E-023

Migemo における高速かつコンパクトなトライの検討
A Study of Fast and Compact Trie for Migemo

小倉 直徒†

1. はじめに

日本語の検索技術として Migemo[1] が知られている．Migemo とは，漢字やひらがなを含んだ文字列を，ローマ字から変換せずに検索することができる検索手法である．

Migemo の C 言語実装である C/Migemo では，辞書データの格納にトライを用いているが，そのデータ構造である二重連鎖木ではデータサイズが大きいという問題がある．Migemo はテキストエディタの起動と同時に読み込まれて利用されることが多く，長時間メモリを占有する．例えば，Migemo 機能を持つ KaoriYa 版 Vim では，初期状態では使用メモリは 7.1MB であるが，Migemo を有効にすると 40.5MB となる．

そこで，簡潔データ構造の一種である LOUDS[2]をデータ構造として検討した．LOUDS は省メモリとして知られている．実装した結果，メモリ使用量が小さくなることが確認できたものの，検索時間が大きくなった．

本研究では，高速かつ省メモリな Migmeo の実装を，文字エンコード及び辞書のツリー構造の候補を複数挙げ，比較することで，最適な実装について議論する．

2. Migemo の拡張

2.1 Migemo の実装

Migemo の検索では，ユーザより入力されたローマ字を，(a)ひらがなに変換し，(b)読みが前方一致する単語を辞書から集め，(c)正規表現を生成している．この処理の中では 4 つのトライが使われており，C/Migemo では全て二重連鎖木で実装されている．二重連鎖木は，実装が簡単で更新に強いというメリットがあるものの，データサイズが大きく，読み込みに時間がかかるという問題がある．そこで，処理の高速化のため，(a)及び(c)で使われるトライを，それぞれ二重配列，三分探索木をデータ構造として採用した．二重配列は検索が高速，三分探索木は更新が高速という特徴がある．ともにデータサイズは大きくなるものの，辞書データと比較し十分小さいため，データサイズへの影響は小さい．

Migemo を高速化するにあたり，実験用の Migemo を Go 言語で実装した．上記のとおりデータ構造を変更すると，検索時間のうち 54%が(b)に費やされていた．そのため，以降は辞書に着目し議論する．

2.2 辞書の文字エンコーディング

英字のみの文字列では，文字数とバイト数は一致するの，日本語文字の場合，文字エンコードにより可変となる．UTF8 は，英字では 1 バイトであるが，多くの日本語の文字では3バイトとなる．一方，UTF16 では英字・日本語文字に関わらずほとんどの文字において 2 バイトである．Migemo で利用する辞書に格納される文字は日本語文字が多いため，1 文字あたり 3 バイトではなく 2 バイトとなる UTF16 のほうが適しているように思われる．

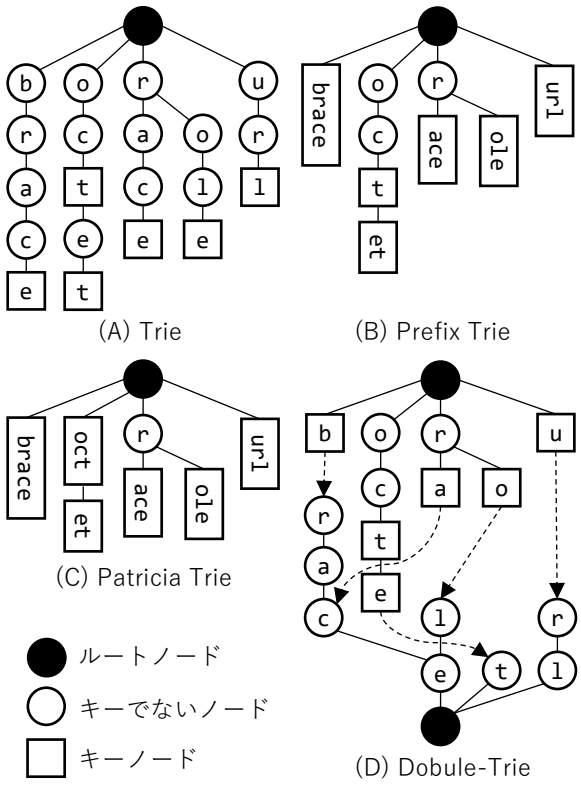

図 1　トライ構造の比較

加えて，UTF8 では最小のバイト単位は 1 バイトのため，日本語文字を表すには 3 ノード必要となる．一方，UTF16 では最小で 2 バイトのため，日本語文字では 1 ノードとなり，ノード数は UTF16 の方が小さくなる．

2.3 辞書のツリー構造

LOUDS はツリー構造を 1 ノードあたり(2 ビット+ラベルサイズ)で表現することができる，非常にコンパクトなデータ構造である．トライでは，ツリー構造を用いることで複数の文字列を効率的に格納することができる．トライの中でも，様々な構造が提案されている．本研究では Trie, Prefix Trie, Patricia Trie, Double-Trie を挙げる．4 つのトライのイメージを図 1 に示す．いずれのトライも，キー集合 { “brace”,“oct”, “octet”, “race”, “role”, “url” }を格納している．

Trie…図 1(A)に例を挙げる．ノードの削減の工夫をしないプレーンなトライ構造となる．トライの特徴として，共通の前半部分が併合されている．例えば，”race”と”role”の共通部分”r”が併合し同一ノードとなっている．また，”oct”が”octet”の経路上に位置している．キーノードは，ルート

†個人　Free

ノードからそのノードまで辿るとキーを得られることを表す．

Prefix Trie…あるノード以降分岐しない連続したノードを 1 つのノード(末尾文字列)としたもの．キーの後半は併合されにくいことに着目している．例えば，図 1(A)の Trie において，”brace”は他のノードと併合していないため，5 文字全てがノードとなっている．図 1(B)に示す Prefix Trie では”brace”を表すノード 1 つとなっている．

Patricia Trie…分岐しない連続するノードを併合し 1 つのノード(パトリシア)としたもの．例えば，図 1(B)の Prefix Trie では，末尾の連続するノードしか併合できていないが，図 1(C)に示す Patricia Trie では，ツリーの途中にある”oct”が 1 つのノードとなっている．

Double-Trie…Prefix Trie の分岐しない末尾文字列を，逆向きのトライに格納したもの．図 1(B)の 2 つの末尾文字列”brace”と”ace”が，図 1(D)の Double-Trie では共通の”ce”の部分が併合している．

3. 実験と議論

3.1 実験の設定

辞書サイズは，C/Migemo に同梱されている migemo-dict ファイルとほぼ同じ単語数を収録して比較した．

検索時間は，有名な文学作品 5 作品に含まれているルビ 4,439 語を入力し，正規表現の文字列を生成するのに要した時間である．

実験に用いたプログラムは，Go 言語を用いて実装した．プログラムはウェブ上で公開している*1．実行環境は一般的な Windows ノート PC である．

3.2 文字エンコーディングの比較

文字エンコーディングを UTF8 の場合と UTF16 の場合で実験の設定に従い比較した結果を，表 1 に示す．

文字エンコーディングが UTF16 の場合は，UTF8 の場合と比較し読みと単語の 2 つのトライの両方において，ノード数が半分以下になっている．また，辞書サイズ・検索時間ともに UTF16 のほうが優れている．

表 1　文字エンコーディングによる比較

文字エンコーディング	辞書サイズ [kByte]	検索時間 [ms]	トライのノード数 読み	トライのノード数 単語
UTF8	2,919	436	784,875	734,043
UTF16	2,513	259	380,448	305,153

表 2　ツリー構造による比較

データ構造	辞書サイズ [kByte]	検索時間 [ms]	トライのノード数 読み	トライのノード数 単語
Trie	2,513	259	380,448	305,153
Prefix Trie	2,513	265	199,955	215,200
Patricia Trie	2,570	260	182,887	208,975
Double-Trie	2,652	259	199,955	215,200

表 3　C/Migemo との比較

プログラム	使用メモリ [MByte]	起動時間 [ms]	検索時間 [ms]
C/Migemo	26.1	198	5053
本実装	12.1	78	5860

*1 https://github.com/oguna/gomigemo-experiments-2020

UTF16 ではノードあたりのサイズが大きくなるものの，ノードが少なく辞書サイズも小さくなっている．検索時間については，UTF16 のほうはノード数が小さいため，探索において目的のノードまでの遷移が高速となっている．

3.3 ツリー構造の比較

辞書のツリー構造を 4 つのトライで比較した結果を，表 2 に示す．

Trie と比較し，他の 3 つのトライは読み・単語のトライ両方においてノードが少なくなっている．これは分岐のない連続したノードの併合や，末尾文字列を別のトライで表す構造の効果が出ている．一方，辞書サイズや検索速度では，Trie と比較しいずれのトライも悪化している．Trie と異なり，ノードの種類により処理を変える必要があり，そのため処理にオーバーヘッドが加わったためだと考えられる．末尾文字列を併合しサイズ減少を図った Double-Trie については，併合により減少したサイズ分より，図 1(D)の破線矢印が表す別のトライへのポインタのサイズが大きかったためである．

3.4 C/Migemo との比較

比較の結果，Migemo において最適な文字エンコーディングは UTF16，ツリー構造は Trie である．この結果により C/Migemo と比較をした．異なるプログラム間での比較のため，語の入力は標準入力を用いたり，語の数を多くしたりなど，実験の設定は一部異なる．

結果を表 3 に示す．C/Migemo より使用メモリは半分以下，検索時間は約 15%大きいことが分かった．

3.5 妥当性への脅威

妥当性について，本研究で用いたプログラムに意図しない欠陥が含まれていた場合，実験結果に影響する可能性がある．実験に用いたプログラムは全て公開しており，再現可能である．

4. おわりに

本研究では，辞書のツリー構造と文字エンコードを，複数の候補で比較し最適な実装を選択することで，高速かつ省メモリな Migemo を実現した．C/Migemo と比較した結果，使用メモリは半分以下，検索時間は 15%大きくなった．

今後の課題として，より優れたツリー構造を利用することが挙げられる．例えば，Marisa-Trie[3]では Patricia Trie におけるパトリシアの文字列を再帰的にトライに格納し省メモリを実現している．実験の設定として，文学作品のルビを用いたが，ユーザによる利用状況を正確に反映しているわけではない．そこで，ユーザの利用状況を収集し，実際の利用を反映した実験題材の作成が，課題に挙げられる．

参考文献

[1] 高林 哲 et al., Migemo: 日本語のインクリメンタル検索, 情報処理学会論文誌, Vol. 43, No. 12, pp. 3698-3705, 2002 年 12 月.

[2] O. Delpratt et al., Engineering the LOUDS succinct tree representation. WEA 2006, pp. 134-145, 2006 年 10 月.

[3] 矢田晋, Prefix/Patricia Trie の入れ子による辞書圧縮, 言語処理学会第 17 回年次大会(NLP2011), pp.576-578, 2011 年 3 月.