D_044

ダブル配列上の一方向分岐に着目した探索手法の提案

Proposal of Search Method Focused on the One-Way Branch on a Double-array Structure

芳本　貴男[†]　　中村　康正[†]　　入江　祐司[†]　　望月　久稔[†]

Takao YOSHIMOTO　Yasumasa NAKAMURA　Yuji IRIE　Hisatoshi MOCHIZUKI

1．はじめに

自然言語処理システムの辞書を中心に広く用いられるトライのデータ構造として，節点間の遷移を $O(1)$ で決定できる高速性とコンパクト性をあわせもつダブル配列 [1] がある．最近の研究では，その更新時間を短縮する手法 [2][3][4] が提案されている．また，トライ上の一方向分岐を圧縮する手法としてパトリシアがあり，ダブル配列を利用して圧縮する手法 [5] も提案されているが，2分木トライであるため遷移数が大きい．

ダブル配列における探索処理は，節点間の遷移数に依存する．そこで本論文では，ダブル配列上に存在する一方向分岐に着目し，その遷移情報を格納する節点を導入した探索手法を提案し，その有効性を評価する．

2．ダブル配列のデータ構造

ダブル配列は2つの配列 BASE，CHECK によりトライを表現し，トライにおける節点 s からラベル a による節点 t への遷移を式 (1) で定義する．ここで，BASE 値には遷移先節点の位置に対するオフセットを，CHECK 値には遷移元節点を格納する．

$$\begin{cases} BASE[s] + a = t \\ CHECK[t] = s \end{cases} \tag{1}$$

また，トライからあるキーを一意に決定できる最前方の節点をセパレート節点 (以下，SP 節点)[1] と呼ぶ．ダブル配列上の節点数を抑えるため，SP 節点以降の遷移をラベルの列として配列 TAIL に格納する．配列 TAIL での格納位置は SP 節点の BASE 値で表し，負値とすることで他節点と区別する．

キー集合 K={“decide#”, “decidable#”, “disk#”} に対するトライとダブル配列を図1に示す．ラベル#は終端記号を表し，#, a, b, …, z の内部表現値をそれぞれ 1, 2, 3, …, 27 とする．

図1の節点8は，出次数が1で一方向分岐となっており，配列 TAIL は SP 節点以降の一方向分岐による遷移を圧縮して格納したものである [1] といえる．ダブル配列上の一方向分岐の圧縮について，配列 BASE の各要素数ビットに遷移情報をもたせてパトリシアを1つの1次元配列に圧縮する手法 [5] が提案されている．しかし，遷移数が大きい2分木トライのパトリシアではなく，多分木トライとして図1の節点8から節点13のような SP 節点より前方で存在する一方向分岐を圧縮する手法ではない．

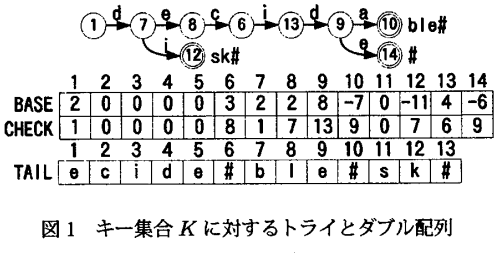

図1　キー集合 K に対するトライとダブル配列

3．ダブル配列上の一方向分岐に着目した探索処理

ダブル配列上の一方向分岐に着目し，その遷移情報を格納する節点を導入した探索手法を提案する．

連続する一方向分岐の節点 $s_1, s_2, \ldots, s_n$ が存在し，s_n が節点 t に遷移するとき，s_1 から t への遷移は一意に決定する．そこで，SP 節点以降の遷移と同様に，一意に決まる遷移を配列 TAIL に格納する．ここで，節点 t に至る遷移を管理するために，圧縮した部分の遷移情報を格納する節点 p を式 (2) で定義し，節点 t に対する prefix 節点と呼ぶ．

$$\begin{cases} BASE[p] = -tail \\ CHECK[p] = -pos \end{cases} \tag{2}$$

ここで，$tail$ は配列 TAIL におけるラベル列の格納位置を表し，pos はキーにおける節点 t の比較位置を表す．したがって，キーの探索において遷移先が節点 t となるとき，配列 TAIL 上の $tail$ に存在するラベル列とキーを比較することで，pos までの遷移を確認できる．なお，BASE 値と CHECK 値をともに負値とし，他節点および未使用要素 [3] と区別する．

prefix 節点は，定数 PCODE による遷移先 $BASE[t]$+PCODE に作成する．ここで，prefix 節点は圧縮した一方向分岐の遷移先節点に対してのみ存在すればよく，各節点に対する prefix 節点の有無を管理する必要がある．これは配列 BASE あるいは CHECK の各要素1ビットを用いれば十分である．以下，節点 x に対する prefix 節点の有無を PNODE(x) で表す．

キー集合 K に対する，prefix 節点を導入したトライとダブル配列を図2に示す．ここで，定数 PCODE を0とする．また，探索関数 Search を以下に示す．従来のダブル配列 [1] は，手順1で状態を初期化した後，手順3によるダブル配列上の遷移を繰り返し，手順5で SP 節点以降の遷移を確認する．これに対して本手法は，SP 節点以前で圧縮した一方向分岐節点を手順2で遷移し，手順4でその遷移確認を行う点が異なる．

関数 Search($key[1 \ldots k]$)

手順1 状態の初期化

現在の節点 x を1，探索キー key 上の走査位置 pos を1に設定する．

手順2 一方向分岐による遷移

PNODE(x) が偽ならば，手順3に進む．x に対する prefix 節点 p を $BASE[x]$+PCODE に設定する．配列 TAIL に格納した一方向分岐部分のラベル数 n を $-CHECK[p] - pos$ に設定し，n, pos, および配列 TAIL におけるラベル列の位置 $-BASE[p]$ を遷移情報としてスタックに格納する．pos を $pos+n$ に更新する．

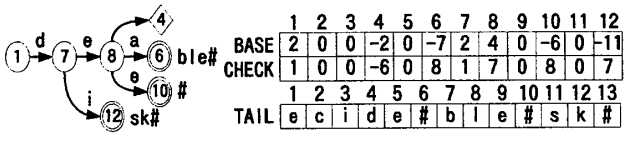

図2　prefix 節点を導入したトライとダブル配列

† 大阪教育大学, Osaka Kyoiku University

手順3 ダブル配列上の遷移

遷移先節点 $next$ を $BASE[x] + key[pos]$ に設定し，$CHECK[next] \neq x$ ならば探索失敗で終了する．x を $next$ に更新し，x がSP節点，すなわち $BASE[x] < 0$ であれば，手順4に進む．pos を $pos + 1$ に更新し，手順2に戻る．

手順4 SP節点以前における一方向分岐部分の遷移確認

スタックに格納したすべての遷移情報に対して，配列TAILと key の該当部分を比較する．すべて一致すれば手順5に進み，一致しなければ探索失敗で終了する．

手順5 SP節点以降の遷移確認

配列TAIL上の $-BASE[x]$ から始まるSP節点以降のラベル列とキーの残りの部分を比較し，一致すれば探索成功，一致しなければ探索失敗として終了する．[関数終]

4． 実験による評価

本手法の有効性を示すため，従来のダブル配列[1](以下，対象手法A)およびダブル配列とは異なる構造であるsuffix array[6](以下，対象手法S)との比較実験をIntel Pentium4 2.8GHz, Fedora Core4上で行った．実験では，キー集合としてURIをランダムに20万語抽出したものを用いた．このキー集合の平均キー長は57.5であり，遷移のラベルとなる各構成文字の内部表現値はASCIIコードとした．

まず，20万語のキー集合より1万語から20万語まで1万語毎にキー数を増やした部分キー集合を作成し，本手法および対象手法Aについて，各キー集合を登録したときの各配列の使用要素数を図3に示す．図3より，本手法における配列BASE, CHECKの使用要素数は，対象手法Aと比較して20万語で約0.40倍に抑えられた．本手法における配列TAILの使用要素数は，すべての一方向分岐節点のラベル列を管理するため，SP節点以前の一方向分岐節点数が加わる．

次に，20万語のキー集合を登録した各配列に要する使用領域を表1に示す．本手法では，SP節点以前の一方向分岐節点を，節点として8byteを要する配列BASE, CHECKから1byteのラベルとして配列TAILに格納するため，対象手法Aに対して20万語では0.65倍に抑えられた．また，対象手法Sは，対象手法Aに対して5.25倍の領域を必要とした．なお，対象手法Sはインデックス部分であるsuffix arrayおよびテキスト部分であるキーを対象に計算した．

さらに，先に作成した1万語から20万語までの各部分キー集合に対し，1万回の成功探索を行った合計探索時間を図4に示す．20万語における本手法の探索時間は対象手法Aに対して0.84倍となった．また，suffix arrayに対して二分探索を行う対象手法Sは，対象手法Aに対して2.22倍の時間を必要とした．

表1 URIキー集合20万語に対する各手法の使用領域

	本手法	対象手法A	対象手法S
使用領域(byte)	7,229,284	11,142,677	58,457,635
対比(倍)	0.65	1.00	5.25

以上のことから，本手法は対象手法Sに対して時間的にも空間的にも効率的であり，対象手法Aに対して時間的にほぼ同等ながら空間的には効率的である結果が得られた．

5． おわりに

本論文では，一方向分岐に着目して，その遷移情報を格納するprefix節点を導入した探索手法を提案し，実験により有効性を示した．今後の課題として，prefix節点数に対する探索性能，および更新処理とあわせた評価があげられる．

参考文献

[1] J. Aoe, K. Morimoto, M. Shishibori and K-H. Park, A Trie Compaction Algorithm for a Large Set of Keys, IEEE Transactions on Knowledge and Data Engineering, Vol.8, No.3, pp.476-491, 1996.

[2] 大野将樹，森田和宏，泓田正雄，青江順一，ダブル配列による自然言語辞書の高速更新法，言語処理学会第11回年次大会，pp.745-748, 2005.

[3] 中村康正，野村優，望月久稔，遷移先節点集合を導入したトライ構造における更新手法の実現，情報処理学会研究報告(FI-82/DD-54), pp.1-6, 2006.

[4] 永井弘之，藤田茂，菅原研次，シングル状態を利用したダブル配列における動的追加の高速化，情報処理学会研究報告(NL-173), pp.29-34, 2006.

[5] 山本一徳，獅々堀正幹，柘植覚，北研二，パトリシアトライの1次元配列構造への圧縮手法，言語処理学会第11回年次大会，pp.688-690, 2005.

[6] Juha K., Peter S., Simple Linear Work Suffix Array Construction, ICALP 2003, LNCS 2719, pp.943-955, 2003.

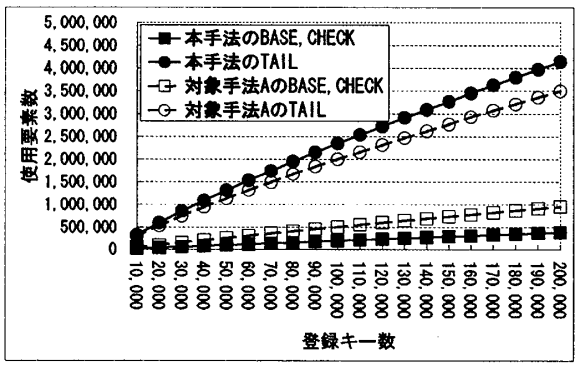

図3 URIキー集合に対する各配列の使用要素数

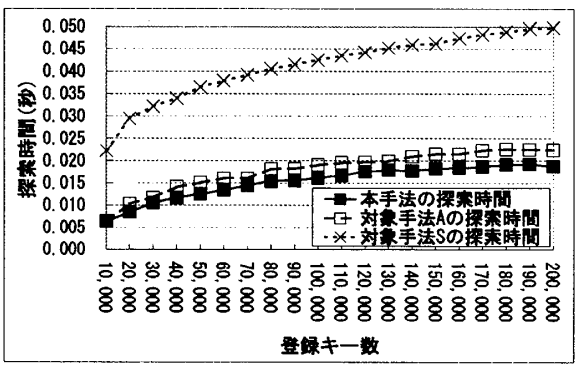

図4 URIキー集合に対する探索時間