LZW圧縮テキストに対する高速文字列照合アルゴリズム

4C－7

喜田 拓也　竹田 正幸　篠原 歩
九州大学大学院システム情報科学研究科

1. はじめに

圧縮されたテキストに対し文字列照合を行なう場合，一時的に展開したのちに，一般的な文字列照合アルゴリズムを用いるという方法が考えられる．この方法は単純ではあるが，圧縮テキストとその展開後のテキストの両方を走査しなければならないので時間がかかる．Amirら[2]は，LZW符号により圧縮されたテキストを展開することなしに，文字列照合を行なうアルゴリズムを提案した．また著者らは，彼らのアルゴリズムをもとに，Aho-Corasickの文字列照合機械を利用することで，複数パターンのすべての出現位置を出力するアルゴリズムを開発した[4]．本稿では，Abrahamson[1]の文字列照合アルゴリズムを利用し，さらなる高速化と高機能化を図ったLZW圧縮テキストに対する文字列照合アルゴリズムを提案する．提案アルゴリズムは$O(m)$時間の前処理を必要とし，$O(N+r)$時間でテキストを走査する．ここでm, N, rはそれぞれ，パターン長，圧縮テキスト長，パターンの出現回数である．

2. 一般化された文字列照合問題

Σを文字の有限集合とする．文字列ξを構成する文字の数をξの長さといい$|\xi|$で表す．文字列$\xi = a_1 a_2 \ldots a_n$のi番目の文字a_iを$\xi[i]$で表し，ξの部分列$a_i a_{i+1} \ldots a_j$を$\xi[i:j]$で表す．

Abrahamson[1]は，次のように一般化された文字列照合問題を提案した．$\Delta = \{X \subseteq \Sigma \mid \emptyset \neq X\}$としたとき，

パターン$\mathcal{P} = X_1 \ldots X_m$ $(X_i \in \Delta)$と
テキスト$\mathcal{T} = \mathcal{T}[1:n]$
が与えられたとき，$\mathcal{T}[j:j+m-1] \in \mathcal{P}$となるような$j$をすべて求めよ．

このようなjが一つでも存在するとき，テキスト$\mathcal{T}$中にパターン$\mathcal{P}$が出現するといい，このjをパターンの出現位置と呼ぶ．また，Abrahamsonは同時にこの問題を解く以下のアルゴリズムを示した．整数pと整数の集合Sに対して$S \oplus p = \{i+p \mid i \in S\}$とおく．$1 \leq k \leq n$に対して

$$R_k = \{1 \leq i \leq m \mid X_1 \ldots X_i \ni \mathcal{T}[k-i+1:k]\}$$

と定義する．また，任意の$a \in \Sigma$について，

$$M(a) = \{1 \leq i \leq m \mid X_i \ni a\}$$

とおく．さらに関数$f(S,a)$を$f(S,a) = ((S \oplus 1) \cup \{1\}) \cap M(a)$と定義しておくと，このとき各$R_k$は

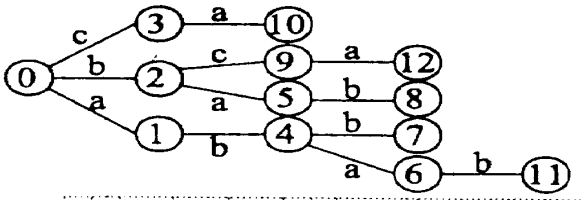
1) $R_0 = \emptyset$
2) $R_{k+1} = f(R_k, \mathcal{T}[k+1])$ $(k \geq 0)$

によって求めることができる．アルゴリズムはテキストの文字を先頭から1文字ずつ読みながら，順次R_kを計算していく．もし$R_k \ni m$ならば$\mathcal{T}[k-m+1:k] \in \mathcal{P}$である．すなわち$k-m+1$がパターンの出現位置である．整数の集合を$m$ビットのビットベクトルで表現すると，関数$f$はビットシフトとAnd,Orの論理演算によって定数時間で高速に求めることができる．またこのアルゴリズムには数多くの変種が開発されており，ミスマッチや複数パターンに対応したもの[3]や，近似文字列照合が可能なもの[5]が提案されている．

3. LZW圧縮法

LZW圧縮されたテキストは整数の列になっている．各整数jは辞書木とよばれる木構造のj番目のノードに対応している．辞書木の各ノードは，木のルートからそのノードまでの枝にふられている文字の連接を表している．この辞書木のノードが示す文字列の集合をDと表し，辞書と呼ぶ．辞書木は，テキストを圧縮する際に適応的に構築される．また圧縮テキストを展開する際にも同じ辞書木を復元することが可能なので，辞書の情報を圧縮テキストに埋め込んでおく必要がない．図1は$\Sigma = \{a,b,c\}, \mathcal{T} = ababbabcababcabab$のときの辞書木を示している．例えば，この図の圧縮テキストにおける'9'は，文字列bcを表している．よって以降では，文字列uとそれを示す整数とを同一視する．

LZW圧縮テキストを展開するには，もとのテキストの長さに比例した計算時間が必要である．しかし，もとのテキストが必要ではなく，辞書木だけを構築するだけならば，その構築時間は$O(N)$時間ですむ（図2）．ここでNは圧縮テキストの長さである．



T = a b a b a b b a b c a b a b c a b a b
Z = 1, 2, 4, 4, 5, 2, 3, 6, 9, 11

図1 辞書木．

Pattern Matching Algorithm in LZW Compressed Text
Takuya Kida, Masayuki Takeda, Ayumi Shinohara
Department of Informatics, Kyushu University, Fukuoka, 812-8581 Japan

Input. LZW 圧縮テキスト $Z = u_1 u_2 \ldots u_N$.
Output. 辞書木 D.
begin
  $D := \Sigma$;
  for $i := 1$ **to** $N-1$ **do begin**
    if $u_{i+1} \leq |D|$ **then**
      a を u_{i+1} の先頭の文字とする.
    else
      a を u_i の先頭の文字とする;
    $D := D \cup \{u_i \cdot a\}$
  end
end.

図 2 圧縮テキストからの辞書木構築アルゴリズム.

4. 提案アルゴリズム

我々の目的は Abrahamson のアルゴリズムを, LZW 圧縮テキストに対して使用できるように拡張することである. LZW 圧縮テキストの 1 文字は辞書木の番号を示す整数であり, 辞書 D の文字列に対応することは既に述べた. まず, 任意の $v \in D$ について,

$$M^+(v) = \left\{1 \leq i \leq m \;\middle|\; v \in \mathcal{P}[i-|v|+1 : i]\right\}$$

と定義する. ただしこの式では $\mathcal{P}[i] = \Sigma \;\; (i \leq 0)$ としておく. この $M^+(v)$ を計算するために次の関係を用いる. 任意の $v \in D$, $a \in \Sigma$ について,

1) $M^+(a) = M(a)$,
2) $M^+(v \cdot a) = f(M^+(v), a)$

である. すなわち新たにノード u が辞書木に付加される時には, u の親ノード v の $M^+(v)$ から $O(1)$ 時間で $M^+(u)$ を計算することができる.

いま圧縮テキストが $Z = u_1 u_2 \ldots u_N$ であるとする. このとき, $1 \leq \ell \leq N$ について,

1) $R_0 = \emptyset$,
2) $R_{k+|u_\ell|} = ((R_k \oplus |u_\ell|) \cup \{1, 2, \ldots, |u_\ell|\}) \cap M^+(u_\ell)$

より R_k をとびとびに求めることができる. しかしながら, このようにして求めた R_k からはパターンの出現を検知できない場合がある. そこで次のような出力関数を用意する.

$$Output(R_k, u_\ell) = \left\{1 \leq i \leq |u_\ell| \;\middle|\; R_{k+i} \ni m\right\}$$

するとこのとき, $\ell' = \Sigma_{j=1}^{\ell-1} |u_j|$ とおくと, パターンの出現位置は $\ell' + p - m + 1 \;\; (p \in Output(R_k, u_\ell))$ である.

任意の $u \in D$ について, $U(u) = \left\{1 \leq i \leq |u| \;\middle|\; u[1:i] \in \mathcal{P}[m-i+1 : m]\right\}$ とする. また $\tilde{u}$ を $U(u)$ の要素で一番大きな整数とし, 整数の集合 S に対して $m \ominus S = \left\{m - i \;\middle|\; i \in S\right\}$ とする. このとき, 次の補題が成り立つ.

補題. $A(u) = \left\{\tilde{u} \leq i \leq |u| \;\middle|\; u[i-m+1 : i] \in \mathcal{P}\right\}$ とおくと. このとき $Output(R_k, u_\ell)$ は

$$Output(R_k, u_\ell) = ((m \ominus R_k) \cap U(u_\ell)) \cup A(u_\ell)$$

と書くことができる. □

したがって, LZW 圧縮テキストに対する文字列照合アルゴリズムは図 3 のように要約できる.

Input. LZW 圧縮テキスト $u_1 u_2 \ldots u_N$.
Output. すべてのパターンの出現位置.
begin
  $\ell' := 0$, $k := 0$, $R := \emptyset$;
  for $\ell := 1$ **to** N **do begin**
    UpdateDictionaryTrie(u_ℓ);
    for each $p \in Output(R, u_\ell)$ **do**
      $\ell' + p - m + 1$ にパターンが出現したと報告;
    $R = ((R + |u_\ell|) \cup \{1, 2, \ldots, |u_\ell|\}) \cap M^+(u_\ell)$;
    $\ell' := \ell' + |u_i|$
  end
end.

procedure *UpdateDictionaryTrie*(u)
begin
  D に新しいノード $u = v \cdot a$ を加える;
  $M^+(u) = f(M^+(v), a)$;
  if $M^+(u) \ni m$ **then**
    if $|u| \leq m$ **then** $U(u) := U(v) \cup \{|u|\}$
    else $A(u) := A(v) \cup \{|u|\}$
  else $U(u) := U(v)$, $A(u) := A(v)$
end;

図 3 文字列照合アルゴリズム.

5. 実験

提案アルゴリズムがどの程度高速なのかを知るために, アルゴリズムを実際に実装し比較実験を行った. 図 4 がその実験結果である.

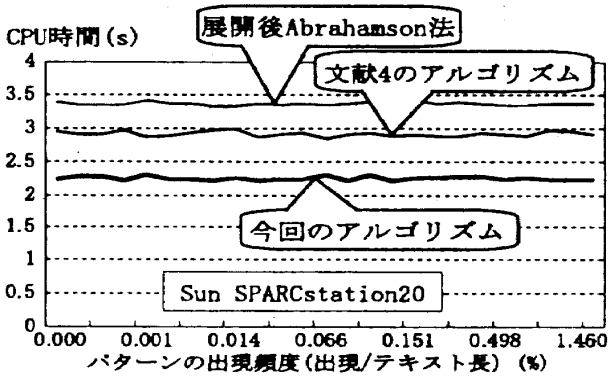


図 4 実験結果 (テキスト約 6.5Mb/圧縮時約 3.4Mb).

参考文献

1) K. Abrahamson. Generalized string matching. *SIAM J. Comput.*, 16(6):1039–1051, December 1987.
2) A. Amir, G. Benson, and M. Farach. Let sleeping files lie: Pattern matching in Z-compressed files. *Computer and System Sciences*, 52(23):299–307, 1996.
3) R. Baeza-Yaltes and G. H. Gonnet. A new approach to text searching. *Communications of the ACM*, 35(10):74–82, October 1992.
4) T. Kida, M. Takeda, A. Shinohara, M. Miyazaki, and S. Arikawa. Multiple pattern matching in LZW compressed text. In J. A. Atorer and M. Cohn eds., *Proceedings of Date Compression Conference '98*, pp. 103–112. IEEE Computer Society, March 1998.
5) S. Wu and U. Manber. Fast text searching allowing errors. *Communications of the ACM*, 35(10):83–91, October 1992.