漸増的最長共通部分列問題

Incremental Longest Common Subsequence Problem

[1, †)]石田祐介，[2,*)]稲永俊介，[3)]篠原　歩，[4,5)]竹田正幸
1)九州大学大学院システム情報科学府, 2) 日本学術振興会特別研究員,
3) 東北大学大学院情報科学研究科
4) 九州大学大学院システム情報科学研究院，5) SORST, JST
[1)]Yusuke Ishida，[2)]Shunsuke Inenaga，[3)]Ayumi Shinohara，[4,5)]Masayuki Takeda
1)Department of Informatics, Kyushu University, 2) Research Fellow of Japan Society for the Promotion of Science, 3) GSIS, Tohoku University
4) Department of Informatics, Kyushu University，5) SORST, JST
*Email: shunsuke.inenaga@i.kyushu-u.ac.jp

キーワード：最長共通部分列，文字列処理，アルゴリズム
Keywords: Longest Common Subsequence, String Processing, and Algorithm

1. はじめに

有限アルファベットΣ上の文字列 $A=a_1a_2...a_n$ に対し，$|A|$はその長さ n を表し，$A[i]=a_i$ とする．列 $A[i_1]A[i_2]...A[i_k]$は，　$1\le i_1<i_2<...<i_k \le n$ のとき，A の長さ k の**部分列**という．文字列 A と B に共通して現れる部分列の中で，長さが最長のものを**最長共通部分列**（*Longest Common Subsequence*）とよび，その長さをLCS(*A*,*B*)と表す．LCS(*A*,*B*)は，動的計画法を用いて $O(nm)$時間で求められることが知られている[1]. ここに n と m はそれぞれ A と B の長さを表す．本稿では，LCS(*A*,*B*)が既に求まっているときに，A や B の前または後に文字 c が追加された文字列に対して，LCS(*cA*,*B*), LCS(*A*,*cB*), LCS(*Ac*,*B*), LCS(*A*,*Bc*)を効率よく求める問題を考える．

2. 動的計画法

文字列 A と B に対し$(m+1)\times(n+1)$の二次元配列 DP を次のように帰納的に定義する．

$$DP[i,j] = \begin{cases} 0 & i=0 \text{ or } j=0 \\ \max\{DP[i-1,j], DP[i,j-1]\} & A[j]\neq B[i] \\ DP[i-1,j-1]+1 & A[j]=B[i] \end{cases}$$

このとき，$DP[m,n]$=LCS(*A*,*B*)であり，これは $O(mn)$時間と領域で計算できる（図 1 上）．

3. Landau らのアルゴリズム

Landau ら[3]は，L = LCS(*A*,*B*)が計算済みであるとき, 任意の文字 c に対して LCS(*cA*,*B*)を効率よく計算するアルゴリズムを提案した．彼らのアルゴリズムは, DP 全体を再計算するのではなく，次のように定義される分割点の集合 P のみを計算するため，従来の手法より高速かつ省領域で動作する．$A[j]=B[i]$のとき，対(i,j)を A と B の間の**一致点**という．また，$DP[i,j]=DP[i-1,j]+1$ が成り立つとき，対(i,j)を DP の**分割点**とよぶ．DP のすべての分割点からなる集合を P で表す．$DP[i,j]=v$ となる分割点$(i,j)\in P$ に対し，$P[v,j]=i$ と定義する．すなわち，$P[v,j]$は DP の第 j 列において，初めて v になる行を表す（図 1）．

定理 1 (Landau ら[3]). L = LCS(*A*,*B*)が計算済みであるとき, 任意の文字 c に対して, LCS(*cA*,*B*)を $O(L)$時間で計算できる．

		a	d	b	d	c	d
b		0	0	**1**	1	1	1
c		0	0	1	1	**2**	2
b		0	0	**1**	1	2	2
d		0	**1**	1	**2**	2	**3**

	b	a	d	b	d	c	d
b	**1**	1	1	**1**	1	1	1
c	1	1	1	1	1	**2**	2
b	**1**	1	1	**2**	2	2	2
d	1	1	**2**	2	**3**	3	**3**

図 1　A=adbdcd, B=bcbd のときの DP（上図）と，A の先頭に b を追加したときの DP^{Ah}（下図）．簡単のため，$i>0, j>0$ の部分のみ記載する．一致点を**太字**で表し，分割点を □ で囲み，DP^{Ah} で新たに分割点になったところを灰色で塗っている．

†現　東日本電信電話株式会社.

4. LCS(*A*, *cB*)の計算

LCS(*A*,*B*)が計算済みであるとし，そのときのDP表を*DP*とし，分割点の集合を*P*とおく．*B*の先頭に文字*c*が追加されたときのLCS(*A*,*cB*)に対するDP表をDP^{Bh}とし，分割点の集合をP^{Bh}とする．この変化に着目するために，$k=\min\{j \mid A[j] = c\}$とする．*k*は$DP^{Bh}$の最初の行のなかの一致点を表す列である．

補題 1. 任意の列$j \geq k$に対し，次式を満たす行E_jが存在する．

$$DP^{Bh}[i,j]=\begin{cases} DP[i,j]+1 & (i < E_j), \\ DP[i,j] & (i \geq E_j). \end{cases}$$

補題1より，*P*をP^{Bh}に更新する際，各列$j \geq k$で行E_jのみに着目すればよいことがわかる．ここで(E_j,j)は*P*をP^{Bh}に更新する際に，列*j*において削除される分割点である．次に示す補題2と補題3より，削除すべき分割点(E_j,j)を効率よく求めることができる．

補題 2. 分割点集合*P*をP^{Bh}に更新する際に*j*-1列および*j*列で削除される分割点をそれぞれ$(E_{j-1},j-1)$および(E_j,j)とし，$DP^{Bh}[E_{j-1},j-1]=v$とする．このとき，$E_{j-1} \leq E_j \leq P^{Bh}[v+1,j-1]$が成り立つ．（図2参照）．

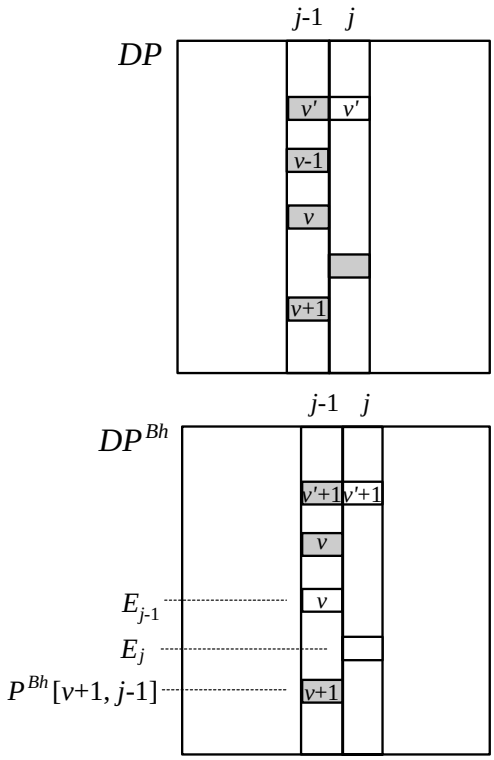

図2. *P*をP^{Bh}に更新する際に第*j*列の分割点E_jが存在する範囲. 灰色で表した点が分割点を表す.

		動的計画法	KP法の応用	提案手法
計算時間	LCS(*cA*,*B*)	$O(mn)$	$O(m+n)$	$O(L)$
	LCS(*Ac*,*B*)	$O(m)$	$O(m)$	$O(L)$
	LCS(*A*,*cB*)	$O(mn)$	$O(m+n)$	$O(n)$
	LCS(*A*,*Bc*)	$O(n)$	$O(n)$	$O(n)$
作業領域		$O(mn)$	$O(mn)$	$O(nL+m)$

表1 計算量の比較. KP法は文献[2]を応用したものである．$|A|=n$, $|B|=m$であり，$L=\mathrm{LCS}(A,B) \leq \min(n,m)$が常に成り立つことに注意.

補題 3. 分割点集合*P*をP^{Bh}に更新する際に*j*-1および*j*列で削除される分割点をそれぞれ$(E_{j-1},j-1)$および(E_j,j)とし，$DP^{Bh}[E_{j-1},j-1]=v$とする．このとき，$P^{Bh}[v,j-1]<x \leq E_{j-1}$を満たす一致点$(x,j)$が存在するときは，$E_j=E_{j-1}$，そうでなければ$E_j=P[v+1,j]$が成り立つ．

補題 4. $k=\min\{j \mid A[j] = c\}$とする．*DP*の*j*列に分割点が存在するときはE_k=null，そうでなければ$E_k=P[1,k]$である．

以上をまとめて，次の定理が得られる．

定理 1. *L*=LCS(*A*,*B*)が計算済みであるとき，任意の文字*c*に対して，LCS(*A*,*cB*)を$O(n)$時間で計算できる．

5. おわりに

表1に本論文の提案手法と従来手法の計算時間の比較結果を示す．第3章と第4章で述べたとおり，LCS(*A*,*B*)が計算済みであるとき，LCS(*cA*,*B*)とLCS(*A*,*cB*)はそれぞれ$O(L)$時間と$O(n)$時間で計算できる．LCS(*Ac*,*B*)とLCS(*A*,*Bc*)は，それぞれ$O(L)$時間と$O(n)$時間で容易に計算できる．詳細は割愛する．

参考文献

[1] Apostolico, A., “String editing and longest common subsequences”, In Handbook of Formal Languages, Vol. 2, Springer-Verlag, 1997, pp. 361-398.

[2] Kim, S.R. and Park, K. “A dynamic edit distance table”, In Proc. 11th Ann. Symp. on Combinatorial Pattern Matching, LNCS 1848, Springer-Verlag, 2000, pp. 60-68.

[3] Landau, G.M., Myers, E., and Ziv-Ukelson, M., “Two algorithms for LCS consecutive suffix alignment”, In Proc. 15th Ann. Simp. on Combinatorial Pattern Matching, LNCS 3109, 2004, pp. 173-193.