データベースシステム 99−35
（1994. 7. 22）

全文検索のための文字成分表方式の改良

小川 泰嗣　岩崎 雅二郎　林 大川
{ogawa,iwasaki,haya}@ipe.rdc.ricoh.co.jp
（株）リコー　情報通信研究所

全文検索法として文字成分表は有効であるが、つぎのような問題点がある：(1) 検索文字列が長くなるに従い検索時間が遅くなる、(2) 文字成分表ファイルのブロックサイズは登録／検索に反する影響を与えるため、登録／検索速度を両立させることができない。本稿では、これら問題点を解決法を提案する。前者に関しては、文字列から抽出されるエントリの重複を削除する、高頻度で出現する長さ m 以上の文字列もエントリとすることで検索文字列から抽出されるエントリ数を削減する、という二つの手法で文字列長の影響を小さくする。後者に関しては、二種類のブロックを導入し、メンテナンス時に小さいブロックを大きくまとめ上げることで、登録速度を維持しながら検索を高速化する。

Enhancing full-text search performance using character bitmap tables

OGAWA Yasushi, IWASAKI Masajirou and HAYASHI Taisen
Information & Communication R&D Center, RICOH Co., Ltd.

Although a *character bitmap table* which maps documents to their contained characters is effective in full text search, there are two problems: 1) longer query strings lead to longer retrieval times. 2) bitmap file block size has opposing impacts on registration performance and retrieval performance. To solve these problems, 1) we reduce the number of entries extracted from the query by adding *string entries*, special entries in the character bitmap organization, as well as eliminate duplicate entries, to reduce the effect of query length on search response. 2) we introduce two sizes for blocks, using small ones during registration and large ones during retrieval. Thus, our method attains high performance in both cases.

1 はじめに

特許広報が CD-ROM 化されるなど、全文検索への要求が高まっている。

全文検索の実現法の一つとして文字成分表がある[3]。文字成分表は文字ごとに出現した文書を記録するものであり、(1) 処理単位が文字なので言語処理が不要であり、システムの構築／メンテナンスが簡単にできる、(2) 文字成分表方式では一文字一文字がある程度の文書識別力を持つことが要求されるため、字彙の多い日本語向きである、等の理由から多くの研究開発が行なわれている。文字成分表方式では、文字の位置情報を捨象するため誤検索が避けられないが、隣接する二文字を記録することで誤検索を削減可能である。

文字成分表に関し、[2] において、われわれもつぎのような提案を行なっている：(1) 隣接文字成分の異なり数をハッシュにより削減し、隣接文字成分表を小型化した。その際、文字種ごとに異なるエントリを割り当てることで、誤検索率も低く押えた。(2) 文字成分表の要素の多くは 0 であるので、圧縮により文字成分表を大幅に小型化した。その結果、検索速度も向上した。

しかし、このような改良を施しても問題点が残っている。以下、2 章でわれわれの方式を含めた文字成分表の二つの問題点を明らかにし、3/4 章でそれら問題点に対する解決策を提案する。さらに、5 章で解決策の有効性の評価結果を示す。

2 文字成分表方式の問題点

2.1 検索文字列長と検索時間の関係

文字成分表方式では、つぎのように検索処理を行なう：(1) 検索文字列からエントリ[1]を抽出し、(2) 各エントリに対応するビット列を読みだし、(3) 全エントリのビット列の AND をとる。したがって、検索文字列が長くなると、抽出されるエントリが増大し、検索時間が遅くなる。図3の実線が、従来方式の文字成分表における文字列長と検索時間の関係を示したものである。この図から文字列の長さに応じて検索時間が長くなることが確認できる。

[1]われわれの方式のように隣接文字からハッシュにより算出される値を使用する場合、文字成分表に記録されるものは文字と一対一の関係はない。そこで、以下では『文字成分』の変わりに『エントリ』という言葉を用いる。

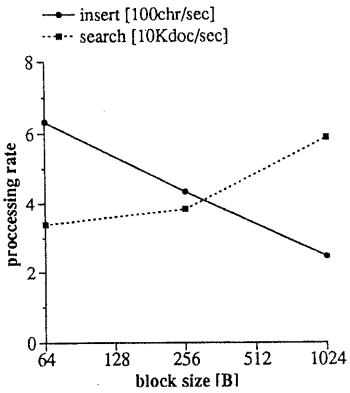


図 1: ブロックサイズと登録／検索速度の関係

2.2 ファイル構成

検索高速化のためには、文字成分表をエントリごとにビット列にアクセス可能であるビットスライス形式とすることが有効である。ビットスライスファイルは、固定長レコード（以下『ブロック』）の集合体として実装すれば、管理を容易にできる。

ブロックの大きさは、登録／検索性能に大きく影響する：登録時にはブロック単位に書き込みが行なわれるため、ブロックが小さい方が登録速度が高い。一方、検索時にはブロックへのアクセス回数を減少できるため、ブロックが大きい方が検索速度が高い。すなわち、登録／検索速度はブロックサイズに関して反比例の関係があり、両立させることができない。この様子を図 1に示す。この図は、5 章で示す 10 万件の文書に対する登録／検索速度の測定結果である。登録速度は 1 sec 当りの登録処理文字数、検索速度はこのＤＢにおける 1 sec 当りの検索処理文書数である。

ファイル構成上のもう一つの問題は、文書登録時に要求発生順に新たなブロックを割り当てていることである。その結果、各ビット列を構成するブロックが連続領域とならず、ビット列読み出し時に余計なシーク／回転遅延が発生し、検索速度が低下する。

3 長い検索文字列の検索時間短縮

まず、長い文字列の検索時間を短縮する方法として、つぎの二つの改良を提案する。

3.1 エントリ重複削除

第一の方法は非常に単純な方法で、検索文字列から抽出されたエントリの重複を削除することである。

例：検索文字列「フリップフロップ」

・従来方式では 15 エントリ抽出される[2]
フ、フリ、リ、リッ、ッ、ップ、プ、プフ、
フ、フロ、ロ、ロッ、ッ、ップ、プ

・重複削除で 11 エントリに削減できる
フ、フリ、リ、リッ、ッ、ップ、プ、プフ、
フロ、ロ、ロッ

その結果、ビット列の読み出し（および AND 処理）回数が削減され、検索時間が短縮される。検索文字列が長い程一つの文字列から同一のエントリが抽出される確率は高くなるため、文字列長の影響を少なくできる。

3.2 文字列エントリ

第二の方法は、文字成分表に m 文字以上の文字列であるエントリ（以下『文字列エントリ』）の導入である。検索語に文字列エントリを構成する文字列が含まれている場合、その部分に対応する従来型エントリは抽出しない。

例：「テキスト」が文字列エントリに含まれ、
検索文字列が「フルテキスト」

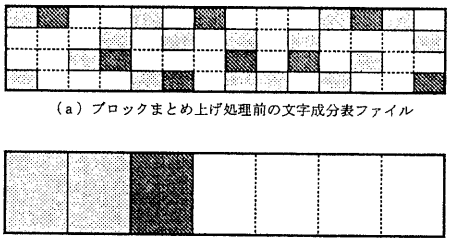
・従来方式では 11 エントリ抽出される
フ、フル、ル、ルテ、テ、テキ、キ、キス、
ス、スト、ト

・文字列エントリで 5 エントリに削減できる
フ、フル、ル、ルテ、テキスト

その結果、検索語から抽出されるエントリ数が減少し、検索が高速化される。検索文字列が長い程、文字列エントリが含まれている確率が高くなるため、文字列長の影響を少なくすることができる。

文字列エントリの考えは、 n>2 の n-gram（n 文字の全ての組合せを登録する方法 [1]）と類似している。しかし、n=2 [3]に対しても、日本語では字彙が豊富なのでエントリ数を減らすべくハッシュを導入して

[2] 実際には、隣接文字はそのままエントリを構成するのではなくハッシュ値を計算してエントリに変換されるが、ここでは簡単のため隣接文字をそのまま示している。



（a）ブロックまとめ上げ処理前の文字成分表ファイル

（b）ブロックまとめ上げ処理後の文字成分表ファイル

図 2: ブロックまとめ上げ処理

いることを考慮すると、n>2 の n-gram は非現実的である。これに対し、われわれは文字列エントリの個数を限定し、出現頻度の高いの文字列のみをエントリとして選択している。その結果、字彙の豊富な日本語に対しても、大きな記憶負荷をかけることなく、検索速度を向上させることが可能となった。

また、この方式は従来型の単語（キーワード）索引にも近い。しかし、(1) 文書登録時の文字列エントリを文字列照合によって実現しており、自然言語処理を用いたキーワード抽出を行なう必要がない。したがって、キーワード抽出に伴う辞書管理などの作業が不要であり、キーワード抽出の精度が問題となることもない、(2) 文字列エントリを構成する文字列を静的なものとしているので、キーワード索引のように B 木等のインデックスが不要である、の二点で異なる。

4 ブロックまとめ上げ処理

ファイル構成上の問題を解決するため、大小二つの大きさのブロックの導入を提案する（以下では小さいブロックを『バケット』、大きいブロックを『コンテナ』と呼ぶ）。その上で、登録処理では新たなブロックとして小さいバケットを割り当てることとし、登録速度を維持する。バケット数が増大した時点で、複数バケットをコンテナに融合する『ブロックまとめ上げ』を行なう。その際、エントリごとにまとめ上げることで、同一エントリを構成するブロック（コンテナ）を連続領域に割り当てる（図 2参照）。その結果、連続領域からコンテナ単位にビット列を読み出すことが可能となり、検索レスポンスを大幅に向上できる。

[3] 隣接文字成分表は 2-gram である。

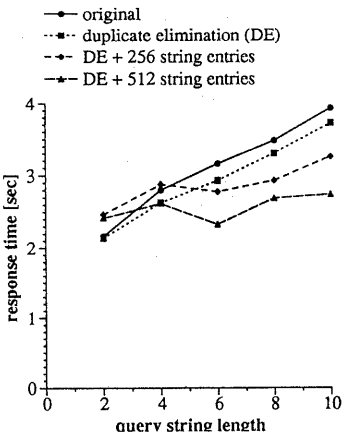


図 3: 文字列エントリの効果

5 評価

われわれが提案した方式の性能測定を行ない、有効性を評価した。測定はつぎの条件で行なった。

- マシン：Sun SPARCstation 10 (model41)
- 登録データ：PATLIS の特許要約文 100,000 件。一件の平均長は 70 文字、計 14 MB。
- 検索文字列：上記要約文からカタカナ文字列を、長さ 2 ～ 10 のもの各 20 個、計 100 個選出。

5.1 重複削除／文字列エントリの効果

文字列長ごとの平均検索時間を (1) 従来方式、(2) エントリの重複削除を行なった場合、(3) 256 個の文字列エントリを加えて重複削除を行なった場合、(4) 512 個の文字列エントリを加えて重複削除を行なった場合、の四つの場合について測定した（ブロックサイズを 64B とし、まとめ上げ処理は行なっていない）。文字列エントリは、対象データにおける 3 文字以上のカタカナ連続部分から出現頻度の高い順に 256/512 個を選択した。

測定結果を図 3に示す。従来方式では文字列長に応じて検索時間が増加していたのに対し、(2)～(4) になるにしたがって、検索時間の増加が押えられることが確認できた。特に (4) は有効であり、文字列長 10 の条件では約 30% の短縮効果があった。

表 1: まとめ上げによる速度向上

バケットサイズ (B)	まとめ上げ前 (Kdoc/sec)	まとめ上げ後 (Kdoc/sec)	向上率
64	34.0	195	5.74
256	38.4		5.08
1024	58.7		3.32

5.2 ブロックまとめ上げの効果

まとめ上げ前後の文字成分表に対する検索速度を測定し、その効果を示す。（登録処理そのものは従来方式のままなので、図 1に示す通りである。）

コンテナの大きさを 1024B とした場合の平均検索速度／向上率を表 1 に示す。ここで、平均検索速度は図 1 に示した値であり、向上率はまとめ上げ前に対するまとめ上げ後の速度の比である。この表から、まとめ上げ処理が極めて有効であることがわかる。なお、バケットサイズがコンテナと同じ 1024 B の場合でも、3 倍以上の高速化が実現されている。これは、ブロックが連続領域に割り当てられたためである。

6 おわりに

文字成分表に関する二つの問題点を解決する方式を提案した。検索文字列長については、文字列から抽出されるエントリの重複を削除する、高頻度で出現する長さ m 以上の文字列もエントリとすることで検索文字列から抽出すべきエントリ数を削減する、という二つの手法で影響を小さくする。ファイル構成については、二種類のブロックを導入し、まとめ上げ処理により、登録速度を維持しながら検索を高速化する。評価の結果、これら手法の有効性が確認できた。

参考文献

[1] M.C. Harrison. Implementation of the substring test by hashing. *Communications of ACM*, Vol. 14, No. 12, pp. 777–779, 1971.

[2] 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. 情報処理学会研究会報告, Vol. DBS97, pp. 1–10, 1993.

[3] 小川 泰嗣. テキストデータベース技術の最近の動向. In *ADBS 93*, pp. 153–162, 1993.