木構造を用いた高速な全文検索について*

5T－7

大野 学 †三木田 哲也 ‡佐藤 隆士 §
大阪教育大学¶

1 はじめに

本稿では，テキスト中から任意の文字列を特に高速に検索するため新しいデータ構造と，それを用いた検索アルゴリズムを提案する．基本的アイデアは，一定長（L:レベル）の文字列全てについて，その文字列が出現するテキスト中の位置を検索部としてコンパクトなデータ構造に記憶する点である．この長さL用の検索部を用いることにより，キー長Lの文字列のみならず，L未満およびLを超える文字列の検索も効率的に行うことができる．

本稿での提案に関連し，日本語を対象とした全文検索システムで1文字単位ないし2文字単位の転置ファイルを利用したシステムの稼働例が上げられているが，日本語の場合，文字数が多いため検索部が大きくなる問題が指摘されており[1]，文字種に応じた複数の検索構造を作成する工夫がなされている[2, 3]．本稿でのレベルLが1ないし2以上の場合に関連しているが，従来の研究では，キー長がL未満の文字列の検索は扱ってない．また，複数文字単位の転置ファイルに基づいている点は共通だが，本稿で提案のインデックス部は二次記憶へのアクセス回数を極力減少するデータ構造に構成されている点でも特徴を有する．

提案の手法は，検索部として用いるデータ構造のみで，当該文字列を含むテキスト中の位置を知ることができるため，署名ファイル，PAT木[4]など存在の可能性のみを知る手法に比べ，精度の高い方法と言える．また，提案の方法は，転置ファイルなど単語ごとのインデックスによるもの[5]ではなく，すべての文字列を検索の対象とするものである．即ち，作成される検索高速化のためのデータ構造は，単語をはじめテキストの文法上のいかなる性質にも依存していないため，広範囲な言語，ビット列や遺伝子情報などのあらゆるストリングマッチに適用可能である．

2 基本アルゴリズム

今回提案する手法では，固定長部分文字列 (n-gram) をもとに木構造にしたものを利用して検索を行う．

木構造データファイルを作るにはまず被検索テキスト全体から隣接する固定長Lの部分文字列群を取り出す．取り出した文字列を昇順に整列し，その文字列のテキスト上の位置情報とともに図1(この場合は$L=2$)に示す木構造を作成する．

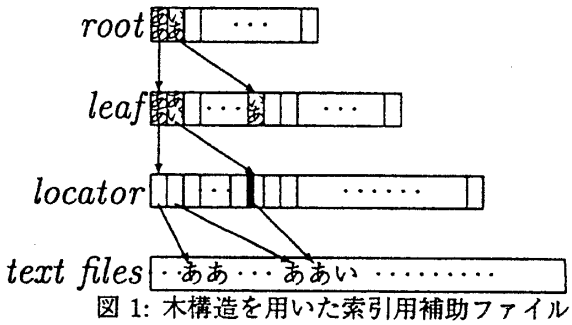

図 1: 木構造を用いた索引用補助ファイル

*root*部，*leaf*部には昇順に整列された文字列と，木の下方に向けてのポインタが格納されている．また，*locator*部には各部分文字列ごとにグループ化された位置情報が列挙されている．

実際の検索時には，全文書中で用いられている文字集合をΣと定義した時，キー長lの検索キーkは$k = c_1c_2\ldots c_l(c_i \in \Sigma; 1 \leq i \leq l)$と表すことができる．

(1):$l \leq L$のとき

$k = c_1c_2\ldots c_lc_{l+1}\ldots c_L$ $(\forall c_j \in \Sigma; l+1 \leq j \leq L)$とキー長を$L$に拡張する事により検索を行う．実際には固定長部分文字列は昇順に列挙されているのでその両端を求めて範囲内全ての位置情報を取り出せば良い．

(2):$l > L$のとき

$l-L+1$個の部分文字列$k_j = c_jc_{j+1}\ldots c_{j+L-1}(1 \leq j \leq l-L+1)$に分解し，それぞれの文字列に対して行った検索結果の積集合を取る事で検索を行う．

---

*Fast string pattern maching using tree structure files
†Manabu Oono
‡Tetsuya Mikita
§Takashi Sato
¶Osaka Kyoiku University 4-698-1 Asahigaoka, Kashiwara 582, Japan

3 改良アルゴリズム

前節でのデータ構造をそのまま用いた場合，実験の結果 L の値にもよるが元の文書データの 90%から200%程度とかなり大きなデータになる事がわかった．そこで，できるだけ高速性を失わないように配慮しつつ圧縮することにした．基本的には各データファイルの要素の内，隣接データが圧縮するのに十分なぐらい小さい場合に，符号を切替えることで圧縮を行っている．

3.1 *locator* 部の圧縮

今回の実験には，位置情報として文書番号を持たせることにしたので 2 バイト整数を割り当てることにしたが，隣り合う位置情報の差分が符号付き 1 バイト整数の上限を下回る場合には半分の 1 バイト整数に圧縮することにした．

3.2 *leaf* 部の圧縮

今回の実験には，分割文字列長 L に 4 を選択し，各文字が 2 バイト文字であるので，計 8 バイト分の領域を割り当てることにしたが，隣り合う文字列の差分が小さく，圧縮可能な場合には半分の 4 バイトに圧縮することにした．

3.3 日本語への対応

基本アルゴリズムの方では日本語への対応については全く触れていなかった．計算機において日本語として使われている文字は一般的に 1 バイト文字である ASCII 文字と 2 バイト文字の仮名・漢字・記号などである．ここで問題となるのは 1 バイト文字と 2 バイト文字との混在である．今回は，線形写像を用いて全ての文字を 2 バイトの整数値に変換することで混在していることを意識しないで済むようにした．また，線形写像を行う際に 1 バイト文字と 2 バイト文字で同じ内容を示すような文字[1]を統一することにし，検索時にどちらでもマッチするようにしている．

4 実験結果

今回の実験では，network news の日本語のニュースグループから約 100MB 分の文書を標本として利用した．また，文字列長 L には，差分計算に都合が良い最大長と考えている 8 バイトになるように，4 文字を採用した．実験用検索文字列にはマッチ数の少ないものと多いものとの 2 種類を選んで実験している．圧縮の効果からか，表 1からもわかるように被検索ファイル中の部分文字列を網羅しているにもかかわらず被検索テキストと比較しても 80%程度になっている．このように十分小さくなっているにもかかわらず検索速度は表 2のように，いずれの場合も $100msec$ 程度におさまり充分高速であると言える．今回の手法では文字列を線形写像により数値化しているため 8 バイト整数演算ライブラリを使用しているのだがこの部分での処理に，多分にコストがかかっており $l = 3$ の実験 2 の場合でかなりの時間を有していることがわかる．

表 1: 各ファイルの大きさ (KB)

被検索テキスト	root	leaf	locator
約 100000	502	28425	55751

表 2: キー長ごとの検索時間 (msec) とマッチ数

	$l = 3$	$l = 4$	$l = 5$
実験 1	85.8/ 1	106.3/ 4	152.2/ 3
実験 2	159.3/2060	90.6/87	135.7/11

5 おわりに

以上のように，被検索テキストと同量程度の補助ファイルを用意することにより，かなり高速な検索を行えることを示せた．今後は，更新操作を考案し，また，今回の問題点である整数演算ライブラリについても改善する必要がある．

参考文献

[1] 菊地芳秀他: 全文検索の技術動向とシステム事例, 情報処理学会情報学基礎研究報告, 92-FI-25 (1992).

[2] 菊地忠一: 日本語文書用高速全文検索の一手法, 電子情報通信学会論文誌 (D-I), Vol. J75-D-I, No. 9, pp. 836–846 (1992).

[3] 岩崎雅二郎, 小川泰嗣: 文字成分表による文字列検索の実現と評価, 情報処理学会データベースシステム研究報告, 93-DBS-92, pp. 1–10 (1993).

[4] Gonnet, G., Baeza-Yates, R. and Snider T.: New Indices for Text: Pat Trees, in *Information Retrieval: Data Structure & Algorithms* chapter 5, Frakes, W. and Baeza-Yates, R. Ed., pp. 66–82 (1992).

[5] Zobal, J., Moffat, A. and Sacks-Davis, R.: An Efficient Indexing Technique for Full-text Database, *Proc. 18th Int. Conf. on VLDB*, 1992, pp. 352–362.

[1] アルファベットなど