ダブル配列によるパトリシアを拡張した基数探索法

Radix Search Method Extended Patricia based on the Double-array

望月 久稔 ♥ 中村 康正 ♦
尾崎 拓郎 ♠

Hisatoshi MOCHIZUKI Yasumasa NAKAMURA
Takuro OZAKI

木構造で表現される基数探索法の探索処理を高速化するため，遷移が 1 つしか存在しない分岐を圧縮したパトリシアや，遷移数を抑制するために多分木としたマルチウェイ基数探索法がある．マルチウェイ基数探索法のデータ構造として，節点間の遷移を定数時間で決定できる高速性をもつダブル配列がある．本論文では，ダブル配列によりパトリシアを拡張した基数探索法を提案する．評価実験の結果，提案手法は探索処理や更新処理において比較手法よりも有効であるとわかった．

Patricia and multiway radix search method are proposed to accelerate search processing expressed tree structure. The double-array structure is an efficient data structure combining fast access with compactness. In this paper, we present radix search method based on the double-array structure. The simulation results turned out that the presented method is more effective than the original methods.

1. まえがき

検索技法である基数探索法は，キーの構成を各遷移とする木構造で表現され，離散探索法とマルチウェイ基数探索法がある．離散探索法はキーの各 1 ビットを遷移とした二分木で表現され，一方向分岐が多数存在する．そこで，木構造上の遷移数を抑制するため，一方向分岐を削除したパトリシアがある [2]．マルチウェイ基数探索法はキーの複数ビットを遷移とした多分木で表現するため，離散探索法より任意のキーに対する遷移数が少ない．この手法に有効なデータ構造として，高速性とコンパクト性を併せ持つダブル配列がある [1]．しかし，任意の節点からすべての遷移種に対する遷移は存在しないので不要な遷移情報が多い．

そこで本論文では，マルチウェイ基数探索法における木構造上の遷移数を抑制して探索処理を高速化するため，ダブル配列を用いてパトリシアを多分木に拡張した基数探索法を提案する．評価実験を行った結果，提案手法は探索処理や更新処理において比較手法よりも有効であるとわかった．

以下，2. でパトリシアとダブル配列を説明し，3. でパトリシアを拡張した基数探索法を提案する．4. で提案手法の評価を与え，5. で本論文のまとめと今後の課題についてふれる．

♥ 正会員 大阪教育大学 motizuki@cc.osaka-kyoiku.ac.jp
♦ 非会員 大阪教育大学 j059610@ex.osaka-kyoiku.ac.jp
♠ 非会員 大阪教育大学 g046807@ex.osaka-kyoiku.ac.jp

2. パトリシアとダブル配列

2.1 パトリシア

パトリシアは，一方向分岐を作成せず，各節点にキーの比較位置情報を格納した手法であるため，不要な遷移情報が存在せず，空間効率が良い．さらに，遷移を抑制しているため，離散探索法に比べて任意のキーに対するトライ上の遷移数が少ない．パトリシア上には，葉に対応したキーすべてを格納する．よって，パトリシアの探索処理は，パトリシア上を根から葉まで遷移し，葉に対応するキーと探索キーを比較する．しかし，パトリシアは二分木で表現されるために木の深さが増し，探索に時間を要する．

2.2 ダブル配列

ダブル配列は，BASE と CHECK の 2 つの一次元配列を用いてトライの節点を実現する．配列 BASE は遷移の基底位置を与え，配列 CHECK はトライ上の親子関係を決定する．つまり，始点 s から遷移 a での終点 t を式 (1)，式 (2) で定義する [1]．ここで，表記記号の内部表現値を単に a とし，ダブル配列上の要素 x に関する BASE 値を $B[x]$，CHECK 値を $C[x]$ とする．

$$B[s] + a = t \tag{1}$$

$$C[t] = s \tag{2}$$

トライ上の葉について，ダブル配列では接尾辞を一次元配列 TAIL に格納する [1]．この際，葉の BASE 値は配列 TAIL の要素番号に設定し，他の節点と区別するために BASE 値を負値とする．以下，配列 TAIL の pos 番目の要素を $T[pos]$ とし，$T[pos]$ から始まる遷移列を $T + pos$ とする．ダブル配列は，離散探索法を多分木とすることで探索時の遷移を抑制する．しかし，ダブル配列上には一方向分岐が存在する．

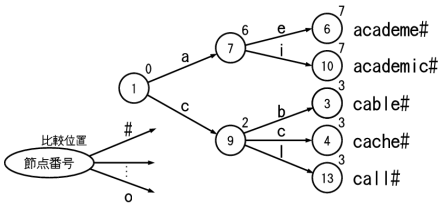
3. パトリシアを拡張した基数探索法

ダブル配列で表現される多分木上の一方向分岐を削除し，二分木のパトリシアを多分木に拡張した基数探索法を提案する．例として'#' を終端記号としたキー集合を用い，記号 '#'，'a'，...，'o' の内部表現値をそれぞれ 0，1，...，15 とする．

3.1 データ構造

ダブル配列にキーの比較位置を格納する配列 POS を新たに用いる．キー集合に対して，提案手法が実現するトライを図 1 に示す．配列 POS を用いることにより，キー key に対する始点 s からの終点 t は式 (3) となる．以下，ダブル配列上の要素 x に関する POS 値を $P[x]$ とする．

$$B[s] + key[P[s]] = t \tag{3}$$



	1	2	3	4	5	6	7	8	9	10	11	12	13
BASE	6		-18	-24		-1	1		1	-9			-30
CHECK			9	9		7	1		1	7			9
POS	0		3	3		7	6		2	7			3

	1	9	18	24	30
TAIL	academe#	academic#	cable#	cache#	call#

図 1 提案手法におけるトライ

Fig. 1 The trie in the proposal technique.

Function: Search(t, key)

(S1) **while:** 節点 t が葉ではない
(S2) $s \leftarrow t$; $t \leftarrow B[s] + key[P[s]]$;
(S3) **if:** $C[t]$ と s が異なる
(S4) **return** FALSE;
(S5) **return** EqualKey($T + (-B[t])$, key);

図 2 関数：Search

Fig. 2 Function: Search.

また，パトリシアと同様に配列 TAIL にキー全体を格納する．

3.2 探索アルゴリズム

提案手法においてキーの探索が成功する条件は，遷移先で式 (3)，遷移元で式 (2) を満足させながらトライ上の根から葉まで遷移し，葉に対応したキーと探索キーが一致することである．

key を探索する関数 Search を図 2 に示す．まず S2 で，key に対する始点 s からの終点を配列 POS を用いて決定する．次に，式 (2) により遷移の存在を判断し，存在すれば S1 から始まる while 文で葉まで遷移する．その後，S5 で遷移列 $T + (-B[t])$ と key が一致するかを関数 EqualKey で判断する．関数 EqualKey(x, y) は，遷移列 x と y が等しければ TRUE を，等しくなければ異なる最小の比較位置を返す．

例 1: 図 1 から “academic#” を探索する．節点 t を根 1 とし，図 2 の S1 で t が葉でないことを確認する．次に S2 で節点 s を $t = 1$ に設定し，s から比較位置 $P[1] = 0$ である ‘a’ で遷移する節点 t を決定する．つまり，式 (3) より t を $B[s]$+‘a’$= 6+1 = 7$ に設定し，節点 1 から ‘a’ での遷移先節点が 7 であることがわかる．同様に，S3 で式 (2)，(3) を確認しながら，節点 7 から $key[P[7] = 6]$ である ‘i’ で節点 10 に遷移する．節点 10 は葉であるので，while 文を終了する．S5 で葉 $t = 10$ に対応した遷移列 “academic#” と key が等しいので，探索成功となる．

（例終了）

3.3 追加アルゴリズム

キー key に対する追加処理は，key の探索が失敗した場合に行う．追加処理で使用する変数と関数を以下に示す．

関数 NewBase(L)
配列上の未使用要素を管理するリスト E-LINK[3, 4] を走査し，遷移集合 L の全要素が遷移可能な BASE 値を返す．

関数 RenewalCheck(new, old)
ダブル配列上の要素番号が old から new へ移動した節点の終点に対し，その CHECK 値を old から new に更新する．

関数 TransNode(s, $newBase$, L)
始点 s の BASE 値を $newBase$ に変更し，遷移集合 L の全要素に対して式 (1) を満足させるために s の終点を移動する．また，式 (2) を満足させるため，関数 RenewalCheck により移動した節点における終点の CHECK 値を更新する．

関数 InsCommon(s, key, $tail$, pos)
トライ内の節点作成時，葉から接尾辞に対する節点を作成する場合と，既存の節点間に節点を作成する場合に用いる．s の更新前の情報を保存しておき，その節点から $tail[pos]$ と $key[pos]$ に対する終点を作成する．

関数 InsLeaf(s, t, key, pos)
始点 s からの新規節点 t を葉として作成する．始点 s から $key[pos]$ での終点 t が未使用要素でない場合，t に衝突が発生するため，関数 NewBase，TransNode を用いてトライ上の節点を移動させて衝突を回避した後，葉を作成する．

提案手法は，図 3 に示す関数 Insert により追加処理を実現する．関数 Insert の入力は，関数 Search により遷移が成功した節点 $terminal$ と追加キー key である．まず，図 3 の I2 で新規節点の比較位置 pos を得るため，key と比較する他のキーを調べる．$terminal$ をもとに葉 $leaf$ を得て，キー情報を取得する．次に，I3 から I9 でトライ上の根から $terminal$ まで遷移しながら，新規節点の追加位置を決定する．ここで，提案手法は一方向分岐の節点を作成していないため，遷移列内に新規節点の比較位置 pos が存在する場合は，I7 で関数 InsCommon により既存の節点間に節点を作成する．新規節点の追加位置が既存の節点間ではない場合，$terminal$ までの遷移が保証されているため，$terminal$ が葉であれば I11 で関数 InsCommon を呼び出し，$terminal$ から遷移 $key[pos]$ での終点が存在しなければ I13 で関数 InsLeaf を呼び出す．

3.4 削除アルゴリズム

提案手法の削除処理は，削除キーに対応する葉の処理と，葉の削除により出現した一方向分岐の削除で構成する．よって，提案手法は最大 2 つの節点を削除する．削除処理を実現する関数 Delete を図 4 に示し，削除処理で使用する関数を以下に示す．

関数 IsOneWayBranch(s)
始点 s からの遷移が一方向分岐であれば s の終点を返す．

削除キーに対応する葉を削除する処理は，図 4 の D2 で削除キーの探索で到達した葉 $leaf$ を削除することで実現する．

Function: Insert($terminal$, key)

(I1) $terminal$ をもとに $leaf$ を得る;
(I2) $pos \leftarrow$EqualKey($T + (-B[leaf])$, key);
(I3) s をトライ上の根に設定する;
(I4) **while:** 節点 s が $terminal$ とは異なる
(I5) $t \leftarrow B[s] + key[P[s]]$;
(I6) **if:** pos が $P[t]$ より小さい
(I7) InsCommon(s, key, $T + (-B[leaf])$, $P[s]$, pos);
(I8) **return**;
(I9) $s \leftarrow t$;
(I10) **if:** 節点 s が葉である
(I11) InsCommon(s, key, $T + (-B[leaf])$, $P[s]$, pos);
(I12) **else:**
(I13) $t =\leftarrow B[s] + key[P[s]]$; InsLeaf($s$, t, key, pos);
(I14) **return**;

図 3 関数：Insert
Fig. 3 Function: Insert.

Function: Delete($leaf$)

(D1) $s \leftarrow C[leaf]$;
(D2) 節点 $leaf$ を削除する;
(D3) $t \leftarrow$IsOneWayBranch(s);
(D4) **if:** s が一方向分岐である
(D5) $B[s] \leftarrow B[t]$;
(D6) **if:** t が葉ではない
(D7) $P[s] \leftarrow P[t]$; RenewalCheck(s, t);
(D8) 節点 t を削除する;
(D9) **return**;

図 4 関数：Delete
Fig. 4 Function: Delete.

まず，D3 で s から t への遷移が一方向分岐であるかを判断し，真であれば D8 で t を削除する．このとき，t が葉でなければ，式 (3) と式 (2) を満足させるため，D5 で $B[s]$ を $B[t]$ に，D7 で t の終点における CHECK 値を s に更新する．

4. 評価

提案手法の有効性を示すため，配列 TAIL を用いない比較手法 D[1, 4]，配列 TAIL を用いた比較手法 DT[1, 3]，パトリシア [2] を比較手法 P とし，探索，追加，削除処理に対して比較評価を行う．ここで，キー数を n，未使用要素数を e とし，キー長を k，キーの遷移種数を b とする．

4.1 理論的評価

比較手法 D は，トライ上の遷移数が k であるので $O(k)$ となる．比較手法 DT は，トライ上の遷移数と葉でのキー比較により $O(k)$ となる．また，パトリシアの高さが最良 $\log_2 n$ であるのに対し，提案手法が実現するトライの高さは最良 $\log_b n$ となる．よって，提案手法の探索処理について，葉でのキー比較による計算量を考慮すると，比較手法 P が $O(\log_2 n + k)$ であり，提案手法は $O(\log_b n + k)$ となる．

追加処理については，関数 InsLeaf と関数 InsCommon をもとに評価を行う．ダブル配列の関数 InsLeaf は関数 NewBase に依存する．関数 NewBase は，要素数 e の E-LINK を走査し，要素数 b の遷移集合による終点の作成を判断するため $O(e \cdot b)$ である [3, 4]．また，要素数 b の遷移集合による各遷移先の CHECK 値を更新する手続きが発生するため $O(b \cdot b)$ である．よって，関数 InsLeaf は $O(e \cdot b + b \cdot b)$ である．比較手法 DT で呼び出される関数 InsCommon は，関数 NewBase と一方向分岐を作成する計算量が多い．BASE 値は E-LINK により $O(1)$ で決定されるので，一方向分岐の作成処理は $O(k)$ である．一方，関数 InsCommon 内で呼び出す関数 NewBase は，要素数が 2 の遷移集合を入力とするので $O(e \cdot 2)$ である．よって，比較手法 DT における関数 InsCommon は $O(k + e \cdot 2)$ となる．以上より，比較手法 D は $O(k + e \cdot b + b \cdot b)$，比較手法 DT は $O(e \cdot b + b \cdot b)$ または $O(k + e \cdot 2 + b \cdot b)$ となる．比較手法 P の追加処理は，パトリシア上を根から葉まで遷移し，追加キーと葉に対応したキーを比較して追加位置を決定する．その後，再度パトリシア上を根から追加位置まで遷移するので，$O(\log_2 n + k + \log_2 n)$ となる [2]．提案手法の追加処理について，トライ上の遷移とキー比較に $O(\log_b n + k + \log_b n)$ が必要である．関数 InsLeaf はダブル配列と同様であり，関数 InsCommon は，一方向分岐を作成しないので $O(e \cdot 2)$ である．よって，提案手法は $O(\log_b n + k + \log_b n + e \cdot 2)$ となる．ゆえに，計算量の変化率が，ダブル配列では分岐数に依存して増加するのに対し，提案手法では低下する．

削除処理について，比較手法 D，DT は，削除キーに対応する葉から根へ遷移しながら，一方向分岐であるかを確認する．よって，$O(b \cdot k)$ である．提案手法の削除処理は，削除キーの探索後に葉と他の一つの節点を削除する．葉以外の節点を削除するかを判断し，削除時は関数 RenewalCheck を呼び，$O(b + b)$ となる．よって，提案手法は $O(\log_b n + k + b + b)$ となる．ゆえに，計算量の変化率が，ダブル配列では分岐数に関わらず一定であるのに対し，提案手法では低下する．

以上より，パトリシアに対して提案手法は，分岐数が 2 から b ($b \geq 2$) になったために深さが抑制され，探索時の遷移数が減少する．また，ダブル配列に対しては，更新処理時間も向上する．

4.2 実験的評価

実験では，キー集合として 1,000 万件をランダムに抽出した URI を母集合とした．

表 1 より，探索において，提案手法は比較手法 D より約 1.97 倍，比較手法 DT より約 1.88 倍，比較手法 P より約 2.13 倍高速である．

表 2 より，1,000 万件のキー集合で提案手法の追加速度は比較手法 P の約 0.79 倍と遅く，比較手法 D より約 1.41 倍，比較手法 DT より約 1.64 倍と高速となった．比較手法 P が高速となったのは，関数 InsLeaf のような計算量の多い処理を必要としないデータ構造をとっているからである [2]．比較手法 DT と提案手法を比較した場合，関数 NewBase の計算回数が減少したため，提案手法が高速となっている．関数 NewBase から，E-LINK を

表 1　探索処理に対する実験結果
Table 1　The simulation results of search processing.

	探索時間（μs）			トライ上の遷移数
	トライ上	配列 TAIL 上	合計	
比較手法 D	4.02	0.00	4.02	58.48
比較手法 DT	3.68	0.18	3.86	48.67
比較手法 P	3.89	0.47	4.36	52.21
提案手法	1.60	0.44	2.04	15.97

表 2　更新処理に対する実験結果
Table 2　The simulation results of updating processing.

キー数		100 万	500 万	1,000 万
追加時間（μs）	比較手法 D	10.41	12.94	14.22
	比較手法 DT	8.54	17.55	16.51
	比較手法 P	7.79	7.93	7.94
	提案手法	13.56	12.12	10.07
削除時間（μs）	比較手法 D	21.12	25.96	29.34
	比較手法 DT	11.72	13.17	13.58
	比較手法 P	6.44	6.60	6.62
	提案手法	5.31	5.43	5.43
関数 NewBase	比較手法 D	1.01	5.80	35.83
の計算回数	比較手法 DT	1.05	32.09	29.44
	提案手法	1.01	27.37	20.79

走査しながら遷移集合の全要素による終点が未使用要素であるかを判断する．つまり，未使用要素が多いと E-LINK の走査における計算量が多くなる．一方，比較手法 D は未使用要素がほとんど存在せず E-LINK の走査における計算量が少ない．よって，関数 NewBase の計算回数は少ない．しかし，追加節点数の増加により追加処理全体の時間を提案手法よりも要する．

削除処理について，1,000 万件のキー集合において提案手法の削除速度は，比較手法 D より約 5.40 倍，比較手法 DT より約 2.50 倍，比較手法 P より約 1.46 倍高速であった．これは，提案手法が最大二つの節点を削除するのに対し，比較手法 D，DT は最大キー長個の節点を削除するためである．また，提案手法が比較手法 P よりも高速となったのは，表 1 より削除キーの探索処理に差があるためである．

表 3 に，トライと配列 TAIL における使用空間を示す．

1,000 万件におけるキー集合の使用空間について，提案手法におけるトライと配列 TAIL での合計使用空間は，比較手法 D の 0.51 倍，比較手法 DT の 1.57 倍，比較手法 P の 1.10 倍である．しかし，提案手法におけるトライの使用空間は，比較手法 D の 0.12 倍，比較手法 DT の 0.54 倍，比較手法 P の 1.56 倍であった．よって，比較手法 DT よりも合計使用空間が大きくなったのは，キー全体を配列 TAIL に格納したことにより配列 TAIL の使用空間が増加したためである．

表 3　使用空間に対する実験結果
Table 3　The simulation results of used area.

キー数		100 万	500 万	1,000 万
トライに関する	比較手法 D	60.78	76.03	76.09
使用空間（MB）	比較手法 DT	15.27	17.36	17.35
	比較手法 P	5.19	6.00	6.01
	提案手法	8.12	9.38	9.38
配列 TAIL の	比較手法 DT	6.54	7.35	7.35
使用空間（MB）	比較手法 P，提案手法	25.31	29.27	29.28

以上より，提案手法は，ダブル配列に対して更新処理が有効である．パトリシアに対しては，探索速度が向上し，使用空間もほぼ同等であり有効であるといえる．

5.　おわりに

本論文では，ダブル配列上の一方向分木を削除し，パトリシアを多分木に拡張した基数探索法を提案し，実験で有効性を示した．今後の課題として，半無限文字列のようなキーが長いデータに対して実験を行い，詳細に評価することが挙げられる．

[文献]

[1] J. Aoe. An efficient digital search algorithm by using a double-array structure. *IEEE Trans. on Software Engineering*, Vol. 15, No. 9, pp. 1066–1077, 1989.

[2] D. R. Morrison. Patricia practical algorithm to retrieve information coded in alphanumeric. *Journal of the ACM*, Vol. 15, pp. 514–534, 1968.

[3] 中村康正, 望月久稔. 圧縮ディジタル探索木における辞書情報更新の高速化手法. 情報処理学会：データベース, Vol. 47, SIG 13（TOD 31）, pp. 16–27, 2006.

[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.

望月 久稔　Hisatoshi MOCHIZUKI
大阪教育大学教育学部教養学科情報科学講座講師．情報検索，自然言語処理，知識表現の研究に従事．情報処理学会，電子情報通信学会，人工知能学会，自然言語処理学会，日本データベース学会各正会員．

中村 康正　Yasumasa NAKAMURA
平成 19 年大阪教育大学大学院修士課程終了．情報処理学会正会員．現在四條畷学園高等学校常勤講師．

尾崎 拓郎　Takuro OZAKI
大阪教育大学教育学部教養学科情報科学専攻在学中．情報検索に関する研究に従事．情報処理学会学生会員．