LD-008

ダブル配列におけるキャッシュの効率化
Cache-Efficienct Double-Array

矢田 晋 Susumu Yata　森田 和宏 Kazuhiro Morita　泓田 正雄 Masao Fuketa　平石亘 Wataru Hiraishi　青江 順一 Jun-ichi Aoe
徳島大学工学部　Faculty of Engineering, Tokushima University

1. はじめに

辞書からキーを検索するという処理は，コンパイラ，索引検索，フィルタリング，形態素解析などの様々な分野で必要となるため，計算機処理における基礎技術の一つとされている [1]．特に，文字単位で照合をおこなうトライは，理論的な検索時間がキーの長さで抑えられる，入力に前方一致するキーを容易に検出できるなどの理由から，自然言語辞書を中心として幅広く利用されている．このトライを実現するデータ構造として，検索の高速性とコンパクト性という特長を併せもつダブル配列が存在する [2]．

ダブル配列に関する研究は，追加や削除の効率化 [3, 4, 5] を目的としたものが中心であり，検索の高速化については，対象を共起関係などに限定している [6]．これは，ダブル配列における理論的な検索時間がキーの長さに比例するためである．しかし，アーキテクチャの複雑化にともない，単純にアルゴリズムの性能を測ることはできなくなっている．

本稿では，ダブル配列の検索時間において，キャッシュミスがボトルネックになることを述べ，検索を高速化する手法を提案する．また，日英単語をキー集合とした実験結果により，提案法は検索を最大で 2 倍以上に高速化できることを示す．

2. ダブル配列

ダブル配列は，2 つの一次元配列 BASE と CHECK を使用して[1]，トライにおけるノード s から t への遷移を次のように定義する．なお，s, t はそれぞれ整数として取り扱われており，c は遷移文字の内部表現値を表している．

$$t = \mathrm{BASE}[s] + c; \quad \mathrm{CHECK}[t] = s \qquad (1)$$

一般に，最終分岐以降の遷移については，遷移文字を連結して配列 TAIL に格納する．また，葉ノードの BASE を負にして TAIL へのリンクとして使用し，葉ノードの判別と接尾辞の照合を可能とする．これにより，接尾辞を文字列として照合できるため[2]，検索を高速化することができる．

[例 1] キー集合 $K = \{$“*ace*”, “*babe*”, “*badge*”, “*be*”$\}$ に対するダブル配列を**図 1** に示す．‘#’ は文字列の終端を表しており，遷移文字の内部表現値は，‘#’ を 0，‘*a*’ ～ ‘*z*’ を 1 ～ 26 としている．（例終）

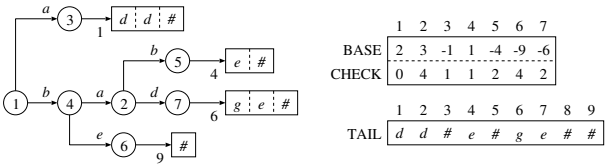

図 1 キー集合 K に対するダブル配列

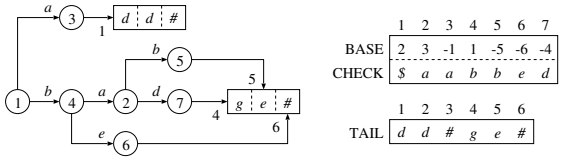

図 2 キー集合 K に対する圧縮ダブル配列

ダブル配列における検索では，遷移を $O(1)$ の時間計算量で実現できるものの，データアクセスがランダムアクセスであるため，キャッシュミスがボトルネックとなりやすい[3]．すなわち，キャッシュを効率化することで，ダブル配列の検索を高速化することができる．

3. ダブル配列におけるキャッシュの効率化

3.1 提案法の概要

本稿では，ダブル配列の検索においてキャッシュを効率化するために，ダブル配列を圧縮する手法と，キャッシュラインを利用する手法を提案する．

3.2 ダブル配列の圧縮によるキャッシュの効率化

ダブル配列は，葉ノードを除くあらゆるノードの組 $\{s, t\}$ において，$\mathrm{BASE}[s] \neq \mathrm{BASE}[t]$ が成立するとき，遷移文字を用いて遷移を確認することができる．そのため，遷移元のノード番号 (4bytes) の代わりとして，遷移文字 (1byte)[4]を CHECK に格納することにより，ダブル配列を圧縮できる．このとき，ダブル配列の各要素を 4bytes 境界に揃えるため，BASE には 3bytes の整数を割り当てる．以上の圧縮により，各要素のサイズは半減する．さらに，後方一致する接尾辞を併合することで，TAIL を圧縮することができる．

[例 2] キー集合 K に対するダブル配列を圧縮したものを**図 2** に示す．‘*$*’ はキー集合及び入力で使用されない文字を表している．CHECK に遷移文字を格納し，接尾辞 “ ”, “*e*” を接尾辞 “*ge*” に併合することで，ダブル配列を圧縮している．（例終）

[1]実装には，BASE と CHECK からなる要素の配列を用いる．
[2]接尾辞が連続空間に配置されることの影響が大きい．
[3]高速な CPU は，キャッシュミスで 100cycles 以上をロスする．
[4]文字列をバイト列として扱うことを想定している．

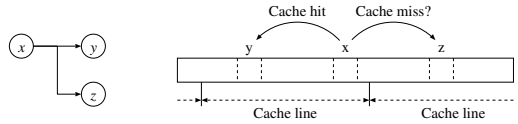

図 3 ダブル配列における遷移とキャッシュラインの関係

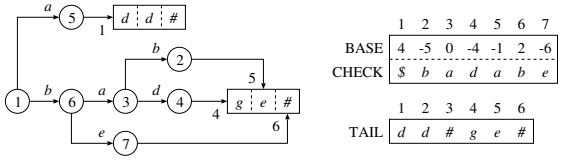

図 4 キー集合 K に対する排他的論理和を用いたダブル配列

3.3 ダブル配列におけるキャッシュラインの効率化

キャッシュミスが発生したとき，基本的にキャッシュライン単位[5]でメモリとの転送がおこなわれる．そのため，遷移元と遷移先のノードを同一ライン上に配置することにより，さらなるキャッシュミスを防ぐことができる．なお，ラインの確認を容易にするため，排他的論理和 $\oplus$ を用いて式 (1) を次のように変更する[6]．

$$t = \mathrm{BASE}[s] \oplus c; \quad \mathrm{CHECK}[t] = c \qquad (2)$$

従来のダブル配列では，前方の未使用要素から順に利用する．これに対し，遷移元と同一ライン上の未使用要素を優先して利用することにより，提案法はキャッシュの効率化を図る．

[例 3] 図 3 に示すダブル配列では，x から y への遷移でキャッシュヒットが保証されるものの，x から z への遷移において，キャッシュミスの可能性がある．また，図 2 のダブル配列に対して排他的論理和を導入したものを**図 4** に示す．（例終）

4. 評価

提案法の有効性を実証するために比較実験をおこなった．実験環境の詳細[7]を**表 1** に示し，キー集合の詳細を**表 2** に示す．実験では，従来法と提案法を用いてダブル配列を静的に構築し，キー集合に含まれているキーをランダム順に検索した．

表 3 は，提案法の適用前後における，各キー集合に対するダブル配列の空間使用量と構築時間及び検索時間を表している．また，**図 5** は，キー数と検索時間の関係を表している．これらの実験結果より，提案法を適用することで，構築時間は長くなるものの，検索時間を短縮できることがわかる．ただし，検索時間の短縮率は実験環境やキー数に大きく影響を受ける (1 ∼ 55%)．

表 1 実験に用いた CPU の詳細

CPU	Clock	L1 size	L2 size	Line size
Via Eden	600MHz	64KB	64KB	32bytes
Intel P4	3.2GHz	16KB	1024KB	64bytes

[5]ラインサイズは 32, 64bytes が主流である．
[6]圧縮を適用しているので CHECK$[t] = c$ となっている．
[7]Vine Linux 3.2 において dmesg により確認した．

表 2 実験に用いたキー集合の詳細

Source	Number of keys	Length (bytes) Average	Max
ipadic-2.6.3	217,404	6.649	52
WordNet-2.1	147,249	11.474	71

表 3 提案法の適用前後におけるダブル配列の比較

Key set	ipadic-2.6.3		WordNet-2.1	
CPU	Via Eden	Intel P4	Via Eden	Intel P4
Size (bytes)				
Before	3,067,026		2,879,377	
After	1,423,576		1,411,566	
Construction time (seconds)				
Before	1.557	0.236	1.147	0.144
After	2.797	0.369	1.763	0.211
Retrieval time (cycles/key)				
Before	902	1381	1196	1688
After	750	723	1048	926

5. おわりに

ダブル配列は，理論的な検索時間がキー数に依存しない優れたデータ構造である．しかし，キャッシュミスがボトルネックとなり，本来の性能を発揮できないことが多い．本稿では，静的なダブル配列を対象として，キャッシュを効率化することにより，実環境における検索時間を短縮する手法を提案した．そして，実験結果を用いて提案法の有効性を示した．

参考文献

[1] 青江順一. トライとその応用. キー検索技法—情報処理学会論文誌, Vol. 34, No. 2, pp. 244–251, February 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, Vol. J71–D, No. 9, pp. 1592–1600, September 1988.
[3] 森田和宏, 泓田正雄, 大野将樹, 青江順一. ダブル配列における動的更新の効率化アルゴリズム. 情報処理学会論文誌, Vol. 42, No. 9, pp. 2229–2238, September 2001.
[4] 大野将樹, 森田和宏, 泓田正雄, 青江順一. ダブル配列におけるキー削除の効率化手法. 情報処理学会論文誌, Vol. 44, No. 5, pp. 1311–1320, May 2003.
[5] 矢田晋, 大野将樹, 森田和宏, 泓田正雄, 吉成友子, 青江順一. 接頭辞ダブル配列における空間効率を低下させないキー削除法. 情報処理学会論文誌, Vol. 47, No. 6, pp. 1894–1902, June 2006.
[6] 森田和宏, 望月久稔, 山川善弘, 青江順一. トライ構造を用いた共起情報の効率的検索アルゴリズム. 情報処理学会論文誌, Vol. 39, No. 9, pp. 2563–2571, September 1998.

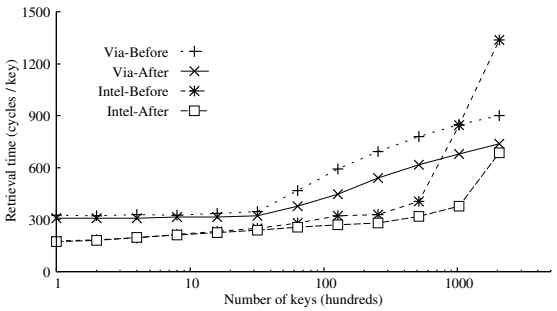

図 5 ipadic-2.6.3 の部分キー集合に対する検索時間