บทที่ 9

การเทียบสายอักขระ (String Matching)

การค้นหาคำ หรือสายอักขระ (String) ในเอกสารใช้อยู่เป็นประจำในโปรแกรมจัดพิมพ์งานเอกสาร (Word processing) และในงานเปรียบเทียบ

การเปรียบเทียบเพื่อที่จะค้นหาว่ามีคำหรือสายอักขระ P อยู่ในข้อความ T หรือไม่นั้น มีข้อตกลงดังนี้

1. ให้ข้อความ T เป็นอาเรย์ T[0.. n-1] ที่มีความยาวเป็น n ส่วนคำหรือสายอักขระที่ต้องการเปรียบเทียบคือ P [0.. m-1] ที่มีความยาวเป็น m

2. กำหนดให้สมาชิกแต่ละตัวใน T และ P เป็นตัวอักขระ ที่ได้มาจากเซ็ตจำกัดของตัวอักขระ Σ

เช่น $\Sigma = \{0, 1\}$ หรือ

$\Sigma = \{a, b, ...z\}$ เป็นต้น

3. เรียกอาเรย์ของตัวอักขระใน T กับ P ว่า สายอักขระ

เรากล่าวว่าสายอักขระ P ปรากฏเป็นส่วนหนึ่งของ T โดยเริ่มต้นที่ตำแหน่ง s ถ้า

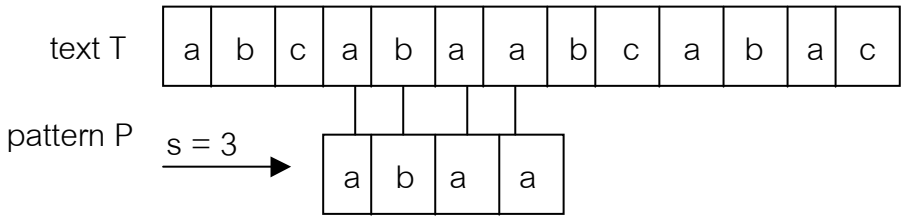
$$T [s .. s + m-1] = P [0.. m-1]$$

เมื่อ $0 = \leq s \leq n - m$

นั่นคือ ถ้าพิจารณาทีละตัวอักขระแล้วจะได้ว่า $T [s+j] = P[j]$

สำหรับ $i \leq j \leq m$

รูปที่ 9.1 ต่อไปนี้แสดงให้เห็นว่า P เป็นส่วนหนึ่งของข้อความ T ที่เลื่อนไป (shifts) = 3



รูปที่ 9.1 แสดงการเทียบสายอักขระ P กับ สายอักขระ T

จากรูปที่ 9.1 จะเห็นว่าสายอักขระ P = abaa ปรากฏเป็นส่วนย่อยของสายอักขระ T = abcabaabcabac ที่โดยเริ่มต้นที่ตำแหน่งที่ 3

อัลกอริทึมในการเทียบสายอักขระมีหลายวิธี แต่ในที่นี้จะกล่าวถึงเพียง 3 วิธี

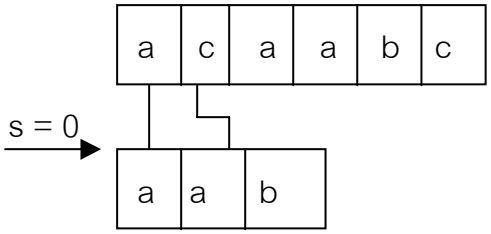
9.1 อัลกอริทึมการเทียบสายอักขระแบบง่าย (Naïve String Matching Algorithm)

อัลกอริทึมต่อไปนี้ จะทำการหาค่า S ที่ทำให้

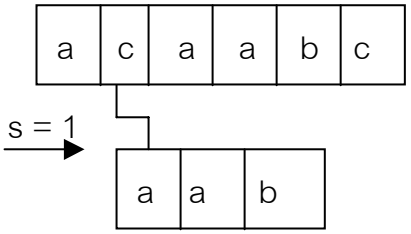
T [s..s + m-1] = P [0..m-1] สำหรับทุกค่าที่เป็นไปได้ของ S นั่นคือ ตำแหน่งที่ $\leq$ n - m

```
NAIVE - STRING - MATCHER (T, P)
// ข้อมูลนำเข้า คือ สายอักขระ T และ P ที่มีความยาว n และ m ตามลำดับ
// ข้อมูลส่งออก คือ ตำแหน่งเริ่มต้นที่พบสายอักขระ P ในสายอักขระ T
1.  n       =       length [T]
2.  m       =       length [P]
3.  for s   =       0  to  n  -  m
4.      do if P [0..m-1] = T [s . . s + m-1]
5.          then print " Pattern P occurs with shift" s    // ตำแหน่งเริ่มต้นที่พบ P คือ s
```

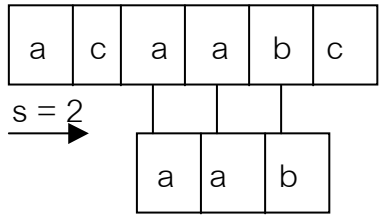
ตัวอย่างที่ 9.1 รูปที่ 9.2 ต่อไปนี้ แสดงการเปรียบเทียบสายอักขระโดยใช้อัลกอริทึม NAIVE - STRING - MATCHER เมื่อ s มีค่าเป็น 0, 1, 2, ...ตามลำดับ



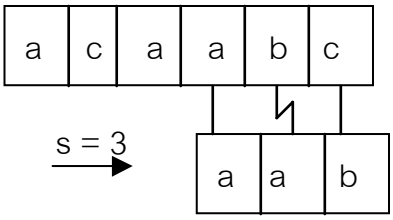
(a)



(b)



(c)



(d)

รูปที่ 9.2 แสดงการเทียบสายอักขระโดยใช้อัลกอริทึม Naïve

จากรูปที่ 9.2 จะเห็นว่าพบสายอักขระ P ในข้อความ T โดยเริ่มต้นที่ตำแหน่งที่ 2

9.2 การเทียบสายอักขระโดยใช้ออโตมาตาสถานะจำกัด

นิยาม 9.1

ออโตมาตาสถานะจำกัด $M = (Q, q_o, A, \Sigma, \delta)$ เมื่อ

1. Q เป็นเซ็ตจำกัดของสถานะ (State)
2. $q_o \in Q$ เป็นสถานะเริ่มต้น (start state)
3. $A \subseteq Q$ เป็นเซ็ตจำกัดของสถานะสุดท้าย
4. Σ คือ เซ็ตจำกัดของข้อมูลนำเข้า
5. δ เป็นฟังก์ชั่นการเปลี่ยนสถานะของออโตมาตา M เมื่อ

$$\delta : Q \times \Sigma \rightarrow Q$$

การทำงานของออโตมาตาสถานะจำกัด

ออโตมาตา M จะเริ่มต้นที่สถานะ q_o ทำการอ่านข้อมูลนำเข้าทีละตัว ซึ่งถ้า M อยู่ที่สถานะ q และอ่านข้อมูลนำเข้า a แล้ว M จะเปลี่ยนสถานะไปอยู่ที่สถานะ δ (q, a) เราจะกล่าวว่าออโตมาตา M จะยอมรับสายอักขระที่อ่านเข้ามาถ้าหลังจากอ่านตัวอักขระตัวสุดท้ายแล้ว M เปลี่ยนสถานะไปอยู่ที่สถานะ $q_f \in A$

ตัวอย่างที่ 9.2 รูปที่ 9.3 แสดงถึงออโตมาตาสถานะจำกัดที่มีองค์ประกอบต่าง ๆ ดังนี้

1. Q = {0, 1}
2. q_o = 0 (ออโตมาตาเริ่มต้นที่สถานะ 0)
3. A = { 1 }
4. Σ = {a, b}
5. ฟังก์ชันเปลี่ยนสถานะ δ กำหนดดังตารางต่อไปนี้

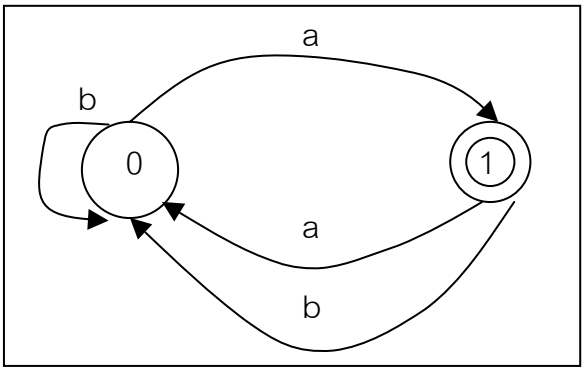
	input	
state	a	b
0	1	0
1	0	1

หรือได้ว่า $\delta(0, a) = 1$

$\delta(0, b) = 0$

$\delta(1, a) = 0$

$\delta(1, b) = 1$



รูปที่ 9.3 ออโตมาตาสถานะจำกัด

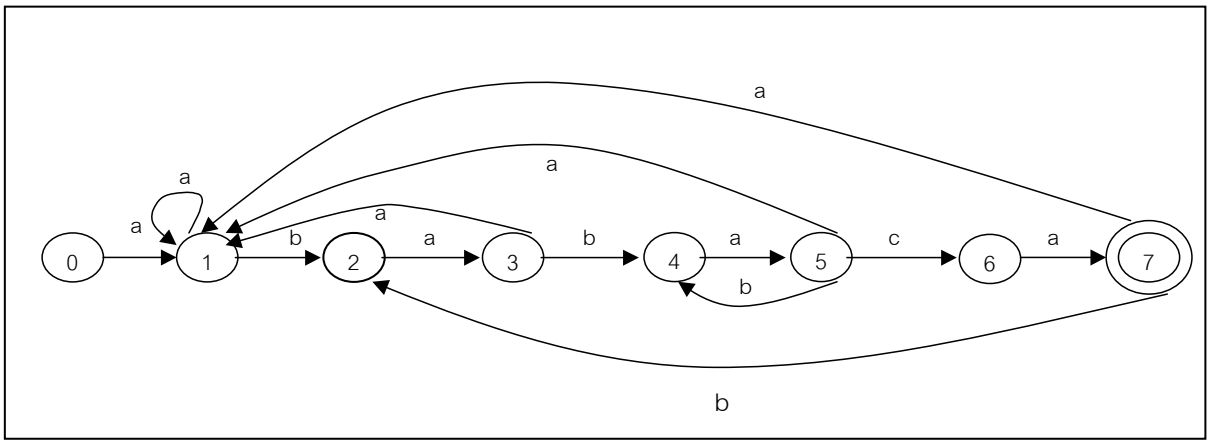
หมายเหตุ

โหนดที่มีวงกลม 2 วง ซ้อนกัน หมายถึงสถานะสิ้นสุด และถ้าเมื่อใดก็ตามที่ออโตมาตาอ่านตัวอักขระเข้ามาแล้วเปลี่ยนสถานะไปอยู่ที่สถานะสิ้นสุด เราจะกล่าวว่า ออโตมาตายอมรับสายอักขระที่อ่านมาแล้วทั้งหมด แต่ถ้าไม่ใช่แสดงว่าออโตมาตาไม่ยอมรับสายอักขระนั้น

เมื่อพิจารณาการทำงานของออโตมาตาในรูปที่ 9.3 จะพบว่าออโตมาตานี้จะยอมรับสายอักขระที่ประกอบด้วยตัวอักขระ 'a' เป็นจำนวนคี่

การเปรียบเทียบสายอักขระโดยใช้ออโตมาตานั้นจะต้องทำการสร้างออโตมาตาสถานะจำกัด สำหรับสายอักขระ P ก่อนที่จะเริ่มต้นค้นหาว่ามีสายอักขระ P อยู่ที่ T หรือไม่

ตัวอย่างที่ 9.3 ออโตมาตาสถานะจำกัดในรูปที่ 9.4 ต่อไปนี้ เป็นออโตมาตาที่ยอมรับสายอักขระ P = ababaca



รูปที่ 9.4 ออโตมาตาสถานะจำกัด

ตารางที่ 9.1 เป็นตารางการเปลี่ยนสถานะของออโตมาตาในรูปที่ 9.4

State	Input		
	a	b	c
0	1	0	0
1	1	2	0
2	3	0	0
3	1	4	0
4	5	0	0
5	1	4	6
6	7	0	0
7	1	2	0

หลักการในการสร้างออโตมาตาสถานะจำกัดที่ยอมรับสายอักขระ P[0..m-1] นั้น จะต้องกำหนดให้ Q = {0, 1, ...,m} สถานะเริ่มต้น $q_0 = 0$ และ สถานะสิ้นสุด คือ m

หลังจากการสร้างออโตมาตาสถานะจำกัดที่ยอมรับสายอักขระ P แล้ว ให้ทำกรเทียบสายอักขระ P ใน สายอักขระ T โดยใช้อัลกอริทึม FINITE-AUTOMATA-MATCHER ดังนี้

```
FINITE - AUTOMATA - MATCHER (T, δ, m)
// ข้อมูลนำเข้า คือ สายอักขระ T ที่มีความยาวเป็น n และฟังก์ชันเปลี่ยนสถานะ δ ของสายอักขระ P
//                ที่มีความยาว เป็น m
// ข้อมูลส่งออก คือ ตำแหน่งเริ่มต้นในสายอักขระ T ที่พบสายอักขระ P
   1. n  =  length [T]
   2. q  =  0
   3. for i = 0  to  n - 1
   4.     do  q  =  δ (q, T [i])
   5.         if  q  =  m
   6.            then s =  i - m
   7.                 print "pattern occurs with shift" s
```

9.3 การเทียบสายอักขระโดยใช้ Knuth-Morris-Pratt Algorithm (KMP)

การเทียบสายอักขระโดยใช้อัลกอริทึม **NAIVE - STRING – MATCHER** นั้น พบว่าเวลาสูงสุดที่ต้องใช้ในการเปรียบเทียบ คือ O(mn) ทั้งนี้เพราะว่า ในการเปรียบเทียบแต่ละครั้งเมื่อพบว่าตัวอักขระในสายอักขระกับข้อความไม่ตรงกันเราจะเลื่อนสายอักขระไปทางขวาเพียง 1 ตำแหน่งเสมอ ไม่ว่าตัวอักขระที่ไม่ตรงกันนั้นจะอยู่ที่ตำแหน่งใดของสายอักขระ แล้วดำเนินการเปรียบเทียบตัวอักขระในรอบต่อไปทีละตัวตามลำดับไปเรื่อย ๆ จนกว่าจะพบ หรือ จนกว่าจะหมดข้อความ ซึ่งการดำเนินการเช่นนี้ทำให้การดำเนินงานไม่มีประสิทธิภาพเท่าที่ควร เพื่อให้สามารถเปรียบเทียบสายอักขระมีจำนวนครั้งไม่เกิน O(m + n) จำเป็นต้องมีการเก็บข้อมูลที่ได้จากการเปรียบเทียบในรอบก่อนหน้ามาใช้ช่วยในการเลื่อนสายอักขระไปที่ตำแหน่งที่เหมาะสม ซึ่งอัลกอริทึมนั้น คือ อัลกอริทึม Knuth-Morris-Pratt ที่มีการคำนวณฟังก์ชัน KMP Failure Function ดังนี้

Failure Function

แนวคิดหลักของอัลกอริทึม KMP คือการประมวลผล (preprocess) สายอักขระ P ก่อนเพื่อคำนวณฟังก์ชัน Failure f ที่สามารถกำหนดระยะทางในการเคลื่อนสายอักขระ P ไปยังตำแหน่งที่เหมาะสมเพื่อหลีกเลี่ยงการเปรียบเทียบตัวอักขระคู่เดิมซ้ำกันหลายครั้ง โดยกำหนดให้ f(j) คือ ค่า

ความยาวที่มากที่สุดของพรีฟิกส์ (prefix) ของสายอักขระ P หรือตำแหน่งเริ่มต้นของซัฟฟิกส์ของ P[1..j] (จงสังเกตว่าในที่นี้ไม่ใช้ P[0..j-1]) นอกจากนี้ยังกำหนดให้ f(0) = 0 อัลกอริทึมฟังก์ชัน failure เป็นดังนี้

```
KMPFailureFunction(P)
        ข้อมูลนำเข้า      : สายอักขระ P[0..m – 1] ซึ่งมีความยาวเป็น m
        ข้อมูลส่งออก      : failure function f สำหรับสายอักขระ P เมื่อกำหนดให้ j
                            คือ ความยาวที่มากที่สุดของพรีฟิกส์ของ P หรือตำแหน่งเริ่มต้นของ P
```

```
i       =  1;
j       =  0;
f(0)    =  0;
while i <= m –1 do
        if  ( P[j] = P[i])    then
    // แสดงว่าพบสายอักขระตรงกัน j + 1 คู่แล้ว ทำการเลื่อน i, j ไปเปรียบเทียบใน
    // ตำแหน่งถัดไป
                f ( i )   = j + 1;
                i         = i + 1;
                j         = j + 1;
    // กรณีที่ไม่พบตัวอักขระที่ตรงกัน แยกพิจารณา เป็น 2 กรณี คือ เมื่อ j > 0 กับ j = 0
        else if   j > 0     then
    //  j เป็นตำแหน่งดัชนีที่ถัดจากพรีฟิกส์ของ P ซึ่งสายอักขระตรงกัน
                j         = f( j – 1 )
         else
                f(i)      =   0;
                i         =    i + 1;
```

ตัวอย่างที่ 9.4 กำหนด T และ P ดังนี้

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
T	a	b	a	c	a	a	b	a	c	c	a	b	a	c	a	b	a	a	b	b

P	a	b	a	c	a	b

Failure function f สำหรับสายอักขระ P เป็นดังนี้

j	0	1	2	3	4	5
P(j)	a	b	a	c	a	b
f(j)	0	0	1	0	1	2

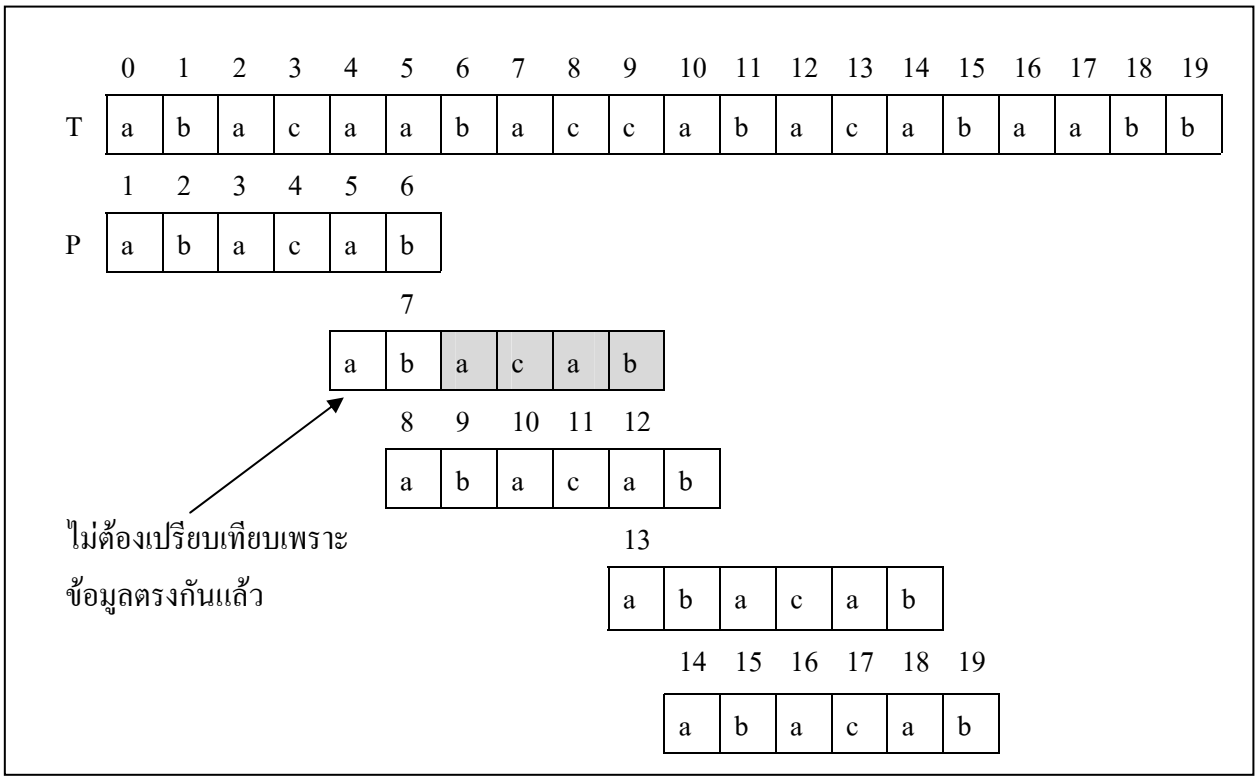
อัลกอริทึม KMPMatcher

อัลกอริทึม KMPMatcher ต่อไปนี้ เป็นอัลกอริทึมที่ทำการเทียบสายอักขระ T กับ สายอักขระ P โดยใช้ failure function f ที่สร้างขึ้นด้วยอัลกอริทึม KMPFailureFunction()

```
Algorithm  KMPFailureFunction(P):
// ข้อมูลนำเข้า   : สายอักขระ T ที่มีความยาวเท่ากับ n และสายอักขระ P ที่มีความยาวเป็น m
// ข้อมูลส่งออก : ตำแหน่งดัชนีเริ่มต้นในสายอักขระ T ที่พบสายอักขระ P ใน T หรือ รายงานว่าสาย
                  อักขระ P ไม่เป็นสายอักขระย่อยของ T

        f  =  KMPFailureFunction(P);   // สร้างฟังก์ชัน failure
        i   = 0;
        j  =  0;
        while i < n do
                if  P[j]  = T[i]  then
                        if  j = m – 1 then
                                return  i – m +1     // พบสายอักขระ P ใน T โดยเริ่มต้นที่ตำแหน่งนี้
                        i = i +1;
                        j = j + 1
                else    if    j > 0  then             // ตัวอักขระใน P และ T ไม่ตรงกัน เคลื่อน P ไป
                                                      // ทางขวา
                        j  =  f(j – 1)                // j เป็นตำแหน่งดัชนีที่ถัดจากพรีฟิกส์ของ P ซึ่งสายอักขระ
                                                      // ตรงกัน
                else
                        i = i + 1
        return                                        // สายอักขระ P ไม่เป็นสายอักขระย่อยของ T
```

ตัวอย่างที่ 9.5 เมื่อใช้ Failure function จากตัวอย่างที่ 9.4 ช่วยทำการเปรียบเทียบ ข้อความ T กับ P จะได้ดังรูปที่ 9.5



รูปที่ 9.5 แสดงการเปรียบเทียบสายอักขระ T และ P ด้วย KMPMatch

จากรูปที่ 9.5 มีจำนวนครั้งในการเปรียบเทียบเพียง 19 ครั้ง โดยในครั้งที่ 6 เมื่อพบว่าตัวอักขระจากทั้งสองสายอักขระไม่ตรงกัน จึงกำหนดค่า j ใหม่ให้เท่ากับ f(j-1) = f(5-1) = f(4) = 1 (ค่า f ดูจากตัวอย่างที่ 9.4 นั่นคือเป็นการกำหนดให้ j = 1) ในครั้งที่ 7 ทำการเปรียบ P[1] กับ T[5] (โดยไม่ต้องเริ่มต้นเทียบใหม่จาก P[0]) พบว่าไม่เท่ากัน จึงกำหนดค่า j ใหม่ให้เท่ากับ f(j-1) = f(1-1) = f(0) = 0 ในครั้งที่ 8 ทำการเปรียบ P[0] กับ T[5] ซึ่งพบว่าเท่ากัน จึงเพิ่มค่า i และ j ขึ้นอีก 1 แล้วเปรียบเทียบครั้งที่ 9 ดำเนินการตามอัลกอริทึมไปเรื่อย ๆ จะพบว่า สายอักขระ P เป็นสายอักขระย่อยของ T โดยมีตำแหน่งเริ่มต้นที่ j = 10

ประสิทธิภาพของอัลกอริทึม KMP

ถ้าไม่รวมการทำงานของการคำนวณฟังก์ชัน Failure ประสิทธิภาพของอัลกอริทึม **KMPmatch** ขึ้นอยู่จำนวนครั้งของการวนลูป while ซึ่งอยู่ในรูป O(n) ในขณะที่ ประสิทธิภาพของ KMPFailureFunction คือ O(m) ดังนั้น ประสิทธิภาพของอัลกอริทึม KMP คือ O(m + n)

แบบฝึกหัด

1. จงแสดงการค้นหาสายอักขระ P = 0001 ใน T = 000010001010001 โดยใช้อัลกอริทึม NAIVE-STRING-MATCHER

2. จงสร้างออโตมาตาสถานะจำกัดเพื่อค้นหา P = aabab ใน T = aaababaabaababaab

3. จงสร้างออโตมาตาสถานะจำกัดที่ยอมรับสายอักขระ ababbabbabbababbabb เมื่อ Σ = {a, b}

4. จงคำนวณฟังก์ชัน failure สำหรับ สายอักขระ P = cgtacgttcgtac

5. จงใช้อัลกอริทึม KMP ในการค้นหาสายอักขระ P = bacbaaa ใน T = bacbacabcbacbbbacabacbabcbbba