ダブル配列を用いたマシン AC の効率的格納手法

信種真人† 森田和宏† 泓田正雄† 青江順一†
†徳島大学大学院 先端技術科学教育部

概要：電子文書を扱う上での基本操作の 1 つとして文字列照合がある．文字列照合には複数のキーを 1 回の走査で検出することができる AC 法がある．AC 法を実現するデータ構造には，リスト構造や配列構造などが考えられるが，検索の高速性と記憶量のコンパクト性がトレードオフの関係となる．一方でキー検索手法において，これら 2 つの利点を兼ね備えたデータ構造を実現するダブル配列法が提案されている．そこで本稿では，AC 法にダブル配列を用いることでキー集合を効率的に格納する手法を提案する．また実験により，リスト構造，配列構造を用いた AC 法と比較して有効性を示す．

1. はじめに

近年，計算機の記憶領域の大容量化とネットワーク化に伴い大量の電子文書があふれている．これらの電子文書を有効に扱うためには，必要な情報を効率よく検索する手段が必要である．

文書中から検索キーと完全一致する部分を探し出す全文検索のうち文字列照合は，文書と検索キーにおいて，それぞれの文字を逐次的に比較して検索する手法である．文字列照合アルゴリズムの一つには，AC 法[1]がある．AC 法は，複数のキーを 1 回の走査で同時に検出することができ，検索速度が非常に高速である．

AC 法を実現するデータ構造には配列構造やリスト構造などがあるが，記憶量と検索速度がトレードオフの関係となる．記憶量がコンパクトかつ高速な検索を実現できれば，AC 法のデータ構造として効率的である．

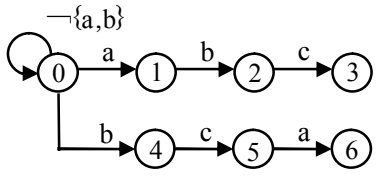
そこで本稿では，ダブル配列[2]を用いた高速かつコンパクトな AC 法のデータ構造を提案する．

2. AC 法

AC 法は，検索要求として与えられたキー集合からマシン AC と呼ばれる有限オートマトンを構成し，これに文字列を入力することで文字列中からマッチする全ての部分文字列を求めるアルゴリズムである．

マシン AC は，goto 関数，failure 関数，output 関数の 3 種類の関数から構成される．図 1 に，キー集合 K={“abc”,“bc”,“bca”}から構成されたマシン AC を示す．

図 1.(a)の状態遷移図は goto 関数を示している．



(a) goto 関数

s	f(s)
1	0
2	4
3	5
4	0
5	0
6	1

(b) failure 関数

s	output(s)
3	{“abc”, “bc”}
5	{“bc”}
6	{“bca”}

(c) output 関数

図 1: マシン AC の構成

例えば，ノード 0 から‘a’とラベル付けされた矢印による遷移はノード 1 へ到達する．この goto 関数による遷移を g(0,‘a’)=1 のように示す．また，未定義の遷移に対しては fail を返す．初期ノードにおいては，‘a’，‘b’以外の文字を表す¬{‘a’，‘b’}に対し，g(0,¬{‘a’,‘b’})=0 が定義されている．

図 1.(b)の failure 関数には，goto 関数による遷移に失敗したとき，つまり fail が返されたときの照合ポインタの移動先を格納する．

図 1.(c)の output 関数はその関数が定義されているノードに照合ポインタが到達したときに，検出されるキーを格納する関数である．

[例 1] 図 1 のマシン AC に対して，文字列“abca”を入力した場合について考える．まず初期ノード 0

において 1 文字目の‘a’に対し，g(0,‘a’)=1 となりノード 1 に遷移する．同様に文字‘b’と‘c’に対して g(1,‘b’)=2, g(2,‘c’)=3 となり，ノード 3 に遷移する．ここで，output(3)={“abc”, “bc”}であるため，“abc”, “bc”が検出される．

更にノード 3 から‘a’に対し，g(3,‘a’)=fail となるので f(3)=5 によりノード 5 へ遷移する．これは，照合中の文字列が“abc”から“bc”に変わったことを意味する．ノード 5 から文字‘a’の照合を再開し g(5,‘a’)=6, output(6) ={“bca”}より，“bca”を検出し，全ての文字について照合が終了する．　　(例終)

3. ダブル配列を用いた AC 法

3.1. ダブル配列法

マシン AC を構成する goto 関数は，g(0,¬{‘a’, ‘b’})=0 なる遷移を除くと，キー集合における共通の接頭辞を併合した木構造であるトライ構造[3]をしている．

このトライ構造を高速かつコンパクトに実現する手法としてダブル配列法[2]が提案されている．

ダブル配列法は，BASE，CHECK の 2 つの配列を用いてトライを実現する．文字‘a’に対する内部表現値を C(‘a’)とすると，g(s,‘a’)= t の遷移はダブル配列によって，以下の 2 式から定義される．

$$t = BASE[s] + C(‘a’) \quad (1)$$

$$CHECK[t] = s \quad (2)$$

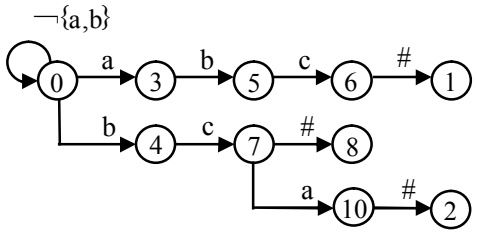
BASE には遷移先へのオフセットの値, CHECK には遷移元の状態番号が設定される．つまり，式(1)では BASE[s]の値と C(‘a’)の和で状態 t への遷移を決定し，式(2)では，s からの遷移が定義されていることを確認する．

3.2. ダブル配列を用いた goto 関数

図 2 にキー集合 K に対する，goto 関数にダブル配列を用いたマシン AC の構成とデータ構造を示す．

図 2.(a)の goto 関数では，終端遷移‘#’を付加することでトライの葉とキーを 1 対 1 に対応させ，キーの終端ノードの BASE 値に output 関数へのポインタ情報を負値で設定する．

図 2.(b)の各配列のインデックスはトライ中のノード番号に対応しており，文字の内部表現値は‘#’を 0, 後に述べる‘$’を 1, 文字‘a’～‘z’を 2～27 とする．



(a) goto 関数

	0	1	2	3	4	5	6	7	8	9	10
BASE	1	-1	-3	2	3	2	1	8	-2	0	2
CHECK	0	6	10	0	0	3	5	4	7	0	7
FAILURE	0	0	0	0	0	4	7	0	0	0	3

(b) goto 関数，failure 関数のデータ構造

-BASE[s]	output(-BASE[s])
1	{ “abc”, “bc” }
2	{ “bc” }
3	{ “bca” }

(c) output 関数のデータ構造

図 2: ダブル配列を用いた AC の構成，データ構造

また，配列 FAILURE には，各ノードからの遷移が fail となる場合の照合ポインタの移動先が格納されている．ただし，ダブル配列上では g(0,¬{‘a’, ‘b’})=fail となるため，配列 FAILURE によって初期ノードに遷移後，照合文字位置を 1 文字ずらすことで g(0,¬{‘a’, ‘b’})=0 を実現する．

図 2.(c)の output 関数では，終端ノードの BASE 値の負値をセットすることにより，検出されたキーを特定する．

3.3. failure 関数の組み込み

図 2.(b)の配列 FAILURE からも分かるように，failure 関数は全てのノードに対して照合ポインタの移動先のノードを格納しているが，その移動先が初期ノードの場合，配列 FAILURE を使用しなくても照合に失敗した場合は初期ノードに移動して照合を再開すればよいので, FAILURE の領域は無駄である．したがって，failure 関数のうち初期ノード以外へ移動するノードの情報さえ保持できればよい．これを実現するため，failure 関数をノードとして定義し，ダブル配列に埋め込む．以後，この failure 関数を埋

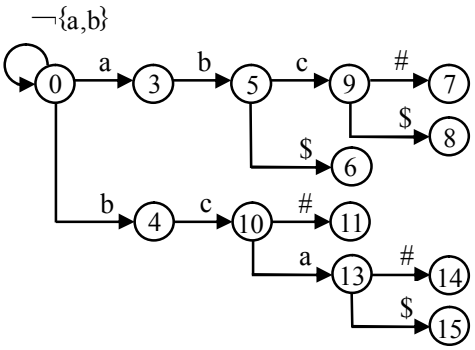


図 3: DA マシン AC の構成

め込んだ構造のマシン AC を DA マシン AC と呼び, その実現方法を説明する.

図 3 にキー集合 K に対する DA マシン AC の goto 関数, failure 関数の構成を示す. failure 関数として定義されたノードには文字'$'により遷移し, このノードのことを以後 failure ノードと呼ぶ. 図 3 において, ノード 6, 8, 15 が failure ノードとなっている. また failure ノードは, 初期ノード以外へ遷移するノードの情報を保持するために付加するので, 全てのノードに付加するわけではない. キーの照合時に failure ノードを持たないノードで照合に失敗した場合には, 初期ノードに戻って照合を再開する.

failure ノードの BASE 値には, failure 関数の遷移先であるノードの BASE 値を格納する. 例えば, 図 3 においてノード 5 では. f(5)=4 である. そこで, ノード 5 の failure ノードであるノード 6 の BASE 値にノード 4 の BASE 値を格納する. これによりノード 4 と 6 は同じ遷移先を持つことになる. なぜなら, 式 (1) からもわかるようにダブル配列の遷移は BASE 値と文字により決定するからである. つまり同じ BASE 値を持つノードにおいて同じ文字で遷移先を決めれば同じノードに遷移する. 同様にノード 8 と 10, ノード 15 と 3 にも同じ BASE 値を格納する. 図 4 に failure ノードを付加したダブル配列のデータ構造を示す.

	0	1	2	3	4	5	6	7	8	9	10
BASE	1	0	0	2	6	5	6	-1	11	7	11
CHECK	0	0	0	0	0	3	5	9	9	5	4

	11	12	13	14	15
BASE	-2	0	14	-3	2
CHECK	10	0	10	13	13

図 4: failure ノードを付加したダブル配列

以上の操作により, failure ノードから式 (1) で遷移先を決定するが, 式 (2) で遷移の確認ができなくなる. 例えば, failure ノード 8 から文字'a'でノード 13 に遷移する場合, CHECK[13]=10≠8 となる. つまり式 (2) では, ノード 8 からノード 13 への遷移を確認できない. そこで, 式 (2) を以下のように変更する.

$$CHECK[t] = C('a') \qquad (3)$$

ダブル配列では, CHECK に遷移元のノード番号を格納していたのに対して, 式 (3) は CHECK に遷移文字の内部表現値を格納している. これにより, 「どのノードから遷移してきたのか」という情報ではなく「どの文字によって遷移してきたか」という情報を保持することになるが, この場合でも BASE 値が重複しなければ遷移の確認は可能である [4]. 図 5 に以上の変更後のキー集合 K に対する DA マシン AC のデータ構造を示す. 図 5 のデータ構造において遷移を確認すると, ノード 13 では CHECK[13]=2 となり遷移文字'a'の内部表現値 C('a')=2 と一致することから, ノード 8 からノード 13 への遷移が確認できる.

また CHECK に格納する 1 要素のサイズは, ノード番号を格納する場合, ノード数の増加に対応するため通常 4byte 必要となるのに対して, 文字の内部表現値を格納する場合, 遷移文字の最大数から 1byte に抑えることが可能となる.

[例 2] 図 5 の DA マシン AC に対して, 文字列"abca" の検索をおこなう. なお output 関数は, 図 2.(c) を使用する.

まず, ノード 0 に対する BASE[0]=1 と C('a')=2 から, t=BASE[0]+C('a')=3 となる. そして, CHECK[t]=CHECK[3]=2 より C('a')=2 と一致し g(0,'a')=3 の遷移が確認できる. 同様に文字'b'と'c' に対して g(3,'b')=5, g(5,'c')=9 の遷移が確認できる. ここでノード 9 では, g(9,'#')=7 の遷移が定義

	0	1	2	3	4	5	6	7	8	9	10
BASE	1	0	0	2	6	5	6	-1	11	7	11
CHECK	0	0	0	2	3	3	1	0	1	4	4

	11	12	13	14	15
BASE	-2	0	14	-3	2
CHECK	0	0	2	0	1

図 5: DA マシン AC のデータ構造

されているので終端ノード 7 において output 関数から output(-BASE[7])=output(1)={ “abc”, “bc” }が検出される．ノード 9 から再開し，文字‘a’に対して g(9,‘a’)=fail となるため，g(9,‘$’)=8 により failure ノード 8 へ遷移する．ノード 8 からは g(8,‘a’)=13 と遷移し，ノード 13 では，g(13,‘#’)=14 があることから output(-BASE[14])={"bca”}が検出される．これで全ての文字列に対して検索をおこなったので終了する． (例終)

4. 実験と評価

提案する DA マシン AC のデータ構造に対して比較実験をおこなった．比較対象には，リスト構造，配列構造を用いたマシン AC を用いた．

実験においての動作環境は，Pentium D 3.40GHz とする. 実験に使用するキー集合は EDR 電子化辞書[4]から単語長が 3 以上の英単語 5 万件を無作為に選び出したものを使用する．また，検索の対象とする文字列は，英語の文書(1MB)とする．図 6,7 は各手法に対する記憶領域と検索時間の実験結果を示す．

図 6 より，提案手法の記憶領域は記憶量のコンパクト性に優れるリスト構造に比べ，記憶量が 56%程度コンパクトになっている．これは，リスト構造は 1 ノードあたりのデータサイズが兄弟遷移，子遷移，failure 関数などのポインタ情報を合計すると 17byte 必要なのに対し，提案手法では，BASE 値の 4byte と CHECK 値の 1byte を合計した 5byte で表現できるためである．また図 7 より，検索時間においても検索速度に優れる配列構造に対して 40%程度に抑えられている．これは，図 3,5 のダブル配列の構造のように各遷移前後のノードがダブル配列のインデック

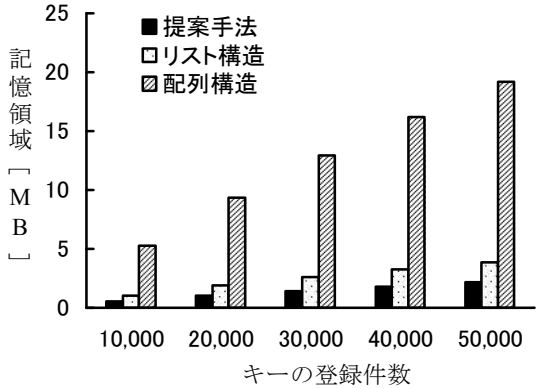


図6：記憶領域の実験結果

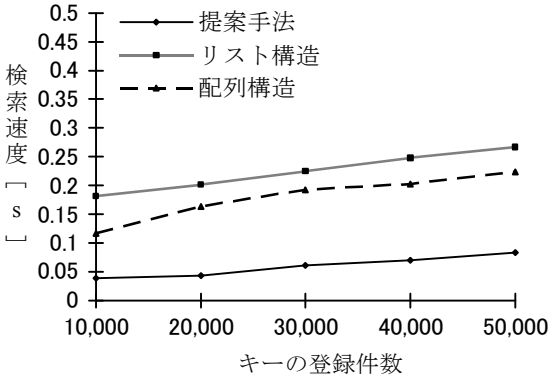


図7:検索速度の実験結果

ス上で近傍に存在する場合が多く，キャッシュのヒット率が上がったためだと考えられる．以上のことから提案手法は，複数文字列照合アルゴリズムを実現するための有効な手法であるといえる．

5. まとめ

本稿では高速性を高め，記憶量を抑えた AC 法のデータ構造を提案し，リスト構造，配列構造を用いたマシン AC との比較実験から本手法の有効性を示した．現在の問題点としては，キー長の短いキーが多く登録されると検索速度が低下するといったものがある．これは，単語長 1 のキーであればトライ中の多くのノードに output 関数に対応した終端遷移‘#’が発生し，検索時に output 関数をチェックする回数が多くなるためである．

今後，これらの問題の解決策を検討する．

参考文献

[1]北研二・津田和彦・獅々堀正幹．情報検索アルゴリズム. 共立出版 2002.

[2]青江順一．ダブル配列による高速検索ディジタル検索アルゴリズム．電子情報通信学会論文誌, Vol.J71-D, No.4, pp.1592-1600, 1988.

[3]Fredkin E. Trie memory. Commum.ACM, Vol.3,No.9, pp.490-500, September 1960.

[4]矢田晋・森田和宏・泓田正雄・平石亘・青江順一．ダブル配列におけるキャッシュの効率化．FIT2006. pp.71-72．2006.

[5]日本電子化辞書研究所. EDR 電子化辞書, 1996.