メモリ上に日本語辞書を展開する一方式

4A−10

宮武　圭子　　瀧塚　孝志　　浅見 徹

国際電信電話株式会社 研究所

1. はじめに

文書を一括して処理するような自然言語処理用の辞書では、高速アクセスと小容量化が求められる。ワープロや機械翻訳用の辞書では、通常先頭の数文字に対するインデックスのみメモリ上に展開し、その先を、磁気ディスク上に展開する手法を採用し、高速なアクセスが要求される場合に全ての見出しをトライ構造に展開する方法を用いている。しかし、トライ構造に展開すると大容量のメモリを必要とするため、トライ構造部の圧縮が必要となる。本稿では、高速アクセスと小容量化を達成するため、見出し語の部分文字列を出現頻度によってハフマン符号化するトライ構造部の圧縮手法を提案する。本方式では、3種類のトライ構造を使用した。

2. 処理手順

まず、見出し語の標準化を行い、文字列の出現頻度をカウントしハフマン符号を得る。それを基に見出し語をハフマン符号化し、符号列の先頭の数ユニットを切り出して、ユニット切り出し型トライ構造またはテーブル型トライ構造で記憶する。ユニット切り出し型トライ構造とテーブル型トライ構造で表されなかった見出し語のハフマン符号列と該当する見出し語の情報を、フラグ型トライ構造(ユニット切り出し型トライ構造またはテーブル型トライ構造で示された先)において記憶する。

3. 符号化

3.1 標準化

まず、見出し語の標準化を行う。ワープロの辞書では、平仮名と片仮名の文字列が、機械語翻訳等の辞書では、漢字かな混じり文字列が見出し語に用いられる。標準化は、見出しに出現する文字の種類を削減したり、辞書の記載項目数を削減するために文字の変換を行う。

平仮名見出しであれば、'ゑ'や'ゐ'のような旧仮名文字を'え'や'い'のような新仮名文字にしたり、'ヴァ'を'バ'にしたりする変換を行う。また、漢字見出しであれば、更に、'々'を前の字にしたり、字体が異なる'剣'の旧字体の'劍'や、俗字の'劒'や、古字の'劔'や、俗字の略字の'剱'を'剣'にしたり、'聯'を書きかえ文字である'連'にする等の変換を行う。

3.2 文字列のハフマン符号

次に、部分文字列の出現頻度を測定し、見出し語のハフマン符号化を行う。

見出し語	出現頻度	ハフマン符号
く	6281	11・01・00
し	5859	11・11・00
る	5752	00・00・00
…		
こう	1561	11・00・11・10
ひ	1557	11・01・10・10
ね	1521	11・01・11・10
…		
しょう	1244	00・10・11・00
どう	1220	00・11・00・00
よ	1200	00・11・00・11
…		
ろう	726	11・11・10・11
…		

表1　出現頻度とハフマン符号の例

(1) 辞書の全見出し語をソートする。

(2) ソートされた見出し語に対し、前後の見出し語との差分文字列中に出現する文字、即ち、トライ構造上に現れる文字の出現頻度をカウントする。

(3) 頻度が指定値THよりも低い文字の頻度を足し合わせた値$SUM(TH)$を求める。

(4) 全文字を接頭語(符号語)とする。

(5) 符号語と文字の連鎖の出現頻度を(2)と同様にしてカウントする。

(6) $SUM(TH)$よりも大きい出現頻度を持つ符号語及び連鎖を新しい符号語として付け加える。

(7) 符号を割り当てる文字列の集合が収束するまで(6)を繰り返す。

部分文字列を出現頻度順に並べ、ハフマン符号の割り当てを行う。この際、1ビット単位ではなくmビット単位で符号化し、処理単位(ユニット)をmビットに統一することにより、高速化を行う。

本稿では$m=2$として符号化する。表1に広辞苑の見出し語の読みに対する出現頻度とハフマン符号の例を示す。

4. トライ構造への展開

辞書の高速アクセスと小容量化を行うため、ユニット切り出し型、テーブル型、フラグ型の3つのトライ構造を用いることにした。

4.1 ユニット切り出し型トライ構造

ユニット切り出し型トライ構造は、先頭の数ユニットを切り出し、テーブルのインデックスとして使用するト

A Structure of On-core Japanese Dictionaries
Keiko MIYATAKE, Takashi TAKIZUKA, Tohru ASAMI
KDD R&D Laboratories

ライ構造である。表1から、‘くし’のハフマン符号は11・01・00・11・11・00である。先頭の3ユニット、2ユニット分の符号ビットを順に切り出して、ユニット切り出し型トライ構造で表した例を図1に示す。ユニット切り出し型トライ構造は、符号列の比較がないために、アクセスが高速である。しかし、テーブルの空き領域が多いとメモリを無駄に使用する。

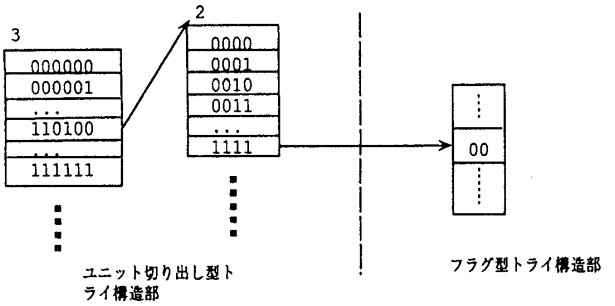


図1　ユニット切り出し型トライ構造

4.2　テーブル型トライ構造

テーブル型トライ構造は、ユニット切り出し型トライ構造から空き領域を除いたものであるが、見出し語のハフマン符号列をインデックスとして使用するのではなく、符号列として記憶する。図2にテーブル型トライ構造部の例を示す。テーブル型トライ構造は、符号列とその内容を記憶しているフラグ型トライ構造までのサイズから成る。例えば符号列0110を検索する場合は、テーブルから0110を検索し、データサイズ$L1$と$L2$を足し込むことによって、その内容を記憶しているフラグ型トライ構造のアドレスを求める。

ユニット切り出し型トライ構造でトライの空き領域が多い場合は、テーブル型トライ構造で記憶した方がメモリ容量が少なくて済むが、符号列を格納しているため、符号列の数が多い場合はユニット切り出しトライ構造よりも、メモリ容量が大きくなる。また、検索時には、符号列の比較とサイズの足し込みを行うため、テーブルサイズが大きいと検索に時間がかかる。

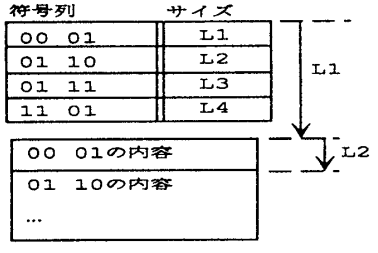


図2　テーブル型トライ構造の例

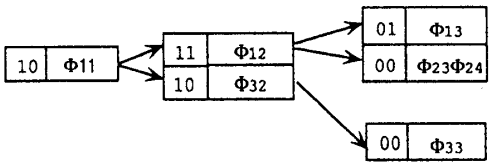
4.3　フラグ型トライ構造

フラグ型トライ構造では、部分木を用いて情報を記憶するが、部分木のデータサイズは用いずに、子の節または葉を持つか、兄弟の節または葉を持つかという2ビットの情報を記憶することにより実現する。図3に3～4ユニットの符号を記憶する場合を例として示す。フラグ型トライ構造は、最小限の容量で記憶することができる。しかし、検索時に最悪の場合、全部の部分木をたどることになるため、アクセス時間が遅くなる。

1：入力見出し語符号

Φ11Φ11Φ11　　Φijは各ユニットの符号列を表す
Φ11Φ12Φ23Φ24
Φ11Φ32Φ33

2：1を表すユニット切り出し型トライ構造



先頭の2ビットはそれぞれ、
子の節または葉を持つ（1）持たない（0）
兄弟の節または葉を持つ（1）持たない（0）
を表す

3：2と同内容のフラグ型トライ構造部

10Φ11　11Φ12　01Φ13
00Φ23Φ24
10Φ32　00Φ33

図3　フラグ型トライ構造の例

5.　トライ構造の使い分け

ユニット切り出し型トライ構造とテーブル型トライ構造部の使い分けを説明する。見出し語のハフマン符号列から数ユニットを切り出して、トライ構造にするとき、使用効率を高くするため、例えば$n \times 2^n$の90%以上の要素が存在するならば、ユニット切り出し型トライ構造を用い、そうでない場合は、テーブル型トライ構造を用いる。この際、テーブルサイズが大きくなると、検索に時間がかかるため、最大のテーブルサイズ(例えば16)を規定し、それをを越える場合はユニット切り出し型トライ構造を用いる。また、使用するデータ容量が、ユニット切り出し型トライ構造にした場合よりも大きくなる場合も、ユニット切り出し型トライ構造を用いる。

ユニット切り出し型トライ構造では、トライが大きくなるとアクセス時間も大きくなるため、1つのトライで表すユニット数の上限値(例えば16)も指定する。

フラグ型トライ構造部で記憶できる見出し語の節及び葉の数は、小さくするとユニット切り出し型トライ構造部とテーブル型トライ構造部の占有割合が大きくなり、大容量のメモリを必要とするため、上限値(例えば50程度)を設けている。

6.　おわりに

標準化された文字列を出現頻度によって符号化し、その符号のビットパターンを切り出してビット切り出し型トライ構造、テーブル型トライ構造及びフラグ型トライ構造として記憶することにより辞書の見出し語の出現分布を均一化し、トライ構造部を圧縮する、日本語辞書のメモリ上への展開方式について説明した。今後は本方式の効果の検証と3つのトライ構造の使い分けの最適化を検討を行う。

最後に日頃御指導頂くKDD研究所小野所長、浦野次長に感謝する。