Prefix/Patricia Trie の入れ子による辞書圧縮

矢田 晋
フリー
susumu.yata@gmail.com

1. はじめに

Trie は辞書データ構造の一種であり，特徴的な検索機能を理由として，自然言語処理において広く用いられている．これまでに多くの内部データ構造が提案されており，高速な検索を特長とするダブル配列であれば，ハッシュ表と比べても遜色ないほどの検索時間を誇る．一方で，簡潔データ構造の一種である LOUDS[1] を用いた Trie 辞書は，文字列の集合を圧縮する目的で使えるほどにコンパクトである．

Prefix/Patricia Trie[2] への特殊化は，構成ノード数の削減によって Trie の空間効率を高める手法であり，辞書の圧縮に有効であることが以前より知られている．特に，Prefix Trie と LOUDS の組み合わせについては，試験的な実装[1]が公開されるとともに，極めてコンパクトに辞書を表現できることが判明している．

本稿では，辞書の圧縮における Patricia Trie の有効性について述べ，Prefix/Patricia Trie の入れ子によって辞書をさらに圧縮できることを示す．これらの圧縮は，キーワード辞書に対して有効であるほか，形態素 n-gram コーパスや，品詞・読みなどを含む形態素辞書にも有効であり，広範な応用が期待される．

2. Prefix/Patricia Trie

2.1 Trie

Trie は文字をラベルとする木構造であり，ラベルを手がかりとして探索することにより，格納されている文字列（キー）を検索したり，復元したりすることができる．また，キー前半の共通部分が Trie 上で併合されることから，文字列の集合に対する効率的な格納手段として用いられることもある．

例として，6 つの単語からなるキー集合 { “*brace*”, “*oct*”, “*octet*”, “*race*”, “*role*”, “*url*” } に対する Trie を**図 1** に示す．この例では，左端の経路に “*brace*” が格納されていることや，“*oct*” が “*octet*” の前半に併合されていることを確認できる．

一般的な実装において，各ノードに割り当てる領域は一定であることから，Trie の空間使用量は各ノードの大きさとノード数の積により求められる．そのため，Trie を圧縮する戦略は，各ノードの縮小とノード数の削減に大別できる．本稿で紹介するのは，LOUDS[1] による各ノードの縮小を前提として，ノード数を削減する目的で Prefix/Patricia Trie を利用し，さらに，取り除いたノードの行き先として新たな Prefix/Patricia Trie を利用する手法である．

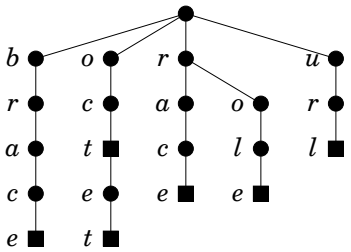

図 1 Trie

表 1 Wikipedia の見出しに対する Trie のノード数

[×1,000]	#keys	Trie	Prefix	Patricia
Japanese[2]	1,147	11,164	2,642	1,564
English[3]	8,279	70,677	18,351	11,301

2.2 Prefix Trie

ただ 1 つのキーと対応する経路の内部にあるノードを Trie から取り除くことで得られる木構造が Prefix Trie である．キーの後半は併合されにくいという Trie の特徴に着目した手法であり，**表 1** に示すように，日英 Wikipedia の見出しを登録した Trie に適用すると，ノード数を 1/4 程度にまで削減することができる．

例として，図 1 の Trie から得られる Prefix Trie を**図 2** に示す．この例では，11 個のノードが取り除かれ，“*brace*”, “*et*”, “*ace*”, “*ole*”, “*url*” という 5 つのラベルにまとめられている．

Prefix Trie では，葉のラベルが可変長になるため，2 文字以上のラベルを格納する補助データ構造が必要となる．一般的には，ラベルを文字列として別に格納し

[1] ux-trie: http://code.google.com/p/ux-trie/

[2] http://download.wikimedia.org/jawiki/20101102/jawiki-20101102-all-titles-in-ns0.gz

[3] http://download.wikimedia.org/enwiki/20110115/enwiki-20110115-all-titles-in-ns0.gz

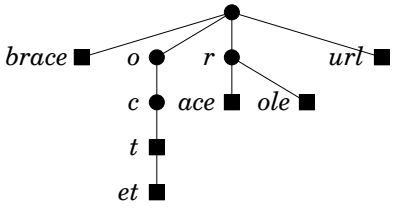

図 2 Prefix Trie

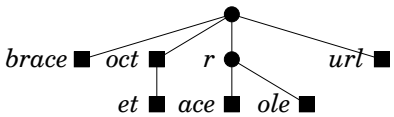

図 3 Patricia Trie

ておき，ラベルの位置を葉に持たせることで，文字によるノードの置き換えを実現する．そして，ノードと文字の差分による領域削減の効果が葉とラベルのリンクにかかるコストを上回る場合，辞書は圧縮されることになる．

2.3 Patricia Trie

同じキー集合と対応する経路の内部にあるノードを Trie から取り除くことで得られる木構造が Patricia Trie[2] である．Prefix Trie と比較すると，取り除くノードの条件が緩くなり，根を除くすべてのノードが可変長のラベルを持つという違いがある．表1 に示すように，Trie から Prefix Trie への変化ほど劇的ではないものの，Prefix Trie より少ないノード数で辞書を構成できるようになる．

例として，図1 の Trie から得られる Patricia Trie を**図 3** に示す．この例では，{ “*oct*”, “*octet*” } と対応する経路の内部にある 2 つのノードが新たに取り除かれ，“*oct*” というラベルにまとめられている．

Patricia Trie はノード数の削減に有効であるものの，探索において，木構造とは別に格納されているラベルを繰り返し参照することになり，キャッシュミスが発生しやすいという欠点を持つ．一方で，与えられた文字列で始まるキーの数を求める場合など，ラベルの参照を省略できる状況では，ノード数の削減が時間効率の面でも有利に働く．

3. Prefix/Patricia Trie の入れ子

3.1 Double-Trie

Prefix/Patricia Trie の入れ子により得られるデータ構造の前身となる存在が Double-Trie[3] である．Double-Trie は，Prefix Trie に含まれる 2 文字以上のラベルについて，2 文字目以降を逆順にして 2 つ目の Trie に格納するというデータ構造であり，ダブル配列との組み合わせによる高速かつコンパクトな辞書検索の実現を目的として提案されている．

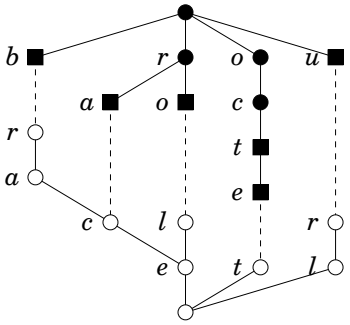

図 4 Double-Trie

例として，図1 の Trie から得られる Double-Trie を**図 4** に示す．図2 の Prefix Trie に相対するように 2 つ目の Trie が配置されており，ラベルの 2 文字目以降である “*race*”, “*t*”, “*ce*”, “*le*”, “*rl*” が逆順に格納されている．

3.2 空間効率を重視した入れ子構造

辞書の圧縮を目的とする場合，ラベルを次の Trie に格納するという考え方は Double-Trie と同様に有効であるものの，木構造の簡潔表現が内部データ構造の有力な候補になるとともに，Patricia Trie や多段の入れ子が現実的な選択肢として挙がってくる．すなわち，探索に時間がかかる木構造を導入することによって，ラベルの参照や Trie 間の移動において発生するキャッシュミスなどの影響を意識する必要性が薄くなる．

そこで，本研究では，LOUDS と Prefix/Patricia Trie を組み合わせ，さらに入れ子にすることで空間効率の向上を目指した．また，2 文字以上のラベルを格納する方法については，1 文字目も含めて次の Prefix/Patricia Trie に格納するように変更したほか，最後の Prefix/Patricia Trie を構築した後，ラベルの文字列化において，重複しているラベルを併合するように変更した．

例として，図1 の Trie から得られる 3 段の Patricia Trie を**図 5** に示す．この例において，“*brace*” は 1 段目の Patricia Trie で分割されず，2 段目の Patricia Trie で “*e*”, “*ca*”, “*rb*” に分割されている．3 段目の Patricia Trie においては，“*ca*” が “*c*”, “*a*” に分割されている一方で，“*rb*” は最終的に文字列として格納されている．

図5 に示されているように，入れ子が深くなるにつれて，キーは徐々に短いラベルへと分割されていく．そして，ラベルの集合が十分に小さくなった時点で，残

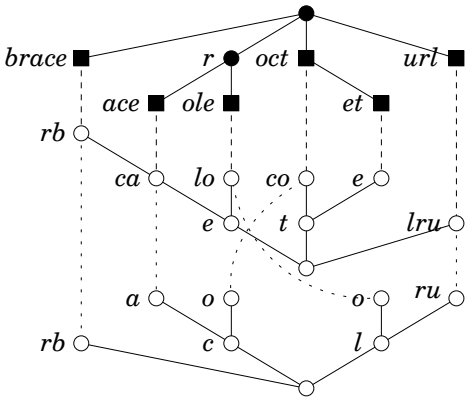

図 5 Patricia Trie の入れ子

表 2 評価に用いたキー集合の詳細

	#keys [×1,000]	total length [kb]	ave. length [bytes/key]
W_J	Japanese Wikipedia page titles[2]		
	1,147	23,809	20.766
N_7	Word 7-grams from NWC2010[5]		
	1,024	40,779	39.838
M_I	CSV files in IPAdic for MeCab[6]		
	392	30,775	78.484

りを文字列として保存することにより，Prefix/Patricia Trie の入れ子，すなわち辞書の圧縮は終了となる．

4. 評価

Prefix/Patricia Trie の入れ子を C++により実装し[4]，**表 2** のキー集合から構築した辞書の詳細として，辞書を構成する Prefix/Patricia Trie の数，ノード数の合計と辞書のサイズに加え，辞書の構築に要した時間とすべてのキーを検索するのに要した時間をキー数で割った値を**表 3** と**表 4** に示す．時間の計測に用いた環境は CPU: Core 2 Duo 1.60 GHz，OS: Ubuntu 10.04 であり，コンパイラには gcc 4.4.3 を使用した．

表 3 と表 4 を比べると，より多くのノードを削減できる Patricia Trie の方が優れた性能を示していることがわかる．入れ子の効果については，深さをパラメータとして，時間と空間のトレードオフになっていることが確認できる．また，キー集合による比較では，複雑な構成を持つ辞書を限界まで圧縮するには，深い入れ子が必要になるということが確認された．

以上の結果は，Prefix/Patricia Trie の入れ子が深さの調整によって幅広い用途に対応できる優れた辞書データ構造であることを示している．

[4]marisa-trie: `http://code.google.com/p/marisa-trie/`
[5]`http://s-yata.jp/corpus/nwc2010/ngrams/`
[6]`http://mecab.sourceforge.net/#download`

表 3 Prefix Trie の入れ子による辞書の詳細

	#tries [×1]	#nodes [×1,000]	size [kb]	build [μs/key]	lookup [μs/key]
W_J	1	2,642	10,693	4.666	3.201
	2	3,382	8,488	5.041	4.500
	3	3,574	8,085	5.093	4.745
N_7	1	3,744	22,058	6.712	5.178
	2	5,175	15,610	7.864	8.216
	3	6,023	14,039	8.480	9.613
	5	6,431	13,203	8.773	10.248
M_I	1	1,074	30,370	6.120	2.499
	2	3,192	23,500	10.609	7.472
	3	4,292	18,659	13.389	11.909
	10	6,177	12,402	18.234	21.141

表 4 Patricia Trie の入れ子による辞書の詳細

	#tries [×1]	#nodes [×1,000]	size [kb]	build [μs/key]	lookup [μs/key]
W_J	1	1,564	9,850	4.465	2.765
	2	2,088	7,057	4.666	4.265
	3	2,264	6,523	4.806	4.727
N_7	1	1,663	19,978	5.910	3.507
	2	2,491	12,410	6.848	6.809
	3	2,925	10,356	7.268	8.382
	5	3,259	9,169	7.542	9.281
M_I	1	568	30,078	5.559	2.142
	2	1,184	22,535	8.212	4.539
	3	1,730	15,988	10.175	7.855
	10	2,670	7,831	12.649	19.178

5. おわりに

Prefix/Patricia Trie の入れ子は，汎用的かつ高性能な辞書データ構造であり，これまで専用のデータ構造で対処していたような複雑な構成を持つ辞書について，同じように圧縮できる可能性を示唆している．

量と質の両面で高度化が進んでいる言語処理において，大規模・複雑な辞書の必要性が高まっていくと考えられるため，今後，幅広い応用が期待される．

参考文献

[1] O. Delpratt, N. Rahman and R. Raman. Engineering the LOUDS succinct tree representation. *WEA 2006*, pp. 134–145 (Oct. 2006).

[2] D. R. Morrison. PATRICIA—practical algorithm to retrieve information coded in alphanumeric. *Commun. ACM*, 15(4), pp. 514–534 (Oct. 1968).

[3] 森本勝士, 入口浩二, 青江順一. 二つのトライを用いた辞書検索アルゴリズム. 信学論 (D-II), J76-D-II(11), pp. 2374–2383 (Nov. 1993).