A-022

圧縮された接尾辞配列を用いた近似文字列照合

Approximate String Matching using Suffix Array Compressed

田中 洋輔 *　小野 廣隆 †　定兼 邦彦 ‡　山下 雅史 †
Yosuke Tanaka　Hirotaka Ono　Kunihiko Sadakane　Masafumi Yamashita

1 導入

あるパターンだけでなく, それに類似したものも同時に検索するという近似文字列照合 (Approximate String Matching, ASM) は, テキスト検索, 計算生物学, パターン認識等に応用を持つ, 重要な機能である. ASM は, 長さ n の文字列 T 中で, 長さ m の文字列 P との編集距離が k 以下である全ての部分文字列の出現位置を得る[1], という問題である. ここで, 2 つの文字列間の編集距離は, それらを同じ文字列にするために必要な, 挿入, 削除, 代用操作の最小回数と定義される.

ASM を実現する様々な手法が提案されており, [1] では, それらを紹介し, 比較実験を行っている. これらの手法は, 索引を用いず逐次的に先頭から照合を行うものと, 索引を用いるものに分類でき, 本論文では索引を用いた方法に着目する. [2] では, 接尾辞配列 (SA) という索引を用いた ASM 手法が提案されている. SA には比較的多くのメモリ容量が必要で, これを圧縮する様々な手法が存在する. SADIV [3] は, 既存の手法と比べ, 容量は大きいが, 検索語の頻度が大きい場合の検索が比較的高速な SA 圧縮法である. 本論文では [2] で用いられる SA の代わりに SADIV を用いる.

2 準備

接尾辞配列 (SA)　接尾辞とは文字列の後ろ部分, 最後の 1 語 ($T[n]$) を含む部分文字列で, T 中の j の位置に対応する接尾辞は $T[j...n]$ である. 文字列 T に対し, SA は, 辞書順が i 番目の接尾辞の出現位置が j のとき $SA[i] = j$ と定義される. SA には $n \lg n$ bit (int 型で $4n$ Bytes) の容量が必要である. 検索は, 2 分探索で P と接頭辞を共有する範囲を探索, その範囲の SA の値を返すことで, $O(m \lg n + occ)$ で実現できる (occ は検索語の出現数).

SADIV　SADIV [3] の索引作成手順を示す. SA を大きさ s のブロックに区切り, その範囲内で SA の値を昇順にソート ($sorted\_SA$), その差分の値を保存 (dif_sSA), またソート前の区間の先頭の SA の値も別途保存する ($samp\_SA$). dif_sSA はゴロム符号という可変長符号を用い圧縮する. 理論的最悪容量は $(n+s)(\lg \frac{n}{s} + 3)$ となる.

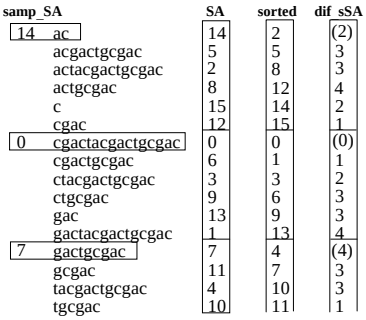

図 1: SADIV

次に検索法を示す. P が与えられたとき, まず $samp\_SA$ のみに関し 2 分探索を行う. この結果, P にマッチするものがどのブロック内に存在する可能性があるかわかるので, その範囲内の $sorted\_SA$ の値を dif_sSA から全て取り戻し, その全ての値に対応する接尾辞を P と逐次的に照合する. 計算時間は $O(ms + m \lg(\frac{n}{s} + 1) + occ)$ となる.

*九州大学大学院システム情報科学府
†九州大学大学院システム情報科学研究院
‡国立情報学研究所情報学プリンシプル研究系
[1] k をエラー数とも呼ぶ.

SA を用いた ASM 手法　[2] では SA を用いて, Suffix Trie (ST) をシミュレートする (ST とは, 接尾辞を登録した木で, 葉にその出現位置を格納したものである). ST を深さ優先で全探索 (DFS), エラー数 k 以下となる箇所を逐次的 ASM 手法を用いて[2] 探すことで, ASM を実現できる. 検索速度は, エラー率 $\alpha = \frac{k}{m}$ が低い場合, 逐次照合より高速となる. さらに [2] では, filtering というテクニックを用い, m が大きい場合の照合を高速化している.

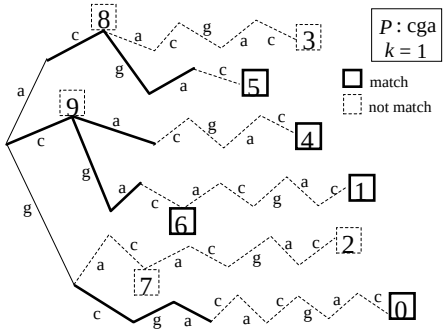

図 2: ST (SA) による ASM 手法

3　提案手法

提案手法は [2] で用いられる SA の代わりに SADIV を用い, 必要なメモリ容量を節約する. [2] での SA (ST) の全探索を, $samp\_SA$ に対して行い, この結果, マッチする (エラー数が k 以下となる) 接尾辞を含む可能性のあるブロックを選別, 選ばれたブロックのみデコードし, 逐次照合を行う[3].

問題点は, 隣接する 2 つの $samp\_SA$ に対応する接尾辞の文字が一致する場合は, ブロック内の文字が全ての同じなので, ブロックの文字を確定させることができるが, 一致しない場合, 文字が複数存在し, 文字が確定しないという点である. 例えば, b 番目のブロックに対し, $T[samp\_SA[b]+0]$ ='u' で $T[samp\_SA[b+1]+0]$ ='w' とすると, このブロックの 1 文字目には u と v と w の 3 つの文字が出てくる可能性がある. しかし, 逆に言えば u, v, w 以外の文字は出現せず, 比較する P の文字がそれ以外の文字ならば, エラー数を増やすことができ, 結果, デコードの必要なブロックを減らせる可能性がある.

図 3 の例[4] に対し $samp\_SA$ に対応する ST を描くと左側の木のようになるが, これに関し DFS を行うと, $samp\_SA$ の間の接尾辞を見逃してしまう. これを避けるには右側の木のような枝に (出現する全ての) 文字の集合を持つような木に対し DFS を行う必要がある. 言い換えると, あるブロックの 1 文字目が s,t, 2 文字目が a,e,h ということは, これらのすべての組み合わせ sa,se,sh,ta,te,th が存在する可能性があり, それらのどれとも近似一致しない場合はデコードの必要はない. これを実現するため, $\frac{n}{s}$ 個のブロック全てで, (辞書順の接尾辞の並びの) ある列内に存在する文字を (行数 d^* までに限定) 覚えておく (図 3 で, $B[b][d]$ は b 個目のブロックの d 文字目に現れる文字の集合を示す).

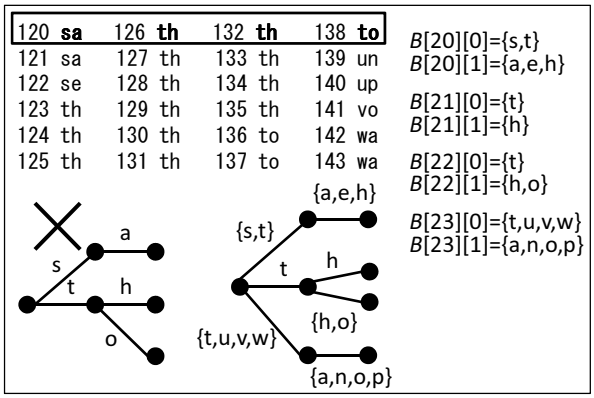

図 3: SADIV を用いた ASM 手法の問題点と解決案

参考文献

[1] Gonzalo Navarro, “A Guided Tour to Approximate String Matching,” ACM Computing Surveys, 33(1):31-88, 2001.

[2] Gonzalo Navarro, Ricardo Baeza-Yates, “A Hybrid Indexing Method for Approximate String Matching,” Journal of Discrete Algorithms, 1(1):21-49, 2000.

[3] 田中洋輔, 小野廣隆, 定兼邦彦, 山下雅史, “高速復元可能な接尾辞配列圧縮法,” FIT2009 第 8 回情報科学技術フォーラム, 2009.

[2] [2] では, 並列化された非決定性オートマトンを用いる.

[3] 提案手法でも filtering のテクニックを用いることができる.

[4] 辞書順での接尾辞の並びの一部抽出で (3 文字目以降は省略), 枠に囲まれたものが $samp\_SA$ に対応. b 番目のブロックは, 辞書順 $sb \ldots s(b+1)-1$ の接尾辞に対応.