Vol. 26 No. 6 情報処理学会論文誌 Nov. 1985

ショートノート

連想記憶に基づくパイプライン型文字列検索アルゴリズム†*

板野肯三†† 佐々木日出美†††** 山形朝義††††

連想記憶上に実現された文字列テーブルとパイプライン方式のハードウェア検索機構に基づいて，文字列を高速に検索するアルゴリズムを開発し，実際に UNIX 中の文字列データを使用して評価を行った．本方式ではテーブル中の文字列を1文字ごとにリンクしておき文字列中の文字を最初から1文字走査するごとに文字列テーブルを連想検索し，最後の文字を走査したときに文字列全体の検索が終了する．文字列はテーブル中で木構造で表現され先頭からの部分文字列が共有されるので，可変長の文字列が取り扱えるだけでなく，必要なテーブルのサイズを節約することができる．

1. まえがき

文字列の検索操作はコンパイラなどの言語処理系におけるシンボルテーブル[1),2)]だけでなく，LISP などにおける記号処理[3)]や文書処理[4)]など多くの分野において基本的に重要な役割を果たしている．文字列は，一般に計算機システムと外界とのインタフェース時のみならず，内部での表現や処理においても文字ごとの逐次的形式をとるので，操作は文字単位が基本となる．したがって，文字列の操作時には文字単位のパイプライン方式が有効なアーキテクチャと考えられる[5)]．本論文では，連想記憶と高速の文字単位パイプラインアーキテクチャ[6)]を前提として，この上で動作するパイプライン型の文字列検索アルゴリズムについて提案し，実際に UNIX[7)]において使用されている文字列データを用いてアルゴリズムの評価を行った[8)]．

2. パイプライン型文字列検索アルゴリズム

パイプライン方式の検索では，文字列を構成する文字を最初から一文字ずつ検索ユニットに送り，最後の文字を送りこんだとき，検索が終了する．検索ユニットは基本的には連想記憶で構成される文字列テーブルと検索制御部で構成されていて，文字列テーブル中には検索の対象となる文字列が格納されている．ここではまず，文字列テーブルの構造とテーブル中での文字列の表現形式を示し，次に検索アルゴリズムの説明をする．

2.1 文字列テーブルの構造

図1に文字列テーブルの基本的な構造を示す．テーブルの各エントリは1文字分の文字コードを格納するフィールドとポインタを格納するフィールドで構成されている．テーブルの第0エントリ（インデックス0）は文字列を終端するために用いられるので通常の文字の格納はできず，文字コードとして存在しない値（ここでは 255）を含んでいる．文字コードフィールドが0であればそのエントリは未使用であり，0以外の値であるときは，そのエントリは使用されていることを表す．

2.2 文字列の表現形式

文字列の表現形式を図2に示す．文字列中の各文字はポインタで逆向きにリンクされていて，最初の文字のリンクは終端されている．もしも文字列の最初から始まる部分文字列が共有できるときは，一意的に共有される．図2の例では4種類の文字列 “a”, “ab”, “abd”, “ac” が表現されていて，“a” は他の三つ “ab”, “abd”, “ac” に共有されており，“ab” は “abd” に共有されている．この文字列の例を実際に文字列テーブル中に格納した例は図1に示されている．

2.3 文字列検索アルゴリズム

文字列テーブルの検索アルゴリズムを検索，登録，削除の三つについて示す．今，検索の対象になっている文字列Sは一次元のストリームであり，最後の文字

† A Pipelined String Search Algorithm Based on an Associative Memory by KOZO ITANO (Institute of Information Sciences and Electronics, University of Tsukuba), HIDEMI SASAKI (College of Information Sciences, University of Tsukuba) and TOMOYOSHI YAMAGATA (Division of Research Facilitation, University of Tsukuba).

†† 筑波大学電子情報工学系
††† 筑波大学情報学類
†††† 筑波大学研究協力課

* 本研究は文部省科学研究費補助金試験研究(1)58850063によって一部を補助された．

** 現在 伊藤忠システム開発

address	char	pointer
0	255	0
1	0	
2	0	
3	0	
4	'a'	0
5	0	
6	0	
7	'b'	4
8	0	
9	'c'	4
10	'd'	7
11	0	
n-2	0	
n-1	0	

図 1　連想文字列テーブルの構造
Fig. 1　Associative string table structure.

が0で終端しているとする．文字列テーブル st はハードウェアの連想記憶で構成されているのでランダムアクセスと連想アクセスの両方が可能である．すなわち，文字列テーブル中の j 番目のエントリの文字フィールドとポインタフィールドはそれぞれ st[j].char，st[j].ptr で表すことにすると，ランダムアクセス時には j を与えてアクセスし，連想アクセス時にはエントリの各フィールドの内容を与えてエントリのインデックスを出力することができる．

（1）検　　索

文字列Sを検索し，その最終文字のエントリのインデックスを文字列識別コードとして与えるアルゴリズムを以下に示す．

step 1. $p \leftarrow 0$;
step 2. $c \leftarrow$the next character of stream S;
　　If $c=0$, then goto step 5;
step 3. Search an entry st[q] where
　　st[q].char$=c$ and st[q].ptr$=p$;
　　／＊ if found, q is given as an index to the found entry ＊／
　　If found, then $p \leftarrow q$; goto step 2;
step 4. Not found;
step 5. End of search;
　　The index of the found entry is in p;

（2）登　　録

文字列Sを検索し，もし存在しなければその文字列を文字列テーブルに登録するアルゴリズムを以下に示す．

step 1. $p \leftarrow 0$;
step 2. $c \leftarrow$the next character of stream S;
　　If $c=0$, then goto step 6;
step 3. Search an entry st[q] where
　　st[q].char$=c$ and st[q].ptr$=p$;
　　／＊ if found, q is given as an index to the found entry ＊／
　　If found, then $p \leftarrow q$; goto step 2;
step 4. Search an empty entry st[q] where
　　st[q].char$=0$;
step 5. st[q].char$\leftarrow c$;
　　st[q].ptr$\leftarrow p$;
　　$p \leftarrow q$;
　　goto step 2;

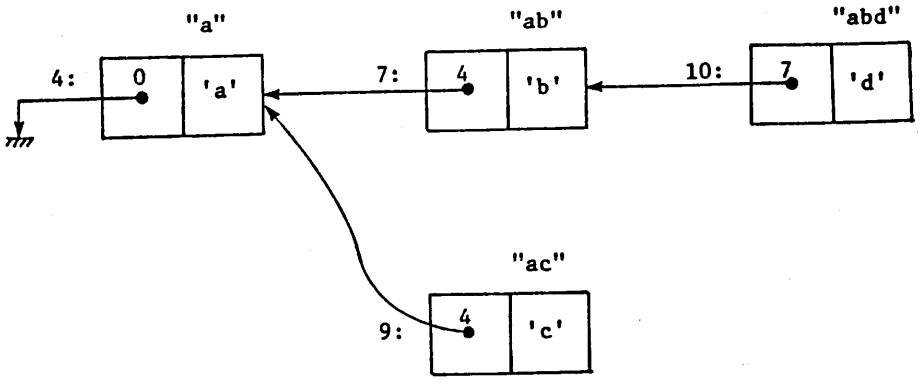

図 2　文字列の表現形式
Fig. 2　String representation scheme.

step 6. End of store;
p indicates the index of the last entry;

(3) 削　　除

文字列Sを検索し，もし他の文字列が部分文字列としてその文字列を共有していなければ，文字列テーブルから削除するアルゴリズムを示す．まず検索アルゴリズム(1)の step 1 から4を実行して*p*に文字列の最終文字のエントリのアドレスを得た後に，次のアルゴリズムを実行する．

step 5. Search an entry st [*q*] where
st [*q*] .ptr=*p*;
If found, goto step 8;
step 6. ／＊ delete the entry ＊／
st [*p*] .char←0;
If *p*=0, goto step 8;
step 7. p←st [*p*] .ptr;
goto step 5;
step 8. End of delete;

3. 検索アルゴリズムの評価

提案したパイプライン型文字列検索アルゴリズムを用いて実際に UNIX BSD 4.2[7]中の C, Pascal 両言語で書かれたソースプログラムおよび英文のドキュメントについて，文字列テーブルに登録および検索を行った．連想記憶テーブルは Hash 法[8]を用いて実現し，VAX 11/750 BSD 4.2 上で測定を行った．ここで文字列とは Pascal，Cのプログラムの場合は言語の字句レベルでのトークンすなわち，名前や予約語，オペレータなどであり，ドキュメントの場合は英語の単語である．表1には実際に使用した文字列データのサイズが，また表2には文字列の走査を行った結果が示されている．ここで“異なる文字列数”は出現した文字列の種類，また“異なる文字列中の総文字数”は部分文字列を共有しない場合に必要なエントリ数，また“使用されたエントリ数”は部分文字列の共有の場合必要であったエントリ数を示している．部分文字列を共有しないと仮定した場合に比較して共有を行う場合でどの程度に必要なエントリが減少したかを表すファクタが“エントリ数短縮率”であり，Pascal, C, ドキュメントでそれぞれ 0.73, 0.50, 0.41 であった．短縮率は文字列の先頭部分が共通であるような類似性を持つ文字パターンの種類が多く現れると小さくなる．例えば，短縮率 0.5 のときは全体として半分の部分文字列がオーバラップしていて共通のエントリで表現できたことを表している．また，この短縮率は，プログラムの場合プログラマの名前のつけ方の好みに依存するのである程度人の個性を反映することになる．例えば，パスカルプログラムの場合 0.73 とやや短縮率が悪いのはプログラマが名前の先頭部分の文字列が同じにならないような名前づけを好んだ結果であると思われる．

表1 測定データのサイズ
Table 1 Size of measured data.

	ファイル数	全文字数	全文字列数
パスカルソース	22	26103	6716
C ソース	69	543153	101670
ドキュメント	514	3625929	1355029

表2 文字列の走査結果
Table 2 Result of lexical scan.

	異なる文字列	異なる文字列中の総文字数	使用されたエントリ数	エントリ数短縮率
パスカルソース	442	2266	1663	0.73
C ソース	4625	27803	14024	0.50
ドキュメント	20933	141412	57922	0.41

4. 本方式の特徴

ここでは，提案方式の検索時と，表現時の特徴について考察する．

(1) 検索時間

文字列のすべての文字をまとめて一つのエントリに入れる方式では検索は文字列について1回ですむことになるが文字ごとにエントリが分かれているときは文字数だけ検索をする必要が生じる．しかし前者の場合でも文字列中の文字を逐次的に比較するとすると，文字数だけ比較することが必要であるばかりでなく，一致しない文字列についても文字の比較が起こるので，文字の比較の回数は本方式の方が一般に少ないと考えられる．また提案方式ではハードウェアの検索機構を前提として，文字単位の逐次的処理に重ねて検索ができる点が重要であり，さらに比較機構が1文字分ですむのでハードウェアのコストの点でも有利である．

(2) 文字列表現の柔軟性

従来方式の場合は，文字列テーブルの1エントリに格納できる文字数は固定長なので，文字列が一定の長さを越えると取扱いが困難となる．これに比較し，提案方式では，文字列の長さは可変長でよく，制限を受

けない．これは文字列テーブルを連想記憶としてハードウェアで実現する場合にハードウェアの設計が格納すべき文字列長に依存しないことを示しており，ハードウェア化に適していると考えられる．

（3） 記憶領域

提案方式の不利な点は，ポインタフィールドが文字ごとに必要であり，これがオーバヘッドとなることである．しかし，格納する文字列を可変長にすることを前提条件とするとリスト構造を導入する必要があり，結局ポインタフィールドが必要になる．したがって，可変長文字列を取り扱う場合は，特に提案方式が不利とはいえない．

5. むすび

文字列の逐次的走査に関連して，文字列テーブルを高速に検索するアルゴリズムを開発し，実際に評価を行った．従来から使用されているソフトウェアによるテーブル検索アルゴリズムとの定量的比較はまだ行っていないが，定性的な比較だけでも，提案方式の有用性が示されていると考える．本アルゴリズムは，現在試作中の高級言語計算機[9]中の字句解析ユニットに使用されており，実際の使用には文字列テーブルにさらにいくつかのフィールドが追加され，高級言語の字句解析を可能にしている．

参考文献

1) Gries, D.: *Compiler Construction for Digital Computers*, John Wiley and Sons (1971).
2) Itano, K.: A Pascal Interactive Direct-Execution Computer, Proc. of the International Workshop on High-Level Language Computer Architecture, pp. 161-169 (1982).
3) Ida, T.: Hashing Hardware and Its Application to Symbol Manipulation, Proc. of the International Workshop on High-Level Language Computer Architecture, pp. 99-107 (1980).
4) Freeman, M., and Levy, L. S.: On Architecture for Document Preparation, Proc. of the International Workshop on High-Level Language Computer Architecture, pp. 196-205 (1980).
5) Itano, K.: CDL Design of a Pipelined Lexical Scanner, Technical Report TR-1093, Department of Computer Science, Univ. of Maryland (1981).
6) 板野肯三：逐次ハッシュ文字列短縮法に基づく字句解析プロセッサの設計，テクニカルノート HLLA-6，筑波大学・電子情報工学系 (1984).
7) UNIX Programmer's Manual, Department of Electrical Engineering and Computer Science, Univ. of California, Berkeley (1983).
8) 板野肯三，山形朝義：逐次ハッシュ法による文字列データのコンパクト化率の測定，テクニカルノート HLLA-31，筑波大学・電子情報工学系 (1984).
9) 板野肯三，杉原敏昭，佐々木日出美，山形朝義：UDEC：汎用直接実行型計算機，第30回情報処理学会大会論文集，pp. 223-230 (1985).

(昭和60年2月25日受付)
(昭和60年6月20日採録)