順序木の簡潔表現を用いたトライ辞書の評価

Evaluation of trie dictionaries based on succinct ordered trees

矢田 晋† 森田 和宏 泓田 正雄 青江 順一
Susumu Yata Kazuhiro Morita Masao Fuketa Jun-ichi Aoe

徳島大学大学院 ソシオテクノサイエンス研究部
Faculty of Engineering, University of Tokushima

†日本学術振興会特別研究員
JSPS Research Fellow

1. はじめに

多くの分野において大規模データが活用されるようになり，様々なデータ構造をコンパクトに表現できる簡潔データ構造 [1] に関する研究が進められている．辞書のデータ構造として有用なトライについても，順序木の簡潔表現 LOUDS[2] を用いた実装[1]が存在し，自然言語処理などに利用されている．本稿では，トライの実装において，ノードの順序や順序木の簡潔表現が検索時間に影響することを説明する．

2. トライ

2.1 トライの基本構造

トライは記号列（キー）の集合を格納する木構造であり，キーを構成する記号を根から葉への経路上にラベルとして付与する．さらに，本研究では，キーの終端を識別するために，各ノードに終端かどうかを示すフラグ（終端フラグ）を持たせる．また，キーの数が m で与えられるとき，各キーには $0 \leq x < m$ を満足する整数 x を ID として割り当てるものとする．

2.2 トライの簡潔表現

順序木の簡潔表現とラベル列を組み合わせることにより，トライをコンパクトに表現できる [3]．この組み合わせは，ノード間の移動をポインタにより実現する一般的なデータ構造と比べて，サイズが 1/10 程度になるため，キー数が 1 億件を超える大規模なキー集合への適用が可能である．

本研究では，BWT (Burrows-Wheel Transfer) を応用した圧縮 [3] などは適用せず，順序木の簡潔表現と検索時間の関係を評価する．また，キー終端を識別するために終端フラグのビットベクトルを追加し，キー ID の算出を高速化するために簡潔データ構造を付加する．

2.3 トライにおけるノードの順序

トライからキーを検索する場合，経路上の各ノードにおいて，入力記号をラベルとして持つ子を検索する必要がある．このとき，二分探索やハッシュ検索の可能な構成でなければ，子のリストを線形探索することになる．そして，通常，ノードの出現頻度は大きく偏るため，出現頻度の高い順に子を並べておくことにより，平均探索時間を大幅に短縮できる．また，キーに関連付けられた重みに応じて子の順序を変更することにより，重み付き入力補完を効率化できる [4]．

次に，キャッシュの利用効率に着目すると，兄弟関係にあるノードの近接配置により，子の検索にかかる線形探索の効率を向上できることがわかる．一方で，深さ優先のノード配置は，親子関係にあるノードが近くに配置されるという特徴から，キャッシュの利用効率向上につながり，検索時間の短縮に貢献する．

3. 順序木の簡潔表現

3.1 ビットベクトルの簡潔データ構造

順序木の簡潔表現は，ビットベクトルに対して定義される以下のメソッド rank, select を用いて実現される．

- rank(x, i): $[1..i]$ の範囲に含まれる x の数を返す．
- select(x, i): 先頭から i 番目の x の位置を返す．

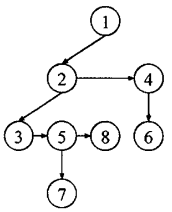
そして，rank, select を高速に実現するための付加情報を，ビットベクトルに対する簡潔データ構造という [1]．これまでに数多くの実装が提案されており [5, 6]，時間と空間のトレードオフ，0/1 の比率などを考慮して選択することが望ましい．また，rank と比べて select は実装が複雑になりやすく，処理に時間がかかるため，rank のみを用いて実現できる簡潔表現が望ましい．

3.2 幅優先な順序木の簡潔表現

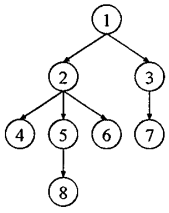
幅優先な順序木の利点は，ビットベクトルの簡潔データ構造を用いてシンプルに実装できるため，サイズが小さいことである．しかし，親子関係にあるノードが近くに配置されないという欠点を持つ．代表的な簡潔表現の概要を以下に示し，それぞれの例を**図 1** に示す．

- LOB (Level-order bitmap)[2]
  順序木の二分木表現に対するビットベクトル表現であり，ノード間の移動を rank のみで実現できる．しかし，二分木表現への変換において兄弟関係が

[1]Tx: Succinct Trie Data structure: http://www-tsujii.is.s.u-tokyo.ac.jp/~hillbig/tx-j.htm



LOB



LOUDS/LOUDS++

LOB[2]		$_1$10 $_2$11 $_3$01 $_4$10 $_5$11 $_6$00 $_7$00 $_8$00
LOUDS[2]		$_1$110 $_2$1110 $_3$10 $_4$0 $_5$10 $_6$0 $_7$0 $_8$0
LOUDS++[7]	Child	$_1$1 $_2$1 $_3$1 $_4$0 $_5$1 $_6$0 $_7$0 $_8$0
	Sibling	$_1$0 $_2$1 $_3$0 $_4$1 $_5$1 $_6$0 $_7$0 $_8$0

図 1 幅優先な順序木の簡潔表現

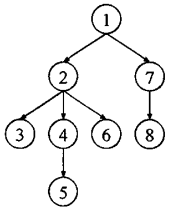
親子関係に置き換えられるため，順序木において兄弟関係にあるノードが近接配置されない．

- LOUDS (Level-order unary degree sequence)[2]
- LOUDS++[7]
  順序木のビットベクトル表現であり，兄弟関係にあるノードが近接配置されるものの，子への移動に rank, select の両方が必要となる．LOUDS はノード ID の取得に rank を用いる．

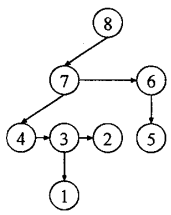
3.3 深さ優先な順序木の簡潔表現

深さ優先な順序木の利点は，親子関係にあるノードを近くに配置できることである．しかし，探索用の補助データ構造や整数列の符号化が必要になるため，幅優先の簡潔表現と比べてサイズが大きいという欠点を持つ．代表的な簡潔表現の概要を以下に示し，それぞれの例を**図 2** に示す．

- BP (Balanced parentheses)[8]
- DFUDS (Depth-first unary degree sequence)[9]
  順序木の括弧列表現であり，対応する括弧の位置を求めるための補助データ構造が必要になる．BP は兄弟への移動に rank を用いるのみであるのに対し，DFUDS はノード間の移動に rank, select の両方が必要となる．どちらも，ノード ID の取得には rank を用いる．
- PODS (Post-order difference sequence)
  順序木をノード ID の差分により表現したデータ構造であり，ほとんどの差分は 0/1 となるものの，親子関係については符号化が必要になる．ノード ID の割り当ては，順序木の構築において，兄弟が確定したノードから順におこなわれる．子への移動に rank を用いるのみであり，兄弟関係にあるノードは近接配置される．



BP/DFUDS



PODS

BP[8]		$_1$($_2$($_3$( ) $_4$($_5$( )) $_6$( )) $_7$($_8$( )))
DFUDS[9]		($_1$(( ) $_2$((( ) $_3$) $_4$ ( ) $_5$) $_6$) $_7$ ( ) $_8$)
PODS	Child	$_1$0 $_2$0 $_3$2 $_4$0 $_5$0 $_6$1 $_7$3 $_8$1
	Sibling	$_1$0 $_2$0 $_3$1 $_4$1 $_5$0 $_6$0 $_7$1 $_8$0

図 2 深さ優先な順序木の簡潔表現

4. おわりに

本稿では，大規模な辞書の運用にトライの簡潔表現が有効であることを述べ，検索時間を左右する要素として，ノードの順序とビットベクトルのデータ構造，および順序木の簡潔表現を説明した．今後は，実験により最適な組み合わせを求めるとともに，補助記憶装置上での運用や共通部分木の併合による効率化を検討する予定である．

参考文献

[1] 定兼邦彦. 大規模データ処理のための簡潔データ構造. 情報処理, Vol. 48, No. 8, pp. 899–902, 2007.

[2] Guy Jacobson. Space-efficient static trees and graphs. In *Proceedings of FOCS 1989*, pp. 549–554, 1989.

[3] Paolo Ferragina, Fabrizio Luccio, Giovanni Manzini, and S. Muthukrishnan. Compressing and indexing labeled trees, with applications. *Journal of the ACM*, Vol. 57, No. 1, pp. 1–33, 2009.

[4] Susumu Yata, Kazuhiro Morita, Masao Fuketa, and Jun-ichi Aoe. Customized tries for weighted key completion. In *Proceedings of ICCEA 2010*, 2010.

[5] Daisuke Okanohara and Kunihiko Sadakane. Practical entropy-compressed rank/select dictionary. In *Proceedings of ALENEX 2007*, pp. 549–554, 2007.

[6] Sebastiano Vigna. Broadword implementation of rank/select queries. In *Proceedings of WEA 2008*, pp. 154–168, 2008.

[7] O'Neil Delpratt, Naila Rahman, and Rajeev Raman. Engineering the LOUDS succinct tree representation. In *Proceedings of WEA 2006*, pp. 134–145, 2006.

[8] Ian J. Munro and Venkatesh Raman. Succinct representation of balanced parentheses and static trees. *SIAM Journal on Computing*, Vol. 31, No. 3, pp. 762–776, 2001.

[9] David Benoit, Erik D. Demaine, J. Ian Munro, Rajeev Raman, Venkatesh Raman, and S. Srinivasa Rao. Representing trees of higher degree. *Algorithmica*, Vol. 43, No. 4, pp. 275–292, 2005.