辞書検索に用いる有限オートマトンの構成と実装

伊東秀夫
リコー情報通信研究所
hideo@ic.rdc.ricoh.co.jp

1 はじめに

辞書検索は、検索キーとなる文字列の集合を決定性の有限オートマトン (DFSA) として表現し入力テキストと照合することで最適な速度で実行できる。しかし辞書が大規模化すると内部記憶に展開される DFSA の記憶量が実用上の問題となる。

一般に DFSA の記憶効率は、DFSA の構成法と実装法に依存する。従来の DFSA の構成法として Trie[4] や DAWG[2] などがあり、その効率的な実装法として単配置法がある [7]。

我々は、従来に比べコンパクトな DFSA の構成法として MDAWG (Minimal DAWG) を提案し、さらにその実装法を得るために単配置法を拡張した。日本語と英語の辞書を用いた実験により、その記憶効率等を検証する。

2 DFSA の構成法

従来の DFSA の構成法について、例として検索キー集合 $K = \{action, acted, abortion, aborted\}$ を用いて説明する。最初に K から図1の状態遷移図で表現される非決定性有限オートマトン (NDFSA) を構成してみる。図中で各ノードは状態を表し通番が付されている。またノード間の有向矢は状態遷移を表し遷移ラベルが付されている。

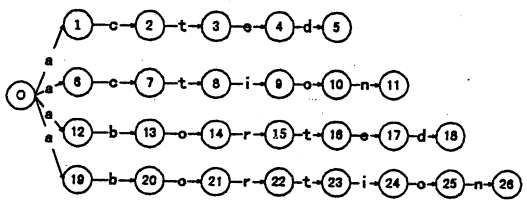

図 1: NDFSA

図1の NDFSA に対し、いわゆる determinization を施すことで構成される DFSA が図 2上の Trie[4] であり、加えて状態数最小化 (minimization) を施すことで構成される DFSA が図 2下の DAWG(Directed Acyclic Word Graph)[2] である。

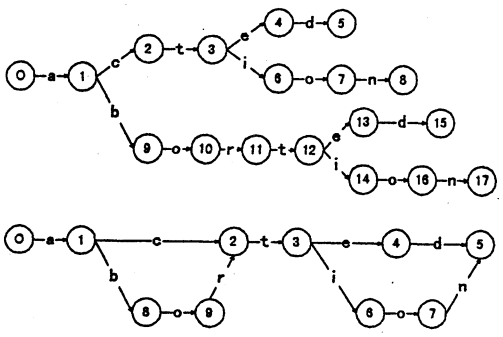

図 2: Trie (上) と DAWG (下)

また、Trie と DAWG に対し、分岐のない状態遷移パスを、文字列を遷移ラベルとする一つの状態遷移に置換すること (compaction) により図3の Patricia trie [6] と Compact DAWG (CDAWG) [3] が各々構成される。

以上説明した従来の DFSA の構成法の中では CDAWG が状態遷移数の最も少ないコンパクトな表現となる。

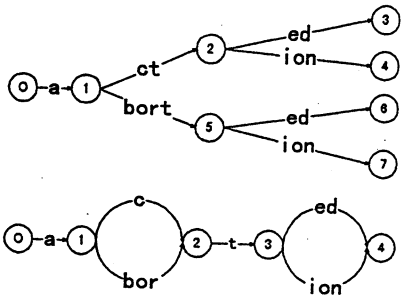

図 3: Patricia trie (上) と Compact DAWG (下)

3 MDAWG

前節の従来法に比べさらにコンパクトなDFSAを与える構成法としてMDAWG (Minimal DAWG)を提案する。図4にKに対するMDAWGを示す。

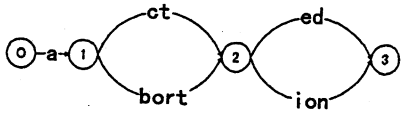

図 4: MDAWG

MDAWGによるDFSAの構成法の概略を図5に沿って説明する。まず検索キーの全体集合を初期状態である状態0の内部表現とする。状態0の要素に対し、各先頭文字毎に最長共通接頭 (longest common prefix) を求め、それを状態0からの遷移ラベルとする。図5では、先頭文字は"a" のみであり、"a" に関する最長共通接頭は"a" となるため、それを遷移ラベルとする。次に遷移ラベル"a" を接頭とする状態0の各要素に関し、その接頭を除いて得られる接尾を要素とする集合を遷移先の状態1の内部表現とする。図5では、遷移ラベル"a" による遷移先である状態1の内部表現は $\{cted, ction, borted, bortion\}$ となる。同様に状態1からは2つの遷移ラベル"ct" と"bort" による遷移が可能であることがわかる。各々の遷移先状態の内部表現は $\{ed, ion\}$ となり一致する。このように内部表現が一致した場合、それらの状態を一つに併合する。新しい状態が生成されなくなるまで、以上の操作を繰り返すことで最終的に前図4に示すMDAWGが得られる。

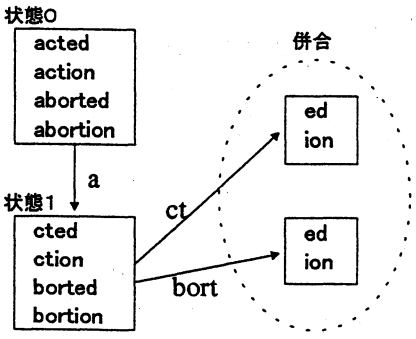

図 5: MDAWGの構成法

MDAWGの特長は次の3点である。

特長1 従来法に比べ最もコンパクトなDFSAを構成できる。

特長2 CDAWGのように中間的にDAWGを経ることなく直接DFSAを構成できる。

特長3 辞書検索の過程で、後続し得る最長の文字列を予測できる。

特長1に関連する事柄として、遷移ラベルの記憶について述べる。図3,4のCDAWGとMDAWGを、状態数と遷移数に関し比較するとMDAWGの方が少ないが、遷移ラベル長の合計はMDAWGの方が"t" の重複分だけ逆に多い。この問題は、遷移ラベル集合を図6に示す逆木構造により共通する接尾を併合し記憶することで解消できる。CDAWGでは、この接尾併合が遷移元状態のみに関し局所的に行なわれるが、逆木構造による接尾併合は全遷移ラベルに関する大局的なものとなるため、遷移ラベルと状態遷移の総体 (DFSAの記憶量) はMDAWGの方が少ない。

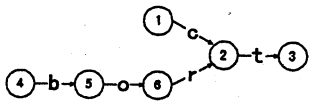

図 6: 遷移ラベルの接尾併合

また特長2により、CDAWGに比べDFSAの構成に要する記憶および時間を節約でき、またアルゴリズムも単純になる。

最後の特長3は、MDAWGが最長共通接頭を遷移ラベルとすることに因る付随的な機能で、辞書検索の応用系として例えばテキスト入力を想定した場合、入力の途中で予測し得る後続文字列を自動補間すること等に使える。

4 DFSAの実装法

DFSAの状態遷移は、二次元の状態遷移表として実装できるが、この表はスパース (疎) になり無駄な記憶が多い。

また DAWG 型の DFSA は、Trie のように各検索キーと木構造の葉が一対一対応することを利用して付属するレコードを得ることができない。

Revuz による DFSA の実装法 [5] は、単配置法 (single displacement method)[7] を基礎とするダブル配列法 [1] と類似した方法で状態遷移表の記憶効率を改善し、さらに、DAWG 型の DFSA において検索キーとレコード情報を一意に対応させることができる。以下に、この方法について図 7 を用いて説明する。

図 7の上表は、前頁の図 4の MDAWG の状態遷移表である。ただし遷移ラベルが文字列の場合は、その先頭文字で表してある。遷移ラベル"$" は検索キーの末尾を表す。Revuz の方法を適用すると、図中の下側に示す三種類の配列 $(label, base, order)$ が得られる。

この内、$label, base$ 配列は状態遷移表の圧縮表現に相当する。各状態 x には固有の配列位置 x' が対応し、各遷移ラベル c には内部コード c' が割当てられている。$label[s' + c']$ には c' が格納され、これを用い遷移可能性をチェックする。状態 s からの遷移ラベル c による遷移先状態の配列位置が $base[s' + c']$ に格納されている。

入力テキスト"ae" を例に、辞書検索の過程をトレースしてみる。ただし、配列位置は 0 起算、内部コードは 1 起算の通番とし、初期状態 s0 の配列位置は 0 と定める。まず $label[s0' + a'] = label[0 + 1] = 1 = a'$ により遷移可能とわかり $base[s0' + a'] = base[0 + 1] = 1$ により遷移先状態 s1 の配列位置として 1 を得る。次に $label[s1' + e'] = label[1 + 4] \neq 4 = e'$ により遷移が失敗し"ae" は検索キーの集合 K に含まれないことがわかる。

	a	b	c	e	i	$
状態 0	1					
状態 1		2	2			
状態 2				3	3	
状態 3						F

↓

label		1		2	3		4	5		$
base		1		2	2		3	3		
order		0		2	0		0	1		

図 7: Revuz の実装法

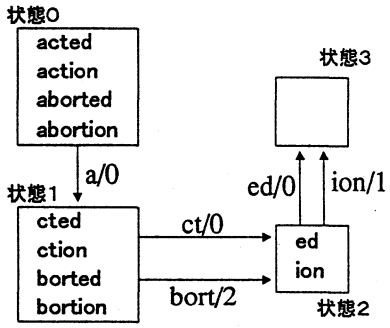

図 8: 順位数の割当て

$order$配列には、図 8に示すように各状態遷移に割当てられた数 (順位数と呼ぶ) が格納される。初期状態を表現する検索キーの集合は予め辞書式順序に並べておき、DFSA の構成時に各遷移が求まる毎に、その遷移ラベルより若い辞書式順位を持つ遷移元の要素の数を、その遷移の順位数として記憶しておく。そして辞書検索時には、終了状態に至った遷移パス上の順位数を加算することで対応する検索キーの辞書式順位が求まる。例えば $abortion$ に対しては 0 + 2 + 1 = 3 により 0 起算の辞書式順位は 3 になる。各検索キーに対し辞書式順位は一意なので、この数を用いてレコード情報を特定できる。

5 単配置法の拡張

単配置法を基礎とするダブル配列法や Revuz の方法は、遷移ラベルとして文字を前提としている。一方、MDAWG のように遷移ラベルとして文字と文字列の両方が混在する場合、辞書検索時に、それらの切替えをどのように検知すればよいかという問題が生じる。この問題を解消するために以下のように Revuz の実装法を拡張する。

まず、文字列 s の内部コードを $c + |\Sigma|$ によって割り当てる。ただし c は s の先頭文字の内部コードであり、$|\Sigma|$ は文字種の総数を表す定数である。これにより一回の加算という低コストな操作により、文字の内部コードと重ならないように任意の文字列に対し内部コードが得られる。

MDAWG の構成時には、文字か文字列かの違いに応じて遷移ラベルの内部コードを切替え、Revuz

の実装法に従って *label*, *base*, *order*配列の値を求めればよい。ただし図9に示すように、遷移ラベルが文字列の場合、*base* 配列には文字列の先頭以外の部分 (これは図 6のように記憶圧縮してもよい) と本来の base 配列値 (遷移先の配列位置) との組を格納した記憶場所への参照値を格納する。

辞書検索時には、各状態において文字 c と文字 c を先頭とする文字列が同時には遷移ラベルにならない点に注目し、まず文字による遷移を調べ、それが失敗した時にのみ、文字列による遷移を調べる。文字列遷移ラベルにより遷移可能かどうかは *label*配列によるチェックと、遷移ラベルの先頭文字以外の部分が入力テキストの後続部と合致するかどうかで判断される。

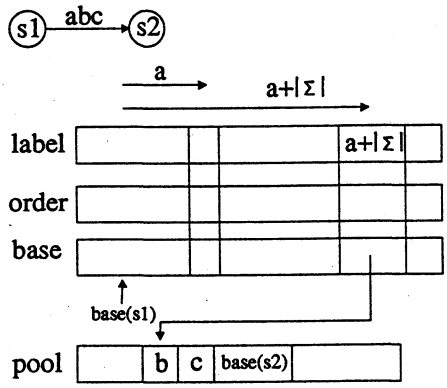

図 9: 単配置法の拡張

6 実験

辞書 (検索キー集合) を MDAWG または Trie に変換し実行形辞書として出力する辞書コンパイラと、実行形辞書と入力テキストを照合する辞書検索プログラムを C 言語で作成し、以下の英語と日本語の辞書を用いた各種の計量実験を行なった。使用計算機は PC(Pentium 133Mz,40MB 主メモリ) である。

辞　書	検索キーの総数	総容量 (KB)
英　語	98667	1029
日本語	101459	747

辞書コンパイラが MDAWG の実行形辞書を得るために要した計算時間は英語 (119sec), 日本語 (175sec) であった。Trie と MDAWG の実行形辞書の容量と、辞書検索時の時間効率を以下に示す。

辞　書	辞書容量 (KB)		速度 (msec/MB)	
	Trie	MDAWG	Trie	MDAWG
英　語	1891	916	917	1013
日本語	1498	829	1089	1146

辞書容量に関し、MDAWG は Trie の約半分になることがわかる。Trie は *base*, *label* 配列のみで実装され、MDAWG はそれらに加えて *order* 配列と遷移ラベル集合の記憶を必要とするが、状態遷移数の差が、それを補う以上の効果を上げている。

辞書検索の時間効率は、MDAWG の方が遷移ラベルの種別検査 (文字か文字列か) が加わるため劣るが、その差はわずかである。

7 おわりに

高速でコンパクトな辞書検索を可能とする DFSA の構成法として MDAWG を提案しその実装法を与えた。今後の課題として、準静的辞書 (比較的少数の検索キーの追加削除) への対応と、テキストインデキシングへの応用 (suffix automaton 等) が考えられる。

参考文献

[1] 青江, ダブル配列による高速ディジタル検索アルゴリズム, 信学論 (D), J71-D, 9, 1592-1600, 1988-09

[2] Blumer, A., et.al., The smallest automaton recognizing the subwords of a text, Thoret. Comput. Sci.,Vol.40, 31–55,1985

[3] Crochemore,M. Rytter W., Text Algolithms, Oxford Univ.Press, 1994

[4] Fredkin, E., Trie memory, Comm. ACM 3, 490-499, 1962

[5] Revuz, D., Dictionnaires et Lexiques, Methodes et Algorithmes, Ph.D.thesis, Universite Paris 7, 1991

[6] Knuth, D.E., The Art of Computer Programming, Vol. 3, Sorting and Searching, 1973

[7] Tarjan,R.E.and Yao A.C., Storing a sparse table, Comm.ACM, Vol.22,No.11,606-611,1979