

文字列の同時置き換えのための パターン・マッチング・マシンについて

九州大学理学部 有川節夫

1. はじめに

筆者等は九州大学大型計算機センターにおいて、文字列を対象とした研究者用の情報システム SIGMA を実現した [1], [2]. このシステムで扱うデータはほとんど文字列データである. そこで Aho-Corasick [3] によるパターン・マッチング・マシンの手法を駆使した一方向逐字処理 [4] を開発し使用している.

SIGMA システム 1 つのコマンドとして、テキスト・データの編集に有用な複数個の文字列の同時置き換えを行うための REPLACE がある. これは、与えられたキーワードの対の集合

$$K = \{(x_1, y_1), (x_2, y_2), \dots, (x_k, y_k)\} \quad (i \neq j \text{ なら } x_i \neq x_j)$$

に対して、テキスト系列を左から右へ一読する間に集合

$$K_X = \{x_i; (x_i, y_i) \in K\}$$

の中のなるべく長いキーワード x_i を探し出し、それを対応

するキーワード y_i で次々に置き換えてできるテキスト系列を作るものである。

SIGMA システムでは、 K_X から Aho-Corasick の方法でパターン・マッチング・マシンを作り、それを走らせるアルゴリズムに工夫をこらして、REPLACE コマンドを実現している。しかし、これでは走査時の負担が大きくなり効率的でないので、新しく固有のパターン・マッチング・マシンを開発した。以下にその概要について報告する。

2. 文字列の同時置き換え

上で述べた「なるべく長いキーワード x_i を探し出しそれを y_i で置き換える」ということは、もう少し正確にいうと次のようなことである。図 1 において直線は入力テキスト系列を表わすものとし、 $x_i \in K_X$ とする。

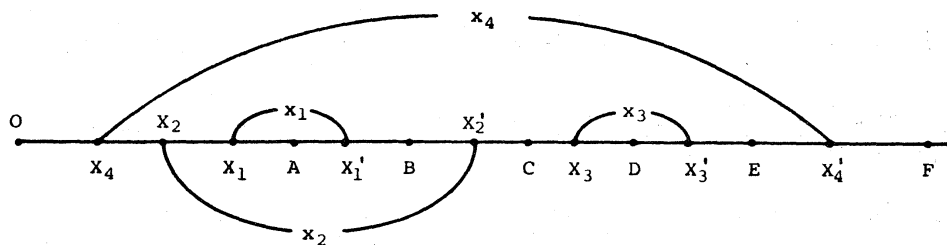


図 1. 入力テキストとキーワード x_i

我々が求めるパターン・マッチングマシンは $[OA]$, $[OB]$, ... 等

で表わされる入力テキスト系列を次のように変換し出力するものである。

text	replaced text
[OA]	[OA]
[OB]	$[OX_1)Y_1(X_1'B]$
[OC]	$[OX_2)Y_2(X_2'C]$
[OD]	$[OX_2)Y_2(X_2'D]$
[OE]	$[OX_2)Y_2(X_2'X_3)Y_3(X_3'E]$
[OF]	$[OX_4)Y_4(X_4'F]$

さて、こうした文字列の同時置き換えのためのパタン・マッチング・マシン (RPMM と書く) は、Aho-Corasick のマシンと同様に goto 関数 g と、failure 関数 f 、出力関数 $output$ からなる。与えられたキーワードの対の集合 K から g , f , $output$ を作るわけであるが、そのアルゴリズムは Algorithm 1 と 2 である。 g を作る Algorithm 1 は Aho 達のもものと $output$ 関数への割当ての所を除いて全く同一である。 Algorithm 2 は、failure 関数 f と Algorithm 1 で部分的に求められた $output$ を完成させるためのものである。

Algorithm 1, 2 によって作られる RPMM を、 $M = (K, g, f, output)$ で示すことにする。

(例) $K = \{(ABCD, \alpha), (CDE, \beta), (BC, \gamma)\}$ に対する RPMM

Algorithm 1. (Construction of the goto function)

Input. Set of keyword pairs $K=\{(x_1, y_1), \dots, (x_k, y_k)\}$.

Output. Goto function g and a partially computed output function output.

Method.

```

begin
  newstate:=0;
  for i:=1 until k do enter( $x_i$ );
  for all a such that  $g(0, a)=fail$  do  $g(0, a):=0$ 
end
procedure enter( $a_1a_2\dots a_m, y$ ):
  begin
    state:=0; j:=1;
    while  $g(state, a_j) \neq fail$  do
      begin
        state:= $g(state, a_j)$ ;
        j:=j+1
      end
    for p:=j until m do
      begin
        newstate:=newstate + 1;
         $g(state, a_p):=newstate$ ;
        state:=newstate
      end
    output(state):=y
  end
end

```

$M = (K, g, f, \text{output})$ を図 2 に示す。この図において、破線を除いた部分が goto 関数 g であり、破線及び破線の出ていない状態から、状態 0 への遷移が failure 関数 f を表わしている。出力関数は、二重丸の印いた状態に対しては、Algorithm 1 により定義され、0 以外の残りの状態に対しては、Algorithm 2 により定義される。

このようにして Algorithm 1, 2 により作られた RPMM $M = (K, g, f, \text{output})$ が、入力テキスト上を Algorithm 3 に従って

Algorithm 2. (Construction of the failure function).

Input. Goto function g and output function $output$ from Algorithm 1.

Output. Failure function f and output function $output$.

Method.

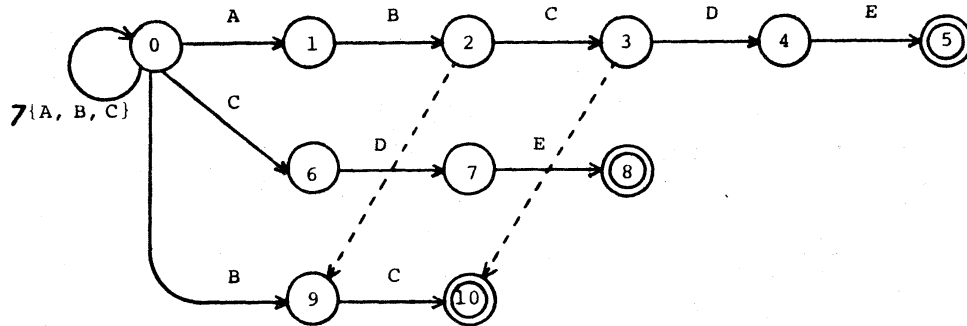
```

begin
  queue:=empty;
  for each a such that  $g(0, a)=s \neq 0$  do
    begin
      queue:=queue.s;
       $f(s):=0$ ;
      if  $output(s)$  is undefined then  $output(s):=a$ 
    end
  while queue  $\neq$  empty do
    begin
      let queue=r.tail;
      queue:=tail;
      for each a such that  $g(r, a)=s \neq fail$  do
        begin
          queue:=queue.s;
          if  $output(s)$  is defined then  $f(s):=0$  else
            begin
              state:= $f(r)$ ;
              while  $g(state, a)=fail$  do
                begin
                   $output(s):=$ if  $output(s)$  is defined
                    then  $output(s).output(state)$ 
                    else  $output(state)$ ;
                  state:= $f(state)$ 
                end
               $f(s):=g(state, a)$ ;
               $output(s):=$ if  $output(s)$  is defined
                then  $output(r).output(s)$ 
                else  $output(r)$ ;
              if  $f(s)=0$  then  $output(s):=output(s).a$ 
            end
          end
        end
      end
    end
  end
end

```

走ることになる。この Algorithm 3 は f による failure 遷移をするたびにその状態に対する出力を関数 $output$ に従って出す。また入力テキストを読み終った時点では、その時の状態から状態 0 に至るまでの failure 遷移をくり返し対応する出力を

$$K = \{(ABCDEFG, \alpha), (BC, \beta), (EF, \gamma)\}$$



s	output(s)
0	undefined
1	A
2	A (output(1))
3	A
4	A γ D (output(3)·output(10)·a)
5	α
6	C
7	CD (output(6)·a)
8	β
9	B
10	γ

図2. R-パターン・マッチング・マシン

出す. なお $g(0, a) = 0$ に対しては a を出力するようになっている点に注意しよう.

(例) 図2のMは入力テキスト DEABCCBCE上で, 図3のように動く. 結局 $0 \rightarrow 0$ 遷移の際の入力記号と破線で示した failure 遷移による出力を綴り合せた DEAYC γ E が出力系列と

して得られる。

Algorithm 3. (Pattern matching machine for multiple replacement)

Input. A text string $z=a_1a_2\dots a_n$ and a pattern matching machine with functions g , f and output.

Output. A replaced text string $w=b_1b_2\dots b_t$.

Method.

```

begin
  state:=0;
  for i:=1 until n do
    begin
      while g(state, ai)=fail do
        begin
          print output(state);
          state:=f(state)
        end
      state:=g(state, ai);
      if state=0 then print ai
    end
  while state ≠ 0 do
    begin
      print output(state);
      state:=f(state)
    end
  end
end

```

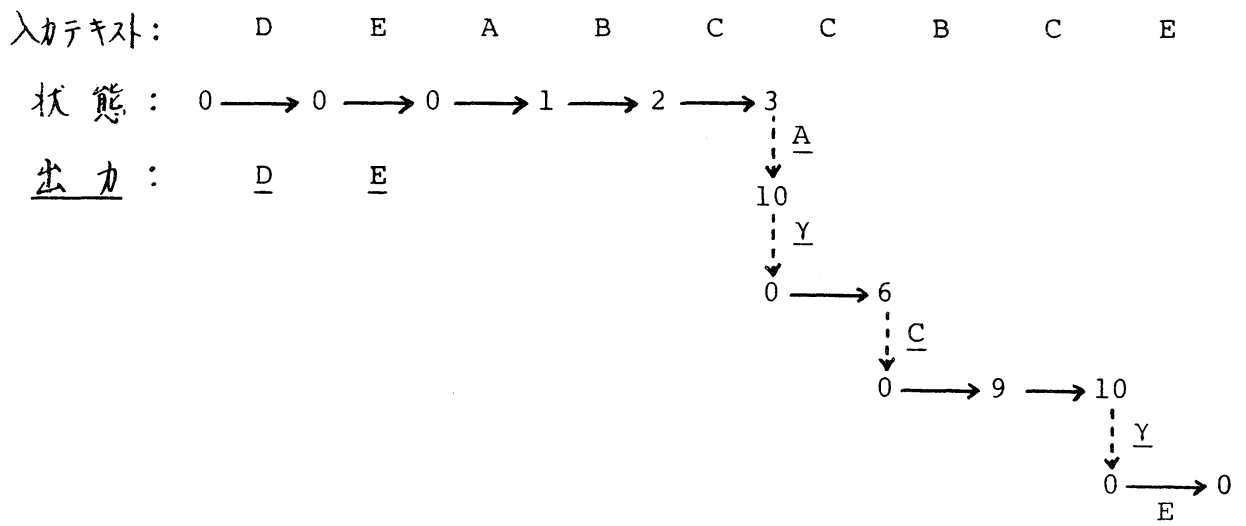


図 3. R-パターン・マッチング・マシン M の動き

3. アルゴリズムの妥当性

Algorithm 1, 2 で作られた RPMM $M=(K, g, f, \text{output})$ は Algorithm 3 に従ってテキスト系列上を走るとき, 我々の目的にかなった妥当なものであるといえる.

いま, $M=(K, g, f, \text{output})$ を RPMM として, 任意のテキスト系列 w に対して, Algorithm 3 を適用して得られる出力系列を $M(w)$ とする. 任意の入力テキスト系列 w に対して, u を

$$(1) \quad w = uxv, \quad x \in K_X$$

となる x が存在するときは, そのような最も短い文字列とし, x をこの u に対して (1) を満たす最も長い文字列とする.

(1) を満たす x が存在しないときは $u=a$ (a は文字), $x=\varepsilon$ (ε は空列) とする.

$$M(w) = M(uxv) = uK(x)M(v)$$

であれば, RPMM M は 妥当 (valid) であるという. ここに, $K(\varepsilon)=\varepsilon$ と仮定する.

定理 1. Algorithm 3 に従って走る RPMM $M=(K, g, f, \text{output})$ は妥当である.

証明は Aho-Corasick [3] と同様に関数 g, f, output の性質に関する補題を用意して行う.

4. 時間計算量

Algorithm 1, 2, 3 の時間計算量 (time complexity) に関しては, [3] と全く同様にしていずれも線形時間であることが示される。即ち,

定理 2. Algorithm 3 が長さ n の入力テキストの変換に要する時間を $T(n)$ とすると

$$T(n) < 3n$$

である。

証明. [3] と違, τ failure 遷移の回数が最悪の場合 $2(n-1)$ 回生じる可能性がある。

定理 3. Algorithm 1 が要する時間は, K_X のキーワードの長さの和に関して線形である。

定理 4. Algorithm 2 が要する時間は, K に含まれる $2k$ 個のキーワードの長さの和に関して線形である。

5. 最適化

Algorithm 1, 2 で作られた RPMM には冗長な failure 遷移が含まれている。例えば, 図 2 の M において, 状態 2 及び 3 からの failure 遷移は, その行き先きである状態 9, 10 で再び 0 への failure 遷移を必要とするので, 冗長である。このような f と output を次の方法で, [3], [4] と同様にして, f' , output' へと最適化することができる。

$$f'(1) = 0,$$

$$f'(i) = \begin{cases} f'(f(i)) & (\forall a \text{ に対して } g(f(i), a) = \text{fail} \text{ 又は } g(i, a) \neq \text{fail} \text{ のとき}) \\ f(i) & (\text{そうでないとき}); \end{cases}$$

$$\text{output}'(1) = \text{output}(1)$$

$$\text{output}'(i) = \begin{cases} \text{output}(i) & (f'(i) = f(i) \text{ のとき}) \\ \text{output}(i)\text{output}(f(i)) & (\text{そうでないとき}), \end{cases}$$

ここに、状態 1, i は深さ 1, i の状態を表わすものである。

この方法を図 2 の M に適用すると、g, f' としては、破線部をとり除いたクラスで表わされるものをもち、出力関数として

$$\text{output}'(2) = AB$$

$$\text{output}'(3) = Ay$$

$$\text{output}'(i) = \text{output}(i) \quad (i \neq 2, 3)$$

をもつ RPMM M' へと最適化される。

6. おわりに

文字列の同時置き換えを行うためのパターン・マッチング・マシンを Aho-Corasick とほぼ同様のアルゴリズムで構成し、走らせることが出来ることを示した。ここで示した方法は最長のマッチングをとるための方法としても使えるので、構文解析の道具としても応用できるであろう。

参考文献

1. 有川, 篠原, 白石, 玉越: 研究者向き情報システム SIGMA について, 九大大型計算機センター 広報, Vol. 14, No.4, 1981, 550-573.
2. Arikawa, S. et al.: SIGMA — An Information System for Researchers Use, Bull. Inform. Cybernetics (1982, in press).
3. Aho, A.V. and Corasick, M.T.: Efficient String Matching: An Aid to Bibliographic Search, CACM 18, 1975, 335-340.
4. Arikawa, S.: One-Way Sequential Search Systems and Their Powers, Bull. Math. Stat., Vol. 19, 1981, 69-85.