5K-10

遷移集合に着目したダブル配列法における追加手法の提案

中村 康正† 野村 優† 望月 久稔†
大阪教育大学†

1. はじめに

自然言語処理システムを中心に広く用いられているトライ法のデータ構造として，2 つの 1 次元配列を用いたダブル配列法が存在する．ダブル配列法は高速性とコンパクト性をあわせもつ有効なデータ構造であるが，動的検索法に比べ追加処理が高速であるとはいえない[1]．そのため現在では，未使用要素を双方向リストとして連結する手法が知られている[3][4]．しかし，この手法は全遷移先節点探索の計算量を抑制するものではない．

そこで本論文では，ダブル配列法に対して，遷移情報を格納する 1 次元配列を新たに用いた追加手法を提案し，有効性を評価する．

2. 双方向リストを用いたダブル配列法

ダブル配列法のデータ構造は，2 つの 1 次元配列 BASE，CHECK により実現する．トライ木の節点 s から内部表現値 a での遷移先節点 t の関係を式(1), (2)で定義する[1]．つまり，配列 BASE は行置換関数を与え，配列 CHECK はトライ木の親子関係を一意に決定するために用いる．以下，節点 *node* の BASE 値を $B[node]$，CHECK 値を $C[node]$とする．また，1 次元配列 TAIL に，他のキーと共有していない遷移を文字列として格納する．これにより，トライ木の節点数を抑制する[1].

$$B[s] + a = t \quad (1)$$
$$s = C[t] \quad (2)$$

ダブル配列上の未使用要素は，以下に示す通り双方向リストとして連結する[3][4]．ここで，ダブル配列上の未使用要素を e_1，e_2，…，e_m とし，未使用要素の CHECK 値をマイナスとすることで使用済み要素と区別する．また，ダブル配列上の要素番号 0(e_0)をダミー要素として用いる．

$$B[e_i] = e_{i+1} \qquad 0 \leqq i \leqq m-1 \quad (3)$$
$$B[e_m] = e_0 \quad (4)$$
$$C[e_i] = -e_{i-1} \qquad 1 \leqq i \leqq m \quad (5)$$
$$C[e_0] = -e_m \quad (6)$$

ダブル配列法の追加処理には，節点 s からの新規遷移による節点がすでに使用されている場合，新規遷移とすでに存在していた遷移が可能な BASE 値に $B[s]$を再決定し，s の全遷移先節点を移動させた後，移動した節点の遷移先節点における CHECK 値を更新する処理が存在する．ここでは，l_{min} は最小遷移，l_{max} は最大遷移の内部表現値とし，以下に示す節点 s の遷移先節点を集合 *T_SET* に格納する関数 TnodeSearch を用いる．また，異なる遷移 a, b による遷移先節点を t_a，t_b，BASE 値再決定による移動後の遷移先節点を t_a'，t_b'とすると，t_aが t_b と等しい場合がある．このとき，関数 TnodeSearch により t_bの遷移先節点を探索すると，t_bの遷移先節点だけでなく，CHECK 値を更新した t_aの遷移先節点が *T_SET* に含まれる．そのため，*T_SET* の全節点に対して，更新済みの節点かどうか判断する処理(以下，判断処理)を必要とする．

関数 TnodeSearch(s, *T_SET*)
手順 1(TS-1):遷移先節点候補の初期化
遷移先節点候補 t に，節点 s の最小遷移先節点候補 $B[s] + l_{min}$をセットする．
手順 2(TS-2):遷移先節点の探索
$C[t]$が s と等しければ *T_SET* に tを追加する．
手順 3(TS-3):遷移先節点候補の更新
t に次の遷移先節点候補 $t + 1$ をセットする．t が s の最大遷移先節点候補 $B[s] + l_{max}$より大きければ終了し，そうでなければ手順 2 へ．

双方向未使用要素リストを用いることにより，BASE 値再決定における計算量を抑制でき，追加処理が高速となる[3][4]．しかしながら，比較回数が常に遷移種数回必要である全遷移先節点探索は頻繁に行われ，多くの計算量を必要とする．

3. トリプル配列法

全遷移先節点探索の計算量を抑制し追加処理を高速化するため，遷移先節点情報を格納する 1 次元配列 SIBLING を用いることを提案する．拡張した関数 TnodeSearch を関数 EX_TnodeSearch として示す．ここで，節点 t の SIBLING 値を $S[t]$とし，節点 s の遷移先節点 t_1，t_2，…，t_k を式(7)，(8)に示す通り環状リストとして連結する．また，未使用要素 e の SIBLING 値は$-e$ とし，関数 MinTnode(s)は，節点 s の最小遷移先節番号を返す．

$$S[t_i] = t_{i+1} \qquad 1 \leqq i \leqq k-1 \quad (7)$$
$$S[t_k] = t_1 \quad (8)$$

関数 EX_TnodeSearch(s, *T_SET*)
手順 1(ETS-1):最小遷移先節点の探索
遷移先節点 t および最小遷移先節点 *min* に，関数 MinTnode(s)の戻り値をセットする．
手順 2(ETS-2):遷移先節点の格納
T_SET に t を追加し，t に $S[t]$をセットする．このとき，t が *min* と等しければ終了し，そうでなければ手順 2 を繰り返す．

提案手法は，最小遷移先節点を探索後，次の遷移先節点を $O(1)$で決定し，全遷移先節点探索を高速

Proposal of Insertion Techniques for a Double-array Structure using the Set of Transition
†Yasumasa NAKAMURA, Yu NOMURA, Hisatoshi MOCHIZUKI, Osaka Kyoiku University

化する．また判断処理に関して，CHECK 値が等しい節点でも，更新前の節点と更新済みの節点は異なる遷移先節点リストに連結されている．よって，最小遷移先節点に対して判断処理を行い，更新済み節点でなければ，配列 SIBLING により連結された節点は更新済み節点でないことが保障される．つまり，すべての節点に対して判断処理を行う必要がない．

節点を追加および削除する際，配列 SIBLING を変更する必要がある．そこで，節点 s の遷移先節点リストに節点 t を追加する関数 InsSibling を，削除する関数 DelSibling を定義し，以下に示す．

関数 InsSibling(s, t)
手順 1(IS-1):最小遷移先節点の探索
最小遷移先節点 *min* に関数 MinTnode(s)の戻り値をセットする．
手順 2(IS-2):遷移先節点リストへの追加
$S[t]$に $S[min]$を，$S[min]$に tをセットする．

関数 DelSibling(t)
手順 1(DS-1):直前遷移先節点の初期化
遷移先節点リストにおける t の直前遷移先節点 p に $S[t]$をセットする．
手順 2(DS-2):直前遷移先節点の探索
$S[p]$が t と等しくなるまで p に $S[p]$をセットし，手順 2 を繰り返す．
手順 3(DS-3):遷移先節点リストから削除
$S[p]$に $S[t]$を，$S[t]$に$-t$をセットする．

4. 評価実験

提案手法の有効性を示すため，双方向リストを用いた手法(以下，対象手法)と比較実験を Intel Pentium4 2.8GHz，FedoraCore3 上で行った．実験では，日本語キー集合(以下，J_SET)として日本語単語辞書約 40 万語から，英語キー集合(以下，E_SET)として英単語辞書約 35 万語から，それぞれ 20 万語をランダムに抽出したものを用いた．

表 1 に，両キー集合の特徴として，平均キー長，追加処理後のダブル配列における平均遷移数を示す．ただし，J_SET における平均キー長は日本語 1 語に対して 2byte とし，平均遷移数はトライ木における内部節点を対象として算出した．さらに両手法の性能として，使用 byte 数，全遷移先節点探索および判断処理の比較回数，追加処理時間を示す．また図 1 に，J_SET より 1 万語から 20 万語まで 1 万語ずつランダムに抽出したキー集合に対して，全遷移先節点探索と判断処理の合計比較回数および追加処理時間を示す．

表 1 および図 1 より提案手法は，全遷移先節点探索および判断処理の比較回数に関して，J_SET では約 86%を，E_SET では約 50%を削減できたことが判る．また図 1 から，対象手法における追加処理は合計比較回数に依存していることも判る．さらに J_SET に関して，提案手法の追加処理は約 2.65 倍，E_SET に関しては約 1.20 倍高速であった．ここで，J_SET における大きな処理速度向上の要因は，判断処理の比較回数に対する高い削減率である．これは，判断処理が平均遷移数に依存するからである．しかし表 1 より，配列 SIBLING を用いることで，対象手法よりも提案手法の使用領域は増加した．

表 1:キー20 万語に対する実験結果

		日本語キー集合	英語キー集合
平均キー長		7.157	9.462
平均遷移数		2.628	2.130
使用 byte 数	対象手法	6,198,824	5,334,459
	提案手法	8,825,832	7,455,199
全遷移先節点探索における比較回数	対象手法	72,819,772	51,954,506
	提案手法	62,002,498	28,830,369
判断処理における比較回数	対象手法	506,650,789	8,217,943
	提案手法	18,724,534	1,150,982
追加処理時間	対象手法	2.102019	0.517965
	提案手法	0.769615	0.436944

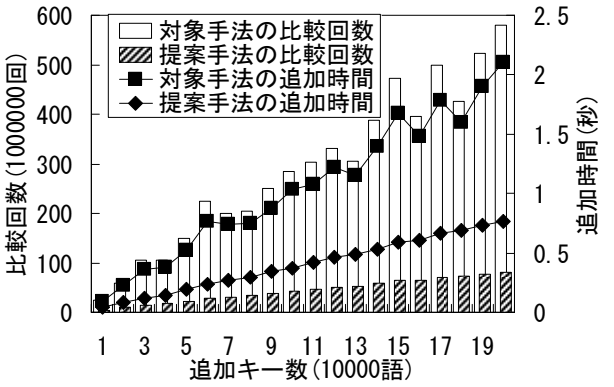

図 1:キー数変動による追加処理に対する実験結果

以上より，使用領域は増加したが，追加処理が依存していた比較回数を削減できた提案手法は有効であるといえる．

5. おわりに

本論文では，遷移情報を用いて追加処理を高速化する手法を提案し，実験により有効性を示した．今後の課題として，遷移情報を用いた削除処理の構築および評価が挙げられる．

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] 入口浩一ほか，グラフ構造に対する効率的記憶検索法，電子情報通信学会論文誌 D-1，Vol.79，No.8，pp.502-510，1996.
[3] 大野将樹ほか，ダブル配列による自然言語処理辞書の高速更新手法，言語処理学会，第 11 回年次大会予稿集，pp.745-748，2005.
[4] 中村康正，望月久稔，自然言語処理における効果的な辞書情報更新アルゴリズム，情報処理学会研究報告(FI-80/NL-169)，pp.117-122，2005.