D-010

関数辞書を用いたダブル配列の圧縮手法
Compression Method of a Double-Array using the Functions Dictionary

土井 優太† 森田 和宏† 泓田 正雄†
Yuta Doi Kazuhiro Morita Masao Fuketa

1. はじめに

トライ法は，トライというキーを構成する文字を分岐条件とする木構造を用いたキー検索法である．トライ法の特徴は，前方一致した文字列を併合できる点と，格納するキー集合に依存しない検索である．これらの特徴から，形態素解析や文章校正の辞書，文章検索の索引など，自然言語処理を中心として幅広く用いられている．

キー集合 K={"ao","aoao","aoba","oba"}に対するトライを図 1 に示す．このとき，'#'は単語の終端を表す．

トライを表現するデータ構造はいくつか存在する．中でも検索速度の面で優れているのがダブル配列である．しかし記憶効率の面では，同じくトライを表現するデータ構造である LOUDS に劣る．これまで，ダブル配列の検索速度を維持した圧縮アプローチは多く提案されている．本研究では，新しい遷移手法を用いて，これまでの圧縮手法よりも優れた記憶効率を持つダブル配列を提案する．

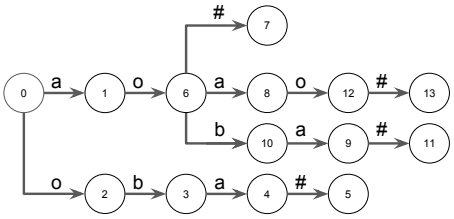

図 1 キー集合 K に対するトライ

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
BASE	0	4	0	3	5	-1	7	-1	10	11	8	-1	13	-1
CHECK		0	0	2	3	4	1	6	6	10	6	9	8	12

図 2 キー集合 K に対するダブル配列

2. トライを表現するデータ構造

2.1 ダブル配列によるトライの表現

ダブル配列[1]は，BASE と CHECK という二つの一次元配列を用いて，トライを表現するデータ構造である．ノード s からノード t へのラベル c による遷移は 以下の式により実現する.ここで関数 CODE は，ラベル c に対応する内部表現値が出力される.

$t := BASE[s]+CODE(c)$, $CHECK[t] = s$ (式 1)

ダブル配列は，上記の遷移式が成立するように BASE と CHECK の値を探索し格納することで，トライを表現する．

図 1 のトライに対するダブル配列を図 2 示す．ダブル配列内の-1 は，終端を表してる．ラベルの内部表現値は，CODE('#')=0，CODE('a')=1，CODE('o')=2，CODE('b')=3 と定義する.

2.2 LOUDS によるトライの表現

LOUDS[2]は，簡潔ビットベクトルを用いて効率的に木構造を表現するためのデータ構造である．木構造のルートから幅優先で木を辿り，遷移先ノードの数と同数の 1 とその後ろに 0 を格納することで遷移先のノードの数を表現している.

LOUDS によるトライの表現は，LOUDS を用いた木構造と，分岐条件となる文字をノードに対応した形で格納した配列の二つで構成されている．遷移処理は，簡潔ビットベクトルを用いた rank，select という操作により実現する．どちらの操作も定数時間で処理することが可能である.

3. ダブル配列の圧縮手法 (XCDA)

XCDA[3]は，DACs[4]という固定長配列の各要素の整数を可変長コードに変換する仕組みを用いて，BASE と CHECK を効率的に格納する圧縮手法である．DACs は，rank 辞書と Vbyte 符号を組み合わせた手法で，各要素が可変長コードの配列に対してランダムアクセスが可能である．したがって，BASE と CHECK に適用することで，従来と同様の遷移処理を維持し，圧縮ができる.

XCDA では，各配列に対して，可変長コードの恩恵を受けるために，インデックス番号と配列要素を xor 演算することで，多くの要素で小さな整数を含む配列に変換している．さらに，この演算の恩恵をより受けるために，BASE 値を決める際にも工夫を加えている.

XCDA では，キー集合によるが，BASE と CHECK の各配列の約 90%以上の要素を 1byte で表現することができる.

4. 提案手法(XXDA)

XCDA の記憶効率は，従来のダブル配列に比べると向上しているが，LOUDS と比べると劣っている．本研究では，関数辞書を用いた新しいダブル配列の圧縮手法を提案する.

関数辞書は，遷移先を算出する関数を格納した辞書である．格納された関数には，識別番号(インデックス番号)が付けられており，識別番号から目的の関数を呼び出すことができる．提案手法の BASE 値には，関数の識別番号が格納されており，遷移の際は，関数辞書から BASE 値が示す関数を呼び出し，遷移先の算出に用いられる．CHECK 値にも関数の識別番号が格納されており，遷移の確認として使用する.

提案手法で使用する遷移式を以下に示す.

$t:=FDict(BASE[s],s)\hat{}CODE(c)$, $CHECK[t]=BASE[s]$ (式 2)

FDict は，関数辞書から関数を呼び出す関数である．第一引数で，呼び出す関数の識別番号を与え，第二引数で，呼び出した関数への引数を与える.

ここで，BASE と CHECK 配列を長さ r 間隔で分割し，その分割した要素をブロックと呼ぶとする．r は，キー集合内の文字の最大内部表現値 n から．$r = 2^{\lfloor \log_2 n \rfloor}$で算出する.

† 徳島大学 Tokushima University

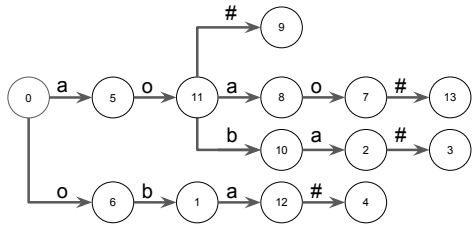

図 3　キー集合 K に対して提案手法を適用したトライ

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
BASE	0	2	4	-1	-1	0	1	1	5	-1	3	1	1	-1
CHECK		1	3	4	1	0	0	5	1	1	1	0	2	1

図 4　キー集合 K に対して提案手法を適用したダブル配列

提案手法の遷移式では，FDict の戻り値と遷移文字との xor 演算で遷移先を算出しているため，FDict の戻り値と遷移先は，同じブロックを示すことになる．この特性から，「同じブロックへの遷移には，同一の関数を使えない」という条件で，ダブル配列を構築することで，誤った遷移を受け付けないトライを表現できる．

提案手法の BASE と CHECK は，関数辞書の識別番号が格納されているため，要素のサイズは，関数辞書の種類に依存する．関数辞書の種類が少ないほど，提案手法は圧縮することができる．

キー集合 K に対して提案手法を適用したトライを図 3 に示し，図 3 のトライに対して提案手法を適用したダブル配列を図 4 に示す．CODE の値は，図 2 のダブル配列と同様である．使用する関数辞書内の各関数を表 1 に示す．また，関数辞書用いられる Xos 関数について以下で定義する．ここで，M は，トライのノード数 x から，$M = 2^{\lfloor \log_2 x \rfloor}$ で算出する．

[Xos(s , b)]
begin
  if b >= 0 then return s ^ ((s << b) & M);
  if b < 0 then return s ^ ((s >> -b) & M);
end

5. 評価実験

提案手法 XXDA を適用したダブル配列を評価するために，二つのキー集合(WordNet3.0 と郵便番号)を使用し比較実験を行なった．キー集合の詳細は，表 2 に示し，実験結果を表 3 に示す．比較対象は，従来のダブル配列 DA(2.1 節)，LOUDS[5](2.2 節)，XCDA[6](3 節)とする．比較項目は，トライの性能を左右する記憶領域とキー検索の速度とする．実験環境は，CPU が Quad-Core Intel Xeon の 2.4GHz，メモリが 16GB のマシンで行なった．

XXDA の検索速度は，手法の中で，3 番目の結果となった．これは，呼び出す関数によっては，検索速度に影響する関数が含まれているためである．しかし，LOUDS に対しては，4 倍以上の速度が確認できるため，ダブル配列の特徴である検索速度の高速性は維持している考えられる．

記憶領域は，LOUDS には劣っているが，どちらのキー集合に対しても，関数の種類は 255 以下で構築できたため，各要素を 1byte で表現できる．したがって，XCDA と比べて全ての配列要素で 1byte を実現してる提案手法の方が記憶効率において優れいていることがわかる．

表 1　図 3 の提案手法で用いる関数辞書

0	s + 4
1	Xos(Xos(s , 1), 2)
2	s + 12
3	Xos(Xos(s , -1), 2)
4	Xos(Xos(s , -1), 4)
5	Xos(Xos(Xos(s , -1) , -3), 3)

表 2　評価実験で用いるキー集合の詳細

	郵便番号	WordNet-3.0
キー総数	120,020	147,306
ノード数	146,525	732,257
平均文字列長	6.9	11.5
文字の種類	10	41

表 3　評価実験の結果

	郵便番号	WordNet-3.0
記憶領域 (MB)		
DA	2.034	6.711
XCDA	0.714	2.213
LOUDS	**0.212**	**1.058**
XXDA	0.539	1.768
検索速度 (μs / str)		
DA	**0.086**	**0.201**
XCDA	0.278	0.422
LOUDS	1.187	2.382
XXDA	0.286	0.487

6. おわりに

本項では，関数辞書を用いた手法により，ダブル配列の検索速度を維持した状態で，高い記憶効率で実現できることを示した．今後は，関数辞書を構成する関数に対する考察が不十分なため，検索速度，記憶領域の面で最適な関数についての選定を試みる．

参考文献

[1] 青江順一, ”ダブル配列による高速ディジタル検索アルゴリズム”, 電子情報通信学会論文誌 D 情報・システム, Vol.71, No.9, pp.1592-1600 (1989)
[2] G.Jacobson, “Space-efcient static trees and graphs.”, In Proc. 30th, pp.549-554(1989)
[3] S.Kanda, K.Morita, M.Fuketa,”Compressed double-array tries for string dictionaries supporting fast lookup.”, Knowl Inf Syst, Vol.51, No.3, pp.1023-1042(2017)
[4] N.Brisaboa, S.Ladra, G.Navarro,”DACs: bringing direct access to variable-length codes.”, InfProcess Manag, Vol.49, No.1, pp.392-404(2013)
[5] Tx: Succinct trie data structure. https://code.google.com/archive/p/tx-trie/
[6] XCDAT: https://github.com/kampersanda/xcdat