทรัยด้วยอาร์เรย์ 1 มิติ เชื่อมต่อแต่ละโหนดของเครื่องสถานะจำกัด (Finite State Machine) โดยขนาดของอาร์เรย์เท่ากับตัวอักษรที่ใช้ เวลาที่ใช้ในการสืบค้นที่มากที่สุดของทรัยไม่สัมพันธ์กับจำนวนคีย์ แต่สัมพันธ์กับความยาวของคีย์ แต่ว่าสมาชิกในอาร์เรย์ของแต่ละสถานะในทรัยมักไม่ค่อยได้ใช้ จึงเป็นจุดด้อยข้อหนึ่งของทรัยที่กินเนื้อที่หน่วยความจำมากเกินไป



รูปที่ 2.2 โครงสร้างข้อมูลทรัยของ K1

ในวิธีทรัยแถวคู่นี้จะแสดงเครื่องสถานะจำกัด (Finite State Machine) เป็นอาร์เรย์ 1 มิติ 2 อาร์เรย์คือ BASE และ CHECK และแสดงการผ่านของ $g(s_r,a) = s_t$ ด้วย

t = BASE[r]+a ; CHECK[t] = r



รูปที่ 2.3 โครงสร้างข้อมูลทรัยแถวคู่

เมื่อกำหนดให้อักษร a มีค่าเชิงจำนวน (Numerical Value) เป็น 1 แล้ว วิธีทรัยแถวคู่เป็นการใช้ความสัมพันธ์ของสถานะที่ r และค่าเชิงจำนวน (Numerical Value) ของ a ส่วนการผ่านกำหนดด้วยค่าดรรชนีของ CHECK ดังในรูปที่ 2.3

การแก้ไขทรัยแถวคู่

1. เงื่อนไขเพิ่มเติมของทรัยแถวคู่

ให้อักษร a,b,c,d $\in$ I ; และสตริง x,y $\in$ I* ส่วนตัวอักษรและสตริงอื่นๆนอกเหนือจากนี้ให้แสดงอยู่ภายในเครื่องหมาย ' ' และ " " ตามลำดับ และกำหนดตัวอักษรว่าง (Null Character) เป็น θ ; สตริงว่าง (Empty String) เป็น ε

จากนี้ไป กำหนดให้ใส่สัญญลักษณ์พิเศษ # ไว้ท้ายคีย์ เพื่อแยกสถานะสุดท้าย (Final State) และสถานะออก (Output State) ในขั้นต้นจะแสดงให้เห็นถึงประสิทธิภาพในการใช้เนื้อที่ของทรัยแถวคู่

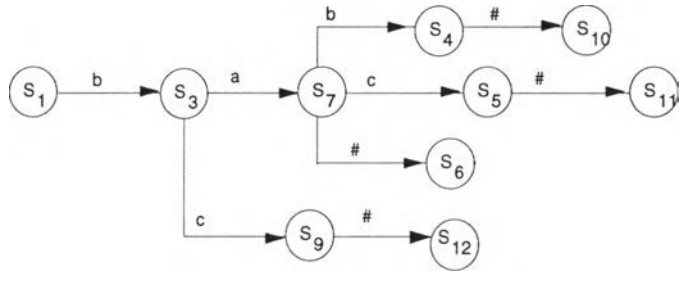
[ข้อกำหนด-1]กำหนดนิยามของสถานะต่างๆในเงื่อนไขของฟังก์ชัน goto ดังนี้

(1) ในลำดับของการผ่านที่ออกจากสถานะ s_t ใน $g(s_r,a) = s_t$ โดย $outdegree(s_t) \geq 2$ ถึงสถานะสุดท้าย (Final State) หากไม่มีสถานะ s_k ที่ $outdegree(s_k) \geq 2$ แล้วเรียกสถานะ s_t นั้นว่าสถานะแยก (Separate State) และให้ S_P เป็นเซ็ตของสถานะแยกเหล่านี้

(2) ในลำดับของการผ่านจากสถานะ s_1 ถึงสถานะแยก (Separate State) เรียกสถานะในลำดับการผ่านนี้(รวมสถานะเริ่มต้นยกเว้นสถานะแยก)ว่าหลายสถานะ (Multi-state) และให้ S_M เป็นเซ็ตของหลายสถานะ (Multi-state) เหล่านี้

(3) ในลำดับของการผ่านจากสถานะแยกถึงสถานะสุดท้าย เรียกสถานะในลำดับการผ่านนี้ (รวมสถานะแยกยกเว้นสถานะสุดท้าย) ว่าสถานะเดี่ยว (Single State)

[ข้อกำหนด-2]จาก $g(s_r,y)=s_t$ โดย $s_r \in S_P$ และ $s_t \in F$ เรียกสตริง y ของ s_r นี้ว่าสตริงเดี่ยว (Single String) แสดงด้วย $STR[s_r]$



รูปที่ 2.4 ฟังก์ชั่น goto ของ K'

[ตัวอย่างที่ 1] ให้ K' เป็นเซ็ตของคีย์คือ {bac#, bc#, ba#, bab#} แสดงฟังก์ชั่น goto ของ K' ได้ดัง รูปที่ 2.4 สรุปสถานะได้ดังนี้

สถานะเริ่มต้น (Start State) คือ s_1;

สถานะสุดท้าย (Final State) คือ $s_{10}, s_{11}, s_6, s_{12}$;

หลายสถานะ (Multi-state) คือ s_1, s_3, s_7 ;

สถานะเดี่ยว (Single State) คือ s_4, s_5, s_9 ;

สถานะแยก (Separate State) คือ s_4, s_5, s_6, s_9 ;

สตริงเดี่ยว (Single String) ของแต่ละสถานะแยก (Separate State) เป็นดังนี้

$STR[s_4] = \#$ $STR[s_5] = \#$

$STR[s_6] = \varepsilon$ $STR[s_9] = \#$

ด้วยวิธีการที่จะนำเสนอนี้ เก็บการผ่านของ $S_M \times (S_M \cup S_P)$ ไว้ในอาร์เรย์คู่คือ BASE และ CHECK และเก็บการผ่านที่เริ่มจากสถานะแยก (Separate State) เป็นสตริงเดี่ยว (Single String) ไว้ในอาร์เรย์ TAIL จากวิธีการที่ปรับปรุงขึ้นนี้จะได้ผลดังนี้

(1) จากจำนวนที่ลดลงของจำนวนการผ่านที่เก็บในอาร์เรย์คู่ สามารถลดจำนวนไบต์ของสมาชิกอาร์เรย์ได้ เพราะว่าในอาร์เรย์คู่จะเก็บ 1 การผ่านด้วยสมาชิกของอาร์เรย์ 2 ตัว แต่ใน TAIL เก็บตัวอักษรที่เป็นสตริงเดี่ยว (Single String) จึงประหยัดเนื้อที่หน่วยความจำ

(2) เนื่องจากการเปรียบเทียบการผ่านตั้งแต่สถานะเดี่ยว (Single State) เป็นการทำการเปรียบแบบ (Pattern Matching) เท่านั้น การสืบค้นจึงกระทำได้รวดเร็ว

ในส่วนของการปรับปรุงที่เพิ่มขึ้นนี้ จำเป็นต้องมีวิธีที่มีประสิทธิภาพในการดูว่ามีสถานะแยก (Separate State) ในการผ่านที่ประมวลผลอยู่หรือไม่ และในการหาตำแหน่งของสตริงเดี่ยว (Single String) ใน TAIL ดังนั้น จึงกำหนดเงื่อนไขเพิ่มเติมดังนี้

[ข้อกำหนด-3]ทรัยแถวคู่ต้องมีลักษณะเป็นไปตามในเงื่อนไข-A ดังนี้

เงื่อนไข-A

(A-1) หาก $g(s_r, a) = s_t$; $s_r \in S_M$; $s_t \in S_M \cup S_P$ แล้ว

$BASE[r] + a = t$, $CHECK[t] = r$

(A-2) สำหรับทุกสถานะที่ r ของ $s_r \in S_P$

$BASE[r] < 0$

(A-3) สำหรับสถานะแยก s_r ใน $STR[s_r] = b_1 b_2 b_3 \ldots b_m$ $(1 \leq M)$ โดย $s_r \in S_P$

$p = -BASE[r]$;

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

เงื่อนไข(A-1) เหมือนกับการเปรียบแบบ (Pattern Matching) ของทรัยแถวคู่ ส่วนเงื่อนไข (A-2) และ (A-3) เป็นการกำหนดว่า หาก BASE[r] เป็นลบแล้ว s_r จะเป็นสถานะแยกและสามารถเข้าถึงสตริงเดี่ยวของสถานะแยกนั้นได้ที่ TAIL ตั้งแต่ตำแหน่งที่ -BASE[r]

นอกจากนี้ในการกำหนดค่ามากสุดของดรรชนีของทรัยแถวคู่ ต้องกำหนดสิ่งต่อไปนี้

[ข้อกำหนด-4]กำหนดให้สมาชิกที่ยังไม่ได้ใช้ของอาร์เรย์คู่มีค่าเป็น 0

และอาร์เรย์คู่ต้องมีลักษณะเป็นไปตามในเงื่อนไข-B ดังนี้

เงื่อนไข-B

กำหนดค่าของ CHECK[1] ที่ตรงกับสถานะเริ่มต้น(สถานะที่ 1) มีค่าเท่ากับค่าดัชนี q ที่มากที่สุดที่ทำให้ CHECK[q] $\neq$ 0

นอกจากนี้ เพื่อให้จัดการกับระเบียนของคีย์ได้ง่ายขึ้น แทนที่จะเก็บเป็นระเบียนของคีย์ใน TAIL จะเก็บ $ ต่อท้ายแต่ละระเบียนเข้าไปใน TAIL

2. การสืบค้นคีย์

อัลกอริธึมการสืบค้นของทรัยแถวคู่ที่ปรับปรุง เป็นดังนี้

Algorithm 1. Retrieval Algorithm.

Input A string x# = $a_1a_2 \ldots a_n a_{n+1}$; a_{n+1} = #;
and the machine M with the double array and TAIL for K.

Output If x $\in$ K then the output is TRUE. Otherwise FALSE

Method

```
begin
        r:= 1; h:= 0
        repeat
            h := h+1; t := BASE[r] + a_h;
(1-1)       if ( t > CHECK[1] ) or (CHECK[t] ≠ r)
            then return(FALSE) else r := t;
        until BASE[r] < 0
(1-2)   if (h = n+1) then return (TRUE)
        else S_TEMP = FETCH_STR(-BASE[r]);
(1-3)   if STR_CMP(a_{h+1} ... a_n a_{n+1}, S_TEMP) = -1
            then return(TRUE) else return(FALSE)
end
```

ตัวแปรและฟังก์ชั่นที่ใช้เป็นดังนี้

S_TEMP ตัวแปรเก็บสตริงที่นำมาจาก TAIL

FETCH_STR[p] ฟังก์ชั่นคืนสตริงของตัวอักษร(รวม #)ที่ได้จาก TAIL ตั้งแต่ TAIL[p] ถึง TAIL[p+K] = # โดย $0 \leq K$

STR_CMP(y,z) หากสตริง y เท่ากับสตริง z แล้วคืนค่า -1 หากไม่เท่ากันแล้วคืนค่าความยาวของเฉพาะส่วนที่เหมือนกัน

ในบรรทัด (1-1) ของ Algorithm-1 ในกรณีที่ t มีค่าเกินดรรชนีที่มากที่สุดของ CHECK หรือการผ่านนั้นยังไม่ถูกกำหนดไว้จะคืนค่า FALSE ในช่วง repeat จะหยุดเมื่อ BASE[r] มีค่าเป็นลบ และในบรรทัด (1-2) นำสตริงเดี่ยว STR[s_r] เก็บใน S_TEMP (นอกจากนี้หาก h=n+1 แสดงว่าเปรียบสำเร็จคืนค่า TRUE) ต่อมาในบรรทัด (1-3) ตรวจสอบว่า x อยู่ใน K หรือไม่ โดยเปรียบเทียบ S_TEMP กับสตริงอินพุตที่เหลืออยู่

นั่นก็คือสำหรับฟังก์ชั่น goto ของเซ็ตของคีย์ K หากทรัยแถวคู่เป็นไปตามเงื่อนไข-A และเงื่อนไข-B แล้ว ใน Algorithm-1 จะคืนค่า TRUE หากอินพุต $x \in K$ และคืนค่า FALSE หากอินพุต $x \notin K$

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

	1	2	3	4	5	6	7	8	9
TAIL	#	$	?	?	#	$	$	#	$

รูปที่ 2.5 ทรัยแถวคู่ของ K'

[ตัวอย่าง-2] แสดงทรัยแถวคู่ของเซ็ตของคีย์ K' ในรูปที่ 2.5 โดยสัญญลักษณ์ a, b, c, # มีค่าเชิงจำนวน (numerical value) เป็น 1, 2, 3, 4 ตามลำดับ สัญญลักษณ์ ? ใน TAIL เป็นการบ่งถึงไบต์ที่ยังไม่ได้ใช้ การสืบค้นคีย์ bc# เป็นดังนี้

t = BASE[1] + 2 = 3

(1-1) : t = 3 $\leq$ CHECK[1] = 9 และ

CHECK[3] = 1

BASE[r] = BASE[3] = 6 > 0,

t = BASE[3] + c = 9

(1-1) : t = 9 $\leq$ CHECK[1] = 9 และ

CHECK[9] = 3

BASE[r] = BASE[9] = -5 < 0

(1-2) : เนื่องจาก h = 2 $\neq$ n+1 =3 จากฟังก์ชั่น FETCH_STR เก็บ "#" ลงใน

S_TEMP

(1-4) : เปรียบเทียบอินพุตสตริงที่เหลืออยู่คือ "#" กับ S_TEMP ด้วยฟังก์ชั่น STR_CMP พบว่าเท่ากัน แสดงว่า bc# ∈ K'

3. การเพิ่มคีย์

ให้ BASE[1] มีค่า 1 และ CHECK[1] มีค่า 0 อัลกอริธึมการเพิ่มคีย์ Algorithm-2 ได้โดยเปลี่ยนบรรทัด (1-1) และ (1-3) คือ return(FALSE) ของ Algorithm-1 เป็นดังบริเวณตัวเอนดังนี้

Algorithm 2. Insert Algorithm.

```
begin
        r:= 1; h:= 0
        repeat
            h := h+1; t := BASE[r] + a_h;
(1-1)       if ( t > CHECK[1] ) or (CHECK[t] ≠ r) then
            begin
                A_INSERT(r, a_h a_h+1 ... a_n a_n+1 ) ;
                return(FALSE) ;
            end
            else r := t;
        until BASE[r] < 0

(1-2)   if (h = n+1) then return (TRUE)
        else S_TEMP = FETCH_STR(-BASE[r]);

(1-3)   if STR_CMP(a_h+1 ... a_n a_n+1, S_TEMP) = -1
        then return(TRUE)
        else
        begin
       /* กำหนดให้ S_TEMP เป็น a_h+1 a_h+2 ... a_h+r b_1 ... b_m ;
        ให้ input string ที่เหลือ อยู่คือ a_h+1 a_h+2 ... a_h+r a_h+r+1 ... a_n a_n+1 ;
        และ b_1 ≠ a_h+r+1 โดย 0 ≤ r ≤ n-h+1 , 0 ≤ m */
            B_INSERT(r, a_h+1 a_h+2 ... a_h+r , a_h+r+1 ... a_n a_n+1 , b_1 ... b_m ) ;
            return(FALSE) ;
```

```
        end
end
```

กระบวนคำสั่ง (procedure) และฟังก์ชั่นที่สำคัญที่ใช้ใน Algorithm-2 เป็นดังนี้

```
procedure A_INSERT( r, b1b2...bn )
begin
(a-1)   t := BASE[r] + b1 ;
(a-2)   if CHECK[t] ≠ 0 then
        begin
(a-3)       LIST1 := SET_LIST(r) ;
(a-4)       LIST2 := SET_LIST(CHECK[t]) ;
(a-5)       if N(LIST1) + 1  <  N(LIST2) then
(a-6)           r := MODIFY(r, r, b1, LIST1) ;
            else
(a-7)           r := MODIFY(r, CHECK[t], θ, LIST2) ;
        end
(a-8)   INS_STR(r, b1 b2 ... bu, POS) ;
end

procedure B_INSERT(r, x, y, z)
/* x = b1 b2 ... bk ; y = c1 c2 ... ch ; z = d1 d2 ... du ; */
begin
(b-1)   old_pos := -BASE[r] ;
        for i := 0 to STR_LEN(b1 b2 ... bk) do
        begin
(b-2)       BASE[r] = X_CHECK(bi) ;
(b-3)       CHECK[BASE[r] + bi] := r ;
(b-4)       r := BASE[r] + bi ;
        end
(b-5)   BASE[r] := X_CHECK({ci, di}) ;
(b-6)   INS_STR(r, d1 d2 ... du, old_pos) ;
(b-6)   INS_STR(r, c1 c2 ... ch, POS) ;
end
```

```
function      MODIFY(current, r, a, LIST)
begin
(m-1)     old_base := BASE[r] ;
(m-2)     BASE[r] := X_CHECK(LIST ∪ {a}) ;
          for each c in LIST do
          begin
(m-3)         t := old_base + c ;  t' := BASE[r] + c ;
(m-4)         CHECK[t'] := CHECK[t] ;    BASE[t'] := BASE[t] ;
(m-5)         if BASE[t] > 0 then
              begin
(m-6)             for each character q such that CHECK[q] = t do
(m-7)                 CHECK[BASE[t] + q] := t' ;
(m-8)             if (t = current)   current := t' ;
              end
(m-9)         BASE[t] := 0 ;       CHECK[t] := 0 ;
          end
          return(current) ;
end

procedure INS_STR(r, e1 e2 ... eq , d_pos)
begin
(s-1)     t := BASE[r] + e1 ;
(s-2)     CHECK[t] := r ;          BASE[t] := -d_pos ;
(s-3)     POS := SET_STR(d_pos, e2 e3 ... eq $) ;
end
```

ตัวแปรและฟังก์ชั่นที่ใช้เป็นดังนี้

SET_LIST(r) : เป็นฟังก์ชั่นคืนเซ็ตของสมาชิกของสัญญลักษณ์ a ที่ทำให้ $g(s_r,a) \neq \{fail\}$

LIST, LIST1, LIST2 : ตัวแปรเก็บสับเซ็ตของ {#} ∪ I

N(LIST) : ฟังก์ชั่นคืนจำนวนสมาชิกของ LIST

POS : ตัวแปรส่วนกลางเก็บค่าความยาวของ TAIL บวกด้วย 1 เริ่มต้นมีค่าเป็น 1

current : ตัวแปรเก็บค่าสถานะใหม่ของสถานะ r ที่เปลี่ยนแปลงไปด้วยฟังก์ชั่น MODIFY

X_CHECK(LIST) : ฟังก์ชั่นคืนค่าดัชนี q ที่เล็กที่สุด โดย c ทั้งหมดที่เป็นสมาชิกของ LIST ทำให้ CHECK[q+c] = 0 และเก็บขนาดของอาร์เรย์คู่ใน CHECK[1]

SET_STR(p, y) : ฟังก์ชั่นนำสตริง y ไปไว้ใน TAIL ที่ตำแหน่ง p เป็นต้นไป หาก p เท่ากับ POS คืนค่า POS บวกด้วยความยาวของสตริง y หากไม่เท่ากันคืนค่า POS

STR_LEN(z) : ฟังก์ชั่นคืนค่าความยาวของสตริง z

ในกระบวนคำสั่ง A_INSERT หากไม่สามารถสืบค้นคีย์ได้ในทรัยแถวคู่ ทำการเพิ่มคีย์นั้นเข้าไป และหากจำเป็นต้องปรับเปลี่ยน BASE[r] การพิจารณาว่าจะเปลี่ยนที่ใดทำโดยเปรียบเทียบ N(LIST1)+1 (ผลรวมของจำนวนการผ่านที่ออกจากสถานะ s_r กับสถานะใหม่ของ $g(s_r,b_1)$) กับ N(LIST2) (จำนวนการผ่านที่ออกจาก CHECK[t]) แล้วจึงเปลี่ยนค่าที่ BASE ของสถานะของที่น้อยกว่าด้วยฟังก์ชั่น MODIFY

ในกระบวนคำสั่ง INS_STR หาการผ่านที่เข้าสู่สถานะแยก (Separate State) แล้วเก็บสตริงเดี่ยว (Single String) นั้นใน TAIL ส่วนในกระบวนคำสั่ง B_INSERT หากไม่สามารถสืบค้นคีย์ได้ใน TAIL ทำการเพิ่มคีย์นั้น โดยอาร์กิวเมนต์ที่ 2 คืออุปสรรค (prefix)
ส่วนที่เหมือนกันของสตริงอินพุตที่เหลืออยู่ xy และ S_TEMP คือ xz ส่วนอาร์กิวเมนต์ที่ 3, 4 คือสตริง y, z ของ xy, xz ที่ตัดอุปสรรค x ออก

ขนาดของอาร์เรย์คู่ที่เก็บใน CHECK[1] ถูกเปลี่ยนแปลงด้วยฟังก์ชั่น X_CHECK ซึ่งเป็นไปตามเงื่อนไข-B

ใน Algorithm-2 เมื่อเพิ่มอินพุต x โดย $x \notin K$ เข้าไปในทรัยแถวคู่แล้ว ทรัยแถวคู่ที่ได้นั้นเป็นไปตามเงื่อนไข-A และเงื่อนไข-B ดังนี้

(1) ในกระบวนคำสั่ง A_INSERT

เมื่อเงื่อนไขในบรรทัด (a-2) เป็นเท็จ เรียก INS_STR ในบรรทัด (a-8) แล้วเก็บการผ่าน $g(s_r,b_1)$ ในอาร์เรย์คู่ในบรรทัด (s-1), (s-2) และเก็บสตริงเดี่ยวใน TAIL ในบรรทัด (s-3) แต่เนื่องจาก t ต้องเป็นสถานะแยก (Separate State) แน่นอน จึงเป็นไปตามเงื่อนไข-A และหากบรรทัด (a-2) เป็นจริงแล้วด้วย MODIFY ในบรรทัด (a-6) ซึ่งได้จากการเปรียบเทียบในบรรทัด (a-5) สามารถแสดงผลของเงื่อนไข (A-1) ได้ 3 อย่างดังนี้

(a) การกำหนดการผ่านที่ได้จาก BASE[r] ใหม่

BASE[r] ใหม่ที่ได้จาก X_CHECK ในบรรทัด (m-2) ทำให้ CHECK[BASE[r]+b] = 0 โดย b คือสมาชิกทั้งหมดของ LIST $\cup$ {a} ในบรรทัด (m-4) เป็นการกำหนดการผ่าน $g(s_r,c)$ โดย $c \in$ LIST และต่อมาในบรรทัด (m-8) เนื่องจาก current (ซึ่ง = r) ไม่เปลี่ยนแปลง เรียก INS_STR ในบรรทัด (a-8) ในบรรทัด (s-1) (s-2) ของ INS_STR จะได้การผ่านใหม่คือ $g(s_r,b_1)$ เก็บไว้ในอาร์เรย์คู่อย่างถูกต้อง

(b) ผลกระทบต่อการผ่านอื่น

ในการเปลี่ยน BASE[r] ในบรรทัด (m-3) สถานะ t เปลี่ยนเป็นสถานะใหม่ t' แต่ในบรรทัด (m-4) เก็บ BASE[t] ใน BASE[t'] และในบรรทัด (m-7) เปลี่ยนค่าสมาชิกของ CHECK ที่ทำให้ CHECK[BASE[t]+b] = t เป็น t' ดังนั้นจึงไม่เกิดข้อขัดแย้งขึ้น

(c) การลบการผ่านเก่า

แสดงไว้ในบรรทัด (m-9) นอกจากนี้เงื่อนไข (A-2) (A-3) ยังแสดงไว้ในบรรทัด (s-2) ถึงบรรทัด (s-3) ของ INS_STR ซึ่งถูกเรียกในบรรทัด (a-8)

และใน MODIFY ซึ่งถูกเรียกในบรรทัด (a-7) มีการเปลี่ยน BASE[CHECK[t]] แทนที่จะเปลี่ยน BASE[r] ดังนั้นสถานะ r จึงมีโอกาสเปลี่ยนแปลงได้ ผลคืออาร์กิวเมนต์ที่ 1 คือ r ถูกส่งไปเป็น current และเนื่องจากหาก r เปลี่ยนไป current จะถูกเปลี่ยนเป็น t' ในบรรทัด (m-8) จึงไม่เกิดขัดแย้งกัน

	1	2	3
BASE	1	0	-1
CHECK	3	0	1

POS = 5

TAIL			
1	2	3	4
a	c	#	$

(a) ผลของ A_INSERT(1, bac#) สำหรับ bac#

	1	2	3	4
BASE	1	-1	1	-5
CHECK	4	3	1	3

POS = 7

TAIL					
1	2	3	4	5	6
c	#	$	?	#	$

(b) ผลของ B_INSERT(3,Ɛ,c#,ac#) สำหรับ bc#

	1	2	3	4	5	6
BASE	1	2	1	-5	-1	-7
CHECK	6	3	1	3	2	2

POS = 8

TAIL						
1	2	3	4	5	6	7
#	$	?	?	#	$	$

(c) ผลของ B_INSERT(2,Ɛ,#,c#) สำหรับ ba#

รูปที่ 2.6 ขั้นตอนการเพิ่มคีย์

(2) ในกระบวนคำสั่ง (procedure) B_INSERT

บรรทัด (b-2) ถึง (b-4) เป็นการกำหนดการผ่านของ $b_1b_2...b_k$ และในบรรทัด (b-5) (b-6) (b-7) เรียก INS_STR ซึ่งในบรรทัด (s-2) จะกำหนดการผ่านของ c_1 d_1 ต่อมาในบรรทัด (b-6) เรียก INS_STR ซึ่งในบรรทัด (s-3) ทำการเก็บ $d_1...d_m$\$ ทับที่ TAIL[old_pos] (บรรทัด (b-1) เก็บ old_pos) ในบรรทัด (b-7) เรียก INS_STR ซึ่งในบรรทัด (s-3) ทำการเก็บ $c_2...c_h$\$ ที่ TAIL[POS] และในการเก็บบันทึกสตริงเดี่ยว (Single String) เหล่านี้ ตั้งค่า BASE[t] เป็น -old_pos หรือ -POS ในบรรทัด (s-2) ผลคือเป็นไปตามเงื่อนไข-A

[ตัวอย่าง-3] ลำดับของการเพิ่มคีย์ bac#, bc#, ba#, bab# แสดงในรูปที่ 2.6 ผลลัพธ์สุดท้ายแสดงไว้ในรูปที่ 2.5 แล้ว และแสดงสมาชิกที่ยังไม่ถูกใช้ของ TAIL ซึ่งถูกเขียนทับในบรรทัด (s-3) ด้วย '?' ต่อไปจะแสดง ลำดับการเพิ่มคีย์ bc#, bab# ร่วมกับหมายเลขบรรทัด

การเพิ่ม bc# จะประมวลผล B_INSERT(3, ε, c#, ac#)

(b-1) old_pos = -BASE[3] = 1

(b-5) BASE[3] = X_CHECK({c, a}) = 1

(b-6) INS_STR(3, ac#, 1)

(s-3) POS = SET_STR(1, c#$) = 5

(b-7) INS_STR(3, c#, 5)

(s-3) POS = SET_STR(5, #$) = 7

การเพิ่ม bab# จะประมวลผล A_INSERT(2, b#)

(a-1) (a-2) t = BASE[2] + b = 4 ,
CHECK[4] = 3 != 0

(a-3) LIST1 = SET_LIST(2) = {c, #}

(a-4) LIST2 = SET_LIST(3) = {a, c}

(a-5) N(LIST1) + 1 = 3 > N(LIST2) = 2

(a-7) MODIFY(2, 3, θ, {a, c})

(m-1) old_base = BASE[3] = 1

(m-2) BASE[3] = X_CHECK({a, c}) = 6

สมาชิก a ของ LIST ถูกประมวลผลดังนี้

(m-3) t = old_base + a = 2
t' = BASE[3] + a = 7

(m-4) CHECK[7] = 3, BASE[7] = 2

(m-5) BASE[2] =2 > 0

(m-7) CHECK[5] = 7, CHECK[6] = 7

(m-8) current = 7 (เปลี่ยนสถานะ 2 ซึ่งเป็นสถานะปัจจุบัน)

(m-9) BASE[2] = 0, CHECK[2] = 0

ส่วนการประมวลผลของสมาชิก c ของ LIST จะละไว้ไม่แสดง MODIFY คืนค่าสถานะ 7 แล้วประมวลผลดังนี้

(a-8) INS_STR(7, d#, 8)

(s-3) POS = SET_STR(8, #$) = 10

4. การลบคีย์

อัลกอริธึมการลบคีย์ Algorithm-3 ได้จากการเปลี่ยน return(TRUE) ในบรรทัด (1-2) (1-3) ของ Algorithm-1 คือเปลี่ยน return(TRUE) ในบรรทัด (1-2) (1-3) เป็นดังบริเวณตัวเอนดังนี้

Algorithm 3. Delete Algorithm

```
begin
        r:= 1; h:= 0
        repeat
            h := h+1; t := BASE[r] + a_h;
(1-1)       if ( t > CHECK[1] ) or (CHECK[t] ≠ r)
            then return(FALSE) else r := t;
        until BASE[r] < 0
(1-2)   if (h = n+1) then
        begin
            BASE[r] := 0 ;      CHECK[r] := 0 ;
            return(TRUE) ;
        end
        else S_TEMP = FETCH_STR(-BASE[r]);
(1-3)   if STR_CMP(a_h+1 ... a_n a_n+1, S_TEMP) = -1
            then
            begin
                BASE[r] := 0 ;  CHECK[r] := 0 ;
                return(TRUE) ;
            end
            else return(FALSE)
end
```

ใน Algorithm-3 เมื่อลบคีย์ x โดย $x \in K$ ออกจากทรัยแถวคู่แล้ว ทรัยแถวคู่ที่ได้นั้นเป็นไปตามเงื่อนไข-A เนื่องจากไม่มีสถานะที่เป็นลบในอาร์เรย์คู่ CHECK[r] = 0 แสดงว่าไม่มีการกำหนดการผ่านของ $g(s_t, a) = s_r$ ที่เข้าสู่สถานะแยก (Separate State) s_r นอกจากนี้ สถานะแยก s_rที่ตรงกับคีย์ x มีเพียงหนึ่งเดียว จึงเป็นไปตามเงื่อนไข-A