長い可変長文字列の挿入操作に関する一考察†

野下浩平†† 角田博保††

本稿では，非常に長い可変長文字列を表現するデータ構造とその操作法を提示して，計算時間と記憶領域の関係を考察する．汎用のテキストエディタや文字列処理システムでは，可変長の文字列を表現する方法として，文字列を一次元配列にべたづめする方法を基本にすることがほぼ定説になっている．そして，非常に長い文字列の挿入や削除の変形操作にも対応するために，配列を基本にして，リストを併用する方法が使われている．本稿では，このデータ構造と挿入と削除の操作について再考し，データ構造として環状配列の環状リストを使い，文字列の挿入操作を行う部分を（操作する部分の近辺しか触わらないという意味で）局所的に限定する方法を提示する．この方法の特徴として，(1)挿入操作が文字列全体の大きさに依存しない時間で実行できること，それと同時に，(2)文字列の表現に要する記憶領域の大きさが（どの時点でも）絶対的に必要な大きさの一定数倍以内ですますことができること，さらに，(3)局所的であるという条件の下で，これより記憶領域が少なくてすむ方法はないという意味で最適なものであることを示す．またこの方法を一般化する．最後に，この方法の実用的意味を議論して，リアルタイム的応用や探索操作の頻度が高い応用に向いていることを指摘する．

1. はじめに

本稿では，非常に長い可変長文字列を表現するデータ構造とそれに対する挿入と削除のアルゴリズムを提示し，（最悪の場合でも）計算時間と記憶領域ともに適当な大きさに限定できることを示す．

文字や 16 進数を扱う汎用のテキストエディタや文字列処理システムの設計では，非常に長い可変長の文字列を表現するデータ構造とその操作用アルゴリズムを定めることが必要になるが，この問題は，古くから詳しく調べられている（例えば文献 2)〜4)，7)）．従来の研究によれば，汎用システムでの文字列の表現は，文字列を一次元配列に“べたづめ”する方法を基本にすることがほぼ定説になっている[4]．さらに Boyer-Moore 法などの文字列探索アルゴリズム[1]は，配列にべたづめする表現を前提にしている．また，いくつかのテキストエディタで見られるように，文字列が行の列からなると仮定できる場合，各行を配列にべたづめして，行の集りをリストでつなぐ方法が使われているが，これは配列表現を基本にしてリストを併用する方法である．最近は，主記憶領域が 10 メガバイト以上であることが普通になり，10 メガバイトを超える辞書やログファイルなど非常に長い文字列が処理の対象になる．それで，素朴なべたづめ方式の配列では，文字列の挿入や削除の変形操作に際して，配列内で非常に長い文字列移動が必要になり，時間と記憶領域に関する非能率さが問題になっており，そのために配列とリストの併用する方法が必要になっている．

このような最近の動向も考慮して，本稿では，この古い問題を次の見方から再考する．従来の研究では，（自明な理論的考察以外は）文字列の操作時間と記憶領域の使用効率の間の関係を実験的あるいは経験的な評価によって示すことが中心になっている．それに対してここでは，長い文字列の変形操作のためのデータ構造とアルゴリズムに対して，理論的な観点から時間的あるいは空間的な能率を考察しよう．

本稿では，非常に長い可変長の文字列を表現するデータ構造として，環状配列の環状リストによるものを提示し，挿入（および削除）の操作用アルゴリズムを工夫することにより，時間と記憶領域の大きさに関するある関係を導く．詳しくは，挿入操作を行う部分を（操作する部分の近辺しか触わらないという意味で）局所的に限定することで，挿入操作が文字列全体の大きさに依存しない時間で実行でき，それと同時に，文字列の表現に要する記憶領域の大きさが（どの時点でも）絶対的に必要な大きさの定数倍以下ですますことができる．さらに，この方法は，局所的であるという条件の下で，これより記憶領域が少なくてすむアルゴリズムは存在しないという意味で“最適”なものであることを証明する．この方法を一般化して，時間と記憶領域の大きさの関係を表す一般式も示す．なお，本稿の方法の実用的な意味と他の方法との比較については，後の考察の章で議論する．

† On Inserting Strings into a Long String by KOHEI NOSHITA and HIROYASU KAKUDA (Department of Computer Science, Faculty of Electro-Communications, The University of Electro-Communications).

†† 電気通信大学電気通信学部情報工学科

2. 文字列の表現と挿入アルゴリズム

まず長さnの文字列を表現するデータ構造を示す．これは，環状の配列を環状のリストでつなぐものである．レコード(ページに相当)の並びを両方向リストで表現し，先頭のレコードと末尾のレコードをつなぎ環状に作る．一般的な様子を図1に示す．なお，レコードの割付けなどの記憶領域全体の動的管理法は，普通の方法で実現するものとする（文献5）などを参照)．

1つのレコードは，大きさsの一次元配列Aであり，文字が最大s個入る．sはかなり大きい一定値とする（例えば$s=128$，文字自身は1または2バイトである)．レコードに実際に入っている文字の数はhで表すことにして，h個の文字からなる部分文字列を環状に表現する．つまり，文字列を(sを法として)添字の連続した領域に並べる．先頭の添字をp，末尾の添字$+1$ (mod s) をqで表す．ここで，$1 \leqq h \leqq s$である．例えば，文字列 "ABCD$" は図2のようになる．

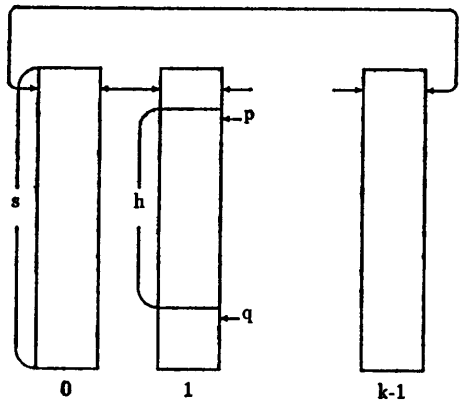


図1 環状配列の環状リスト（s：レコードの大きさ，h：レコード内文字数，p：先頭の添字，q：末尾の添字+1，k：レコード数)

Fig. 1 A circular list of circular arrays (s: record size, h: string length, p: index of the first character, q: index of the last character +1, k: the number of records).

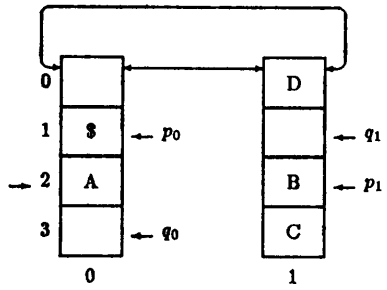


図2 文字列 "ABCD$" の表現例 ($s=4$)

Fig. 2 Representation of string "ABCD$" ($s=4$).

レコードの表現には，その他に，リンク用のポインタ2つ，数h，p，qをいれる場所が少なくとも必要である．両方向リストをなすレコードに対して，その先頭から順に0から$k-1$まで番号を振り，レコードjにある文字数をh_jとすると，当然 $n=h_0+h_1+\cdots+h_{k-1}$ が成り立つ．文字列全体が環状に表現されているので，文字列の先頭と末尾が同一のレコード内で隣合うことが普通である．なお，空の文字列は，ダミーの文字を1つ使って長さ1の文字列として表現する．

文字列に対する基本操作は，いろいろな種類があるが，当面の議論に必要な基本操作である "挿入" だけを取り上げる．"削除" は挿入の逆操作として簡単に実現できるが，後でも説明する．

次に，挿入アルゴリズムを説明する．ここでは，上記のように表現された長さnの文字列 $\alpha=a_1a_2\cdots a_n$ に対して，挿入位置（レコードの添字jとその中の文字の添字x）が与えられて，その直前に長さmの新しい文字 $\beta=b_1b_2\cdots b_m$ を挿入する．ここで，文字列βは，入力バッファの中の文字列であるとか，あるいは，同じように表現された文字列の部分文字列から複製して作るものであったりする．

まず，文字列の挿入が1つのレコード内だけで実行できる場合の挿入手続き pinsert を用意する．記述の簡潔さのため，$A[p:d]$ という表現でレコード内の添字pではじまる文字d個分の領域を表す．この手続きの考え方は，挿入に際して移動する文字数を（最悪の場合でも）高々 $h_j/2$ に抑えるものである．つまり，レコード内の文字列は挿入位置xによって $A[p:(x-p+s) \bmod s]$ と $A[x:(q-x+s) \bmod s]$ に2分される．それぞれの長さを比較して短い方を移動する．

手続き pinsert(j, x, β):

{ここで，jはレコードの番号，xはレコード内の添字，βは挿入すべき文字列である．ただし，βの長さ $m=|\beta|$ は，$m+h_j \leqq s$ を満たすものとする}

```
if (q+s-x) mod s≧h_j/2 then
  begin A[p:(x-p+s) mod s] を m 文字分，先
        頭の方に移動する；p:=(p-m+s) mod s；
        A[(x-m+s) mod s:m] に β をコピーす
        る
  end
else
  begin A[x:(q-x+s) mod s] を m 文字分，末
        尾の方に移動する；q:=(q+m) mod s；
```

$A[x:m]$ にβをコピーする

end

この手続き pinsert は，次の insert の説明中に陽に出てこないが，コーディングに際して基本的なルーチンになる．pinsert によって，これから頻繁にでてくるレコード間の文字列の移動が直接関係ない文字を触わらないで実行できるようになる．

これからの説明で，“文字列をレコードの先頭に移動する”という場合，その文字列をそのレコードにある文字列の先頭に移動して，文字列を連接することを意味する．“末尾に移動”も同様である．

一般の挿入アルゴリズム insert を示すために，ここで，文字列αの長さnは（それでレコードの個数kも）十分大きいとする．（実際には$k\geqq 4$とする．$k\leqq 3$の場合は，特別扱いする手続きを作ればよいが，後でも触れる．）

以下では，簡単のため，レコードの大きさsは4の倍数とする．

手続き insert (j, x, β):

{$m=|\beta|$ と定義する．$k\geqq 4$ である．

insert の実行の前後に次の条件 C を満たす．

〈条件C〉 $3s/4\leqq h_j\leqq s$ $(0\leqq j\leqq k-1)$

この条件の意図は，（全体の文字列 α がごく短い場合を除いて）各レコードの記憶領域の使用効率をいつも 75% 以上に保持することである．}

if $m+h_j\leqq s$ **then**

begin pinsert (j, x, β); return **end**

以下では，$m+h_j>s$ とする．すなわち挿入される文字列βがレコードjの中に収容しきれない場合を扱う．

3つの文字列 $A[p:(x+s-p) \bmod s]$, β, $A[x:(q+s-x) \bmod s]$ をこの順に連接してできる文字列をγとよぶ．（注意：実際に連接を実行してγを作るわけでない．理解の容易さのためγを用いるが，実際のコーディングでは，添字の集りでγを表しておいて，移動で必要になる度に，簡単な添字の計算だけで部分文字列の場所を計算する．）

次に，文字列γをレコードに分配して収容する手順を示す．ここで文字列内の文字の並びの順序を保存するように実現していることにも注意されたい．

これから，レコードの添字はkを法として計算するが，混乱のおきない場所では mod k を省略する．

$R_{j-1}=h_{j-1}-3s/4$

$R_{j+1}=h_{j+1}-3s/4$

と定義する．また，

$N=|\gamma|=m+h_j$ $(>s)$

と定義する．次式を満たすようにQとRを求める．

$N=(s/4)Q+R$

ここで，$Q\geqq 4$, $0\leqq R<s/4$ を満たす．

次に，場合(1)～(5)のQの値に応じて，次式を満たすように整数 u, v, w $(\geqq 0)$ を求める．これですべての場合を尽していることに注意されたい．この場合分けの意図については，手順の記述のあとでも詳しく説明する．

$Q=4u+3v+w$

ここで，uは長さsの部分文字列をもつレコードの個数，vは長さ $3s/4$ の部分文字列をもつレコードの個数，wは長さ $s/4$ の半端な部分文字列の個数という意味をもつ．

(1) $Q=4$, $R>0$ ならば $u=0$, $v=1$, $w=1$

(2) $Q=5$ ならば $u=0$, $v=1$, $w=2$

(3) $Q=8$, $R>0$ ならば $u=0$, $v=2$, $w=2$

(4) $Q=8$, $R=0$ ならば $u=2$, $v=0$, $w=0$

(5) その他のQに対しては，$u\geqq 0$, $v\geqq 1$, $w=0$ とする．($Q=3(u+v)+u$ と書けることに注意．実用にはuをなるべく大きくとる．)

for $i:=1$ **to** $u+v$ **do**

$i\leqq u$ ならば $d=s$, $i>u$ ならば $d=3s/4$ とする．文字列γの末尾から長さdの部分文字列を切り取り，それを新たに割り付けたレコードに移動して，そのレコードをレコードjの直後に挿入する．

この繰返し文で，もともとのγの先頭にあった長さ $(s/4)w+R$ の部分文字列がいまの文字列 γ になっている（つまりこの時点で $|\gamma|=(s/4)w+R$).

レコードの $j+1$ 番目以降は，新しい番号でよぶことにする．例えば，もとの A_{j+1} が現在の A_{j+2} になることがある．

上記(1)～(5)の場合のうち，(4)の場合には，γが空であるので，return する（A_j はゴミになるのでシステムへ返す）．（注意：実際のコーディングでは，A_{j+1} の割付けを延ばせば，A_j をゴミにしないように最適化できる．これで A_j の中にすでにγ（の一部）があることも利用できる．以下の return の部分でも同様である．）

場合(5)では，γをレコード A_{j+1} の先頭に移動して，return する（A_j はゴミになるのでシステムへ返す）．ここで，$|\gamma|<s/4$ であり，A_{j+1} には $s/4$ の余裕

があることに注意しよう ($v \geqq 1$).

次に，(1)，(2)，(3)の場合を扱う．各場合に応じて，次のように定義する．

(1) $S=s/2$

(2) $S=s/4$

(3) $S=s/4$，改めて $R_{j+1}=0$ とする．

[場合A] $R_{j-1}+R+R_{j+1} \leqq S$

γの先頭の長さ $s/4-R_{j-1}$ の文字列を切り取り，それを A_{j-1} の末尾に移動する．

[A.1] 残ったγに対して，$|\gamma| \leqq s/4$ であれば，γを A_{j+1} の先頭に移動して，return する (A_j はゴミになるのでシステムへ返す)．

[A.2] 残ったγに対して，$|\gamma| > s/4$ であれば，A_{j+1} の末尾の長さ $s/4-R_{j+1}$ の文字列を切り取り，A_{j+2} の先頭に移動する．γを A_{j+1} の先頭に移動して，return する (A_j はゴミになるのでシステムへ返す)．

[場合B] $R_{j-1}+R+R_{j+1} > S$

A_j に γ を設定する．A_{j-1} の末尾の長さ R_{j-1} の文字列を切り取り，A_j の先頭に移動する．A_j の長さが $3s/4$ を超えたら，return する．そうでなければ，A_{j+1} の先頭から長さ R_{j+1} の文字列を切り取り，A_j の末尾に移動する．A_{j+2} の先頭から R_{j+1} の文字列を切り取り，A_{j+1} の末尾に移動する．そしてreturn する．

以上説明してきた手続き insert は複雑にみえるが(実際のコーディングでは場合分けの作業が面倒である)，次の考察から文字列の挿入が正しく行われるとともに，条件Cを常に満たすことがわかる．

まず，$m+h_j \leqq s$ の場合の正しさは明らかである．それで $m+h_j > s$ の場合をみよう．すなわち，文字列γがレコード A_j からあふれる場合の扱いに注目する．

まず，場合(4)では，γはちょうど長さ s の部分文字列2個に切断できる．場合(5)では，γを切断して，長さRの部分文字列 γ' と，長さがそれぞれ $3s/4$ と s の部分文字列の列にできる．A_{j+1} の長さは $3s/4$ になるので，γ' を A_{j+1} に収容できる．

場合(1)，(2)，(3)では，[場合A，B]の判定の時点では，A_{j-1}，A_{j+1}，A_{j+2} の長さがそれぞれ $3s/4+R_{j-1}$，$3s/4$，$3s/4+R_{j+1}$ になっている．そこで，[場合A]では，(この時点の) γを切断して，A_{j-1}，A_{j+1}，A_{j+2} の空き場所を使って分配する．A_{j+2} は，場合(1)，(2)で空きが $s/4-R_{j+1}$ であり，場合(3)で空きが $s/4$ ある．一方，[場合B]では，γを A_j にまず置いて，その長さの不足分を A_{j-1}，A_{j+1}，A_{j+2} から文字列を移動する．(実際に，各レコードも部分文字列が過不足なく $3s/4$ と s の間におさまり，条件Cを保存することは，各場合をチェックして確かめることができる．)

手続き insert の説明の最後として，$k \leqq 3$ の場合を扱う．挿入によりできる文字列の長さ $n+m$ が $3s$ 以下である場合には，$k \leqq 3$ にできる．一方，$n+m \geqq 3s+1$ の場合には，上記の条件Cを満たす必要があるが，これは常に可能である．なぜなら，すでにみたように，$k \geqq 4$ になると，少なくとも4つの(連続した)レコードの文字列の長さを $3s/4$ から s までの長さに設定できるので，条件Cを満たしつつ，1から s までの任意の長さの部分文字列を収容する余裕ができるからである．(簡単であるので実現の詳細は省略する．)

以上で，挿入手続き insert とその正しさの説明を終るが，削除手続き delete について説明しておこう．削除手続きは，挿入に相補的な手続きであり，手続き insert による条件Cの保存の仕方を逆にたどることによって作ることができる．すなわち，delete は，文字列の先頭の位置を表す添字 i と x，末尾の位置 j と y が与えられて，条件Cを保存するように，文字列を削除すればよい．もっと簡単には，削除する文字列の先頭の文字が属するレコードと末尾の文字が属するレコードに注目して，削除後に各々のレコードの中に残る2つの部分文字列を insert することによって実現できる．これで自然に条件Cが満足する．その他の変形操作として，切断や連結なども必要になるが，挿入や削除を同様に考えれば，条件Cを保存する実現には困難がない．

3. 時間と記憶領域の関係

前章で提示した挿入アルゴリズムに対して，時間と記憶領域の関係を示す．まず，記憶領域の大きさについては，文字列の構成単位の文字の大きさを1として，文字列の長さ n に対する相対的な大きさを評価する．レコードは最大 s 文字を収容する．すでに触れたように，レコードには，ポインタ2個，添字2個，文字数カウンタ1個が少なくとも必要であるが，これらの合計は，s に較べて小さい一定値として，以下の議論では無視することにする．計算時間については，挿入操作のみを取り扱うが，長さ n の文字列 α に長さ m の文字列 β を挿入するものとして，挿入する際の文字

の移動回数で計算時間を評価する．ここでも，ポインタのつけかえ，レコードの割付けなどの時間がかかるが，本稿の範囲内ではいずれも小さい一定値として仮定して差し支えないので，以下の議論では無視する．

次の定理は，時間と記憶領域に関して最悪の場合の上界を与えるものである．

定理 1

十分大きい n に対して，長さ n の文字列 α の表現に要する記憶領域の大きさは，約 $4n/3$ 以内になるとともに，長さ m の文字列 β の挿入に要する時間は，高々 $O(m)$ ですむ．文字の移動回数は，多く見積っても $m+7s/4$ ですむ．ここで，s はレコードの大きさ（定数）である．いいかえれば，挿入される文字1個当りの挿入時間はある定数でおさえられる．

証明：記憶容量については，前章における正しさの説明からわかるように，挿入操作が条件Cを保存することより成り立つことがわかる．それで，次の式が成り立つ．

$$(3s/4)k \leqq n \leqq sk$$

これより，$n/s \leqq k \leqq 4n/(3s)$ となり，最悪の場合でもレコードの個数は約 $k=4n/(3s)$ となる．

計算時間については，操作の局所性に着目する．まず，$m+h_j \leqq s$ の場合，文字の移動回数は，高々 s としてよい．

次に，$N=m+h_j>s$ の場合をみる．文字の移動回数は，場合(4)と(5)で高々 N である．場合(1)，(2)，(3)において移動する文字数の上界は次のとおりである．

[場合A.1] N

[場合A.2] $N+s/4$

[場合B] $N+3s/4$

それで，挿入操作で生じる文字の移動回数の上界は，$N+3s/4 \leqq m+7s/4$ になる．文字の移動以外に要する時間は，一定とみなしてよいので，挿入にかかる計算時間は $O(m+s)$ になる． □

次に，記憶領域の下界を考察しよう．下界を証明するには，計算のモデルをきちんと定義する必要があるが，ここでは，局所的な挿入操作の列による文字列の推移を定めるものとして挿入アルゴリズムを定義する．

第 t 回の挿入操作が行われる直前の文字列 α が次のような部分文字列の列から構成されているとする（α が環状に表現されているが，いまの議論に無関係である）．

$$\alpha=\alpha_0\alpha_1\alpha_2\cdots\alpha_{j-1}\alpha_j\alpha_{j+1}\cdots\alpha_{k-1}$$

ここで，$k \geqq 1$ であり，$1 \leqq |\alpha_i| \leqq s$ であるとする $(0 \leqq i \leqq k-1)$．（厳密には各 α_i は文字列を値とする変数の名前である．）

第 t 回の挿入操作 INSERT(j, x, β) は，文字列 α に対して，添字 j の文字列 α_j の x 番目の文字の直前に文字列 β を挿入するものとする $(1 \leqq x \leqq |\alpha_j|)$．また，挿入の結果できる文字列 α' は，次の構成をしているものとする．

$$\alpha'=\alpha'_0\alpha'_1\alpha'_2\cdots\alpha'_{j-1}\cdots\alpha'_{k'-1}$$

ただし，次が成り立つものとする．

$$\alpha_0=\alpha'_0,\ \alpha_1=\alpha'_1,\ \cdots,\ \alpha_{j-2}=\alpha'_{j-2},$$

$$\alpha_{j+2}=\alpha'_{k'-k+j+2},\ \cdots,\ \alpha_k=\alpha'_{k'},$$

ここで，文字列 α から α' への変化は，“局所的な”操作によって生じることを表している $(\alpha_{j-1}, \alpha_j, \alpha_{j+1}$ 以外は変化しない$)$．初期値は，$\alpha(0)=\alpha_0=$“$”（ダミーの文字1個）とする．

こうして，t 回の挿入操作 INSERT(j_t, x_t, β_t) の列によって，

$$\alpha(0), \alpha(1), \cdots, \alpha(t)$$

が定義される．（計算量理論の言葉でいうと，ここでのアルゴリズムは“オンライン”である．）

第 t 回の挿入の直後の文字列が $\alpha(t)=\alpha_0\alpha_1\alpha_2\alpha_3\cdots\alpha_{k-1}$, $n=|\alpha(t)|$ であれば，記憶領域の使用効率は，1文字当り sk/n になる．

定理 2

十分大きい任意の t に対して，t 回の挿入によってできる文字列 $\alpha(t)$ の長さを n とする時，いかなる局所的な挿入アルゴリズムによっても，最悪の場合，$n/k \leqq (3s+1)/4$ が成り立つ．すなわち，レコードの数 k は少なくとも約 $4n/(3s)$ が必要になる．ここで，s はレコードの大きさ（定数）である．

証明：任意の局所的な挿入アルゴリズムを選んで固定する．これに対して，次の第1回の挿入を考える．

$$\text{INSERT}(0, 1, \beta),\quad |\beta|=3s$$

これによって，$\alpha(1)=\alpha_0\alpha_1\alpha_2\cdots\alpha_{k-1}$ とすると，$k \geqq 4$ になる．2回目以降の挿入の列として，次のようなものを考える．第 $t-1$ 回の挿入でできた文字列を $\alpha(t-1)$ として，

$$\alpha(t-1)=\alpha_0\alpha_1\alpha_2\cdots\alpha_j\cdots\alpha_{k-1}$$

であるとする $(k \geqq 4)$．いま，$0 \leqq j \leqq k-1$ に対して $|\alpha_{j-1}|+|\alpha_j|+|\alpha_{j+1}|$ が最大になるような添字 j を z とする $(0 \leqq z \leqq k-1)$．（添字は mod k で計算することに注意．）第 t 回の挿入は，

INSERT $(z, 1, \beta)$,

$|\beta|=3s-(|\alpha_{z-1}|+|\alpha_z|+|\alpha_{z+1}|)+1$

とする.

局所的な操作に限定されているどんな挿入アルゴリズムでも，このような t 回の（意地の悪い）挿入が行われると，

$n/k \leqq (3s+1)/4$

が成り立つことを示す．証明は帰納法による．

まず，$t=1$ の時は明らかに成り立つ．一般に第 $t-1$ 回の挿入で上式が成り立っていると仮定する．第 t 回の挿入でできる文字列の長さ n'，レコード数を k' とする．

$n'=n+3s-(|\alpha_{z-1}|+|\alpha_z|+|\alpha_{z+1}|)+1$

$k' \geqq k+1$

である．さて，z の選び方より，

$k(|\alpha_{z-1}|+|\alpha_z|+|\alpha_{z+1}|) \geqq 3n$

が成り立っている．よって，次式をうる．

$n' \leqq n+3s-3n/k+1$

以上より，次が成り立つ（$k \geqq 4$ に注意）．

$n'/k' \leqq (n+3s-3n/k+1)/(k+1)$

$=(n(1-3/k)+3s+1)/(k+1)$

この式の n に仮定の不等式を代入して整理すると，

$n'/k' \leqq ((1-3/k)(3s+1)k/4+3s+1)/(k+1)$

$=(3s+1)/4$

となる．s は十分大きい定数であるので，最悪の場合には，1文字当りの占有記憶領域は少なくとも約 4/3 になることがわかる． □

この定理によって，本稿の挿入アルゴリズムは，局所的な操作に限定するあらゆるアルゴリズムの中で，(最悪の場合) 記憶領域の能率に関して最適のものであることがわかる．

これまでのアルゴリズムの技法と計算量の解析から，次のような一般化ができる．“p-局所的”な挿入とは，挿入の対象になるレコード j に対して，添字が $j-p, j-p+1, \cdots, j-1, j, j+1, \cdots, j+p-1, j+p$ である $(2p+1)$ 個のレコードに操作を許すものとする．$p=1$ の場合が本稿で詳しく調べたものである．上記の2つの定理の一般化として，次の定理を導くことができる．

定理 3

挿入アルゴリズムで p-局所的なものを作ることができて，それによると，長さ n の文字列に長さ m の文字列を挿入するのに，(最悪の場合に) 高々 $O(m)$ 時間で実行できる．それと同時に，文字列の表現に必要なレコードの個数は，高々 $(n/s)(2p+2)/(2p+1)$ で十分である．一方，p-局所的な（オンライン）挿入アルゴリズムはどんなものでも，最悪の場合，約 $(n/s)(2p+2)/(2p+1)$ 個のレコードを必要とする．ここで，s はレコードの大きさ（定数）である．

これより，一般化した p-局所的なアルゴリズムは，定理1と2と同様の意味で，やはり最適であることがわかる．

上界と下界の証明は，定理1と2の証明をそれぞれ一般化すればよい．ここでは，証明の主要部分を説明する．(定理1と2と同程度に詳しく証明することは興味をもった読者に委ねる.)

前半の上界を示す．まず，この上界を実現する p-局所的な挿入アルゴリズムを作る．ここでは，考え方を簡潔に説明するために，文字列の文字の順序を保持するための手順の部分を考慮せず（そのための修正は定理1のアルゴリズムにならえばよい），挿入する文字列を収容するための空き場所を調整する手順を中心に述べる．定理1の条件Cに相当する条件は，次のとおり．

<条件C> $(2p+1)s/(2p+2) \leqq h_i \leqq s$

以下でレコードの大きさ s は $(2p+2)$ の倍数とする．

まず，挿入する文字列 β の長さ m が大きい場合 $(m \geqq s)$ をみる．$m=w*s+m'$ とする $(w \geqq 1, 0 \leqq m' < s)$．この場合は，$w$ 個のレコードを新たに割り付ければ，文字列 β の長さ $w*s$ の部分を収容できる $(m'=0$ ならばそこで終了する)．これより，一般性を失わず，$0<m<s$ の場合を調べればよい．ここで，$(2p+1)s/(2p+2) \leqq m < s$ の場合は，新たなレコード1個を割り付けると β を収容できるので，それで終了する（当然条件Cを満たす）．以上より

$0<m<(2p+1)s/(2p+2)$

と仮定して話を進める．次に用いる記号 Σ は添字 i が添字 $j-p, j-p+1, \cdots, j+p-1, j+p$ にわたって動くとして総和を表す．

[場合1] $m \leqq \Sigma(s-h_i)$

これは，レコード j を中心とする $2p+1$ 個のレコードに β の長さ以上の空きがあることを意味する．それで（新たにレコードを割り付けることなく）β を収容できる．

[場合2] $m > \Sigma(s-h_i)$

文字列 β と，レコード j を中心とする $2p+1$ 個のレコードの中にある文字列で長さ $(2p+1)s/(2p+2)$ を

超える部分とを併せたもの（文字列の集り）をδとよぶ．δの長さをdで表すと，次が成り立つ．

$$d=m+\Sigma(h_i-(2p+1)s/(2p+2))$$

これで次の2つの場合が生じる．

（なお，$d\leqq(2p+1)s/(2p+2)$にはならない．なぜなら $d=(m-\Sigma(s-h_i))+(2p+1)s/(2p+2)>(2p+1)s/(2p+2)$ であるからである．）

[場合 2.1] $(2p+1)s/(2p+2)<d\leqq s$

新しいレコード1個割り付ければ，βを収容できる．ここで，条件Cをみたすために，このレコードには，β以外のδの文字列の長さ$(d-m)$分に相当する文字列を収容する．

[場合 2.2] $d>s$

ここで $s<d<m+s<(2p+1)s/(2p+2)+s$ であるが，この式の最も右側の値は，新しいレコード1個分（大きさs）ともとの$(2p+1)$個のレコードの空き場所と（β以外の）δが占める場所とを併せたものの大きさを表す．すなわち，新しいレコードを1個割り付け，もとの$2p+1$個のレコードを利用すると，（δ全体を収容できるので）βを収容できる．条件Cを満たすために，新しいレコードに長さsの部分文字列を収容する．

以上がアルゴリズムの要点であるが，これで計算時間が$O(m)$になることは，定理1の証明と同様に，文字列βがp-局所的な操作によって挿入されることをみればわかる．（なお，定理1と同様に，この挿入操作をプログラムとして実現する場合は，$O(m+s+p)$時間ですむようにするのがよい．sやpは定数であるが，mの定数係数に含まれないようにできる．）

次は，下界の証明であるが，これも定理2と同様に行う．ここでも重要な点は，任意のp-局所的なアルゴリズムに対して，最悪の場合を生じさせる意地の悪い入力（挿入操作の列）を作ることである．一般に，第t回の挿入は，連続した$(2p+1)$個のレコードのうち，空いている場所の大きさの総和が最大となる部分に，その総和+1の長さの文字列を挿入するものを考える．つまり，INSERT$(z,1,\beta)$において

$$|\beta|=(2p+1)s-(|\alpha_{z-p}|+\cdots+|\alpha_{z+p}|)+1$$

とする．ここで，添字zは$0\leqq j\leqq k-1$に対して，

$|\alpha_{j-p}|+\cdots+|\alpha_{j+p}|$が最大になるような$j$である（添字は mod k で計算する）．このような挿入操作の列が与えられると，定理2と同様に，

$$n/k\leqq((2p+1)s+1)/(2p+2)$$

が成り立つ．証明は帰納法による．なお，その中における式の変形も定理2の証明にならって行えばよい．

4. 考　　察

前章までで，局所的な挿入操作に限定するという枠組みの下で，記憶領域の大きさの上界と下界を示し，それが一致することをみた．その基礎になるデータ構造として，（個々の構成部品はよく知られているが）環状配列の環状リストによる表現法を提示して，文字列処理特有の工夫を示した．本章では，主として実用という観点から考えよう．

本稿の方法を実用にするには，$p=1$で十分であると思われる（$p=2$でもよい）．定理1の評価は最悪の場合であり，平均的な時間と記憶領域の能率ははるかによいと考えられる．それでも，記憶領域が計算中に不足してきた場合，（最悪 $O(n)$ 時間かけて）文字列全体を走査して圧縮することのほかに，pを可変にして増やしていくという方法も考えられる．

次に，“普通の方法”と簡単な比較をする．例として文献7）の方法をとりあげる．これは，レコードからあふれた分の文字列をいれるレコードを新しく割り付けるという簡単なものである．（なお文献7）は環状配列も環状リストも使っていないし，計算量の上下界にも触れていない．）この方法は，最悪の場合に記憶領域の能率が極端に悪くなる（平均的にもあまりよくない）ので，新しいレコードの割付け要求が頻繁におきる．例えば，先頭のレコードにある部分文字列の真ん中辺りに1文字挿入することを繰り返す場合である．割付け用のレコードが尽きると，文字列の圧縮を行うのであろうが（ゴミも回収する），この圧縮には時間がかかる．それで，圧縮を頻繁に行えない状況では，（レコードの個数が多くなり）1レコード当りの記憶領域の使用効率が相対的に悪くなる．さらにBoyer-Moore法などの高速探索の効果が小さくなる．この議論と対照させると，本稿の方法は，リアルタイム処理が重要である状況（つまり文字列操作の単位でみて最悪の場合の能率が悪いと困るような応用）や探索操作がよく現れる状況などに向いていることがわかる．Boyer-Moore 法もわずかの修正で利用できる．もちろん，実際のシステムへの応用経験にもとづく評価は今後の課題である．なお，これまで考えてきた状況とまったく異なるが，レコードが二次記憶装置のページに対応し，挿入と削除を許す文字列を実現する場合，（途中での圧縮操作が禁止的になるので）本稿の方法がうまく応用できる可能性がある．

本稿の方法は，最悪の場合の性能を保証するものであるが，平均的な場合の理論解析は，今後の研究に任せる．ここで，探索用の B-木 (2-3 木)[6] との類似点をみておこう．B-木 (2-3 木で $s=3$ に対応) では，要素の挿入が1個に限り ($|\beta|=1$ に対応)，レコードのあふれもレコードがちょうど一杯になっている時に限る．それで記憶領域の上界が $p=0$ に対応して2倍 (普通の B-木)，$p=1$ に対応して 4/3 倍になるようにすることはごく簡単にできる．(本稿の定理1は，出発となる考え方が類似しているが，工夫のしどころなど技術的詳細が異なる．) 一方，一般の大きい p に対応するものを実現するのは (木構造のため) 簡単でない．平均の場合については，B-木において，データのランダムな挿入に対するレコード (実は最低レベルの節点) の平均個数を求める問題は，本稿の問題から解釈するとごく特殊な場合に対応するが，文献8) や9) などの研究にかかわらず，難しく完全には解かれていない．それで，文字列の場合，平均的な場合の定義が上手にできたとしても，理論解析はさらに難しいと予想される．

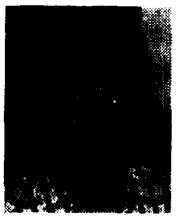
参考文献

1) Boyer, B. S. and Moore, J. S.: A Fast String Searching Algorithm, *Comm. ACM*, Vol. 20, No. 10, pp. 762-772 (1977).
2) Dewar, R. B. K. and McCann, A. P.: Macro SPITBOL—a SNOBOL 4 Compiler, *Softw. Pract. Exper.*, Vol. 7, No. 1, pp. 95-113 (1977).
3) Hanson, D. R.: A Portable Storage Management System for the Icon Programming Language, *Softw. Pract. Exper.*, Vol. 10, No. 6, pp. 489-500 (1980).
4) 角田博保：SNOBOL 3 言語のべたづめ方式処理系とその評価，情報処理学会論文誌，Vol. 22, No. 5, pp. 472-478 (1981).
5) Knuth, D. E.: *The Art of Computer Programming, Vol. 1 (Fundamental Algorithms)*, 2nd ed., Addison-Wesley (1973).
6) Knuth, D. E.: *The Art of Computer Programming, Vol. 3 (Sorting and Searching)*, 2nd print., Addison-Wesley (1975).
7) Pike, B.: The Text Editor Sam, *Softw. Pract. Exper.*, Vol. 17, No. 11, pp. 813-845 (1987).
8) Wright, W. E.: Some Average Performance Measures for the B-tree, *Acta Inf.*, Vol. 21, No. 6, pp. 541-557 (1985).
9) Yao, A. A.: On Random 2-3 Trees, *Acta Inf.*, Vol. 9, No. 2, pp. 159-170 (1978).

(平成2年8月13日受付)
(平成3年2月12日採録)

野下 浩平 (正会員)

1943年生．東京大学工学部計数工学科卒業．現職：電気通信大学情報工学科教授．工学博士 (東京工業大学)．専門：アルゴリズム解析，組合せゲームの理論と実験．ACM, EATCS, CSA 等の各会員．



角田 博保 (正会員)

昭和25年生．同49年東京工業大学理学部情報科学科卒業．同51年同大学院博士課程修了．同57年同大学院博士課程修了．同年電気通信大学計算機科学科助手，平成2年同大学情報工学科講師．理学博士．文字列処理，プログラミング方法論，ヒューマンインタフェース等に興味を持つ．ACM, 日本ソフトウェア科学会各会員．