ソフトウェア科学会 論理と自然言語研究会
「自然言語処理用辞書の諸相」ワークショップ 資料

電子国語辞典「電明解」の構成と実現

１９８８年３月２２日

東京工業大学工学部情報工学科
今津 英世　田中 穂積

目次

第１章　序論

１．１　研究の背景

国語辞典・英和辞典などの辞書を引く作業はデータの検索に他ならない。また、辞書は自然言語のデータベースともいえる。データの蓄積・検索は計算機の得意とするところであるから辞書は計算機上に実現されていることが望ましい。計算機上に実現されていれば紙の辞書では不可能だった様な検索が可能にもなるだろう。また、辞書の電子化は辞書編纂作業の電子化の側面もある。

新明解国語辞典・コンサイス英和辞典・講談社和英辞典・ロングマン英英辞典・広辞苑などのテキスト-データが計算機可読な状態で存在していること、記憶装置が大容量・安価・高速になってきていること、日本語を扱う計算機環境が整備されてきたことなど辞書を計算機上に実現する環境は整いつつあるといえる。

１．２　研究の目的

本研究の目的は人間が引いて使う辞書の計算機上での効率的・効果的実現である。研究は直接的には株式会社三省堂の新明解国語辞典を計算機上で実現することを目的として行なったが、本研究の諸成果は容易に他の国語辞典や別の種類の辞典（英和辞典・和英辞典など）にも応用可能である。新明解国語辞典のテキスト-データは電子技術総合研究所推論機構研究室と三省堂との共同作業により計算機可読なものが作成されている[2]。

辞書データを用いた研究としては[3][4][5]がある。[3]は新明解国語辞典・コンサイス英和辞典・ロングマン現代英英辞典の辞書データを扱った論文の集合である。この中にはコンサイス英和辞典の計算機上の実現に関する論文がある。[4]はロングマン現代英英辞典を関係データベースに蓄積する研究である。また、[5]は新明解国語辞典の語釈文に関する研究である。[3][4][5]の中には国語辞典の電子化に関するものはない。

１．３　研究の経過

現在、電子化新明解国語辞典「電明解」がサン-ワークステーション（ＯＳはUNIX 4.2 bsd 相当）上に実現されており、研究室内で使用されている。辞書データは変更が容易なファイル構造で格納されており、また、変更の為のインターフェースも用意している。既に研究の過程で数十の誤りを発見・修正している。辞書デー

タの追加・削除は行なっていない。

現在可能な辞書引きは「いずれ」・「セレナーデ」といった見出しからの辞書引きのみである。入力された仮名綴りに対して最長一致検索を行なう。辞書引きの例を以下に示す。

(新明解) いずれ <改行>
【いずれ】 00イヅレ
1 [《何れ》(代) 「どれ・何・どこ・どちら」の意の雅語的表現。
「―か一つを選ぶ・―〔＝どちらも〕劣らぬ・真偽―にもせよ〔＝本当であっても、無くても〕・―アヤメかカキツバタ〔＝二つの物がよく似ていて、区別しにくいたとえ〕」
2 (副) 〔どういう経過をたどるにしても、の意〕
1)おそい・速いの違いは有るが、同じ結果になることを表わす。どの道。
「幾ら隠したって―〔＝どうせ、いつかは〕分かることだ」
2)余り遠くない将来において。
「詳しい事は―〔＝近いうち〕お目にかかって申し上げます・―〔＝間もなく〕雨もあがろう」
〔1は、「＜孰れ」とも書く〕
[―にせよ]
1)二つの場合のどちら・を選ぶ(になる)にしても。
2)どうなるとしても。
[―も様 〈さま〉 03]「おとくい・(お近づき)の皆様」の意の老人語。

辞書データは構造化されており、この様に語釈の論理構造を反映したインデンテーションを付けた表示がなされる。最長一致検索をしている為、「せれなーど」と入力しても「セレナーデ」が表示される。

(新明解) せれなーど <改行>
【セレナーデ】 03 〔ド Ｓｅｒｅｎａｄｅ＝愛人の窓の下で聞かせるための、甘美な歌曲〕(小) 夜曲。〔セレナード03は、フランス語形〕

検索対象には「*」を1つ含めることができる。例えば「*すい」と入力すると以下の様に表示される。

(新明解) *すい <改行>

【あき　やす・い】　04［飽き(易い］（形）どんな事も続けてすることが出来ない性質だ。

1/197

これは「すい」で終わる語は 197 あってその１つ目を表示していることを表わしている。ここで <改行> を入力すると

【い　かすい】　02キ―［胃下垂］　胃が異常にたれさがって、重苦しさを感じる病気。

2/197

と表示される。「あい*」と入力した時は「あい」で始まる語が、「あ*ん」と入力した時は「あ」で始まって「ん」で終わる語が表示される。

辞書引きはトライ構造のインデックスにより行なわれ、辞書引きに要するＣＰＵタイムは「*」を含まない場合は数十ミリ秒以内、「*」を含む場合は最大で１秒程度である。

もともとの辞書データは、文字の羅列で構造は持っていない。そのデータをプログラムによりデータ全体に渡って構造化した。ロングマン英英辞典のデータはもともと構造化されており、[3]の中の幾つかの研究と[4]はそれをどう加工するかという研究である。これに対して本研究では構造化されていないテキスト-データを構造化している。テキスト-データの構造化は[3]にある研究でもコンサイス英和辞典に対してなされているが、データに誤りが多いなどの理由で全体に渡って完成してはいない。

プログラムは全てＣ言語で記述している。研究の過程でのコーディング量は 5000 行程度である。

実際に製品として売られている規模の大きな電子辞書としては[6]がある。これは本研究の対象としているものとかなり近いものではあるが、[6]の電子辞書は読み取り専用で更新・追加はないし、またデータの構造化もなされてにない。

１．４　本論文の構成

本論文の構成は次の通りである。まず、２章で辞書データそのものについて、その大きさやデータ化のなされ方などを述べる。次に３・４・５章で電子化新明解国語辞典に関する考察を述べる。３章ではファイル構造、４章ではの辞書引き、５章では辞書データの構造化について考察する。６章で現在実現されている辞書コマンドの使用法の概要を述べ、そして７章が結論である。

尚、以下の文中ではオリジナルの（紙の）新明解国語辞典を「紙の新明解」、新明解国語辞典のテキスト-データを「新明解のデータ」と呼び、どちらと解釈しても問題がない場合は単に「新明解」と呼ぶ。また、本論文中では複数の単語から成る外来語の単語の区切りに「=」（二重ハイフン）を用いる。

第２章　辞書データの諸元

この章では、実現する電子国語辞典のもととなる新明解国語辞典のテキスト-データに自体について、その構成・データ量・データ化のなされ方などについて述べる。

２．１　辞書データの構成

紙の新明解には所々に枠で囲まれた部分があり、そこで「亜」や「哀」といった漢語の造語成分の説明がされている。新明解のデータはこの造語成分のデータと通常の本文のデータの２つのファイルから成っている。本文データはエントリー数約5万8000、１文字すべて２バイトでエントリーの終端を表わすのに１文字使う様な形式（シーケンシャル-ファイル）で格納すると約7.1Mバイト（約370万字）の大きさである。造語成分データは約250k バイトである。本研究で扱っているのは現在のところ本文だけで造語成分は扱っていない。

次に新明解国語辞典の辞書エントリーに入っている情報の種類とそのデータ化のなされ方を述べる。以下、説明の為に新明解のデータからの引用を幾つか示す。

> @いずれ@00イヅレ　1[《何れ］（代）　「どれ・何・どこ・どちら」[1]の意の雅語[*2]的[3]表現。「一か一つを選[7]ぶ・一〔＝どちらも〕劣らぬ・真偽一にもせよ〔＝本当で[8]あっても、無くても〕・一アヤメかカキツバタ〔＝二つの物がよく似ていて、区別しにくいたとえ〕」＼2（副）　〔どういう経過をたどるにしても、の意〕1)おそい・速いの[*9]違いは有るが、同じ結果になることを表わす。どの道。[*a]「幾ら隠したって—〔＝どうせ、いつかは〕分かることだ」＼2)余り遠くない将来において。「詳しい事は—〔＝近いうち〕お目にかかって申し上げます・—〔＝間もなく〕雨もあがろう」〔1は、「＜孰れ」とも書く〕［一にせよ］1)二つの場合のどちら・を選[5]ぶ（にな[b]る）にしても。2)どうなる[c]としても[*d]。［一も様＜さま＞03］「おとくい・（お近づき）の皆様」の意の老人語。[c]
>
> @いち　のう@0002［一能］　一つの技能・才能。[*e]=>一芸

@うり　つなぎ@00［売（り）＜繋（ぎ）］1)〔株式の取引で〕相場の下落を予想して、市場へ売りに出しておくこと。<>買い繋ぎ＼2)家財などを売って、生活を繋ぐこと。（動）$_{g}$売繋ぐ04（^{*f}他五）

@うり　わた・す@04［売（り）渡す］（他五）　売って先方に渡す。<>買い受ける（名）$_{g}$売渡し00

@おぞまし・い@04（形）　いやな感じがする様子だ。うとましい。¢$_{h}$—さ¢$_{h}$03　¢$_{h}$—げ¢$_{h}$04

@おちゃっぴい@04—$_{i}$な・—$_{i}$に　おしゃべりで、ちゃめ・であること（な少女）。

@おど　おど@01（副）—$_{i}$と・—$_{i}$する　こわがったり、自信が無かったりして、落ち着かない様子。「—（と）した態度」

@おぼし・い@03［《思しい］（形）　（…と）思われる・（見える）様子だ。「犯人$_{*j}$≦と—$_{*j}$≧男」　〔「覚しい」とも書く〕

以下、上に引用したものの各部分についての説明を述べる。これは[2]を参照しつつ筆者の観察の結果を加えたものである。

(1) 見出しは「@」又は「＊」又は「＃」で囲まれている。「＊」で囲まれているのは重要語、「＃」で囲まれているのは最重要語である。
(2) アクセント番号。この場合は全体でアクセントが共通しているが、大分類ごとに違うことがあるし、また、小分類ごとに違うこともある。
(3) 旧仮名遣い。「あおくさい」の旧仮名遣いが「アヲ—」となっている様に現代仮名遣いと同じ部分は省略される。
(4) 語釈大分類。
(5) 最初の部分に［］で囲んで正書法が示してある。最後に〔〕で囲った部分があり、そこにその他の表記についての記述があることがある。正書法が大分類ごと・小分類ごとに違うことがあるのはアクセントと同様。
(6) 常用漢字音訓表にない読みを表わす。
(7) 品詞。活用のある語はその活用型も記してある。
(8) 用例。

(9) 改行。紙の辞書で行の途中で改行されているところはデータ中にもその旨の情報が入っている。改行の位置に、はっきりとした規則性はない。
(a) 語釈小分類。
(b) 常用漢字以外の漢字を表わす。
(c) 派生語。派生語の見出しには正書法・アクセントが記されている。
(d) 派生語にも語釈には大分類・小分類がある。
(e) 参照先。「=>置く１４」の様に大分類・小分類番号が付されていることがある。
(f) 対義語。大分類・小分類番号が付されていることがあるのは参照先と同様。
(g) 対応する別品詞の語。名詞に対する動詞、動詞に対する名詞、自動詞に対する他動詞、他動詞に対する自動詞、の４種類がある。
(h) 形容詞・名詞に現れる。「－さ」・「－げ」・「－がる」という形が使われるかどうかを示す。用例が示されることもある。
(i) 名詞・副詞のサ変動詞・形容動詞語幹としての用法。
(j) 用例の中で結合が局限されているものは紙の新明解では太字で示してある。データでは太字の部分が特別なコードで囲んである。
(*) 2, 4, 9, a, e, f, jは JISコードにない特殊なコードを使っている。上に掲げた引用の中では適宜JISにあるコードを使っている。

２．２　紙の新明解との違い

新明解のデータはこの様に紙の新明解をかなり忠実にデータ化したものとなっているが、紙の新明解には記されていてデータでは抜けている情報もいくらかはあるのでそれについて列挙する。

(1) エ列長音・オ列長音などの表記に関して現代仮名遣いには発音との食い違いがある。こういうものに関して紙の新明解では

かくりょう（オ）　0200－レウ［閣僚］……

といった具合に発音が記されていたがデータには発音の情報は入っていない。

(2) 紙の新明解には概ね明朝体が使われていて、正書法中の学習漢字は教科書体で記されているが、データにはこの区別はない。

(3) 参照先や対語を示す時などに、参照先の語や対語の正書法に加えて、その

語釈の中の大分類・小分類も記されていることがある。また、表記に関する説明に大分類番号・小分類番号が現れることもある。紙の新明解ではこういう所に記されている大分類番号・小分類番号は語釈の大分類・小分類を示す為に記されている大分類番号・小分類番号とは大きさが違うのであるが、データでは区別がない。例えば下に示す様な場合、

@ア@　（略）1)アジア。「中央—03」2)アフリカ。「南〈ナン〉—01」3)（日本）アルプス(a)。「南〈ナン〉—連峰(b)」〔1)は「亜(c)」、2)は「<阿(d)」とも書く〕

(a)と(c)は、紙の新明解では文字の大きさが違うのであるが、データでは同じコードが使われている。(b)と(d)についても同様である。このことは5章で述べる構造化の際に問題となった。

第３章　ファイル構造

３．１　ファイル構造の条件

電子辞書のデータを格納するファイル構造には以下のことが要求される。

(1) レコード長可変
辞書エントリーは最小のものと最大のもとではデータ量に100倍以上の差がある。

(2) 直接アクセス可能
インデックスを用いた検索を検索の為にはレコードの直接アクセスが可能でなければならない。

(3) 順次アクセス可能
辞書を最初から順に走査することもあろうし、また、ある辞書エントリーの前後を見たいこともある。

(4) レコード内容の修正が容易
新明解のデータには紙の新明解との食い違い（つまりデータ化時の誤り）が見うけられる。また、小分類番号が１つしかないエントリーに小分類番号が記されているといった論理的誤りが紙の新明解にあり、それがデータに踏襲されている例もあった。

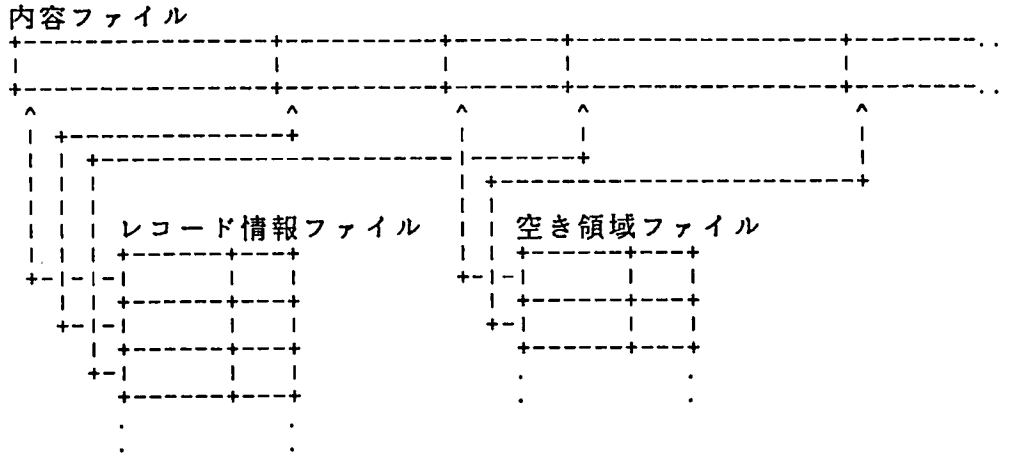
尚、現段階ではレコード（辞書エントリー）の修正は行なうが、追加・削除は行なわない。

べた詰めのシーケンシャル-ファイルはレコード長が可変で、順次アクセス可能であるが、データの大きさが７Ｍバイト以上となるとデータの修正が非常に困難である。「何バイト目からのレコード」という形でレコードを identify すれば直接アクセスも可能である。しかし、これでは修正によるデータ量の増減に対処できない。

これらを考慮の結果 現在 用いているファイル形式について以下に述べる。

３．２　ファイル構造の概要

まず、ファイル構造の概念図を以下に示す。



このファイル構造は「内容ファイル」・「レコード情報ファイル」・「空き領域ファイル」の3つのファイルを使う（参考までに言うと UNIX ではファイルの形式はバイト列のみで、システムコールのレベルで、任意の位置から任意のバイト数の読み込み／書き込みができる）。

まず、各エントリーの内容は順に内容ファイルに入れる。このとき各エントリーが128の倍数バイトの領域を占める様に適宜パディングする。各エントリーが占める領域をあるバイト数の整数倍に切り上げるのは修正の結果できる空き領域の管理の手間を考えてのことである。レコードの占める領域の単位を32・64・128・256とした時の内容ファイルの大きさは下の通りである。現在、辞書データは構造化されているが、下記の数字は構造化されていない状態でのデータである。

レコードの占める領域の単位（バイト）	内容ファイルの大きさ（Mバイト）
32	8
64	9
128	10
256	16

この数字を鑑みて占める領域の単位を128バイトとした。

レコード情報ファイルに各エントリーの内容の開始位置とバイト数を入れる。このバイト数はその内容が占めているバイト数（128の倍数）ではなく、実際のバイト数レコード情報ファイルのレコードをC言語の構造体で表わすと以下の様になる（実際にこの型の変数に読み込み、この型の変数から書き込んでいる）。

```
struct zrecinfo {
        long pos;               内容ファイルの開始位置
        unsigned short size;    内容のバイト数
};
```

各辞書レコードは0から始まるレコード番号で identify する。エントリーの記録順序は紙の新明解と同じである。

レコード情報ファイルでは各レコードの開始位置とバイト数しか関知していない。レコードの内容は現在は構造化されているが、構造化される前（辞書テキストがベタ詰めになっていた）も全く同じファイル構造を用いていた。現在の構造化は必ずしも十分なものではなく、今後、更に詳細かつ効率の良い構造にしなければならないが、その時もこのファイル構造はそのまま使える。

3．3　アクセス方法

レコード番号iのレコードを読み込む手順は以下の通りである。レコード情報ファイルの各レコードの大きさはRバイトであるとする。レコード番号iの辞書レコードのレコード情報はレコード情報ファイルの先頭からR×iバイト目から入っているから、まず、そこからRバイト読み込み、その情報をもとに内容ファイルから読み込むのである。この様子をC言語のプログラムで記述したものを以下に示す[7]。

```
char    buf[16384];         内容を読み込むバッファー
int     i;                  レコード番号
struct zrecinfo recinfo;    レコード情報
FILE    *recinfof,          レコード情報ファイル
        *contf;             内容ファイル

fseek(recinfof, sizeof (struct zrecinfo) * i, 0);
  レコード情報ファイルの該当部分へシークする
fread((char *)&recinfo, sizeof recinfo, 1, recinfof);
  レコード情報を読み込む
fseek(contf, recinfo.pos, 0);
  内容ファイルの該当部分へシークする
fread(buf, 1, recinfo.size, contf);
  内容を読み込む
```

辞書を順に読み込むにはレコード情報ファイルを先頭から順に読み込んで、その情報をもとに内容ファイルを読み込めばよい。順次アクセス、レコード番号による直接アクセス、ともに容易にできる。

3．4　データ量変化を伴うレコードの更新

レコード内容修正の結果バイト数に増減があった場合の対処は以下の通りである。

まず、バイト数が減った時。レコード内容が占める領域は128の倍数に切り上げるから、レコード内容が占めている領域のうち常に何バイトかは空いているのであるが、修正の結果、空きが128 バイトを超えた場合は空きの部分のうち128の倍数バイト分を空き領域ファイルに登録する。

次に増えた時。多くの場合はもとの場所に収まるが、収まらなくなることもある。そういった場合は、まず空き領域ファイルを見て修正結果を収められる空き領域を探す。適当なものがみつかればそこを使う。なければ内容ファイルの最後に新たに領域を確保して使う。修正前にレコードが収められていた領域は空き領域として登録する。

3．5　レコード数増減への対応

このファイル構造では辞書レコードの削除はその辞書レコードに対するレコード情報をレコード情報ファイルから削除することであり、また、辞書レコードの追加は、レコード情報ファイルへのデータの挿入を伴う。従って辞書レコードの追加・削除が行なわれるとレコード番号が変わってしまう。インデックスなどにはレコード番号をレコードのidentifierとして記録するが、レコード番号が変わるとそのインデックスは使えなくなる。レコードを削除した時はレコード情報ファイルの該当部分に削除された旨を書き込むことにすれば削除に関してはレコード番号の変更を伴わなくすることができるが、追加に関してはこの様なことはできない。

レコード情報ファイルのレコードを以下に示す構造体の様に順序を示すリンクを持つものに変更すれば辞書レコードの追加・削除が可能となる。

```
struct zrecinfoa {
        long prev, next;        直前・直後の辞書エントリーへのリンク
        long pos;
        unsigned short size;
};
```

先に述べたファイル構造では辞書の順次アクセスはレコード情報ファイルを「順に」読むことにより行なっている。そこで順に読むのではなく、リンクを辿ることにする訳である。

現段階では辞書エントリーの追加・削除は行なわないからこのファイル構造は用いていない。

第４章　見出しによる辞書引き

国語辞書の使い方で最も基本的で、かつ頻繁に行なわれるのは「いやさか」・「ともがら」といった見出しから語釈を引くことである。こういった辞書引きを順次検索により行なうことは辞書のデータ量を考えると現実的ではない。見出しの文字列とレコード番号を対応付けるインデックスを作っておいてそれを用いて検索することになる。インデックスの編成法としてはハッシュを使ったものとトライを使ったものを試みた。後で述べる様にハッシュによる辞書引きは高速ではあるが機能的に不十分な点があり、現在はトライによる辞書引きのみを行なっている。尚、辞書引きでは平仮名・片仮名は区別しない。

４．１　ハッシュを用いた辞書引き

４．１．１　実現

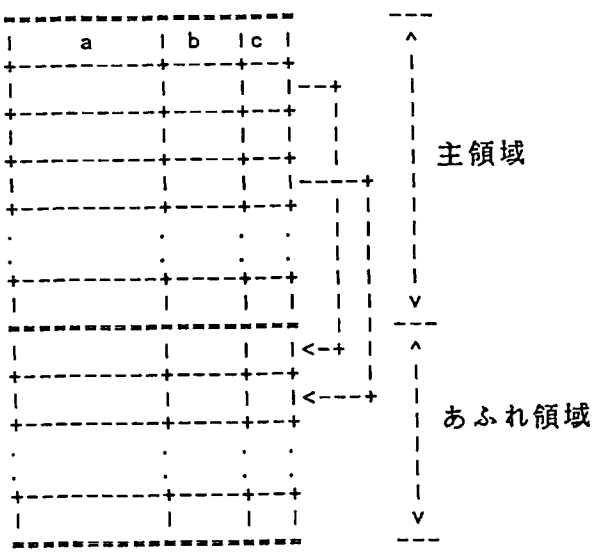
現段階では辞書エントリーの増減は考えていないし、また、それを行なったとしてもデータ全体の大きさに比べればその変化は僅かである。こういった状況だと必要とする記憶容量と検索スピードの点ではハッシュを使ったインデックスが優れている。

ハッシュ関数としては以下のものを用いた。

```
len ← 見出しの長さ
ハッシュ値 ← 0
i を len から 1 まで 1 ずつ減らしながら繰り返す
    ハッシュ値 ← (hashval * 128 + 見出しの i 文字目の文字コード)
                 mod ハッシュ表の主領域の大きさ
```

+ 辞書引きのときは平仮名・片仮名は区別しない。
+ 見出しは仮名のみで構成されているから、見出しの文字コードは 0 から 127 の範囲に収まる。

実現したハッシュ表の概念図を下に示す。

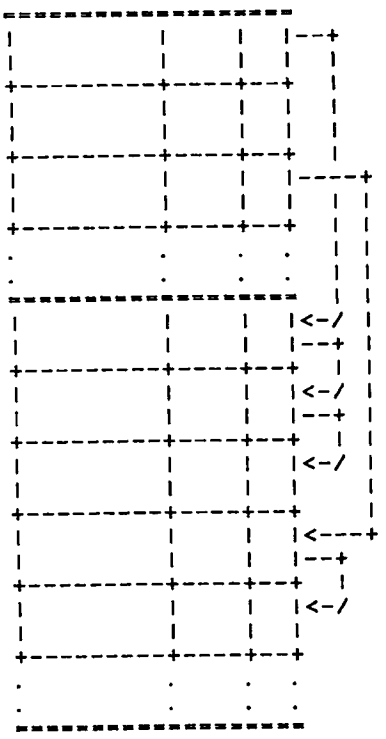


(a) 見出しの文字列。

(b) 辞書エントリーのレコード番号

(c) 同一ハッシュ値の別エントリーへのリンク

ハッシュ値の衝突にはあふれ領域で対応する。ハッシュ表の主領域の大きさは49139エントリー、あふれ領域は16384エントリーとした。同一の見出し文字列を持つエントリーは一般には複数あるがハッシュ表に登録するのはその中で最初に現れるもののみとする。人間が引いて使う辞書の為にはそれで十分だからである。インデックスを付ける対象（辞書データ）が予め固定されているので一旦ハッシュ表を作った後、あふれ領域を整理して同一ハッシュ値に対するエントリーを1箇所にまとめる。その様子を以下に図示する。



ハッシュ表を収めたファイルの大きさは約１Ｍバイトである。辞書データを収めるのに使用しているファイルが10Ｍバイトほどであることを考えるとインデックスの大きさは決して無視できる大きさではない。

４．１．２　考察

ハッシュ関数は多対１の関数であるからハッシュ表にはキーの値を記録しておくことになる。ハッシュ表を引く時はハッシュ関数をもとにハッシュ表のエントリーを読み込んだ後、それが求めるものなのかどうか、引こうとしているキーと読み込んだキーとを比較することになる。先に述べた様にハッシュ表には同一の見出し文字列を持つ辞書エントリーのうちの最初のもののレコード番号を記録している。新明解には異なる見出し文字列は 43995 種類あるが、そのれらが何回のキー比較で対応するレコード番号を知ることができるかを以下の表に示す。

比較回数	その比較回数でレコード番号が分かる見出し文字列が何種類あるか	見出し文字列の種類数の全体に対する割合（%）
1	28486	64.8
2	11150	25.3
3	3344	7.6
4	809	1.8
5	172	0.4
6	27	0.1
7	7	0.0
合計	43995	

比較回数が最大でも７ということは、ある同一ハッシュ値を持つ見出し文字列は最大で７種類ということである。あふれ領域の中では同一ハッシュ値に対するエントリーはまとめてある。そのまとまりは一番大きなものでも６であり、ディスクの入出力は８Ｋバイトごとに行なっているため、１つのまとまりは必ず１回のディスク- アクセスで読み込める。従って見出し文字列から対応するレコード番号を知るのに最大２回のディスク-アクセスで済む。

ハッシュによる検索の利点はその高速性にある。一方ハッシュによる検索には問題もある。見出し文字列をキーにハッシュを作ったとすると、ある語を引くには見出しと全く同じものを与えないと引けないことことである。新明解では例えば「双眼鏡」は「双眼」の派生語として登録されており、「そうがん」という見出しはあっても「そうがんきょう」という見出しはない。今述べたハッシュのインデックスを使って辞書引きをしている限り「そうがんきょう」を引こうとしても、そういう見出しがないこと以外の情報は得られない。また、逆に新明解には「躁鬱病」はあっても「躁鬱」はない為、「そううつ」を引こうとしても引けない。「セレナード」は「セレナーデ（ドイツ語）」のフランス語形であるが、新明解には「セレナーデ」という見出しはあっても「セレナード」はない為、「セレナード」を引こうとしても何も出力されない。

見出しから辞書を引くに時、与えられた見出し文字列がハッシュ表になかった場合、その文字列から最後の１文字を除いた文字列をハッシュ表に探し、それでもなければ更に１文字除いたものについて探し、ということを繰り返すことで、「そうがんきょう」については「そうがん」を引くことができるが、「そううつ」・「セレナード」に関してはだめである。紙の辞書を使っている時はこういった問題は起きないのであるから、このままでは十分ではない。この問題は次に述べるトライを使った辞書引きにより解決されている。

４．２　トライを用いた辞書引き

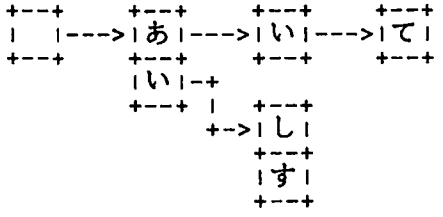
4．2．1　実現

辞書の見出しの向きのインデックス編成法としてはトライ（trie）構造によるものがある[8]。ハッシュを使ったインデックスでは、同一の見出し文字列を持つエントリーが複数ある時、最初のものについてだけレコード番号を記録したが、トライを使ったインデックスではそれぞれの見出し文字列についてその見出しを持つ辞書エントリーを全て記録した。これは、ある文字列で見出しが始まる／終わるエントリーを全て探すことを行なう為である。

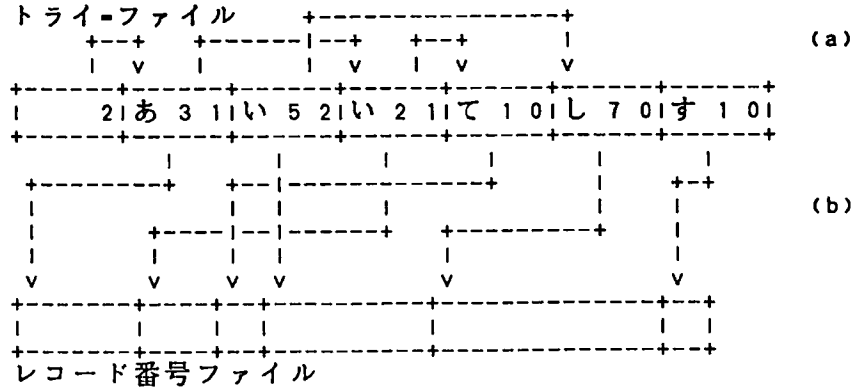
例として以下の様な非常に小さな辞書について考える。

見出し	同じ見出しを持つ語の数
あ	3
あい	2
あいて	1
いい	5
いし	7
いす	1

この辞書に対するトライは以下の様になる。一番左の文字が入っていない四角はトライの根を示している。



トライを用いたインデックスの情報はトライ-ファイルとレコード番号ファイルの２つのファイルに格納する。先に例として挙げた小さな辞書のトライによるインデックスが格納される様子の概念図を以下に示す。



トライ-ファイルは固定長のレコードの集まりであり、そのレコードは以下の構造体で表わされる。トライの各ノードはトライ-レコードの集まりで構成される。

```
struct ztrierec {
        char val;
```

そのトライ-レコードが担当している文字。

```
        long recnos;
```

そのトライ-レコードが表わす文字列を見出しとする辞書レコードのレコード番号の集合がレコード番号ファイルのどこから格納されているか。

先に示したトライ情報格納の概念図の(b)の矢印に相当する。

```
        unsigned short nrecnos;
```

そのトライ-レコードが表わす文字列を見出しとする辞書レコードは幾つあるか。

```
        long children;
```

先に示した概念図の(a)の矢印に相当する。

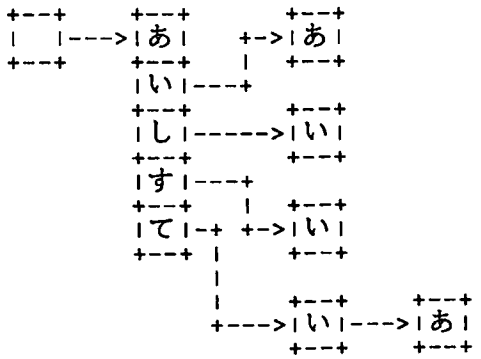
```
        unsigned char nchildren;
```

(a)の矢印ので指された位置から始まるノードが幾つのレコードから構成されているか。

```
};
```

各ノードの中ではレコードは担当している文字（上の構造体ではval）の昇順に並べられており、トライを探索する時はノードを構成するレコードを一旦 全て読み込んで、２分検索する。トライ-ファイルの最初のレコードはトライの根である。

これまで示してきたのは見出しの文字列を前から見て作ったトライで、いわば「順方向トライ」とでもいうべきものであった。見出し文字列を後ろから見て作る「逆方向トライ」というものも考えられる。先に例として挙げた小さな辞書の逆方向トライは以下の様になる。



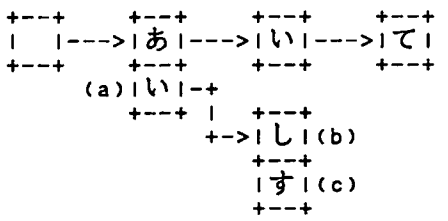
トライによるインデックスは順方向トライによるものと逆方向トライによるものの両方を作成した。

現在のトライの実現では辞書レコードの追加・削除に対応できないが、辞書ファイルで用いている手法を用いれば対応することも可能である。

4．2．2　辞書引き手順

トライによる辞書引きは最長一致により行なう。その手順を以下に示す。

(1) 検索対象文字列に従って順方向トライを辿れるところまで辿る。



トライが上の様になっていたとして検索対象として「いた」が与えられると(a)の所まで到達できる。

(2) 到達した位置から先（その位置を含む）にあるトライ-レコードに登録されているレコード番号のうち最も小さいものを検索結果とする。上のトライでいうと「(a)から先にあるトライ-レコード」とは(a)・(b)・(c)である。

「せれなーど」が検索対象として与えられたとすると以下の様になる。ハッシュのところで述べた通り新明解には「セレナーデ」という見出しはあるが、「セレナード」という見出しはない。「せ」、「れ」、「な」、「ー」までは順に辿って来れるが次に「ど」へ行こうとして失敗する。「ー」から先にあるトライ-レコードに登録されているレコード番号の最小のものは「セレナーデ」に対するものである。従って「せれなーど」により「セレナード」が検索できる。「そうがんきょう」・「そううつ」についても同様に、「双眼」・「躁鬱病」が検索結果となる。ハッシュ表による辞書引きでの問題がトライでは極自然に解決されている。

最長一致検索の他、ある文字列で見出しが始まる／終わる語の検索がトライを用いて行なえる。ある文字列で見出しが始まる語の検索の手順は以下の通りである。順方向トライのある位置まで辿る。次に、そこを含めてそこから先の全てのトライ- レコードに登録されている全てのレコード番号の集合を作るのである。そのレコード番号の集合は、その位置が表わす文字列で見出しが始まる辞書エントリーの集合となっている。逆方向トライで同様の集合を作ると、その位置が表わす文字列で見出しが終わる辞書エントリーの集合となる。また、ある文字列Ａで始まる語の集合とある文字列Ｂで終わる語の集合Ｂの積をとればＡで始まりＢ

で終わる語の集合が得られる。

4．2．3　性能評価

現在の実現環境ではＣＰＵタイム計測の精度は1/60秒である。検索に要するＣＰＵタイムをシステム=コールにより測定した結果は以下の通りである。

最長一致検索	0～50ミリ秒
「…」で始まる語の検索	100～900ミリ秒
「…」で始まり、「…」で終わる語の検索	300～1300ミリ秒

トライによる検索は先に述べたハッシュ表による検索より時間がかかるのであるが、この様に実用上十分なスピードが得られている。

トライ=ファイルの大きさは順方向トライで約900Ｋバイト、逆方向トライで約790Ｋバイトで、レコード番号ファイルは順方向・逆方向同じ大きさで、約230Ｋバイトである。インデックスの為に使われる記憶は合計で約２Ｍバイトである。

4．3　今後の課題

最長一致による検索は「セレナーデ」を「せれなーど」から引く時の様に入力した仮名綴りの終わりの方に不一致がある場合は有効に働くが、「フィルム」（「ィ」は小さい「イ」）を「ふいるむ」（「い」が小さくない）で引こうとした場合の様に仮名綴りの始めの方に不一致がある場合は良い結果が得られない（「ふいるむ」を引くと「不意」が表示される）。

小さい文字・大きい文字、「づ」と「ず」、オ列長音の「お」と「う」といった入力誤り／表記の揺れに対応する為のヒューリスティックを辞書引きに導入することは今後の課題である。

第５章　辞書データの構造化

もともとの辞書データは２章に示した様に文字の羅列であるが、各辞書エントリーは明らかに論理構造を持っている。電子国語辞典の高度な利用の為には辞書データが構造化されていることが必要である。本章では、まず辞書の観察の結果得られた辞書エントリーの論理構造を示し、次に実際の構造化作業に就いて述べる。

５．１　辞書データの論理構造

以下 辞書の観察の結果 得られた辞書エントリーの一般的論理構造を述べる。説明に当たっては理解の便宜を考慮して、まず詳細を省いた基本的論理構造を示し、その上で詳細な論理構造を例と共に示す。

辞書エントリーの基本的論理構造は以下の通りである。

辞書エントリー ::= 大分類+ 派生語* 。
大分類 ::= 小分類+ 派生語* 。
派生語 ::= 派生語大分類+ 。
派生語大分類 ::= 派生語小分類+ 。

「?」はその構造要素が０個又は１個そこに来ることを、「*」は０個以上任意個来ることを、「+」は１個以上来ることを表わす。

辞書エントリーの詳細な論理構造は以下の通りである。

辞書エントリー
::= 見出し 重要度 アクセント* 旧仮名遣い? 書法? 接尾情報?
品詞? 全体的説明? 対義語* 参照先* 大分類+ 派生品詞*
派生語* 。
大分類 ::= アクセント? 旧仮名遣い? 書法? 接尾情報? 品詞?
大分類の説明? 対義語* 参照先* 小分類+ 派生品詞* 派生語* 。
小分類 ::= 書法? アクセント* 接尾情報* 説明? 用例* 対義語* 参照先* 。
派生語 ::= 書法 アクセント* 旧仮名遣い? 接尾情報? 品詞? 派生語の説明?
対義語* 参照先* 派生語大分類+ 派生品詞*。
派生語大分類 ::= 接尾情報? 品詞? 派生大分類の説明? 対義語* 参照先*
小分類+ 派生品詞* 。
派生語小分類 ::= 接尾情報? 説明? 用例* 対義語* 参照先* 。

派生品詞 ::= 書法? アクセント+ 用例* 。

<接尾情報>・<対義語>・<参照先>は<小分類>・<派生語小分類>の構成要素とされている他、<辞書エントリー>・<大分類>・<派生語>・<派生語大分類>の構成要素ともされている。<接尾情報>・<対義語>・<参照先>は一般には<小分類>・< 派生語小分類>に属するのであるが、（派生語）大分類中の各（派生語）小分類について共通しているときは<（派生語）大分類>のレベルに書かれ、各（派生語）大分類に共通しているときは<辞書エントリー>（<派生語>）のレベルに書かれる為、この様な構造とした。同様のことは<アクセント>・<旧仮名遣い>・<書法>・<接尾情報>・<品詞>についても言える。

これに従って２章で説明の為に引用した「いずれ」の論理構造を示す。

見出し： いずれ
重要度： 0
アクセント： 00
旧仮名遣い： イヅレ
大分類１：
書法： 《何れ〔「＜孰れ」とも書く〕
品詞： 代名詞
小分類１：
「どれ・何・どこ・どちら」の意の雅語的表現。
用例：
―か一つを選ぶ
―〔＝どちらも〕劣らぬ
真偽―にもせよ〔＝本当であっても、無くても〕
―アヤメかカキツバタ〔＝二つの物がよく似ていて、区別しにくいたとえ〕
大分類２：
品詞： 副詞
大分類の説明： 〔どういう経過をたどるにしても、の意〕
小分類１：
説明：
おそい・速いの違いは有るが、同じ結果になることを表わす。どの道。
用例：
幾ら隠したって―〔＝どうせ、いつかは〕分かることだ
小分類２：
説明：
余り遠くない将来において。

用例：

詳しい事は—〔＝近いうち〕お目にかかって申し上げます

—〔＝間もなく〕雨もあがろう

派生語１：

書法：　—にせよ

大分類１：

小分類１：

説明：　二つの場合のどちら・を選ぶ（になる）にしても。

小分類２：

説明：　どうなるとしても。

派生語２：

書法：　—も様〈さま〉

アクセント：　03

大分類１：

小分類１：

説明：　「おとくい・（お近づき）の皆様」の意の老人語。

この論理構造にデータを編成するには元の字面を区切るだけでなく、例えば「1)は「…」とも書く」といった字面を解釈してそれを書法の情報としなければならないし、また、この構造自体かなり複雑である。人間の手で構造化作業を行なうのであれば、データを直ちにこの構造に編成することも可能であるが、プログラムにより行なうことは容易ではない。そこで現段階の構造化作業では字面を区切ることによりできる範囲の構造化を行なうこととした。

５．２　現段階で用いる論理構造

現段階の構造化で用いる論理構造は以下の通りである。

辞書エントリー ::=
　　見出し 重要度 全体的説明? 大分類+ 派生品詞? 全体的捕捉?
　　派生語* 。
大分類 ::= 大分類の説明? 小分類+ 派生品詞? 大分類の捕捉? 派生語* 。
派生語 ::= 派生語見出し 派生語の説明? 派生語大分類+
　　派生語の捕捉? 。
派生語大分類 ::=
　　派生語大分類の説明? 小分類+ 派生品詞?
　　派生語大分類の捕捉? 。

小分類 ::= 小分類の説明? 用例? 小分類の捕捉? 。

この構造に若干の説明を加える。大分類が複数ある場合に見出しより後ろで大分類番号１より前の部分が<全体的説明>である。大分類が１つなら<全体的説明>はない。同様に、ある大分類の中に小分類が複数ある場合に大分類の最初から小分類番号１より前の部分が<大分類の説明>であり、小分類が１つなら<大分類の説明> はない。辞書エントリーに派生語がある場合は最初の派生語の直前〔〕で囲まれた部分があり、その部分に大分類番号があればそれは全体的捕捉である。派生語がなければ辞書エントリーの最後の部分に〔〕で囲まれた部分があり、その部分に大分類番号が含まれていればそれは全体的捕捉である。大分類の最後の部分に〔〕で囲まれた部分があり、その部分に小分類番号が含まれていればそれは大分類の捕捉である。「…の説明」・「…の捕捉」とはこういったものである。用例・派生品詞は一般には複数あるが、現段階の構造化では用例を一つ一つ分けたり、派生品詞を一つ一つ分けたりはせず、どちらも１つのものとして扱う。こういう訳で上に示した論理構造が「用例*」・「派生品詞*」とはなっていず「用例?」・「派生品詞?」となっているのである。

この論理構造にそった「いずれ」の論理構造を５．１に習って以下に示す。

見出し： いずれ
全体的説明： 00イヅレ
大分類１：
　小分類１：
　　説明：
　　［《何れ］（代）　「どれ・何・どこ・どちら」の意の雅語的表現。
　　用例：
　　ーか一つを選ぶ・ー〔＝どちらも〕劣らぬ・真偽ーにもせよ〔＝本当であっても、無くても〕・ーアヤメかカキツバタ〔＝二つの物がよく似ていて、区別しにくいたとえ〕
大分類２：
　大分類の説明：（副）　〔どういう経過をたどるにしても、の意〕
　小分類１：
　　説明：
　　おそい・速いの違いは有るが、同じ結果になることを表わす。どの道。
　　用例：
　　幾ら隠したってー〔＝どうせ、いつかは〕分かることだ
　小分類２：
　　説明： 余り遠くない将来において。
　　用例：

詳しい事は―〔＝近いうち〕お目にかかって申し上げます・―〔＝間もなく〕雨もあがろう

捕捉：

〔1は、「＜孰れ」とも書く〕

派生語１：

派生語見出し：　―にせよ

派生語大分類１：

小分類１：

説明：　二つの場合のどちら・を選ぶ（になる）にしても。

小分類２：

説明：　どうなるとしても。

派生語２：

派生語見出し：　―も様〈さま〉03

派生語大分類１：

小分類１：

説明：　「おとくい・（お近づき）の皆様」の意の老人語。

5．3　データ構造

先に述べた論理構造を以下に示す構造体に格納する。

```
typedef struct strwl {
        unsigned short  len;
        unsigned short  *pos;
}       strwl_t;
```

文字は全て２バイトで、unsiged short型とする。文字列は開始位置（pos）と長さ（len）で表わす。posが差しているものを「文字列の実体」と呼ぶ。

小分類：

```
struct zsasyoubun {
        strwl_t setumei;   小分類の説明     (A)
        strwl_t yourei;    用例             (B)
        strwl_t hosoku;    小分類の捕捉     (C)
};
```

派生語大分類：

```
struct zsahadaibun {
        strwl_t             setumei;     派生語大分類の説明 (v)
        unsigned char       nsyoubuns;   小分類の数         (w)
        struct zsasyoubun *syoubuns;     小分類             (x)
        strwl_t             haseihinsi;  派生品詞           (y)
        strwl_t             hosoku;      派生語大分類の捕捉 (z)
};
```

派生語：

```
struct zsahaseigo {
        strwl_t              midasi;        派生語見出し     (q)
        strwl_t              setumei;       派生語の説明     (r)
        unsigned char        ndaibuns;      派生語大分類の数 (s)
        struct zsahadaibun *daibuns;        派生語大分類     (t)
        strwl_t              hosoku;        派生語の捕捉     (u)
};
```

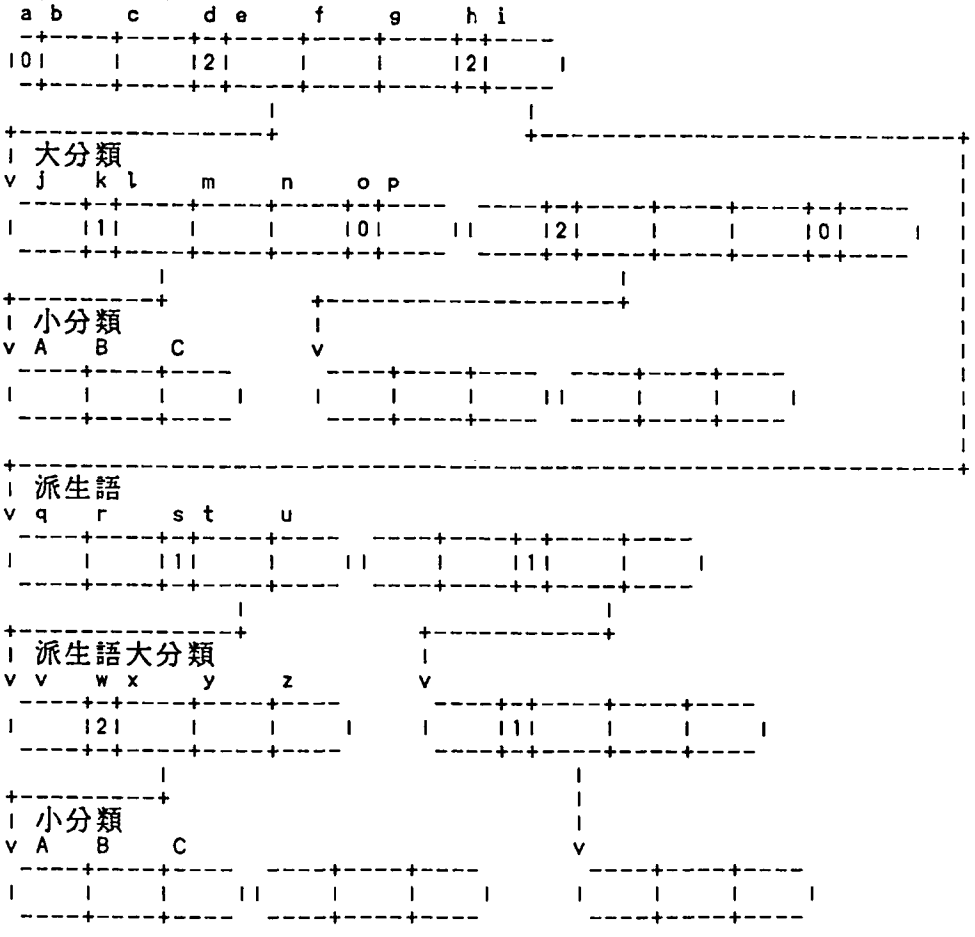
大分類：

```
struct zsadaibun {
        strwl_t             setumei;       大分類の説明     (j)
        unsigned char       nsyoubuns;     小分類の数       (k)
        struct zsasyoubun *syoubuns;       小分類           (l)
        strwl_t             haseihinsi;    派生品詞         (m)
        strwl_t             hosoku;        大分類の捕捉     (n)
        unsigned char       nhaseigos;     派生語の数       (o)
        struct zsahaseigo *haseigos;       派生語           (p)
};
```

辞書エントリー：

```
struct zsatop {
        unsigned char       zyuuyoudo;      重要度          (a)
        strwl_t             midasi;         見出し          (b)
        strwl_t             setumei;        全体的説明      (c)
        unsigned char       ndaibuns;       大分類の数      (d)
        struct zsadaibun    *daibuns;       大分類          (e)
        strwl_t             haseihinsi;     派生品詞        (f)
        strwl_t             hosoku;         全体的捕捉      (g)
        unsigned char       nhaseigos;      派生語の数      (h)
        struct zsahaseigo   *haseigos;      派生語          (i)
};
```

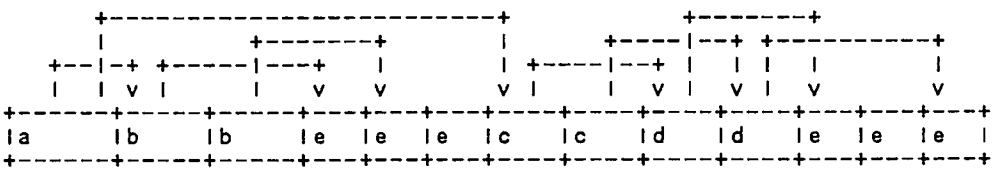
「いずれ」のデータが、これらの構造体に格納される様子を以下に示す。上に示した構造体の右端の()に囲まれたローマ字は以下の図のローマ字と対応している。また、以下の図では文字列（strwl_t 型の構造体要素）については省略してある。



この様に複数の構造体に格納された辞書データをファイルに格納する方法を以

下に述べる。この説明は概念的なもので実際にプログラム中で行なわれている手順とは少々異なる。

まず、参照構造を保ったまま構造体を１つの連続した領域にコピーする。その様子を以下に示す。参照構造を保つのであるから単にコピーするだけではなく、コピーした後でポインターを書き換えなければならない。



a) 辞書エントリー

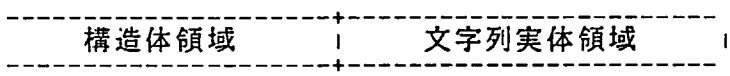
b) 大分類

c) 派生語

d) 派生語大分類

e) 小分類

次に、構造体がコピーされた領域の直後の連続した領域に全ての文字列の実体をコピーし、同時に文字列の実体へのポインター（strwl_t 型の pos 要素）がそこを差す様に変更する。これで辞書エントリーの全てのデータが１箇所にまとめられたことになる。



次に、上の図でいう構造体領域が０番地から始まっていると仮定した値に全てのポインターの値を変更する。こうやってできた領域全体（構造体領域と文字列実体領域を合わせたもの）をファイルに保存する。データを読み込む時は、まず１つの辞書エントリーのデータを連続した領域に読み込み、ポインターの値を変更する。

５．４　構造化の実行

辞書データ全体を５．２に示した論理構造に従って構造化し、それを５．３に示したデータ構造により格納した。構造化を行なうプログラムの形態としては別途 書かれた構造化規則を使うものも考えられる（[3]のコンサイス英和辞典ではそうしている）が、本研究では構造化規則が埋め込まれた形のプログラムを使った。構造化を行なうプログラムの構造化ルーチンは約900行である。２．２で述べた様に大分類・小分類を区切る為の大分類番号・小分類番号とそれ以外の目的の大分類番号・小分類番号がデータでは区別されてない為、構造化作業はそれほ

ど trivial な作業ではなかった。

構造化の過程で辞書データの入力誤りと紙の新明解自体の誤り合わせて60個ほどみつかった。新明解のデータは全体で約370万字あることを考えるとデータの品質はは非常に高いといえる。構造化作業の完了はデータの質の高さにもよるところが大きかった。

5．5　考察

構造化されたデータの要する領域はは約16Mバイトである。[3]ではコンサイス英和辞典の構造化を行なっているがその結果（実際に構造化が全体に対して完成したとして）は約37Mバイトである。以下の点を考えると両者の所用領域は一概には比較できない。

(1) データの持つ論理構造が新明解とコンサイスとでは違う。
(2) 従って構造化のなされ方も異なる。
(3) コンサイスのテキスト-データは約10Mバイトで新明解の約７Mバイト。
(4) 実現された計算機が全く異なる。

しかし、構造化されていないベタ詰めのデータが10Mバイトを要するのに対して構造化されたデータが16Mというのはそれほど悪い数字ではない。

大分類の数・派生語大分類の数・小分類の数の頻度は以下の通りである。

大分類の数	その数の大分類を持つエントリーの数	その割合（%）
1	56400	97.3
2	1442	2.5
3	113	0.2
4	18	0.0
5	4	0.0

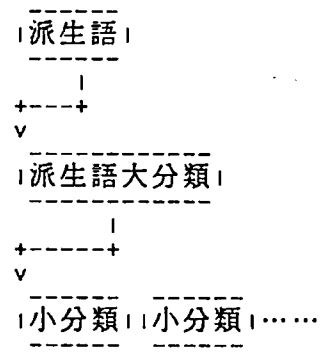
派生語大分類の数	その数の派生語大分類を持つ派生語の数	その割合（%）
1	8202	99.8
2	20	0.2

小分類の数	その数の小分類を持つ大分類または派生語大分類の数	その割合（%）
1	56259	82.9
2	9336	13.7
3	1777	2.6
4	419	0.6
5	100	0.1
6	40	0.1
7	16	0.0
8	7	0.0
9	2	0.0
10	1	0.0

構造化に使った論理構造では以下の様になっている為、

派生語 ::= … 派生語大分類+ …
派生語大分類 ::= … 小分類+ …

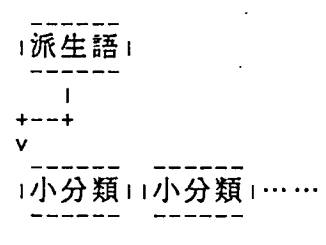
派生語大分類が１つの時、構造体の参照関係は以下の様になる。



派生語大分類が１つの派生語が全体の99.8%を占めているが、仮に、これが100%であるとすると、論理構造は

派生語 ::= … 小分類+ …

となり、構造体の参照関係は以下の様になる。



つまり、僅か0.2%の派生語の為に<派生語大分類>という構造要素は存在し、この構造要素の為に記憶が使われている。派生語大分類が１つの時は派生語が直接小分類を参照する様にすると約190Kバイト節約でき、同様に大分類が１つの時は辞書エントリーが直接小分類を参照する様にすると約1.6Mバイト節約できる。

この点では現在のデータ構造・論理構造には改善の余地がある。
構造化の作業を進めるうちに気が付いたことを以下に列挙する。

(1) 構造化に使用した論理構造では派生語は全体に属するか大分類に属するか、なのであるが、文面から考えて小分類に属するとされている派生語を持つエントリーが５つあった。

(2) 小分類の中に更に「細分類」があることがある。細分類を持つ小分類は辞書全体で44個あった。その例を以下に示す。「a)」・「b)」が細分類を示している。

【よび　だし】　00［呼（び）出（し）］
1)呼び出すこと。
「警察からの一」
2)〔←呼出し電話05〕a)電話を持たない人が近所の電話で取り次いでもらうこと。また、その電話。　b)遊びに来いなどという誘いの電話。
3)〔←呼出しやっこ05〕〔すもう場で〕東西の力士の名を呼び上げて、登場させる係。

小分類は辞書全体で82874あり、44というのは0.1%にも満たないから論理構造を

小分類 ::= … 細分類+ …

と変更し、データ構造もそれに合わせて変更することは得策とは言えない。大分類が１つのエントリーが全体の97.3%で、派生語大分類が１つの派生語が全体の99.8%であることと考え合わせると現在の論理構造・データ構造とは違った「思想」の論理構造・データ構造を考えるべきかもしれない。

５．６　今後の課題

辞書データの構造化は割合単純な「区切ることのみ」により可能な構造化が完成しているに過ぎない。５．１に示した論理構造やそれに近い論理構造への「より高度な」構造化は今後の課題である。また、現在完成している構造化の論理構造・データ構造も「区切ることのみ」による構造化の為の論理構造・データ構造として必ずしも満足の行くものではない。より高度な構造化をする前にこちらの見直しをすべきかも知れない。

計算機上に実現した新明解国語辞典は「jjd」というコマンド名で、引数なしで

jjd <改行>

という操作で起動する。起動すると「(新明解)」というプロンプトを表示する。jjd コマンドは自分自身でローマ字仮名変換を行なう。仮名の入力をフロント-エンド-プロセッサーに頼るのは操作上好ましくないからである。引きたい語の仮名綴りを入力して <改行> を入力すると４．２．２で述べた手順で検索を行ない結果を表示する。

表示は論理構造を反映たインデンテーションを施してなされる。論理構造を反映した体裁のものはベタ詰めのものより格段に見易いのであるが、紙の辞書は紙面の都合でベタ詰めで印刷されている。内容の「見易さ」も電子化の大きなメリットである。表示例については第１章を参照願いたい。表示が１画面以内に収まる時はそのまま表示され、収まらない時は１画面ごとに表示が止まる。何も入力しないで <改行> を入力すると直前に表示されたエントリーの次の辞書エントリーが表示され、「- <改行>」を入力すると一つ前の辞書エントリーが表示される。

以下に示す様に特定の文字列をその中に含む辞書エントリーを検索することも可能である。

(新明解) /船舶
【うき　に】 00 [浮（き）荷]
1)〔海に投げ込まれたり、波にさらわれたりして〕海上に漂う、船舶の積み荷。
2)引取人が決まらないうちに、荷主が船積みして売り出す品物。

辞書を引いていてデータの誤りに気が付いた時は一番最近表示されたエントリーを

(新明解) > ファイル名 <改行>

とすることによりファイルに書き出すことができる。書き出されたファイルをエディターにより修正した後に

(新明解) < ファイル名 <改行>

とすることにより辞書データの修正結果として取り込むことができる。

第７章　結論

以上、本論文では電子国語辞典「電明解」に関して、主にファイル構造・見出しによる辞書引き・辞書データの構造化などの点について論じた。これらは実際に国語辞典を計算機上に実現する過程での考察であるが、英和辞典・和英辞典などにも通ずることがらも多いと思われる。

構造化に関しては、もととなる辞書のテキスト-データが十分に高品質ならばプログラムを使ってデータを構造化することが可能であることを示した。

本研究の実現では効率に重点を置いており、全データの格納に要する記憶は20Mバイト程度であり、また、ＣＰＵの能力もそれほどは要求しない為、本研究の電子辞書をパーソナル-コンピューターに移植することも十分可能である。

電子国語辞典の実現は、まだ その端緒についたばかりの段階であるが、現在の実現でも以下の点で紙の辞書より優れている。

(1) 可能な辞書引きの方法
紙の辞書を人間が引く作業は概ね最長一致検索でカバーされる。「*」を含む検索は紙の辞書では（事実上）不可能である。

(2) 辞書引きの手間
人間の操作は仮名綴りを入力するだけであるから紙の辞書を引くことに比べて人間の手間は圧倒的に少ない。しかも検索は数十ミリ秒で終わる。

(3) 表示の見易さ
紙の辞書は物理的に実現可能でなければならず、また、頁数が多くなると引く手間が増えるから文字をきっちり詰め込まざるを得ないが、電子国語辞典ではその様なことはないから見易い表示ができる。

一方 劣っている点もなくはない。紙の新明解には図はないが、「清朝体」という書体の例や、音符、温泉のマーク、JIS マークなどは現れる。新明解のデータではそれぞれに適宜コードが割り当てられているのであるが、こういったものは「図形情報」なのであって、それにコードを割り当てるだけでは十分ではない。紙の辞書には、たとえ図はなくても一部 図形情報を含んでいるのである。現在の電子国語辞典の実現では文字情報しか扱うことができず、この点では紙の新明解に比べ劣っているのである。

今後の課題としてはこれまでに述べた辞書引きのヒューリスティック（４章）や構造化（５章）に関することの他にも多々 考えられる。以下それを列挙する。

* 現在のところ辞書を仮名見出しから引くことが実現されているのみでこれだけでは「電子国語辞典ならではの利用形態」とは言えない。派生語・例文・表記からの検索や、参照先・対義語の自動参照なども実現すべきである。表記からの検索には漢字の入力が必要であるが、その際は漢字の読みからだけでなく、画数や部首などから漢字を入力できることが必要である。

* 以下の様なエントリーもあるが、

  【あく　き】　01［悪鬼］->あっき

  「あっき」の方に「あくき」とも表記する旨のことを書いておいてこのエントリー自体は削除し、「あくき」で「あっき」が引ける様にすべきであろう。

* 現在はエントリーの追加・削除ができない。電子化国語辞典には辞書編纂の電子化の側面があることを考えると追加・削除はできてしかるべきである。

* 辞書データはネットワーク中の１つの計算機に置いておいてそれを他の計算機から使う様なＬＡＮの環境に合わせた実現。

* 多くの計算機環境では漢字の入力には漢字入力フロント-エンド-プロセッサーが用いられているが、電子新明解国語辞典を包含して、随時それを呼び出せる様なフロント-エンド-プロセッサーは非常に便利であろう。

謝辞

本研究を進めるに当たって有益な助言を下さった田中研究室の皆様に感謝します。また、本研究に用いた新明解国語辞典のデータは非常に高品質で誤りが少なく、本質的でない苦労が非常に少なかったことは特筆に値する。高品質なデータを提供して下さった電子技術総合研究所・株式会社三省堂の皆様に感謝します。

参考文献

[1] 金田一京助ら：新明解国語辞典第３版、1982年、株式会社三省堂

[2] 横山、萩野：「国語辞典磁気テープのドキュメント」、
電総研彙，48，8，PP.672-677、1984年

[3] 長尾真：昭和55,56年科学研究費補助金 試験研究(1)研究成果報告書、
1982年

[4] 中村、長尾：辞書情報検索ツール —ＬＤＯＣＥ／ＲＤＢ—、
特定研究「言語情報処理の高度化」第２回研究発表資料、1987年

[5] 鶴丸弘昭：「日常辞書の機械化とその応用」、
第１回「大学と科学」公開シンポジウム『日本語の特性と機械翻訳』予稿集
PP.99-115、1987年

[6] ＯＡＳＹＳ100ＧＸＣＤ 広辞苑ＣＤ－ＲＯＭシステム操作説明書、1987年、
富士通株式会社

[7] UNIX Programmer's Manual Chapter 3

[8] D. E. Knuth、The Art of Computer Programming Volume 3 pp. 481-489、
Addison-Wesley、1973年