D-046

ダブル配列における動的更新アルゴリズムの高速化
Fast Computation of Dynamic Update Algorithms for a Double-array Structure

中村 康正 *　　望月 久稔 *
YASUMASA NAKAMURA　　HISATOSHI MOTIZUKI
j059610@ex.osaka-kyoiku.ac.jp　　motizuki@cc.osaka-kyoiku.ac.jp

1. はじめに

トライ法は，キー自体で構成される他の検索技法とは異なり，キーの表記記号を遷移としてもつ木構造で表現される．そのため，トライ法は自然言語処理システムの辞書情報構築を中心に広く用いられている．

トライのデータ構造として，配列を用いた手法とリストを用いた手法がある[2]．前者は，配列の特性から遷移を $O(1)$で辿ることができるが，大きな空間を必要とする．また，後者は，不必要な遷移情報をもつ必要がないので小さな空間ですむが，遷移を $O(1)$で辿ることが出来ない．これらを解決するためにダブル配列法がある[1][2]．ダブル配列法は，2つの一次元配列 BASE および CHECK を用い，配列の高速性とリストのコンパクト性とをあわせもつ．また，もう1つの一次元配列 TAIL を用いてトライ上の節点数を抑制している．しかしダブル配列法は，動的検索法に比べキーの追加時間が高速であるとはいえない．そこで現在では，ダブル配列上の未使用要素を単方向リストとして連結する手法が知られている[4][5].

本論文では，ダブル配列法を動的検索法として確立するため，配列 TAIL を用い，未使用要素を双方向リストとして連結することにより，追加処理を高速化する手法を提案し，有効性を評価する．

以下，2節では単方向リストを用いたダブル配列法とその問題点を述べる．3節で追加処理の高速化手法を提案し，4節では提案手法を実験により評価する．最後に，5節で本論文のまとめと今後の課題についてふれる．

2. 単方向リストを用いたダブル配列法と問題点

ダブル配列法における追加操作は，節点 *node* からの遷移集合 L に含まれるすべての要素が遷移可能である BASE 値(以下 $B[node]$とし，配列 CHECK についても同様に $C[node]$とする)を求める関数 NewBase に依存する[3]．関数 NewBase では，BASE 値をダブル配列の先頭から設定し，BASE 値をインクリメントさせながら遷移集合 L に含まれるすべての要素が遷移可能であるかどうかを調べる．よって，ダブル配列のサイズを n，遷移種数を t とすると，関数 NewBase は $O(n*t)$ の計算量が必要となる[3]．これは追加操作が大きなコストとなり，動的検索法としては不十分であることを示している．

そこで，ダブル配列の未使用要素を単方向リストとして連結することにより，関数 NewBase を高速化する[4][5]．これは，式(1)，(2)に示す通り配列 CHECK に次の未使用要素を格納することにより実現する．ここで，未使用要素数を m とし，ダブル配列の未使用要素番号を昇順に e_1，e_2，…，e_m とする．また，未使用要素をマイナス値とし，使用済み要素と区別する．

$$C[e_i]=-e_{i+1} \quad 1\le i\le m-1 \qquad (1)$$

$$C[e_m]=-e_1 \qquad (2)$$

関数 NewBase は，遷移集合 L の最小要素が遷移可能である BASE 値を $O(1)$で求めることができる．よって，関数 NewBase は、ダブル配列の要素数 n ではなく未使用要素数 m に依存し，$O(m*t)$ の計算量が必要となる．

この拡張は，節点を挿入および削除する際にリストを再連結させる処理を必要とする．そこで，以下に節点 *node* を挿入する関数 InsNode，削除する関数 DelNode，およびこれらに呼び出される節点 *node* における直前の未使用要素を探索する関数 PreEmptyNode を示す．ここで，変数 *eTail* は未使用要素リストの最終要素をもつ．

関数 InsNode(*node*)
手順 1(IN-1):前未使用要素の探索
関数 PreEmptyNode(*node*)を呼び出すことにより，節点 *node* よりも前方に存在する未使用要素を探索し，その戻り値を変数 *pre* にセットする．
手順 2(IN-2):*eTail* の更新
eTail と *node* が等しければ，*eTail* に *pre* をセットする．
手順 3(IN-3):リストからの削除
$C[pre]$に $C[node]$をセットし，未使用要素リストを連結しなおす．　(関数終)

関数 DelNode(*node*)
手順 1(DN-1):前未使用要素の探索
関数 PreEmptyNode(*node*)を呼び出すことにより，節点 *node* よりも前方に存在する未使用要素を探索し，その戻り値を変数 *pre* にセットする．
手順 2(DN-2):*eTail* の更新
eTail と *pre* が等しければ，*eTail* に *node* をセットする．
手順 3(DN-3):リストへの追加
$C[node]$に $C[pre]$を，$C[pre]$に $-node$ をセットし，未使用要素リストを連結しなおす．
手順 4(DN-4):節点の初期化
$B[node]$に配列 BASE の初期値0をセットし，未使用要素 *node* を初期化する．　(関数終)

関数 PreEmptyNode(*node*)
手順 1(PE-1):初期化
未使用要素を表す変数 *empty* に $-C[eTail]$をセットする．
手順 2(PE-2):探索
$-C[empty] > node$ であれば探索成功であるので，*empty* を返して終了．そうでなければ *empty* に次の未使用要素である $-C[empty]$をセットし，手順2をくり返す．(関数終)

未使用要素リストを用いることにより，追加操作における大幅なコスト削減を実現することができる[5]．しかし，節点 *node* を追加および削除する際に用いる関数 PreEmptyNode が新たに必要となる．またこの関数は，*node* の値に関係なく先頭未使用要素から線形に *node* の直前未使用要素を探索するために $O(m)$の計算量を要する．

3. 双方向リストを用いた動的追加アルゴリズム

単方向であった未使用要素リストを双方向リストに拡張することにより，関数 InsNode で呼び出す関数 PreEmptyNode の計算量を削減する手法を提案する．

* 大阪教育大学
Osaka Kyoiku University

双方向未使用要素リストは式(1)，(2)を以下に拡張することにより実現する．森田ら[5]が示した単方向リストとは異なり，提案手法では配列 BASE には未使用要素リストにおける末尾方向への連結を，配列 CHECK には先頭方向への連結を格納する．

・末尾方向への連結　$B[e_i]=e_{i+1}$　$1 \le i \le m-1$
　　$B[e_m]=e_1$
・先頭方向への連結　$C[e_i]=-e_{i-1}$　$2 \le i \le m$
　　$C[e_1]=-e_m$

単方向リストの場合と同様，配列 CHECK の要素がマイナス値であれば，その節点が未使用であると判定する．また，単方向から双方向へ拡張するにあたって，配列 BASE を拡張するために使用領域は変化しない．拡張後の関数 InsNode を関数 EX_InsNode とし以下に示す．

関数 EX_InsNode(*node*)
手順 1(EIN-1):前未使用要素の設定
節点 *node* よりも前方に存在する未使用要素を表す変数 *pre* に−*C*[*node*]をセットする．
手順 2(EIN-2):*eTail* の更新
eTail と *node* が等しければ，*eTail* に *pre* をセットする．
手順 3(EIN-3):リストからの削除
B[*pre*]に *B*[*node*]を，*C*[*B*[*node*]]に *C*[*node*]をセットし，未使用要素リストを連結しなおす．　(関数終)

提案手法では，関数 EX_InsNode で呼び出す関数 PreEmptyNode が不要となる．よって，$O(m)$であった手順 1(IN-1)の計算量を，手順 1(IND-1)では $O(1)$に抑制する．

4. 評価実験

提案手法の有効性を示すために，単方向リストを用いた手法(以下，対象手法)と比較実験を行った．提案手法を約 800 行，対象手法を約 800 行の C 言語で実装し，Intel Pentium 4 2.8GHz，Fedora Core3 上で稼動している．実験では，英語キー集合として英単語辞書約 12 万語，日本語キー集合として日本語単語辞書約 40 万語[6][7]から，それぞれ 10 万語をランダムに抽出したものを用いた．

表 1 に上記のキー集合の平均キー長，およびそれらを追加したダブル配列の要素使用状況を示す．ただし，日本語キー集合における平均キー長は日本語 1 語に対して 2byte とする．また，表 1 には対象手法を基準値 1 とした提案手法の相対追加速度も示す．さらに図 1 に，英語キー集合 10 万語を母集団とし，1 万語から 10 万語まで 1 万語ずつ語数を変動させたキー集合を追加した際の実験結果を示す．

表 1 より英語キー集合に関して，提案手法の追加処理は対象手法より約 1.512 倍高速であり，また図 1 より，10 万語以下の語数を追加した場合でも約 1.5 倍高速であることが判る．すなわち，対象手法において関数 InsNode で呼び出される関数 PreEmptyNode の計算量が全体の約 1/3 であったといえる．また，日本語キー集合に対しては追加速度が対象手法よりも約 2.854 倍高速であり，英語キー集合よりも大きく向上していることが判る．これは表 1 より，英語キー集合よりも未使用要素が多いからであると考えられる．すなわち，関数 PreEmptyNode は未使用要素が多いほど計算量が増加することを示している．

以上の実験結果より，未使用要素数が全要素数の 0.2%未満という少ない英語キー集合に対しても提案手法が有効であることが判った．また，未使用要素が存在しない場合でも対象手法と等しい追加性能が得られる．

表 1:キー 10 万語に対する実験結果

		英語キー集合	日本語キー集合
平均キー長(byte)		9.859	8.614
節点数		191,894	155,596
最大節点番号		192,207	171,627
未使用要素数		313	16,031
相対速度	対象手法	1.000	1.000
	提案手法	1.512	2.854

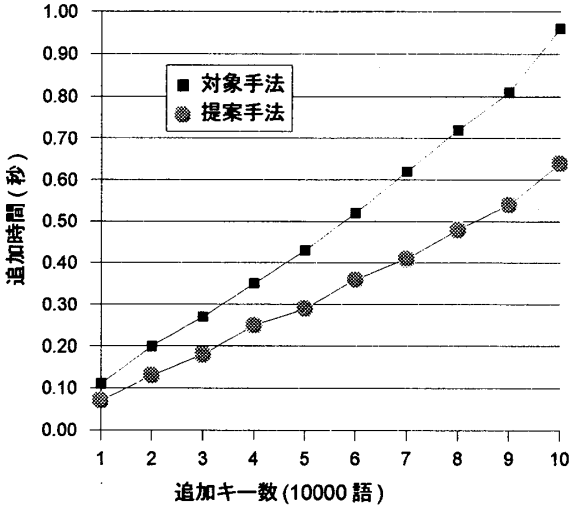

図 1 :キー数を変動させた追加処理に対する実験結果

5. おわりに

本論文では，ダブル配列を動的検索法として確立するために，追加処理を高速化する手法を提案し，実験により提案手法の有効性を示した．関数 DelNode 内で呼び出される関数 PreEmptyNode についても同様にコストを削減することで，さらに高速化をはかることが可能である．

今後の課題として，提案手法を実際の自然言語処理システムに実装し，より詳細に評価することが挙げられる．

参考文献

[1]青江順一:自然言語辞書の検索-ダブル配列による高速ディジタル検索アルゴリズム-，bit，Vol-21，No.-6，pp.36-44，1989.
[2]青江順一:キー検索技法-トライとその応用-，情報処理学会論文誌，Vol-34，No.-2，pp.244-251，1993.
[3]青江順一，森本勝士:ダブル配列法によるトライ検索の実現法，自然言語処理，Vol-85，No.-2，pp.9-16，1991.
[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.1311-1320，2003.
[5]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.2229-2238，2001.
[6](財)新世代コンピュータ技術開発機構:ICOT 形態素辞書，ftp://ftp.icot.or.jp.
[7]情報処理振興事業協会技術センター:IPA 日本語辞書，http://www.ipa.go.jp/STC/NIHONGO/IPAL/ipal.html.