パトリシアトライの一次元配列構造への圧縮手法

A Compression Method of the Patricia Trie by Using a Single Array Structure

山本 一徳 †　**獅々堀 正幹** †　**柘植 覚** †　**北 研二** ‡

Kazunori Yamamoto　Masami Shishibori　Satoru Tsuge　Kenji Kita

1. はじめに

近年，計算機システムの高性能化とネットワーク化に伴い，膨大な電子化された情報が計算機上でアクセス可能になっている．これらの膨大な情報の中から必要な情報を見つけるために情報検索の技術は必要不可欠である．特に，文書データや自然言語辞書を効率よく検索する索引手法の一つであるトライ法には様々な圧縮手法が提案されている．

本稿では，2 進木トライのデータ構造の 1 つであるパトリシアトライに注目する．一般に，パトリシアトライでは 1 本の枝しか持たないノードを削除することで木の深さを浅くしている．しかし，削除ノードに関する情報が別途必要になるため，索引のサイズが大きくなるといった問題点がある．現在，索引のコンパクト性を実現するために，様々なパトリシアトライの圧縮手法が提案されているが，各圧縮手法には時間効率が低下してしまうといった欠点がある．そこで本稿では，パトリシアトライを一本の一次元配列構造に圧縮する手法を提案する．本手法では，一次元配列内の各要素の下位数ビットに削除ノードに関する情報を，残りの上位数ビットにトライ上の各ノード遷移に関する情報を割りあてることで空間効率と時間効率を改善している．

以下，第 2 章でトライの圧縮法，第 3 章でパトリシアトライの圧縮手法について述べる．第 4 章で提案手法について説明し，第 5 章で本手法の実験を行い，最後にまとめと今後の課題を述べる．

2. トライの圧縮法

トライとは登録キーの文字自身によって分岐する枝を決定する, 木を用いた検索アルゴリズムのことである．本章では，トライの圧縮法であるダブル配列法 [1] について述べる．

ダブル配列法では BASE，CHECK と呼ばれる 2 つの一次元配列によってトライを実装する．BASE，CHECK の 2 つの配列は以下のように定義される．

BASE：CHECK に記憶した情報に対するオフセットを格納
CHECK：どのノードからの遷移かを確認するためのノード番号を格納

この BASE，CHECK2 つの一次元配列を用いて，ノード s からノード t へ向かう文字 a による遷移 g(s,a)=t を以下の式で定義する．

$$t = \mathrm{BASE}[s] + \mathrm{code}(a) \ (\mathrm{code}：内部表現値)$$
$$\mathrm{CHECK}[t] = s$$

例として図 1 に登録キー K={“bad#”,“be#”} (#は終端記号) に対するトライを示す．なお，文字 “#”，“a”，⋯，“z” の内部表現値を 1，2，⋯，27 とする．

また，図 2 に図 1 のトライに対するダブル配列表現を示す．なお，終端ノードに対応する BASE 値には遷移の終わりを意味する負の値を格納する．

† 徳島大学工学部
‡ 徳島大学高度情報化基盤センター

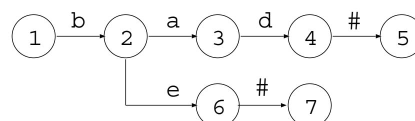

図 1: トライ

	1	2	3	4	5	6	7
BASE	1	-2	0	1	5	-1	1
CHECK	0	7	4	1	3	5	4

図 2: ダブル配列

ダブル配列法は検索キーの長さを k とすると，k 回だけ二つの式を確認するだけですむので計算量 O(k) で検索をすることができ，非常に高速である．また，記憶用量も無駄がなく，非常にコンパクトである．

3. パトリシアトライの圧縮法

3.1 パトリシアトライとは

登録キーの文字自身ではなく，登録キーのビット表現 1 つ 1 つによって分岐する枝を決定するトライとして，2 進木トライがある．2 進木トライは完全 2 進木トライ，正規 2 進木トライ，パトリシアトライ [2] の 3 種類の木構造に分類される．

完全 2 進木トライとは，登録キーのビット表現全てを登録した木構造のことである．正規 2 進木トライとは，完全 2 進木トライを終端ノードから遡って分岐がおこるまでのノードを全て削除した木構造のことである．削除ノードに関する情報は正規 2 進木トライの終端ノードが保有する．パトリシアトライとは，正規 2 進木トライから 1 本の枝しか持たないノードを全て削除した木構造のことである．削除ノードに関する情報は各ノードが保有する．

キー集合 K={“reach”,“read”,“trie”,“try”} に対するパトリシアトライを図 3 に示す．ノード内の番号はノード番号，各ノードの近くにある () 内にある数字は各ノードが保有する削除ノード数を意味している．

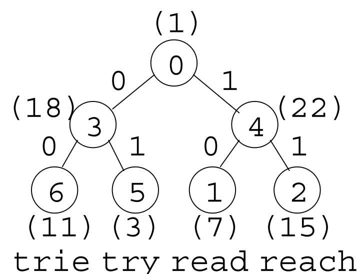

図 3: パトリシアトライ

3.2 パトリシアトライの圧縮法

パトリシアトライの圧縮法として，PAT array 及び PaCB 木を取り上げ，それらの圧縮法を説明する．

3.2.1 PAT array

PAT array はパトリシアトライを配列構造に圧縮したもので，Manber ら [3] によって提案された．パトリシアトライでは索引の構築時間が文書中の文字数に依存するのに対し，PAT array では，索引の構築時間が文字数に依存しない．PAT array のデータ構造は，登録キーを ASCII の文字コード順にソートした後，各始点位置を配列構造に格納したものである．

PAT array は索引の構築時間が文字数に依存しないため，索引を高速に構築できるといった長所があるが，索引のサイズが大きい，検索速度が遅い，ソート処理の計算コストが高いといった欠点がある．

3.2.2 PaCB 木

PaCB 木 [4] はパトリシアトライをコンパクトなビット構造で表現する圧縮法である．PaCB 木は以下に示す treemap，nodemap と呼ぶ 2 つのビット列と TBL と呼ぶ 1 つのテーブルから構成される．

treemap はパトリシアトライの木の状態を表すビット列，nodemap は各内部ノードが保持している削除ノード数の状態を表すビット列，TBL は登録キーの始点位置を格納する表構造である．

PaCB 木は索引がコンパクトになるが，検索時にすべてのビットを走査する必要があるため登録データが大規模になると検索効率が悪化してしまう欠点がある．

4. パトリシアトライの一次元配列構造への圧縮手法

ダブル配列法は，空間効率，時間効率ともに非常に優れた圧縮手法である．このダブル配列法の圧縮アルゴリズムをパトリシアトライに適用した場合，パトリシアトライが 2 進木トライであるため，BASE の一次元配列だけで実装することが可能になる．しかし，ダブル配列法には削除ノードに関する情報を格納する機能がないため，ダブル配列法をパトリシアトライに適用することはできない．

そこで本稿では，ダブル配列法をパトリシアトライに適用できるように改良し，パトリシアトライを 1 本の int 型一次元配列に圧縮する手法を提案する．本手法では，パトリシアトライを pTree と呼ばれる一次元配列に圧縮する．

一次元配列配列 pTree の各要素での負の値はキー集合が格納されるバケットファイルへのファイルポインタの値，また，各要素での正の値は上位 (32-N) ビットと下位 N ビットに分けられ，それぞれが base,eliminate と呼ばれる変数に格納される．なお，base は推移先の配列要素番号，eliminate は削除ノード数を示している．

例として，キー集合 K={“be”,“bt”} のパトリシアトライと，そのパトリシアトライを配列 pTree に圧縮した結果を以下の図 4 に示す．また，配列 pTree の各要素での正の値を base,eliminate に分ける方法を以下の図 5 に示す．なお，図 5 では．N=10 として base,eliminate に分けた例である．

パトリシアトライを配列 pTree に圧縮することでコンパクトな索引になる．また，検索時間は登録データの大きさに依存せず，検索キーの長さに依存する．よって，大規模なデータでも高速に検索することができる．

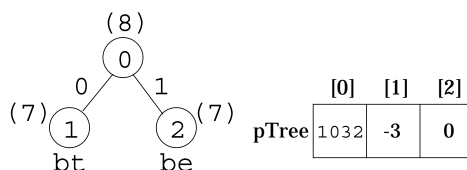

図 4: パトリシアトライと配列 pTree

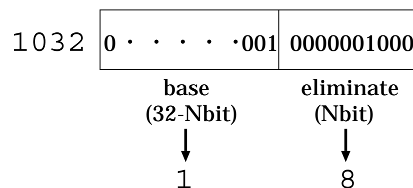

図 5: base と eliminate

以下に配列型パトリシアトライを用いた検索アルゴリズムを示す．

step1 検索キーをビット表現に変換して，配列 Key に格納.

step2 配列 pTree の要素 pTree[n] の値を取りだす．(n の初期値は 0)

step3 step2 で取りだした値が 0 以下だった場合は step11 へ．そうでなければ、step4 へ.

step4 step2 で取りだした値 (int 型) の上位 (32-N)bit(base) と下位 Nbit(eliminate) を求める．

step5 次式より次回参照すべき要素番号 next_node を求める．

next_node = eliminate + all_eliminate(初期値 0)

step6 next_node が配列 Key の要素数以上なら，検索結果候補が複数あると判断し，step9 へ．そうでなければ，step7 へ.

step7 次式よりこれまで辿った枝の数の合計 all_eliminate を更新する，

all_eliminate += eliminate + 1

step8 次式より n を更新して，step2 に戻る．

n = Key[next_node] + base

step9 pTree[base],pTree[base+1] の値を取りだす.

step10 step9 で取りだした 2 つの値それぞれに対して，0 以下なら step11 へ．正の値なら上位ビットから base を求めて step9 へ.

step11 取りだされた負の値を正の値に変換．変換した値をバケットファイルのファイルポインタとして，求めたファイルポインタから始まる行を全て取得する．

step12 取得した行に検索キーが含まれているかチェック．検索キーが含まれていれば，検索結果として出力する．

表 1: 実験結果

-	手法	URL データ					郵便番号データ				
件数	-	10000	15000	20000	25000	30000	40000	60000	80000	100000	120000
サイズ	-	0.42	0.64	0.85	1.05	1.27	0.32	0.46	0.64	0.80	0.96
比率	-	0.167	0.156	0.156	0.157	0.154	0.050	0.048	0.047	0.048	0.049
索引	リスト	2.82	3.89	5.01	6.15	7.25	1.03	1.54	2.07	2.60	3.12
サイズ	ダブル	1.98	2.73	3.53	4.34	5.08	0.70	1.04	1.39	1.75	2.10
(Mbyte)	提案	0.50	0.76	1.01	1.25	1.51	0.61	0.92	1.24	1.55	1.86
索引	リスト	0.16	0.23	0.29	0.36	0.42	0.30	0.38	0.49	0.59	0.67
構築時間	ダブル	0.28	0.45	0.63	0.81	0.97	0.13	0.22	0.28	0.37	0.45
(Sec)	提案	0.47	0.71	0.93	1.15	1.42	2.01	2.69	3.52	4.51	4.83

5. 評価実験

評価実験では URL データ，郵便番号データ [5] を用いて，索引構築時間，索引サイズ，検索時間を計測し，既存手法との比較を行った．なお，比較した既存手法としては，通常のトライをリスト構造で実装した手法とダブル配列法を基にした辞書作成プログラム darts[6] を用いた．

5.1 実験条件

実験で使用した計算機は CPU : Celeron 2.5GHz，メモリ : 256MB，削除ノードに使用するビット数 N は 10 とした．各種実験データの件数のファイルサイズを「サイズ」(単位 : MByte)，削除ノード (子供を 1 つしか持たないノード) の比率を「比率」として表 1 内に示す．なお，削除ノードの比率は以下の式を用いて求めた．

削除ノードの比率 = (削除ノード数)/(全ノード数)

また，URL データと郵便番号データの二種類のデータを用いたのは，削除ノードの割合で実験結果がどのように変わるかを調べるためである．

5.2 実験結果

索引サイズ・索引構築時間の実験結果を表 1 に，検索時間を図 6，7 に示す．なお，表 1 内の「リスト」がリスト構造,「ダブル」がダブル配列による手法を示す．また，図 6，7 においては，◯ が本手法の結果，× がリスト構造，△ がダブル配列の結果である．また，検索時間は登録データすべてを検索し，その合計時間を計測した．

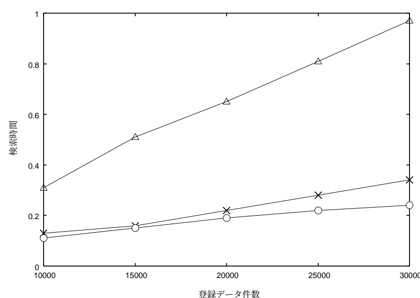

図 6: URL データに対する検索時間

評価実験より索引サイズ，検索時間に関しては，従来手法より精度の向上が見られた．しかし，索引構築時間に関しては従来手法より遅くなってしまったが，実用に耐えるものである．

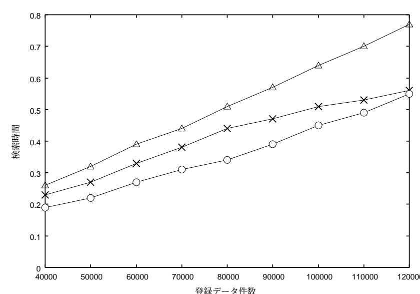

図 7: 郵便番号データに対する検索時間

また，郵便番号データより URL データの方がいい結果が得られた．これは，URL データより郵便番号データの方が削除ノードが多かったためだと考えられる．よって，本手法は，郵便番号データのようなキーの密度が高いデータより，URL データのようなキー文字列が長く密度が低いデータの方が構築，検索ともにより有効であると考えられる．

6. まとめ

本稿では，配列型パトリシアトライへの圧縮手法を提案した．評価実験より従来法と比べ，索引サイズ，検索時間に関しては良い結果が得られた．今後，削除ノードに使用するビット数 N と検索効率との関係を明らかにし，キー集合によって最適な N の値を設定できる機能を追加したい．

参考文献

[1] 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.4，pp.1592-1600

[2] 北研二，津田和彦，獅々堀正幹，“情報検索アルゴリズム”，共立出版

[3] Manber，U. and Myers，G.，“Suffix array： A new method for on-line string searches”，SIAM Journal on computing，Vol.22，No.5，pp.935-948.

[4] Shishibori，M.，Okada，M.，Sumitomo，T.and Aoe. J.，“Design of a compact data structure for the patricia trie”，IEICE Transactions on Information and Systems，Vol.E81-D，No.4，pp.364-371.

[5] http://www.post.japanpost.jp/zipcode/download.html

[6] http://chasen.org/ taku/software/darts/