D-006

Xorshift を用いたダブル配列の圧縮手法
Compression Method of a Double-Array Using Xorshift

土井 優太† 森田 和宏† 神田 峻介† 泓田 正雄†
Yuta Doi Kazuhiro Morita Shunsuke Kanda Masao Fuketa

1. はじめに

トライ法[1]は，自然言語を中心に広く用いられるキー検索法であり，トライを実現するデータ構造として，ダブル配列[2][3]がある．ダブル配列は，二つの一次元配列を用いた手法で，キーの検索が高速という特徴を持つ．ダブル配列は，各配列の要素にトライノード番号と対応した値を格納するが，トライノード数は，キーの総数に依存するため，大規模なキー集合を格納する場合，記憶効率は低下する．本稿では，Xorshift という演算を用いて，ダブル配列の各要素にトライノード番号に対応しない値を格納することで，記憶領域を圧縮する方法を提案する．

2. ダブル配列

ダブル配列は，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．ダブル配列では，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)　(式 1-1)

CHECK[t] = s　(式 1-2)

式 1-1 では，BASE[s]とラベル c の内部表現値との和で遷移先 t が算出され，式 1-2 が成立することで正しい遷移であることを確認できる．

BASE の各要素には，遷移先のインデックス番号とラベル c の内部表現値の差が格納され，CHECK の各要素には，遷移元のインデックス番号 s が格納される．ここで，ダブル配列上のインデックス番号はトライノード番号と一対一対応するため，各配列の要素はトライのノード番号に対応した値が格納される．

キー集合 K={“ag”,”age”,”bad”,”badge”,”bot”}に対するダブル配列を図 1 に示す．このとき，'#'は単語の終端を表す．

3. Xorshift と x パターン

Xorshift[4]は，擬似乱数生成器として使用されており，ビットシフトと排他的論理和のみで演算が構成されているため，他の擬似乱数生成器と比べて高速に乱数を生成できる．本稿では，Xorshift を擬似乱数生成器ではなく，トライの遷移を行うために用いる．

入力 s を右または左にビットシフトした値と s の排他的論理和を Xorshift の基本演算とし，関数 xos に示す．

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

関数 xos の引数である変数 b が正値なら左に b シフトし，負数なら右に-b シフトする．ビットシフトは，与えられた

†徳島大学先端技術科学教育部, Tokushima University

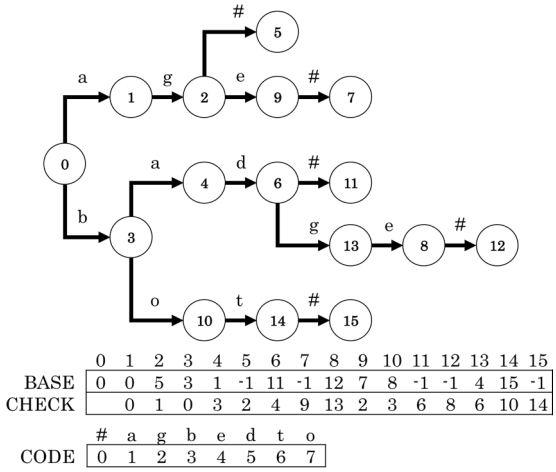


図 1 ダブル配列

データのビットサイズを超えるシフトを行うと超過分は無視される．この性質をアルゴリズムで反映させるため．マスク値として M を設定する．

基本演算の出力を入力とし基本演算をもう一度行う．この繰り返しを 1 回以上行う演算が Xorshift である．基本演算で使用したシフト数とシフト方向によって，Xorshift の出力は変動する．本稿では，Xorshift の演算で用いるシフト回数とシフト方向の集合を x パターンと呼称する． 入力を s，x パターンを B=(b_1,b_2,b_3)とすると Xorshift は，XOS(s , B) = xos(xos(xos(s , b_1) , b_2) , b_3)となる．

4. 提案手法

本手法では，ダブル配列の各要素にトライノード番号に対応しない値を格納することで，大規模なキー集合に対しても効率的に圧縮する手法を提案する．

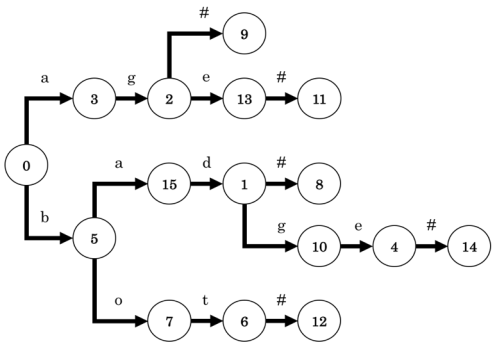
提案手法では，従来のダブル配列で用いられる BASE と CHECK に加え，x パターンを格納した配列 PT を用いてトライを表現する．提案手法での BASE に格納する値は，子ノードへ遷移する x パターンが格納された PT のインデックス番号，CHECK に格納する値は，親ノードを特定するために遷移元が使用した x パターンが格納された PT のインデックス番号である．したがって各要素には，トライノード番号ではなく PT のインデックス番号が対応するため，記憶領域は PT に格納された x パターンの数に依存する．x パターン数の変動については実験により検証する．

提案手法のダブル配列では，ノード s からノード t へのラベル c による遷移を以下のように表現する．

t := XOS((s<<8)+CODE(c) , PT[BASE[s]])　(式 2-1)

CHECK[t] = BASE[s]　(式 2-2)

式 2-1 は，前節で説明した XOS 関数を利用している．一つ目の引数は，ノード番号 s を左に 8 ビットシフトした値と CODE 値の和である．この処理により XOS の入力値



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
BASE	0	3	5	4	8	1	7	6	-1	-1	6	-1	-1	9	-1	2
CHECK		2	4	0	6	0	6	1	3	5	3	9	7	5	8	1

			#	a	g	b	e	d	t	o
CODE			0	1	2	3	4	5	6	7

PT		
	0	(1)
	1	(1, -2, -4. 10)
	2	(2, 4, 9)
	3	(10, -5, 5)
	4	(7, 8)
	5	(3, -6, 3, -9, 6)
	6	(7, 9, 10)
	7	(-7, 7)
	8	(-1, 2, -8, 8)
	9	(-5, -10, 5)

図 2 提案手法

にすべてのトライのノードにおいて独立した値を渡す．XOS では，同一の x パターンを使用した場合は一対一写像になるため，XOS の引数が常に独立であれば，同一の x パターンを使用して異なる遷移元から同じ遷移先を示すことはない．これにより式 2-2 が有効となり確認処理を行うことができる．

キー集合 K に対する提案手法を図 2 に示す．関数 xos で用いる M は，2^{12}-1 である．各要素の記憶領域は，PT 配列に格納されている x パターン数は 10 であるため， 4bit で表現できる．

5. 評価

5.1 x パターンの変動

従来手法では，各要素の記憶領域は，トライノード数に依存したが，提案手法では，x パターン数に依存する．そこで，キー集合数の増加による x パターン数とトライノード数のそれぞれの関係を実験により検証する．キー集合は，英語の Wikipedia のページタイトルからランダムに 10 万語から 100 万語とした．実験環境は CPU が Xeon(R) E5540 の 2.53GHz，メモリが 32GB のマシンで行なった．

図 3 にキー集合数と x パターン数，トライノード数の関係を示す．結果から，x パターン数，トライノード数ともにキー集合数に比例して増加していることがわかる．しかし，比例係数は，トライノード数が 12.56 に対して，x パターン数は 0.0067 であるため，x パターン数の方が大幅に少ないことがわかる．このことから，提案手法では，キー集合数が増加しても，各要素の記憶領域は，従来手法に比べ圧縮できると考えられる．

5.2 圧縮率

提案手法を適用したダブル配列が従来のダブル配列と比べ記憶領域が小さくなっていることを実験により検証する．また，その際のキーの検索速度についても検証する．使用

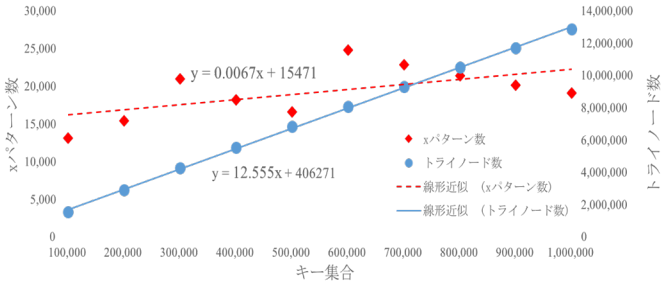


図 3 x パターンとキー集合数の関係

表 2 キー集合詳細

	JapaneseWiki	EnglishWiki	uk-urls
キー総数	1,700,943	13,179,334	39,459,925
ノード数	18,641,080	128,076,946	748,571,709
平均文字列長	21.5	20.4	71.4

表 1 実験結果

	JapaneseWiki	EnglishWiki	uk-urls
総サイズ (MB)			
DA	149.1	1,024.6	5,988.6
XSDA	77.8	582.3	3,398.1
(xパターン数)	(25,218)	(39,739)	(36,387)
検索速度 (μs /key)			
DA	0.808	0.934	2.783
XSDA	2.235	2.706	8.712

したキー集合を表 1 に示す．実験環境は，前節と同じマシンを使用する．従来のダブル配列を DA と表し，各要素の記憶領域は，32bit とする．提案手法を XSDA と表し各要素の記憶領域は，x パターン数で決定する．

表 2 にキー集合ごとの結果を示す．全てのキー集合に対して，x パターン数を 2^{16} 以下で構築できたため，各要素の記憶領域は 16bit となり，約 50%の圧縮に成功した．総サイズに関しても，約 55%の圧縮に成功した．検索速度は約 3 倍低速になっているが単位が μs であることから，実用的な速度であると考えられる．

6. おわりに

本稿では，Xorshift を用いた手法により，ダブル配列の各要素にトライノード番号に対応しない PT のインデックス番号を格納することで，大規模なキー集合に対しても効率的に圧縮できることを示した．今後は，x パターン数を減らすことでさらなる圧縮を試みる．

参考文献

[1] 青江順一, ”トライとその応用(<連載講座>キー検索技法 4)”, 情報処理, Vol34, No.2, pp.224-251 (1993)
[2] 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)
[3] J.Aoe, “An Efficient Digital Search Algorithm by Using a Double-Array Structure”, IEEE Transctions on Software Engineering, Vol.15, No.9, pp.1066-1077 (1989)
[4] Marsaglia G, “Xorshift RNGs”, Journal of Statistical Software, Vol.8, No.14, pp.1-6 (2003).