5T－6

多段階に圧縮された補助ファイルを用いた文字列検索*

松尾 通博† 杉山 良輔‡ 佐藤 隆士§
大阪教育大学¶

1 はじめに

テキスト処理における文字列検索などで，ハードディスクなどの二次記憶上に置かれたファイルに対して高速に文字列検索を行なう場合が発生する．文字列検索高速化の手法としては逐次検索の高速化[1, 2]，署名ファイル[3]を用いた方法などが知られている．今回提案する手法は二次記憶上に置かれたファイルに対して多段階に圧縮されたファイルをあらかじめ用意しておく．検索文字列の長さに応じて検索に最適な圧縮ファイル上で予備検索を行ない，候補を見つけて元ファイルにアクセスして候補を本当に検索できたかどうかチェックするものである．この方法の利点として，検索文字列が長ければ長いほど，圧縮率の大きな圧縮ファイルの予備検索で十分に候補を絞ることが可能で高速な検索が期待できる点である．

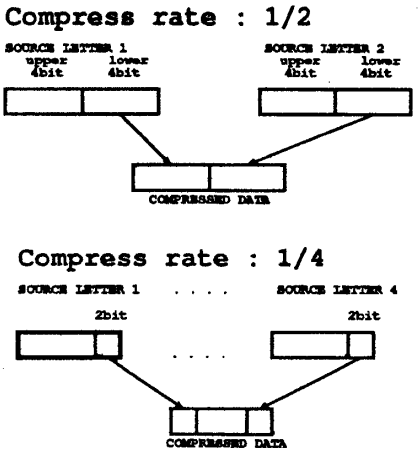
2 圧縮ファイル

文字列の被検索対象ファイルSは二次記憶上に置かれている．以降，Sのことをもとのファイルという．主記憶と二次記憶間のデータ転送はブロック単位に行われる．１ブロックの大きさはB[byte]とし，アルゴリズムのコストはブロック転送回数とする．与えられた検索文字列Kを見つける際，コストをできるだけ小さくし，検索効率を上げることを目的とする．提案の手法では，Sとは別にSを何段階かに圧縮した圧縮ファイル$S_1, S_2, \cdots$をあらかじめ用意しておく．ここで，$S_i(i = 1, 2, \cdots)$はSを$1/2^i$に圧縮したファイルで，必ずしもすべてのiについて用意する必要はない．

もとのファイルの大きさをm[byte]とする．任意の圧縮ファイルともとのファイルとの大きさの比を圧縮率$\alpha(0 \leq \alpha \leq 1)$という．従って，$S_i$の圧縮率は$\alpha = 1/2^i$である．検索文字列$K$の長さは$L$[byte]，ビット数で勘定して$l(= 8L)$[bit]とする．

今回は圧縮ファイルを準備するに際し単純なアルゴリズムを使用した．テキストのビット移動による圧縮であり，非可逆変換である．

半角文字は８ビットのデータで構成されている．これをビット移動によって圧縮する．例えば２分の１圧縮であるのなら，８ビット中の上位４ビットを削除し下位４ビットを左に４ビットだけシフトする．次の文字列も同様に下位４ビットをシフトして来て先の文字の４ビット分にくっつける．これを繰り返せば圧縮率５０％の補助ファイルが出来上がる．下図に示す．



なお，４分の１圧縮，８分の１圧縮なども同

*Fast string pattern matching using differently compressed data files
†Yukihiro Matsuo
‡Ryousuke Sugiyama
§Takashi Sato
¶Osaka Kyoiku University 4-698-1 Asahigaoka, Kashiwara 582, Japan

様に行なう.

3 検索アルゴリズム

提案のアルゴリズムの概略は次のようになる.

C1. [圧縮ファイルの選択] 検索のコストの期待値を最小とする圧縮ファイル S_iを選ぶ.

C2. [検索文字列の変換] 検索文字列 Kを選ばれた圧縮ファイルと同一形式に変換し K_iを得る.

C3. [候補探索] S_i上で K_iを逐次探索し，文字列候補を見つける．その S_i上での位置を P_iとする．もし，S_iの終りまで探索したならば終了.

C4. [確認] P_iをもとのファイル S上での位置 Pに変換し，Sを調べ Kと一致すれば Pを記録する．C3 に戻る.

次に C1 での圧縮ファイルの選び方について考察する．C2 で検索文字列も選ばれた圧縮ファイルと同一方法で変換されるので，C3 の逐次探索で候補の絞込みに有効なビット数は $l\alpha$となる．従って，逐次探索の選択度は，$1/2^{l\alpha}$である．Sは長さ m[byte] なのでその中に長さ L[byte] の文字列は $m-L+1$ 個存在する．圧縮ファイル上でも同様であり，前述の選択度を掛けると C3 での候補数の期待値 gは，

$$g \simeq m/2^{l\alpha} \qquad (1)$$

となる．ここで，一般的に $m \gg L$ であるとし近似を行った．S のブロック数は $n_B = m/B$(端数は無視，以下同様) であるので，C4 で行われる g回の確認が n_B個のブロックに一様に分布すると仮定すると，Sに関するブロックアクセス回数の期待値 fは，

$$f = n_B\{1-(1-1/n_B)^g\} \qquad (2)$$

となる．これに C3 の圧縮ファイルのブロックアクセス回数 $m\alpha/B$を加えた，

$$C = n_B\{1-(1-1/n_B)^g\} + m\alpha/B \qquad (3)$$

が全体のコストの期待値になる．従って，(3) を最小とするαを求め，その値に近い圧縮率をもつ圧縮ファイルを C1 で選択すればよいことになる.

今回は圧縮ファイル上で予備検索を行なうが，予備検索の結果が正しいかどうかはわからない．それは，圧縮した時点で失われた情報量が多いため結果に不正確さが出て来る事が予想されるからである．そこで，予備検索を行って候補を捜し出した上で圧縮前の元ファイルにアクセスして，予備検索の結果が正しいかどうか確認するという手続きが最後に必要となる.

4 計測結果

ここでは検索キーを直接照合する手法，力任せ法での計測結果を以下に示す.

検索文字列:communicate

圧縮率	未圧縮	1/2 圧縮	1/4 圧縮
検索数	242	273	4228
時間 (s)	122	67	93

検索文字列:comp.software.international

圧縮率	未圧縮	1/2 圧縮	1/4 圧縮
検索数	56	56	56
時間 (s)	121	95	93

5 おわりに

思ったほどの高速化の実現は不可能だった．原因としては，圧縮率が高ければ高いほど予備検索の時点で誤った文字列を検索してしまう可能性が高くなり，二次記憶装置へのアクセス回数が増加し，そこがボトルネックになる事，同様に，文字列の bit 処理にかける CPU の負担が予想以上に重くなっていることの 2 点が挙げられる．今後の課題としては，より効果的な圧縮方法の開発と，この検索方法に適した検索アルゴリズムの開発が挙げられるだろう.

参考文献

[1] Boyer, R. S. and Moore, J. S.: A Fast String Searching Algorithm, *Comm. ACM*, Vol. 20, No. 10, pp. 762–772 (1977).

[2] R. セジウィック著 (野下他訳) : アルゴリズム 第 2 巻，近代科学社 (1992)

[3] Faloutsos, C.: Access Methods for Text, *ACM Comput. Surv.*, Vol. 17, No. 1, pp. 49–74 (1985).