数理解析研究所講究録
第353巻 1979年 58-67

最大共通部分系列問題と共通連続部分系列問題のアルゴリズム

京都大学　工学部　　中津　楢男
上林　弥彦
矢島　脩三

1. まえがき

本稿では，与えられた2つの系列の最大共通部分系列及び共通連続部分系列を求めるアルゴリズムを与える。

最大共通部分系列の問題に関しては，特定の場合を除いて最悪の場合$O(mn)$（但し，m,nは各々系列長を表わす）で処理できるアルゴリズムだけが知られている。ここでは，最大共通部分系列の長さが長い場合に，ほとんど線形時間で処理できるアルゴリズムを提案する。この問題は，2つの異なるファイル間の相異を求める問題，蛋白質分子等の分子間の関係を求める問題や音声認識の一部と関係している。またこの問題は系列内に誤りを許した場合のパターンマッチング問題と考えることができ，2つの系列の近さを測る一手段ともなる。

共通連続部分系列の問題は，系列のデータ圧縮等に利用できる。この問題は，Weinerの示したアルゴリズム[5]を若干変

更することにより線形時間で解決可能であるが，ここでは，カウンタオートマトンを用いたアルゴリズムを提案する。

2．諸定義

$\Sigma=\{a_1, a_2, \cdots, a_k\}$を記号の有限集合とし，$\Sigma$の要素によって生成される全ての系列の集合を$\Sigma^*$とする。$|\Sigma|$は集合$\Sigma$の要素数を示し，$|u|$ $(u\in\Sigma^*)$は系列uの長さを示すものとする。系列λとμ $(\lambda, \mu\in\Sigma^*)$の連接を$\lambda\mu$とかく。

〔定義１〕

Σ上の系列$\lambda=\lambda(1)\lambda(2)\cdots\lambda(n)$ $(\lambda(i)\in\Sigma,\ 1\leq i\leq n)$に対して，$\lambda'=\lambda(i_1)\lambda(i_2)\cdots\lambda(i_k)$ $(1\leq i_1<i_2<\cdots<i_k\leq n)$を$\lambda$の部分系列（subsequence）と呼ぶ。特に，$\lambda''=\lambda(i)\lambda(i+1)\cdots\lambda(j)$を$\lambda$の連続部分系列（consecutive subsequence）と呼び，$\lambda(i)\lambda(i+1)\cdots\lambda(j)$を$\lambda(i:j)$と書くことにする。

〔定義２〕

Σ上の２つの系列λ，ωに対し，λとωの両方の部分系列であってかつ長さが最大の系列を最大共通部分系列（Longest Common Subsequence，以下ＬＣＳと略す）と呼ぶ。与えられた２つの系列のＬＣＳを求める問題をＬＣＳ問題と呼ぶ。

λとωの両方の連続部分系列となる系列を，λとωの共通連続部分系列（Common Consecutive Subsequence，以下ＣＣＳと略す）と呼ぶ。与えられた２つの系列のＣＣＳを長さの長い

順に全て求める問題をCCS問題と呼ぶ。

3、最大共通部分系列(LCS)の計算アルゴリズム

2つの系列A, W ($\in \Sigma^*$) が与えられた時に、$|A|=|W|=n$ とした時、$|\Sigma|$ が有限でない場合にはLCS問題を解決するのに $O(n^2)$ の計算量を必要とすることがAhoらによって示された。$|\Sigma|$ が有限の場合には線形アルゴリズムの存在する可能性がAhoらによって示されている[1]。m を2つの系列の内長い方の系列長、n を短い方の系列長とし、p を2つの系列のLCSの長さとする ($p \leq n$)。Huntらは $O((r+n)\log n)$ アルゴリズムを示した[2]。ここで r は平均すると $mn/|\Sigma|$ である。Hirschbergは $O(mp)$ と $O(p(n-p)\log m)$ なる2つのアルゴリズムを提案した[3]。ここでは、$O(m(n-p))$ と新たな $O(p(n-p)\log m)$ アルゴリズムを示す。これらのアルゴリズムの最悪の場合の計算量はいずれも $O(mn)$ 或いは $O(n^2\log m)$ である。アルゴリズムの計算量は従来最悪ケースで議論されてきたが、実行時間は本来入力の性質によって変化するものであり、入力に依存した形でアルゴリズムの計算量を表現し、入力の性質に適したアルゴリズムを選択することも、実行時間を考慮すれば重要であると考えられる。

[定義3]

与えられた2つの系列 $A=A(1)A(2)\cdots A(n)$, $W=W(1)W(2)\cdots$

$w(m)$ に対して，$s(i:n)$ と $w(l:m)$ が長さ k のＬＣＳを持つ様な l の最大値を $L_i(k)$ で表わす（但し，$m \geq n$）。

この時次の３つの補題が成り立つ。証明は省略する。

〔補題１〕

$\forall i\ (1 \leq i \leq n)$，$L_i(1) > L_i(2) > \cdots$ が成り立つ。

但し，$L_i(k)$ の値が存在しない場合を除く。

〔補題２〕

$\forall i\ (1 \leq i \leq n-1)$，$\forall j\ (1 \leq j \leq n)$，$L_{i+1}(j) \leq L_i(j)$ が成り立つ。

〔補題３〕

$$L_i(j) = \begin{cases} Max(L_{i+1}(j), l) & \text{但し } l \text{ は，} s(i) = w(l) \text{ かつ} \\ \quad l < L_{i+1}(j-1) \text{を満足する最大の数.} \\ L_{i+1}(j) & \text{上記の条件を満足する } l \text{ が存在しない時.} \end{cases}$$

ここで $Max(x, y)$ は，x と y の大きい方を示す。

与えられた系列 s、w のＬＣＳを求めるための，$O(m(n-p))$ アルゴリズムを次に示す。このアルゴリズムにおける探索方法を変更することにより $O(p(n-p)\log m)$ アルゴリズムを得る。

２つの系列 s，w $(|s| = n \leq |w| = m)$ に対し，$(n+1) \times n$ の行列Ｌを用意する。Ｌの (i, j) 要素は，定義３の $L_{n-i+1}(j)$ を示す。ＬＣＳを求めるために大きさ n のリスト ＬＣＳ および ｐｏｓを用いる。

〔アルゴリズム１〕 $O(m(n-p))$

```
(1) p←n, i←n, len←0, LCS(k)←0, pos(k)←0 for all k
(2) j←1, q←m+1
(3) if i=0 then go to (7)
(4) if p=n or j > len then L(j,i+1)←0
(5) for x←q-1 step -1 until L(j,i+1) do
      begin
        if s(i)=w(x) then begin
                            q←x, go to (6)
                          end
      end of loop (5)
    q←L(j,i+1)
(6) if q=0 and len ≧ p-1 then go to (7)
    if q=0 and len < p-1 then begin
                                p←p-1, i←p, go to (2)
                              end
    L(j,i)←q, len←Max(len, j), j←j+1, i←i-1
    go to (3)
```

(7) s と w の LCS の長さは len である。$L(len, k)$ $(i \le k \le n)$ の最大値を $LCS(len)$ とし、それを満たす k の最大値を $POS(len)$ とする。

```
(8) for j←1 step 1 until (len-1) do
      begin
        L(len-j, k) (POS(len-j+1) ≦ k ≦ n) の最大値を LCS(len-j) とし、
        それを満たす k の最大値を POS(len-j) とする。
      end of loop (8)
```

(9) s と w の LCS の 1 つは、$w(LCS(len))\, w(LCS(len-1)) \cdots w(LCS(1)) = s(POS(len))\, s(POS(len-1)) \cdots s(POS(1))$ である。

〔例 1〕

$s = bcdabab$, $w = cbacbaaba$ に対してアルゴリズム 1 を適用した後の行列 L を右図に示す。LCS は、

	b	c	d	a	b	a	b
c						9	⑧ 0
b					8	⑦	0
a				7	⑤	0	
c			3	③	0		
b	①	0	0				
a	0						
a							
b							
a							

$w(1)w(3)w(5)w(7)w(8)=\Lambda(2)\Lambda(4)\Lambda(5)\Lambda(6)\Lambda(7)=cabab$ である。

4. 共通連続部分系列の計算アルゴリズム

与えられた2つの系列を記憶する場合に，両方に共通な部分を新しい記号で置き換えたり，片方の系列は他方の系列と異なる部分だけを記憶する等は最も初歩的なデータ圧縮の方法である。また，1つの系列の中で何度も繰り返し出現する系列を短かい記号で置きかえることにより，40%のデータ圧縮が得られることが知られている[4]。これらの応用に関しては，2つの系列のCCSを求めることが重要となる。1つの系列内の繰り返し系列を求める問題は既に，系列長に比例する時間で解けることがWeinerによって示されている[5]。CCS問題も，このアルゴリズムを若干変更することによって線形時間で解決できるが，後処理が必要な場合もある。ここではオートマトンを用いた方法を示す。すなわち，与えられた系列の連続部分系列を全て受理する有限オートマトンの効率良い構成アルゴリズムを次に示す。

〔定義4〕

オートマトンSPMM(Substring Pattern Matching Machine)は，$(\Sigma, S, s_0, \delta, f, r)$なる6字組で定義される。ここで$\Sigma$は入力記号集合，$S$は状態集合，$s_0$は初期状態を示す。$f$は$S\to S$なる写像でfailure関数と呼ぶ。$r$は$S\to\Sigma$（非

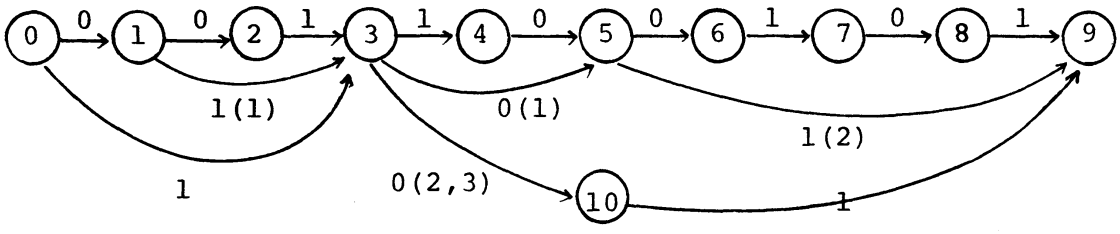
負の整数の集合）なる写像でリセット関数と呼ぶ。δは次状態関数と呼ばれ，$\Sigma \times S \times I \to S \cup \{\Lambda\}$ なる写像である。

SPMMは，状態s_iにおいて，入力a及び，状態s_iにおけるカウンタの値によって，次の様に遷移を行なう。

```
if δ(a,s_i,count)≠Λ then begin
                            enter the state δ(a,s_i,count)
                            count←count+1
                          end
                     else begin
                            if s_i=s_0 then enter the initial state
                                       else begin
                                              count←r(s_i), s_i←f(s_i)
                                              and repeat this process
                                              recursively.
                                            end
                          end
```

〔例２〕

系列Λ＝００１１００１０１に対するＳＰＭＭの状態遷移図を下図に示す。遷移枝には，入力記号と遷移可能なカウンタの値が示されている。



各状態に対するfailure関数，リセット関数の値を次に示す。

状態	0	1	2	3	4	5	6	7	8	9	10
f(i)	0	0	1	0	3	1	2	3	5	3	5
r(i)	0	0	1	0	1	1	2	3	2	2	2

ここで、状態3において、入力が1ならば常に状態4へ、入力が0の時、カウンタの値が1ならば状態5へ、2又は3ならば状態10へ、それ以外の場合はΛ(未定義)とする。カウンタの値が明示されていない枝は、任意のカウンタ値で遷移できるものとする。

与えられた系列Λに対してSPMMを作り、SPMMに他の系列Wを入力し、状態遷移を行ない、カウンタの値を調べることにより、系列ΛとWの共通連続部分系列を|W|に比例した時間で求めることができる。またSPMMのリセット関数の値を調べることにより、Λの中の繰り返し系列を全て求めることもできる。SPMMの構成アルゴリズムを次に示す。

〔アルゴリズム2〕

(1) 与えられた系列 $\Lambda=\Lambda(1)\Lambda(2)\cdots\Lambda(n)$ を受理する一次元の状態遷移図を作る。各枝の遷移可能カウンタ値は任意とする。

```
(2) f(0)←0, f(1)←0, r(0)←0, r(1)←0, avstate←n+1( n=|s| )
(3) for i=2 to n do
        begin
          p←i-1, a←f(p), call CONST(a,p,i)
        end of loop (3)
procedure CONST(a,p,i)
    if δ(a,s(i),r(a)) is defined then begin
                                        f(i)←δ(a,s(i),r(a))
```

```
          r(i)←r(p)+1, return
        end
else begin
       if  δ(a,s(i),*) is not defined (i.e. for any j(>0),
        then begin                              δ(a,s(i),j) is undefined)
              状態 a から S, 状態 i へ, 入力 s(i) に対する遷移枝を作る.
              if a=0(=s0) then begin
                                  f(i)←0, r(i)←0
                                  新しい枝の遷移可能カウント値←0
                                  return
                                end
                           else begin
                                  新しい枝の遷移可能カウント値←[r(a)+1,r(p)]
                                  call CONST(f(p),a,i)
                                end
            end
       else begin
              新状態 avstate をつくり, 状態 a より avstate へ
              s(i) で遷移させる.
              avstate←avstate+1
              δ(a, s(i), ℓ) が定義される様な最大の値を ℓ とする。
              f(avstate)← δ(a,s(i),ℓ), r(avstate)← ℓ+1
              f(i)←f(avstate), r(i)←r(avstate)
              return
            end
    end
end of procedure CONST
```

〔定理1〕

$n = |\Delta|$ とした時, アルゴリズム2は $O(n)$ の計算時間を必要とする。 (証明略)

〔性質1〕

状態iにおける failure 関数，リセット関数の値を各々$f(i)$ $r(i)$とする。この時，$A(i-r(i)+1:i)=A(f(i)-r(i)+1:f(i))$が成り立つ。

アルゴリズム2の正当性は全ての状態が性質1を満足しており，Aの部分系列以外の入力に対しては必ず，次状態が未定義である場合があることを示すことによって証明できる。

5. 結論

本論文ではLCS問題に関して$p>n/2$（pは，LCSの長さ，nは2つの系列の短かい方の長さ）の時に，従来のアルゴリズムより効率の良いアルゴリズムを示した。CCS問題に対して，別の線形アルゴリズムを与えた。このアルゴリズムは従来のアルゴリズムに比べて，特定の応用に関しては必要記憶容量が少なくてすむという利点がある。

謝辞　討論戴いた本学矢島研究室の皆様に心より感謝する。

〔参考文献〕

[1] Aho,A.V., Hirschberg,D.S. and Ullman,J.D., "Bounds on the Complexity of the Longest Common Subsequence Problem", J.ACM, Vol. 23, No. 1, pp. 1-12, Jan. 1976.

[2] Hunt,J.W. and Szymanski,T.G., "A Fast Algorithm for Computing Longest Common Subsequences", C.ACM, Vol. 20, No. 5, May 1977.

[3] Hirschberg,D.S., "Algorithms for the Longest Common Subsequence Problem", J.ACM, Vol. 24, No. 4, pp. 664-675, Oct. 1977.

[4] Mayne,A. and James,E.B., "Information Compression by Factorizing Common Strings", The Computer Journal, Vol. 18, May 1975.

[5] Weiner,P., "Linear Pattern Matching Algorithms", 14th SWAT, 1973.