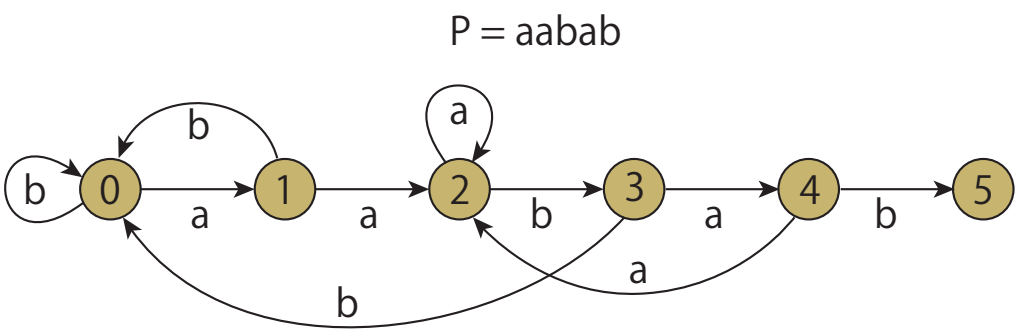
計算機科学
– String Matching 文字列照合 –

Yoshitsugu Yamamoto
山本 芳嗣
3F1007 (029-853-5001), 3E410 (029-853-5395)
yamamoto@sk.tsukuba.ac.jp

revised 2011-11-17, 2012-02-29+08-22



1　定義

△　Σ：有限アルファベット、$\Sigma = \{0,1\}, \{0,1,2,\ldots,9\}, \{a,b,c,\ldots,z\}$

△　Σ^*：アルファベットの有限長さの列の全体

△　$T = T[1..n] \in \Sigma^*$：テキスト

△　$P = P[1..m] \in \Sigma^*$：パターン、通常 $m << n$

△　P は T にシフト s でマッチしている　$\leftrightharpoons$　$T[s+1..s+m] = P[1..m]$

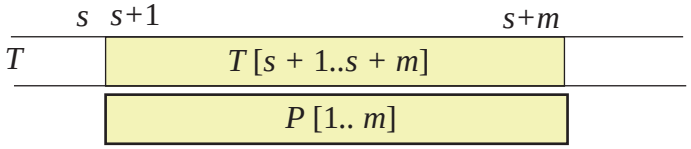


図 1: シフト s でマッチ

2　Naive Algorithm　素朴な算法

2.1　Naive-String-Matcher

シフト s をゼロから順番に増やして $T[s+1..s+m] = P[1..m]$ をチェック。

Naive-String-Matcher

```
1:  n:=length[T]
2:  m:=length[P]
3:  for s:=0 to n-m
4:      do if P[1..m]=T[s+1..s+m] then print "Pattern occurs with shift" s
```

表 1: Naive-String-Matcher

s	a	c	a	a	b	c
0	a	a	b			
1		a	a	b		
2			a	a	b	
3				a	a	b

2.2　計算複雑度

Naive-String-Matcher の計算複雑度は $\Theta((n-m+1)m)$。例えば、$T = \mathtt{aaa}\ldots\mathtt{aa}, P = \mathtt{aa}\ldots\mathtt{a}$ で、考えるべきシフト s は $0,1,\ldots,n-m$、各シフトで 4 行目の照合を行うのに m 個のアルファベットの照合が必要。

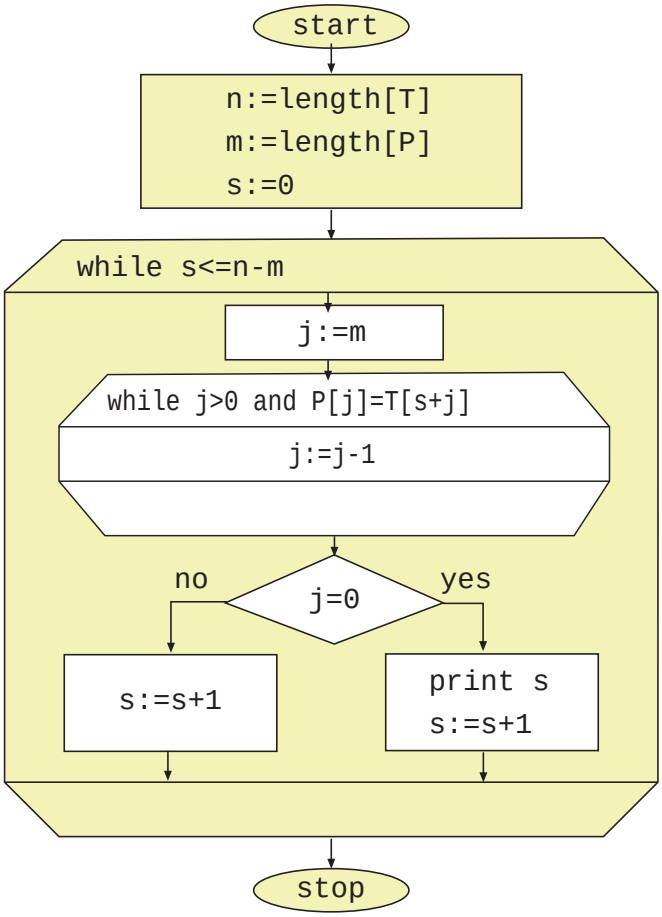


図 2: Flowchart of Naive-String-Matcher

2.3 演習問題

1. $P = \texttt{0001}, T = \texttt{000010001010001}$ に対して Naive-String-Matcher を動かせ。
2. Naive-String-Matcher が最初のパターンを見つける最悪の計算時間が $\Theta((n-m+1)(m-1))$ となることを示せ。

3 Rabin-Karp Algorithm　ラビンーカープの算法

3.1 基本的アイディア

ラビンーカープの算法はアルファベット Σ が $\{0,1,2,\ldots,9\}$ の場合に使える方法である。文字列 $P[1..m]$ を 10 進 m 桁の数値と見なして、その値を p と書き、文字列 $T[s+1..s+m]$ を 10 進 m 桁の数値と見なして、その値を t_s と書く。つまり、

$$p = P[m] + 10^1 P[m-1] + \cdots + 10^{m-2} P[2] + 10^{m-1} P[1]$$

である。そうすると、

$$P[1..m] = T[s+1..s+m] \quad \Leftrightarrow \quad p = t_s \tag{3.1}$$

従って Naive-String-Matcher の 4 行目の照合を $p = t_s$ のチェックで置き換えることができる。この $p = t_s$ の判定は m 個の文字の照合よりも短時間で実行できる。

3.2 $p, t_0, t_1, \ldots, t_{n-m}$ の計算

p と t_0 は Honer 法によって

$$p = P[m] + 10(P[m-1] + 10(P[m-2] + \cdots + 10(P[2] + 10P[1]) \cdots) \tag{3.2}$$

$$t_0 = T[m] + 10(T[m-1] + 10(T[m-2] + \cdots + 10(T[2] + 10T[1]) \cdots) \tag{3.3}$$

計算時間はいずれも $O(m)$。

t_s が得られているとき t_{s+1} は次の漸化式で計算できる。

$$t_{s+1} = 10(t_s - 10^{m-1} T[s+1]) + T[s+m+1] \tag{3.4}$$

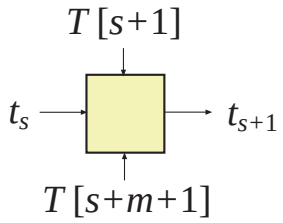


図 3: Compute t_{s+1}

これは

$$t_s = 10^0 T[s+m] + \cdots + 10^{m-2} T[s+2] + 10^{m-1} T[s+1]$$

$$t_{s+1} = T[s+1+m] + 10^1 T[s+m] + \cdots + 10^{m-1} T[s+2]$$

から容易に分かる。10^{m-1} をあらかじめ計算して変数 h に格納しておけば、上の漸化式による計算は m, n に依存しない一定時間。従って、$p, t_0, \ldots, t_{n-m}$ すべての計算時間は $O(n+m)$。

例：$T[s+1..s+m] = 31415, t_s = 31{,}415, T[s+m+1] = 2$ のとき $h = 10^{5-1}$

$$t_{s+1} = 10(31{,}415 - h \times 3) + 2 = 10 \times 1{,}415 + 2 = 14{,}152$$

3.3 計算複雑度

$p, t_0, \ldots, t_{n-m}$ が計算されれば、(3.1) を $n-m+1$ 回繰り返せばよい。(3.1) のチェックは単位時間でできるので、この計算時間は $O(n-m+1)$。従って、全体の計算時間は $O(n+m)$。

3.4 パターンの長さ m が大きいときの問題点と対処法

m が大きいと p, t_s が大きな数値となり、(3.1) の $p = t_s$ の比較が一定時間でできるとの仮定は妥当でない。そこで、適当な大きさの自然数 q を決めて p, t_s の計算を q を法として (mod q) 行う。このとき (3.1) は

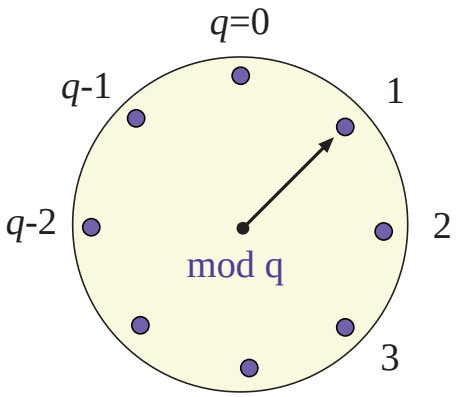


図 4: mod q の計算は時計計算

成り立たないが、

$$P[1..m] = T[s+1..s+m] \quad \Rightarrow \quad p = t_s \pmod{q} \tag{3.5}$$

は成立。つまり、

$$p \neq t_s \pmod{q} \quad \Rightarrow \quad P[1..m] \neq T[s+1..s+m] \tag{3.6}$$

例：表 2 の例のように mod q で計算したため、2回目のマッチングはミスマッチ。その部分だけ $P[1..m] =$

表 2: Rabin-Karp-Matcher

2	3	5	9	0	2	3	1	4	1	5	2	6	7	3	9	9	2	1

↓ mod 13

8	9	3	11	0	1	7	8	4	5	10	11	7	9	11

7	→

$P =$ 31415

$T[s+1..s+m]$ をチェックする。

Rabin-Karp-Matcher

```
1: n:=length[T]
2: m:=length[P]
3: h:=d^(m-1) mod q
4: p:=0
```

```
5:  t_0:=0
6:  for i:=1 to m
7:      do  p:=(dp+P[i]) mod q
8:          t_0:=(dt_0+T[i]) mod q
9:  for s:=0 to n-m
10:     do  if p=t_s
11:         then if P[1..m]=T[s+1..s+m]
12:             then "Pattern occurs with shift" s
13:         if s<n-m
14:             then t_{s+1}:=d(t_s-T[s+1]h)+T[s+m+1] mod q
```

3.5 計算複雑度

Naive-String-Matcher に同じ。ただし、平均的には速い。

3.6 演習問題

1. $q = 11, P = 26, T = 3141592653589793$ としたとき、正しくないマッチが何回起こるか。
2. $n \times n$ の配列のテキスト中に、$m \times m$ のパターンを探す問題に Rabin-Karp-Matcher を拡張せよ。ただし、パターンの回転は考えなくても良い。

4 The Boyer-Moore Algorithm ボイヤーームーアの算法

4.1 Naive-String-Matcher の変種

まずは Naive-String-Matcher に少し手を加えた次の算法をみよう。3 行目が抜けている理由は後で分かる。

```
    Naive-String-Matcher with from-right-to-left comparison
1:  n:=length[T]
2:  m:=length[P]
4:  s:=0
5:  while s<=n-m
6:      do j:=m
7:          while j>0 and P[j]=T[s+j]
8:              do j:=j-1
9:          if j=0
10:             then    print "Pattern occurs at shift" s
11:                     s:=s+1
12:             else    s:=s+1
```

これは Naive-String-Matcher の $P[1..m] = T[s+1..s+m]$ の判定を、お尻から行っているだけ。

4.2 演習問題

1. パターンのすべてのアルファベットが異なっていることが分かっているとき、Naive-String-Matcher の変種を $O(n)$ 時間で動くようにせよ。（ヒント：$P[1..m] = T[s+1..s+m]$ の判定を後ろから行え。もしも $P[m] \neq T[s+m]$ なら、シフトを 1 つ進める。$P[m] = T[s+m]$ なら $P[m-1] = T[s+m-1]$ の判定を行う。この判定の過程でもしも $P[1..m] \neq T[s+1..s+m]$ が分かったら、どこまでシフトを増加させることができるかを考えよ。）

2. パターンのすべてのアルファベットが同じであることが分かっている場合はどうか。

3. パターンのアルファベットが昇順（あるいは降順）に並んでいることが分かっている場合はどうか。

4.3 Bad-Character Heuristic

下の例ではパターン `reminiscence`[1] を後ろから見ていって、3 番目の `n` でエキストの `i` とマッチしていない。パターンを後ろから見て初めて出くわす `i` がテキストの `i` の位置にくるまでシフトしても、マッチを見逃すことはない。もしも、パターン中に `i` が含まれていないときには、パターンはテキストの `i` の次までシフトできる。

$P[j] \neq T[s+j]$ のとき、

$$k = \begin{cases} \max\{\, \ell \mid T[s+j] = P[\ell] \,\} & T[s+j] \text{ がパターンに含まれるとき} \\ 0 & T[s+j] \text{ がパターンに含まれないとき} \end{cases}$$

[1] Someone's reminiscences are things that they remember from the past, and which they talk or write about. Reminiscence is the process of remembering these things and talking or writing about them. A formal word.

表 3: Bad-Character Heuristic

	$s=2$											$j=10$					
	w	r	i	t	t	e	n	_	n	o	t	i	c	e	_	t	h	a	t
$s\rightarrow$		r	e	m	i	n	i	s	c	e	n	c	e	$k=6$					
	w	r	i	t	t	e	n	_	n	o	t	i	c	e	_	t	h	a	t
$s+4\rightarrow$							r	e	m	i	n	i	s	c	e	n	c	e	

とすると s の更新は

$$s := s + \begin{cases} j & k = 0 \text{ の場合} \\ j-k & k < j \text{ の場合} \\ 1 & k > j \text{ の場合} \end{cases} = s + \max\{1, j-k\}$$

とできる。

注意

1. $k = j$ は定義よりあり得ない。
2. $k > j$ は必ずしも $P[j]$ より左にアルファベット $T[s+j]$ が含まれないことを意味しない。つまり、次に紹介する λ は $P[j]$ の左側のどこに $T[s+j]$ が現れるかを見逃している。これは欠点ではあるが、λ の計算負担を軽くしている。

4.4 演習問題

1. 表 4.3 の次の段階ではどこまでシフトを増加できるか。

4.5 Last-Occurrence Function λ

$a \in \Sigma$ について、パターンの中で a が最後に現れる場所を与える関数 (last-occurrence function)$\lambda : \Sigma \rightarrow \{0, 1, 2, \ldots, m\}$ を以下のように定義する。

$$\lambda[a] = \begin{cases} \max\{\, \ell \mid a = P[\ell] \,\} & a \text{ がパターンに含まれるとき} \\ 0 & a \text{ がパターンに含まれないとき} \end{cases} \tag{4.1}$$

Compute-Last-Occurrence-Function

```
1:  for each character a∈ Σ
2:      do λ[a]:=0
3:  for j:=1 to m
4:      do λ[P[j]]:=j
5:  return λ
```

これを用いてはじめのプログラムの次の 2 行を修正すればよい。

```
3:  Compute the Last-Occurrence Function λ
12:             else    s:=s+max{1, j-λ[T[s+j]]}
```

関数 λ が用意されていれば、s の更新は一定時間。

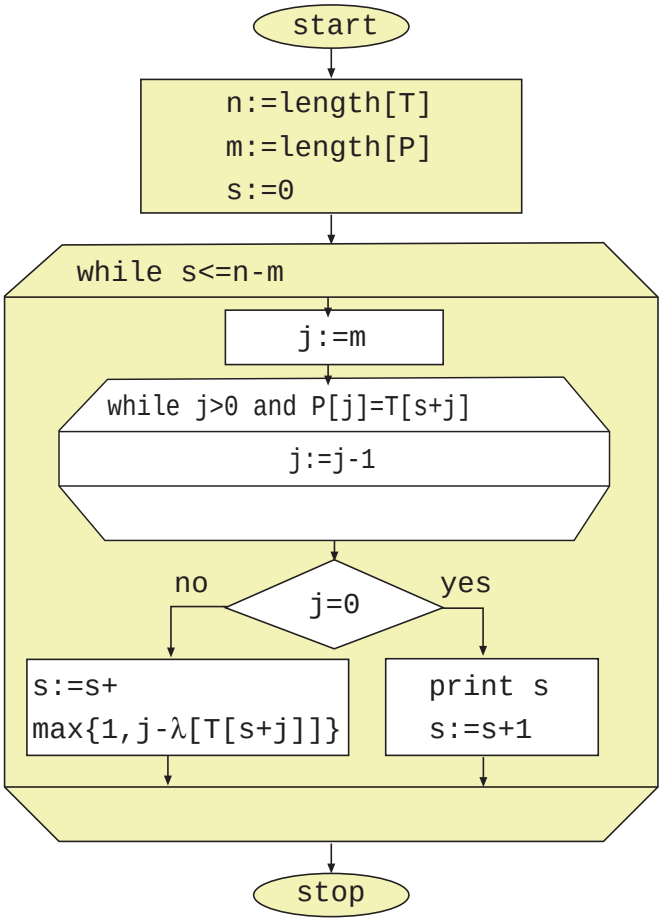


図 5: Flowchart of Boyer-Moore

4.6 Extended Last-Occurrence Function Λ

関数 λ を拡張して関数 $\Lambda : \Sigma \times \{2, 3, \ldots, m\} \to \{0, 1, 2, \ldots, m-1\}$ を

$$\Lambda[a, j] = \begin{cases} \max\{\, \ell \mid a = P[\ell]; \ell < j \,\} & a \text{ が } P[1..j-1] \text{ に含まれるとき} \\ 0 & a \text{ が } P[1..j-1] \text{ に含まれないとき} \end{cases} \tag{4.2}$$

と定義する。これは P の j より前に現れるアルファベット a の最後の位置を与える関数である。表 4 参照。この関数を用いるとシフト s の更新は次のように一定時間でできる。

```
3:   Compute the Extended Last-Occurrence Function Λ
12:              else    s:=s+j-Λ[T[s+j],j]
```

関数 Λ は次のように計算できる。2 重の `for` ループがあるため、m や Σ が大きいときには Λ の計算負担は λ のそれに比べて大きい。

```
     Compute-Extended-Last-Occurrence-Function
1:   for each character a∈ Σ
2:       do Λ[a,1]:=0
3:   for j:=1 to m-1
```

表 4: Extended Last-Occurrence Function Λ for `reminiscence`

	(1)	2	3	4	5	6	7	8	9	10	11	12
a	0	0	0	0	0	0	0	0	0	0	0	0
b	0	0	0	0	0	0	0	0	0	0	0	0
c	0	0	0	0	0	0	0	0	8	8	8	11
⋮												
i	0	0	0	0	4	4	6	6	6	6	6	6
⋮												
r	0	1	1	1	1	1	1	1	1	1	1	1
⋮												
z	0	0	0	0	0	0	0	0	0	0	0	0

```
4:      for each character a∈ Σ
5:          do Λ[a,j+1]:=Λ[a,j]
6:      do Λ[P[j],j+1]:=j
7:  return Λ
```

4.7 演習問題

1. P =`reminiscence` に対して関数 Λ を計算せよ。
2. 関数 Λ を用いた Bad-Character-Heuristic のプログラムを作れ。
3. パターン中に個々のアルファベットが現れる場所を複数個あるいは全部記憶しておくデータ構造の例を 3 種類を挙げよ。
4. そのデータ構造を用いた Bad-Character-Heuristic のプログラムを作り、示したデータ構造の優劣を比較せよ。

5 Knuth-Morris-Pratt Algorithm　クヌースーモリスープラットの算法

5.1 prefix function

Boyer-Moore Algorithm の Bad-Character Heuristic とは逆に、パターンがテキストとマッチしているかをパターンの頭から調べる、つまり j を増やしながら $P[j] = T[s+j]$ を見ることによって判定している場合を考える。そして、パターンの初めの q 文字がテキストとシフト s でマッチしており、つまり $P[1..q] = T[s+1..s+q]$ で、かつ、その次の文字はマッチしなかった、つまり $P[q+1] \neq T[s+q+1]$ であることが分かったとする。図 6 参照。このとき

$$s' = s + \min\{\, r \mid r \geq 1;\ P[1..q-r] = T[(s+1)+r..s+q] \,\} \tag{5.1}$$

$$= s + \min\{\, r \mid r \geq 1;\ P[1..q-r] = P[1+r..q] \,\} \tag{5.2}$$

とする。これはマッチを見逃すことなくパターンの先頭を動かすことのできる新しいシフトである。

例 5.1. $T = \texttt{abcaaababc}$、$P = \texttt{abcab}$ とする。シフト $s = 0$ で先頭から比較すると $P[1..4] = T[1..4]$ と $p[5] \neq T[5]$ が分かる。つまり $q = 4$ である。新しいシフトを s' としたとき、シフトの増分 $s' - s$ を考える。$P[1..4]$ を 1 つずつ右に移動して自分自身と比較すると

$P[1..4]$	a	b	c	a
$P[1..3]$		a	b	c
$P[1..2]$			a	b
$P[1..1]$				a

からシフトの増分として 3 が得られる。

パターン $P[1..m]$ の初めの q 文字からできる文字列 $P[1..q]$ を考え、その先頭の k 文字 $P[1..k]$ を右に、$P[1..k]$ の右端が $P[1..q]$ の右端にそろうまで、ずらせてみる。このとき $P[q-k+1..q] = P[1..k]$ となるとき、つまり $P[1..q]$ がその末尾に $P[1..k]$ を含んでいるとき

$$P[1..q] \succcurlyeq P[1..k]$$

と書くことにしよう。また prefix function $\pi : \{1, \ldots, m\} \to \{0, \ldots, m-1\}$ を

$$\pi[q] = \max\{\, k \mid k < q;\ P[1..q] \succcurlyeq P[1..k] \,\} \tag{5.3}$$

とすれば、上記の s' は

$$s' = s + q - \pi[q]$$

で与えられる。実際、

$$\begin{aligned} s + q - \pi[q] &= s + q - \max\{\, k \mid k < q;\ P[1..q] \succcurlyeq P[1..k] \,\} \\ &= s + \min\{\, q - k \mid k < q;\ P[1..q] \succcurlyeq P[1..k] \,\} \\ &\text{ここで } r = q - k \text{ と置くと} \\ &= s + \min\{\, r \mid r \geq 1;\ P[1..q] \succcurlyeq P[1..q-r] \,\} \\ &= s + \min\{\, r \mid r \geq 1;\ P[1..q-r] = P[1+r..q] \,\} \end{aligned}$$

である。以降では、

$$\Pi[q] = \{\, k \mid k < q;\ P[1..q] \succcurlyeq P[1..k] \,\} \tag{5.4}$$

と書くことする。ただし長さゼロの文字列 $P[1..0]$ はいつも $P[1..q] \succcurlyeq P[1..0]$ を満たしていると仮定して、$0 \in \Pi[q]$ が常に成り立っているものとする。この記号を使うと

$$\pi[q] = \max \Pi[q] \tag{5.5}$$

である。

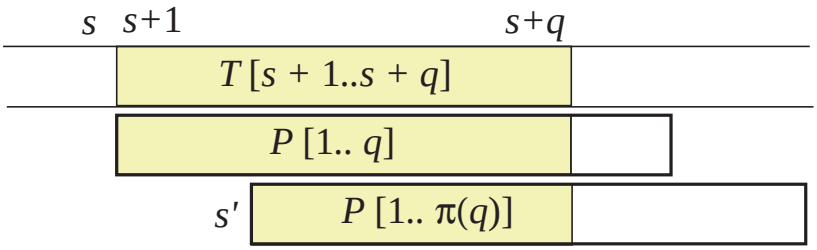


図 6: prefix function π

例 5.2. $P =$ abcaaababc$\cdots$、$q = 10$ として $P[1..q] \succcurlyeq P[1..k]$ なる k を考えると表 5 から分かるように $\pi[q] = 3$ となる。

表 5: $P =$ abcaaababc$\cdots$, $q = 10$ に対する prefix function の値 $\pi[10] = 3$

T	a	b	c	a	a	a	b	a	b	c
$P[1..q]$	a	b	c	a	a	a	b	a	b	c
$P[1..9]$		a	b	c	a	a	a	b	a	b
$P[1..8]$			a	b	c	a	a	a	b	a
$P[1..7]$				a	b	c	a	a	a	b
$P[1..6]$					a	b	c	a	a	a
$P[1..5]$						a	b	c	a	a
$P[1..4]$							a	b	c	a
$P[1..3]$								a	b	c

Knuth-Morris-Pratt Matcher は Naive-String-Matcher のシフトの更新にこの prefix function を用いたものである。下にその pseudo code とフローチャートを示す。

```
    Knuth-Morris-Pratt-Matcher
1:  n:=length[T]
2:  m:=length[P]
3:  Compute the Prefix-Function π
4:  q:=0
5:  for i:=1 to n
6:       do   while q>0 and P[q+1]≠T[i]
```

```
7:                  do q:=π[q]
8:            if P[q+1]=T[i]
9:                  then q:=q+1
10:           if q=m
11:                 then print "Pattern occurs with shift" i-m
12:                      q:=π[q]
```

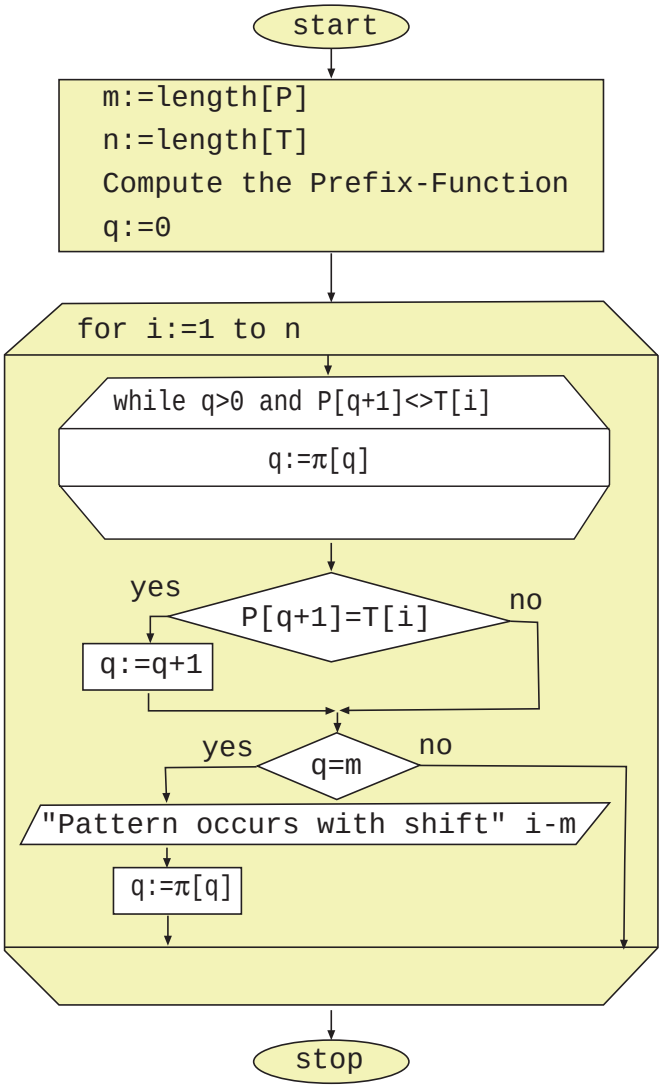


図 7: Knuth-Morris-Pratt-Matcher

例 5.3. $P = \texttt{abca}$ に対する π を定義に従って計算した過程を下に示す。

- $\pi[1] = 0$
- $\pi[2] = \max\{k \mid k < 2;\ p[1..2] \succcurlyeq P[1..k]\} = 0$
  - $P[1..2] = \texttt{ab}$

- $P[1..1] = \texttt{a} \quad \Rightarrow \quad \not\succcurlyeq$

- $\pi[3] = \max\{k \mid k < 3;\ p[1..3] \succcurlyeq P[1..k]\} = 0$
  - $P[1..3] = \texttt{abc}$
  - $P[1..2] = \texttt{ab} \quad \Rightarrow \quad \not\succcurlyeq$
  - $P[1..1] = \texttt{a} \quad \Rightarrow \quad \not\succcurlyeq$

- $\pi[4] = \max\{k \mid k < 4;\ p[1..4] \succcurlyeq P[1..k]\} = 1$
  - $P[1..4] = \texttt{abca}$
  - $P[1..3] = \texttt{abc} \quad \Rightarrow \quad \not\succcurlyeq$
  - $P[1..2] = \texttt{ab} \quad \Rightarrow \quad \not\succcurlyeq$
  - $P[1..1] = \texttt{a} \quad \Rightarrow \quad \succcurlyeq$

次にこの π を用いて $T = \texttt{abcaaa}$ としたときの実行結果は以下のようになる。

表 6: $P = \texttt{abca}$ と $T = \texttt{abcaaa}$ に対する実行結果

i	$P[q+1]$		$T[i]$	q
				0
1	$P[1]$	$=$	$T[1]$	1
2	$P[2]$	$=$	$T[2]$	2
3	$P[3]$	$=$	$T[3]$	3
4	$P[4]$	$=$	$T[4]$	$4 = m$
				$\pi[q] = \pi[4] = 1$
5	$P[2]$	$\neq$	$T[5]$	

さて、残された課題は重要な prefix funciton π の計算である。それは以下の Compute-Prefix-Function(P) で可能で、いくつかの補題を示してその妥当性を証明していくが、その前に直感的な説明を与えよう。

今、$\pi[1], \ldots, \pi[q-1]$ が既に計算されており、次に $\pi[q]$ を計算する段階に来ているとする。もしも、$P[q] = P[\pi[q-1]+1]$ ならば、P の頭から $\pi[q-1]+1$ 文字 $P[1..\pi[q-1]+1]$ が $P[1..q]$ の末尾に等しいことになり、$\pi[q-1]$ の最大性から、$\pi[q] = \pi[q-1]+1$ が分かる。次に、$P[q] \neq P[\pi[q-1]+1]$ の場合を考えよう。P の頭から切り出す文字数を減らして、例えば $\ell+1$ 文字 $P[1..\ell+1]$ が $P[1..q]$ の末尾に等しくなったとしよう。図から分かるようにこのときは $P[1..\ell]$ は $P[1..\pi[q-1]]$ の末尾に等しい。このような ℓ で最大のものは定義から $\pi[\pi[q-1]]$ である。よってこのときは $\pi[q] = \pi[\pi[q-1]]+1$ が得られる。以下同様にして、π の冪（べき）乗によって $\pi[q]$ が計算できることが見て取れる。なお、$\pi[\pi[q-1]]$ は π の合成関数であり、$\pi[q-1] \times \pi[q-1]$ とは別物なので、混同しないように。

```
    Compute-Prefix-Function(P)
1:  m:=length[P]
2:  π[1]:=0
3:  k:=0
4:  for q:=2 to m
5:       do   while k>0 and P[k+1]≠P[q]
```

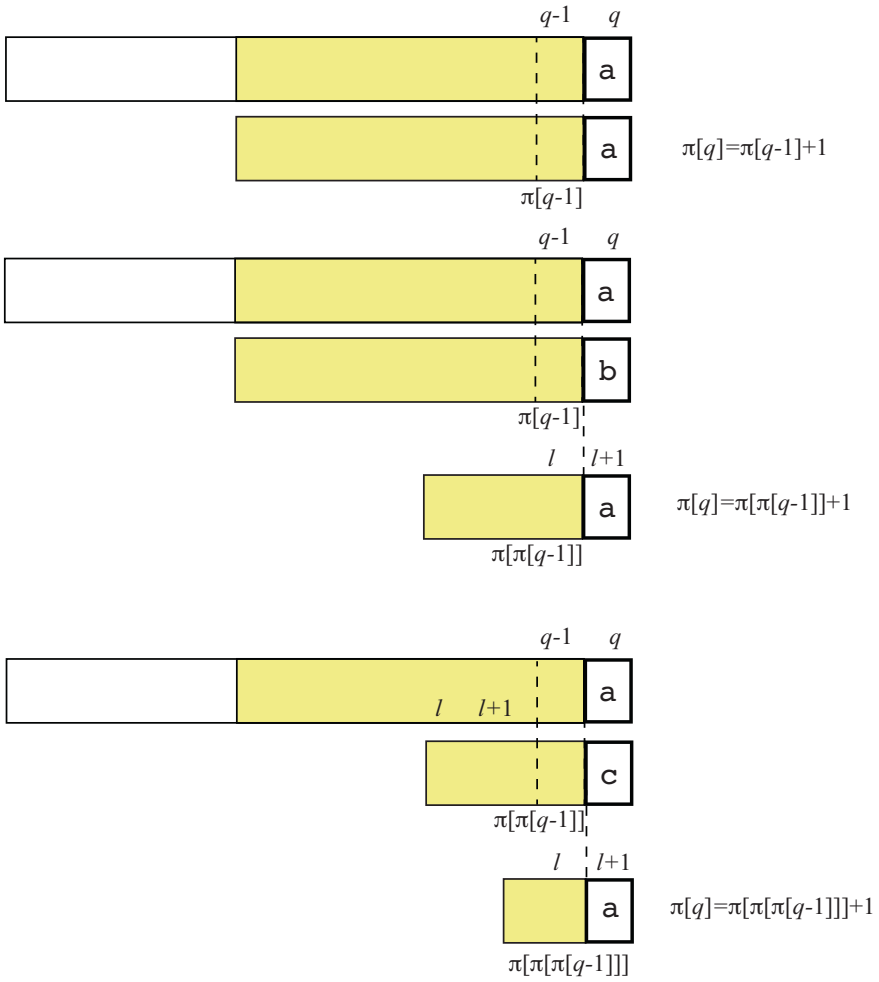


図 8: π の冪乗が現れる理由

```
6:                  do k:=π[k]
7:              if P[k+1]=P[q]
8:                  then k:=k+1
9:              π[q]:=k
10:  return π
```

例 5.4. $P =$ `acaacab` について prefix function π を上のアルゴリズムに従って構成すると表 7 となる。

このアルゴリズムの `while` ループの中で $k := \pi[k]$ が繰り返されている。つまり、このループが繰り返されると、まず $k := \pi[k]$ となり、その k に対して再び同じ演算が繰り返されるので、初めの k に対して $\pi[\pi[k]]$ と π を 2 回施した値が k に代入される。この $\pi[\pi[k]]$ を $\pi^2[k]$ と書くことにしよう。一般的には以下のように π の冪乗を定義する。

定義 5.5. prefix function π に対して

$$\pi^1[q] = \pi[q] \tag{5.6}$$

$$\pi^i[q] = \pi[\pi^{i-1}[q]] \quad i \geq 2 \text{ に対して} \tag{5.7}$$

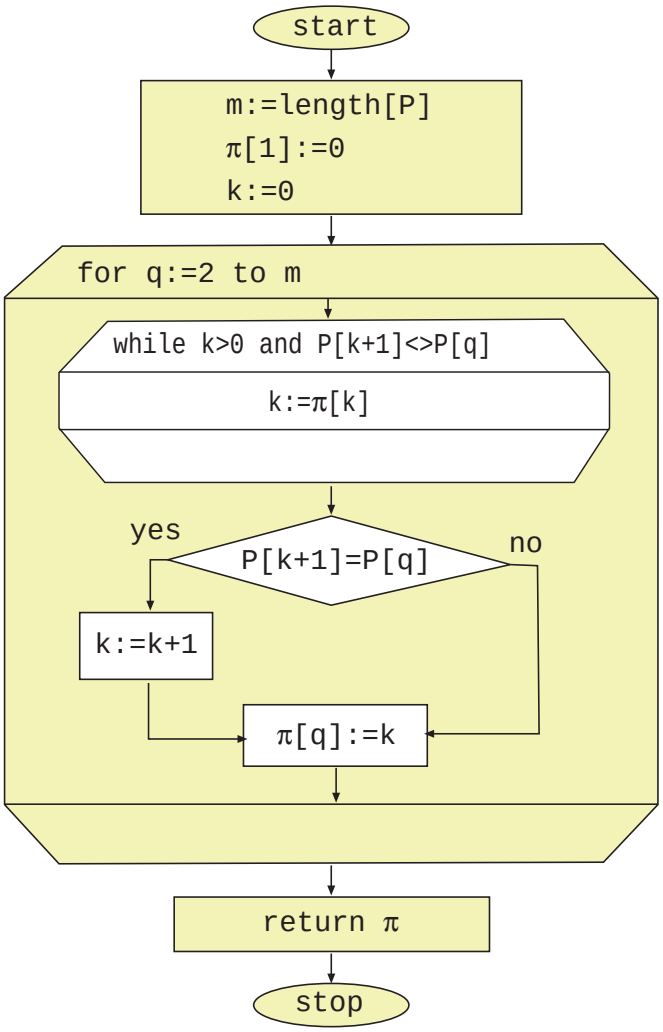


図 9: Compute-Prefix-Function

表 7: $P = \texttt{acaacab}$ について prefix function π の構成

q	k		π
			$\pi[1] = 0$
2	0	$P[1] \neq P[2]$	$\pi[2] = 0$
3	1	$P[1] = P[3]$	$\pi[3] = 1$
4	0	$P[2] \neq P[4]$	
	1	$P[1] = P[4]$	$\pi[4] = 1$
5	2	$P[2] = P[5]$	$\pi[5] = 2$
6	3	$P[3] = P[6]$	$\pi[6] = 3$
7	1	$P[4] \neq P[7]$	
	0	$P[2] \neq P[7]$	
		$P[1] \neq P[7]$	$\pi[7] = 0$

さらに

$$\pi^*[q] = \{\pi^1[q], \pi^2[q], \ldots, \pi^t[q]\} \tag{5.8}$$

と定義する。ただし t は初めて $\pi^t[q] = 0$ となる自然数である。

$\pi[q]$ の定義 (5.3) より $\pi[q] < q$、同様に $\pi^2[q] = \pi[\pi[q]] < \pi[q]$ となり、指数 i の増加に伴って $\pi^i[q]$ は狭義に減少する。従って $\pi^t[q] = 0$ となる自然数が存在する。この prefix function π の冪 $\pi^*[q]$ と式 (5.4) の $\Pi[q]$ の間に次の重要な性質が成り立つ。

補題 5.6. $\forall q = 1, 2, \ldots, m$ について

$$\pi^*[q] = \Pi[q] \tag{5.9}$$

が成り立つ。

Proof. $\pi^*[q] \subseteq \Pi[q]$ の証明：
i を $\pi^*[q]$ の任意の元とすると、$u \in \{1, \ldots, t\}$ が存在して $i = \pi^u[q]$ となっているので、この u についての帰納法を用いる。$\pi^1[q] \in \{\, k \mid k < q;\ P[1..q] \succcurlyeq P[1..k] \,\}$ は $\pi[q]$ の定義 (5.3) より自明。帰納法の仮定として

$$\pi^1[q], \ldots, \pi^u[q] \in \{\, k \mid k < q;\ P[1..q] \succcurlyeq P[1..k] \,\}$$

を置き、$i = \pi^{u+1}[q]$ とすると $\pi[\cdot]$ の定義 (5.3) より $P[1..\pi^u[q]] \succcurlyeq P[1..\pi^{u+1}[q]] = P[1..i]$ となる。ここで関係 $\succcurlyeq$ の推移性を使うと $P[1..q] \succcurlyeq P[1..\pi^{u+1}[q]] = P[1..i]$ が得られる。従って、$i \in \{\, k \mid k < q;\ P[1..q] \succcurlyeq P[1..k] \,\}$ が示された。
$\pi^*[q] \supseteq \Pi[q]$ の証明：
$\{\, k \mid k < q;\ P[1..q] \succcurlyeq P[1..k] \,\}$ に含まれる任意の元をを j とする。つまり j は

$$j < q;\ P[1..q] \succcurlyeq P[1..j]$$

を満たしている。これと $\pi[q]$ の定義より

$$j \leq \pi[q]$$

が得られる。もしも $j = \pi[q]$ であれば $j \in \pi^*[q]$ であるから証明を終わる。従って以降は

$$j < \pi[q]$$

の場合を考えればよい。上の不等式より $\pi^*[q]$ の中には j より大きいもの（少なくとも $\pi[q]$ がその 1 つである）が存在する。そこで、

$$j' = \min\{\, k \mid k > j;\ k \in \pi^*[q] \,\}$$

が定義できて、$j' > j$ である。ここで、ある u が存在して $j' = \pi^u[q]$ となっていることを記憶しておくように。まず j の選び方より $P[1..q] \not\succcurlyeq P[1..j]$。さらに $j' \in \pi^*[q] \subseteq \{\, k \mid k < q;\ P[1..q] \not\succcurlyeq P[1..k] \,\}$ より $P[1..q] \not\succcurlyeq P[1..j']$（$\pi^*[q] \subseteq \{\, k \mid k < q;\ P[1..q] \not\succcurlyeq P[1..k] \,\}$ はこの証明の前半で示した）。ここで $j' > j$ を思い起こすと $P[1..j'] \not\succcurlyeq P[1..j]$ を得る。これから π の定義より、$\pi[j'] \geq j$ が分かるが、j と j' の選び方より

$$\pi[j'] = j$$

が導かれる。実際、

$$\pi[j'] = \pi[\pi^u[q]] = \pi^{u+1}[q] \in \pi^*[q]$$

と $\pi[j'] < j'$ に注意すれば、$\pi[j'] > j$ と仮定すると、$\min\{\, k \mid k > j;\ k \in \pi^*[q] \,\}$ として、j' が選ばれていることに矛盾する。以上のことは $j = \pi[\pi^u[q]] = \pi^{u+1}[q] \in \pi^*[q]$ を意味する。 □

$\pi[q]$ の定義 (5.3)、$\Pi[q]$ の定義 (5.4)、補題 5.6 を用いると、結局

$$\begin{aligned}
\pi[q] &= \max\{\, \ell \mid \ell < q;\ P[1..q] \not\succcurlyeq P[1..\ell] \,\} \\
&= \max\{\, \ell \mid \ell < q;\ P[1..q-1] \not\succcurlyeq P[1..\ell-1];\ P[q] = P[\ell] \,\} \\
&= 1 + \max\{\, k \mid k < q-1;\ P[1..q-1] \not\succcurlyeq P[1..k];\ P[q] = P[k+1] \,\} \\
&= 1 + \max\{\, k \mid k \in \Pi[q-1];\ P[q] = P[k+1] \,\} \\
&= 1 + \max\{\, k \mid k \in \pi^*[q-1];\ P[q] = P[k+1] \,\}
\end{aligned}$$

が得られる。つまり $\pi[q]$ は $q-1$ 以下に対する π の値と条件 $P[q] = P[k+1]$ とから帰納的に計算できる。それを次の定理にまとめておく。

定理 5.7.

$$\pi[q] = 1 + \max\{\, k \mid k \in \pi^*[q-1];\ P[q] = P[k+1] \,\} \tag{5.10}$$

ただし、$k \in \pi^*[q-1]$ かつ $P[q] = P[k+1]$ なる k が存在しない場合は $\pi[q] = 0$。

例 5.8. $P = \texttt{acaacab}$ について $\pi[1..7] = 0011230$ である。実際 $\pi[5] = 2, \pi[\pi[5]] = \pi[2] = 0$ なので $\pi^*[5] = \{\pi[5], \pi[2]\} = \{2, 0\}$ となり、

$$\begin{aligned}
&\{\, k \mid k \in \pi^*[5];\ P[6] = P[k+1] \,\} = \{\, k \mid k \in \{2,0\};\ \texttt{a} = P[k+1] \,\} = \{2\} \\
&\pi[6] = 1 + \max\{\, k \mid k \in \pi^*[5];\ P[6] = P[k+1] \,\} = 1 + 2 = 3
\end{aligned}$$

となる。

定理 5.7 より前掲の Compute-Prefix-Function(P) が $\pi[q]$ を正しく計算していることが導ける。実際、$\pi^*[q-1] = \{\, \pi^1[q-1], \pi^2[q-1], \ldots, \pi^t[q-1] \,\}$ でかつ $\pi^1[q-1] > \pi^2[q-1] > \cdots > \pi^t[q-1] = 0$ であるから、$\pi^1[q-1], \pi^2[q-1], \ldots, \pi^u[q-1], \ldots, \pi^t[q-1]$ の順で $\pi^*[q-1]$ の要素を見ていき、$P[q] = P[\pi^u[q-1]+1]$ が初めて成り立つ番号 $\pi^u[q-1]$ を見つければよい。Compute-Prefix-Function(P) はその通りのことを行っている。

5.2 演習問題

1. 定理 5.7 の「ただし、$k \in \pi^*[q-1]$ かつ $P[q] = P[k+1]$ なる k が存在しない場合は $\pi[q] = 0$」を証明せよ。

2. 定理 5.7 を用いて Compute-Prefix-Function(P) が $\pi[q]$ を正しく計算していること示せ。

3. $P =$`reminrreremiremin` に対して prefix function π を計算せよ。

6 Finite Automaton 有限オートマトン

次節の Finite-Automaton-Matcher のために有限オートマトンを定義する。

△ 有限オートマトンは $(Q, q_0, A, \Sigma, \delta)$ の 5 つ組

- Q は有限の状態集合
- $q_0 \in Q$ は初期状態
- $A \subseteq Q$ は受理状態
- Σ は入力アルファベット
- $\delta : Q \times \Sigma \to Q$ は状態推移関数

△ $\phi : \Sigma^* \to Q$ は最終状態関数

- $\phi(\varepsilon) = q_0$、ε は空のパターン
- $\phi(wa) = \delta(\phi(w), a)$、$w \in \Sigma^*, a \in \Sigma$

注 6.1. ここで注意すべきは、オートマトンは状態推移関数 δ に従って動作するのであって、最終状態関数 ϕ によるのではないという点である。文字列照合を行うオートマトンを作る際には、「望ましい最終状態関数 ϕ を与える状態推移関数 δ はどのように構成できるか」が問題となる。

δ	a	b
0	1	0
1	0	0

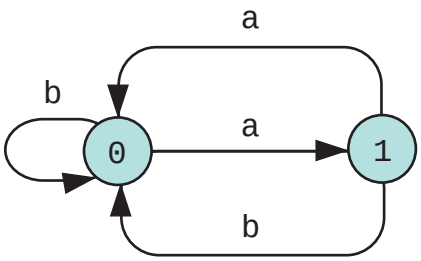


図 10: 有限オートマトンの例

7 String-Matching Automata 文字列照合オートマトン

7.1 Finite-Automaton-Matcher

次図は $P = \texttt{ababaca}$ に対応するオートマトンである。$Q = \{0, 1, 2, 3, 4, 5, 6, 7\}, q_0 = 0, A = \{7\}, \Sigma = \{\texttt{a}, \texttt{b}, \texttt{c}\}$ で δ は表 8 に示してある。各状態を状態 0 に推移させる入力については対応する枝を示していない。

$S \in \Sigma^*$ に対して、P の初めの k 文字が作る文字列 $P[1..k]$ が S の末尾と一致しているような最大の k を与える関数 $\sigma : \Sigma^* \to \{0, 1, \ldots, m\}$ を suffix function という。$P[1..0] = \varepsilon$ との便法を用いれば

$$\sigma(S) = \max\{\, k \mid S \succcurlyeq P[1..k] \,\} \tag{7.1}$$

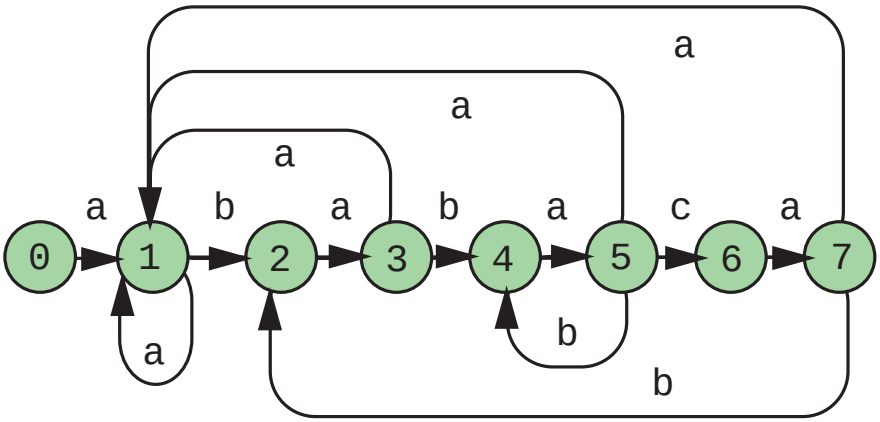


図 11: String-Matching Automaton

表 8: 状態推移関数

	input		
state	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

例：$P = \texttt{ab} \quad \Rightarrow \quad \sigma(\varepsilon) = 0, \sigma(\texttt{ccaca}) = 1, \sigma(\texttt{ccab}) = 2$
定義より

$$\sigma(T[1..s+m]) = m \quad \Leftrightarrow \quad \sigma(T[s+1..s+m]) = m \quad \Leftrightarrow \quad T[s+1..s+m] = P[1..m] \tag{7.2}$$

最終状態関数 $\phi : \Sigma^* \to Q$ がこの $\sigma : \Sigma^* \to Q$ となる有限オートマトンを構成する。そのようなオートマトンの状態推移関数 δ を用いると Finite-Automaton-Matcher は以下のようになる。

```
    Finite-Automaton-Matcher
1:  n:=length[T]
2:  q:=0
3:  for i:=1 to n
4:      do  q:=δ(q,T[i])
5:          if q=m
6:              then    s:=i-m
7:                      print "Pattern occurs with shift" s
```

状態推移関数 δ の決める最終状態関数 ϕ が $\phi = \sigma$ を満たしていれば、この Finite-Automaton-Matcher が正しく動くことは自明である。

表 9: Suffix Function

S	c	c	a	c	a			k
P						a	b	0
					a	b		$1 = \sigma(S)$
				a	b			×

7.2 計算複雑度

δ が用意されていれば $O(n)$。

7.3 演習問題

図 11 のオートマトンの最終状態関数 ϕ が $P =$ ababaca に対する suffix function σ と一致することを確かめよ。

7.4 オートマトンの構成

△ $Q = \{0, 1, \ldots, m\}$

△ 状態推移関数 δ を

$$\delta(q, a) = \sigma(P[1..q]a) \tag{7.3}$$

と定義すると、後に定理 7.4 に示すように δ から決まる最終状態関数 ϕ は suffix function σ と一致する。従って、(7.2) より受理状態 m に到達すれば、マッチするシフトを見いだせることになる。

7.5 Suffix Function の性質と $\phi = \sigma$ の証明

補題 7.1. $\forall X, Y \in \Sigma^*$ について

$$X \succcurlyeq Y \quad \Rightarrow \quad \sigma(X) \geq \sigma(Y) \tag{7.4}$$

Proof. $k = \sigma(Y)$ とすると下の図より明らか。

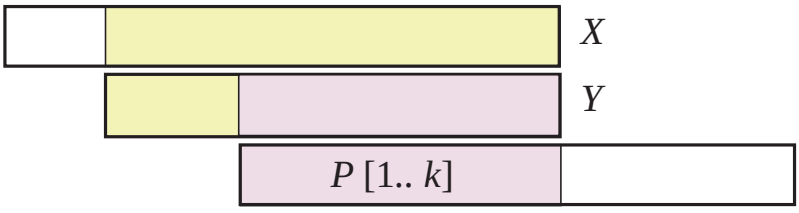


図 12: proof of Lemma 7.1

□

補題 7.2. $\forall X \in \Sigma^*$ $\forall a \in \Sigma$ について

$$\sigma(Xa) \leq \sigma(X) + 1 \tag{7.5}$$

Proof. 図 13 参照。$r = \sigma(Xa)$ とする。$r = 0$ なら (7.5) は自明。$r > 0$ とすると、σ の定義より $Xa \succcurlyeq P[1..r]$。従って $X \succcurlyeq P[1..r-1]$。よって再び σ の定義より

$$r - 1 \leq \sigma(X)$$

□

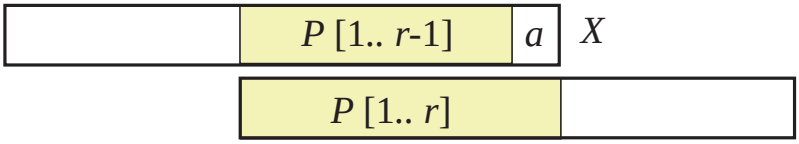


図 13: Suffix-function inequality

補題 7.3. $\forall X \in \Sigma^* \; \forall a \in \Sigma$ について

$$q = \sigma(X) \quad \Rightarrow \quad \sigma(Xa) = \sigma(P[1..q]a)$$

Proof. 仮定 $q = \sigma(X)$ と σ の定義より $X \succcurlyeq P[1..q]$ であるから、この両辺の文字列に 1 文字 a を追加しても同じ関係

$$Xa \succcurlyeq P[1..q]a \tag{7.6}$$

が成り立つ。つぎに、$r = \sigma(Xa)$ とするとまず σ の定義より

$$Xa \succcurlyeq P[1..r] \tag{7.7}$$

が成り立つ。さらに補題 7.2 より

$$r \leq \sigma(X) + 1 = q + 1 \tag{7.8}$$

が得られる。以上の 3 つの結果 (7.6),(7.7),(7.8) から

$$P[1..q]a \succcurlyeq P[1..r] \tag{7.9}$$

が結論できる（図 14)。

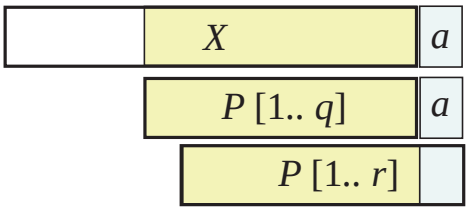


図 14: $Xa, P[1..q]a$ and $P[1..r]$

よって補題 7.1 と r の決め方から

$$\sigma(P[1..q]a) \geq \sigma(P[1..r]) = r = \sigma(Xa) \tag{7.10}$$

となる。ここで、$\sigma(P[1..r]) = r$ は σ の定義に戻れば自明である。

一方 $Xa \succcurlyeq P[1..q]a$ に補題 7.1 を適用すれば

$$\sigma(P[1..q]a) \leq \sigma(Xa) \tag{7.11}$$

が得られる。以上の不等式 (7.10) と (7.11) から補題の主張が得られる。 □

定理 7.4. パターン P に対応して作られたオートマトン (7.3) に $T[1..n]$ が入力されたとする。このとき $i = 0, 1, \ldots, n$ について

$$\phi(T[1..i]) = \sigma(T[1..i])$$

Proof. $i = 0$ については $T[1..0] = \varepsilon$ より自明。$\phi(T[1..i]) = \sigma(T[1..i])$（これを q とおく）を帰納法の仮定として置き、$\phi(T[1..i+1]) = \sigma(T[1..i+1])$ を示す（どこに帰納法の仮定を使ったかが少々見えにくい証明になっているので注意)。以下 $T[i+1]$ を a と書くことにする。

$$\begin{aligned}
\phi(T[1..i+1]) &= \phi(T[1..i]a) && a \text{ の定義 } a = T[i+1] \text{ より} \\
&= \delta(\phi(T[1..i]), a) && \text{最終状態関数の性質より} \\
&= \delta(q, a) && q \text{ の定義 } q = \phi(T[1..i]) \text{ より} \\
&= \sigma(P[1..q]a) && \delta \text{ の定義 (7.3) より} \\
&= \sigma(T[1..i]a) && \text{補題 7.3 より} \\
&= \sigma(T[1..i+1]) && a \text{ の定義 } a = T[i+1] \text{ より}
\end{aligned}$$

□

7.6 δ の計算

Compute-Transition-Function

```
1: m:=length[P]
2: for q:=0 to m
3:     do for each character a∈ Σ
4:         k:=min(m+1,q+2)
5:             repeat k:=k-1 until P[1..q]a ≽ P[1..k]
6:             δ(q,a):=k
7: return δ
```

この計算複雑度は $O(m^3|\Sigma|)$（`P[1..q]a` $\succcurlyeq$ `P[1..k]` の判定にも m 程度の計算時間がかかる点に注意。この部分が m の肩の指数を 2 ではなく 3 にしている)。

$P =$ `aabab` に対して上の計算を実行すると表 10 のようになる。ただし、$\Sigma = \{\mathtt{a}, \mathtt{b}\}$ としている。

7.7 演習問題

1. $P =$ `aabab` に対応するオートマトンを完成させよ。それを $T =$ `aaababaabaababaab` に用いよ。
2. $P =$ `ababbabbababbababbabb` に対応するオートマトンを構成せよ。
3. $P[1..q] \succcurlyeq P[1..k]$ のとき常に $k = 0$ あるいは $k = q$ となるパターンについて、オートマトンを構成せよ。

表 10: $P = \mathtt{aabab}$ に対する状態推移関数

q	a	k	
0	a	2	
		1	$P[1..0]\mathtt{a} \not\succcurlyeq P[1..1]$
			$\delta(0, \mathtt{a}) = 1$
	b	2	
		1	$P[1..0]\mathtt{b} \not\succcurlyeq P[1..1]$
		0	$P[1..0]\mathtt{b} \succcurlyeq P[1..0]$
			$\delta(0, \mathtt{b}) = 0$
1	a	3	
		2	$P[1..1]\mathtt{a} \succcurlyeq P[1..2]$
			$\delta(1, \mathtt{a}) = 2$
	b	3	
		2	$P[1..1]\mathtt{b} \not\succcurlyeq P[1..2]$
		1	$P[1..1]\mathtt{b} \not\succcurlyeq P[1..1]$
		0	$P[1..1]\mathtt{b} \succcurlyeq P[1..0]$
			$\delta(1, \mathtt{b}) = 0$
			$\vdots$
4	a	6	
		5	$P[1..4]\mathtt{a} \not\succcurlyeq P[1..5]$
		4	$P[1..4]\mathtt{a} \not\succcurlyeq P[1..4]$
		3	$P[1..4]\mathtt{a} \not\succcurlyeq P[1..3]$
		2	$P[1..4]\mathtt{a} \succcurlyeq P[1..2]$
			$\delta(4, \mathtt{a}) = 2$
	b	6	
		5	$P[1..4]\mathtt{b} \succcurlyeq P[1..5]$
			$\delta(4, \mathtt{b}) = 5$

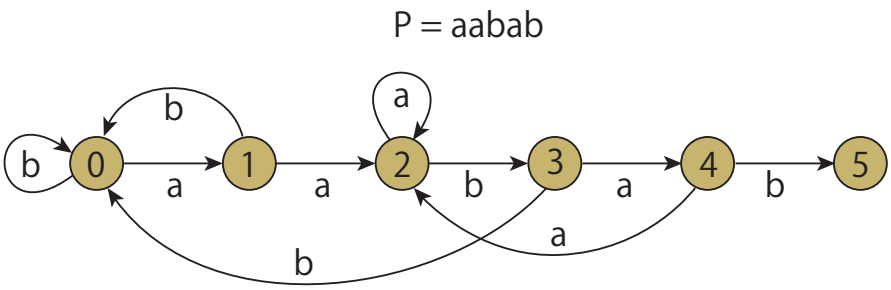


図 15: $P = \mathtt{aabab}$ に対するオートマトン

参考文献

1. T.H. Cormen, C.E. Leiserson and R.L. Rivest, *Introduction to Algorithms*, MIT Press, 1990, ISBN:0-262-53091-0; 浅野哲夫 他 訳「アルゴリズムイントロダクション」近代科学社, 1995, ISBN:4764902451, 476490246X, 4764902478

2. R. Sedgewick, *Algorithms*, 2nd ed., Addison-Wesley, 1988, ISBN:0201066734; 野下 浩平, 星 守, 佐藤 創, 田口 東 訳「アルゴリズム」近代科学社, 1992, ISBN:4-7649-0189-7

3. R. Sedgewick & Philippe Flajolet, *Analysis of Algorithms*, Addison-Wesley, 1996, ISBN:0-201-40009-X

4. 西原清一「データ構造」オーム社, 1993, ISBN:4-274-12955-1

5. 茨木俊秀「アルゴリズムとデータ構造」 昭晃堂, 1989, ISBN:4785601191

6. N. Wirth, *Algorithms and Data Structures*, Prentice-Hall, 1986, ISBN:0130219991; 浦 昭二, 国府方 久史 訳「アルゴリズムとデータ構造」近代科学社, 1991, ISBN:4-7649-0162-5

7. D. Gusfield, *Algorithms on Strings, Trees and Sequences*, Cambridge University Press, 1997, ISBN:0-521-58519-8.

8. 定兼邦彦「文字列検索の索引構造とその効率化」第 14 回 RAMP シンポジウム論文集, Sep. 2002, 30–46.