2005－NL－168（3）
2005／7／22

階層構造データ列の簡易な高速検索アルゴリズム

村田　真樹[†]　内山　将夫[†]　金丸　敏幸[††,†]　井佐原　均[†]

† 独立行政法人 情報通信研究機構
〒 619-0289 京都府相楽郡精華町光台 3-5
†† 京都大学大学院人間・環境学研究科
〒 606-8501 京都府京都市左京区吉田二本松町
E-mail: †{murata,mutiyama,isahara}@nict.go.jp, ††kanamaru@hi.h.kyoto-u.ac.jp,kanamaru@nict.go.jp

あらまし　本稿では，京都テキストコーパスの形態素情報の品詞，品詞細分類，単語などの階層構造をもったデータの列を簡易に高速検索するアルゴリズムを記述する．本稿のアルゴリズムでは各データについて抽象度の低い階層のデータを二つの同じ抽象度の高いデータで挟んだデータを作成し，それらをつなげて一つのテキストとしそれをデータベースに格納する．データベースからの検索には suffix array を利用する．実際に実験を行なった結果，本稿で提案する手法は比較手法に比べて，速いときで 194 倍速く，平均でも 24 倍速かった．本稿のアルゴリズムは他の形態の階層構造にも利用できる．本手法の応用としては，Web テキストなどにおいて各単語に下位の意味ラベル，中位の意味ラベル，上位の意味ラベルなどの意味的な階層の情報を付与しておきそのデータに対して本手法を適用することが考えられる．このようにすると Web テキストにおいて「下位の意味ラベル：行政機関」と「単語：の」と「中位の意味ラベル：技術」をつなげた検索キーのようなものも検索できるようになり，従来の WEB 検索よりもより汎用的で便利な検索が実現できるようになる．WEB の検索は一般的に需要が大きく，本稿のアルゴリズムはそういう需要の大きな課題にも利用できるものである．

キーワード　階層構造，高速検索，データ列，suffix array，コーパス，ウェブ

Fast Search Algorithm for Sequences of Hierarchically Structured Data

Masaki MURATA[†], Masao UTIYAMA[†], Toshiyuki KANAMARU[††,†], and Hitoshi ISAHARA[†]

† National Institute of Information and Communications Technology
3-5 Hikaridai, Seika-cho, Soraku-gun, Kyoto, 619-0289, Japan
†† Graduate School of Human and Environmental Studies, Kyoto University
Yoshida-nihonmatsu-cho, Sakyo-ku, Kyoto, 606-8501, Japan
E-mail: †{murata,mutiyama,isahara}@nict.go.jp, ††kanamaru@hi.h.kyoto-u.ac.jp,kanamaru@nict.go.jp

Abstract　We developed an algorithm for quickly searching sequences of hierarchically structured data, such as the Kyoto Text Corpus, where each word includes information on its part of speech (POS), minor POS and word itself. Using our method, we first make a data item where each data item in a lower level is surrounded by two data items in a higher level. We then connect these data items to make a long string and store the string in a database. We use suffix arrays to retrieve queries on the database. Our experiments showed that our method was 450 times faster than a conventional method at fastest and 64 times faster on average. Our method can be used for other kinds of hierarchically structured data, such as Web applications. Methods that can be used on such data are in high demand. For example, our method can be used to retrieve Web text, that includes hierarchical information of low, middle, and high semantic levels. If we use our method for such Web text, we can query using the terms, "Middle semantic level: technique", "Word: of", and "Low semantic level: administrative organ"; in other words, our retrieval method is more useful and convenient than conventional web retrieval.

Key words　Hierarchical Structure, Fast Search, Sequence of Data, Suffix Array, Corpus, Web

1. はじめに

本稿では，階層構造の形式をとったデータ列を簡易に高速に検索するアルゴリズムについて記述する．例えば，京都テキストコーパス[1]は一つ一つの形態素には単語，品詞細分類，品詞のように抽象度が低い単語から抽象度が高い品詞の情報が格納されている．本稿でいう階層構造の形式は上述のように抽象度が低いデータから抽象度が高いデータがあることをいう．そして，この階層構造の形式を持つ形態素が連続して記述される京都テキストコーパスは，階層構造の形式をとったデータの列とみることができる．本稿で記述するアルゴリズムは，そういう階層構造の形式をとったデータ列を簡易に高速に検索するものである．本稿で記述するアルゴリズムは，京都テキストコーパスに限らず，階層構造の形式をとったデータ列に対してすべて適用できる．(注1)

本稿のアルゴリズムの用途としては，まず上述のような単語，品詞細分類，品詞のような階層をもったコーパスでの検索があげられる．このようなコーパスを利用した言語処理システムで高速検索を実現したい場合，本稿のアルゴリズムが役に立つ．また，言語学者，または，言語的情報を利用したいユーザにとっては，実際に上述のような品詞情報などの文法データが付いたデータを検索したい場合がある．そのようなときにも役に立つ．また，本稿のアルゴリズムは，上述のような単語，品詞細分類，品詞のような階層構造以外のデータにも利用できる．例えば，テキストを単語に分割し，その各単語に対して，単語，下位の意味ラベル，中位の意味ラベル，上位の意味ラベルのように，その単語の意味ラベルを下位のものから上位のものまで付与し，そういう階層構造を持ったデータに対しても本稿のアルゴリズムを利用できる．具体的には，例えば，単語が郵政省ならその上位の意味である，「行政機関」「役所」「組織」が下位の意味ラベル，中位の意味ラベル，上位の意味ラベルとなり，そういう意味ラベルが付与されたデータでの検索にも本稿のアルゴリズムが役に立つ．単語に対して意味ラベルをつけることはシソーラスなどの単語に対する階層的な知識が記述されたデータベースを利用すれば簡単にできる．WEBのテキストに対してこのような意味ラベルをつけておき，そのテキストに対して本稿のアルゴリズムを利用すると，例えば，「意味ラベル：役所」と「単語：の」と「意味ラベル：技術」が連接した表現を検索するということも可能になる．このような検索が一般のWEBテキストに対して使えれば，従来のWEB検索よりもより汎用的で便利な検索が実現できるようになる．WEBの検索は一般的に需要が大きく，本稿のアルゴリズムはそういう需要の大きな課題にも利用できるものである．

2. アルゴリズム

2.1 データベースの格納法

本稿のアルゴリズムでは検索にはsuffix array[3]～[5]を利用する．そしてデータを格納するデータベースでは，データを特殊な形式で格納することで高速検索できるようにする．

表1 形態素列データの例

単語	品詞細分類	品詞
郵政省	組織名	名詞
の	接続助詞	助詞
技術	普通名詞	名詞
。	句点	記号

データは以下のような形式で格納する．階層構造の形式をとった各データは，それぞれ抽象度の低い下位階層のデータがそのデータに対応する二つの同じ抽象度の高い上位階層のデータによって挟まれる形で表現され，この形で表現されたデータを連続して記述したテキストをデータベースに格納する．そして，このデータベースに対してsuffix arrayを利用して，インデキシング処理をして高速検索できるようにする．

例えば，京都テキストコーパスをこの方法でデータベースに格納することを考えてみる．ここでは特に，表1の形態素列データを格納することを考えてみよう．表1の各データは，単語，品詞細分類，品詞という抽象度の低いものから高いものまである階層構造を持ったものである．ここではそういうデータが各行に配置して書いてある．このデータを本稿のアルゴリズムのデータベースに格納することを考える．「郵政省」の行のデータを考える．このデータで最も抽象度の低いデータは「郵政省」である．これの次に抽象度の低いデータは「組織名」である．そこで，「郵政省」を囲うように「組織名」を配置する．そうすると，「組織名：郵政省：組織名」が得られる．ここではデータの切れ目がわかるように「：」の記号を入れている．次に抽象度の低いデータは「名詞」である．そこで，「組織名：郵政省：組織名」を囲うように「名詞」を配置する．そうすると，「名詞：組織名：郵政省：組織名：名詞」が得られる．ここでもデータの切れ目がわかるように「：」の記号を入れている．これで「郵政省」のデータは完成である．これと同じように「の」「技術」「。」も同様の形式に変換し，それらすべてをつなげた以下のデータを得る．

／名詞：組織名：郵政省：組織名：名詞／
助詞：接続助詞：の：接続助詞：助詞／
名詞：普通名詞：技術：普通名詞：名詞／
記号：句点：。：句点：記号／

それぞれのデータの切れ目がわかるように先頭と末尾とデータの間に「／」の記号を入れている．ここでは見やすさのために一形態素を一行ずつにわけて表示している．実際にデータベースに格納する場合はそのように複数の行にわけずに一行につなげたものを格納する．(以降も同様に見やすさのために行をわけて表示する．) 表1のデータを上述の形式に変換してデータベースに格納する方法が本稿の方法である．

2.2 検索の方法

検索したい表現が与えられると，その表現も，データベースの格納の際に行なったように抽象度の低いデータを抽象度の高いデータが囲むように整形しその整形したものを検索キーとし

(注1)：本稿のアルゴリズムについて特許[2]を取得している．

て利用して検索を行なう．ただし，検索したい表現のデータ列の先頭と末尾のデータついては上記と異なる整形を行なう．検索したい表現のデータ列の先頭のデータについては，右から抽象度の高い順にデータを並べる．検索したい表現のデータ列の末尾のデータについては，左から抽象度の高い順にデータを並べる．

例えば，検索したい表現が以下のものだったとする．

組織名：名詞
の：接続助詞：助詞
普通名詞：名詞

つまり，一つ目の形態素は品詞が名詞で品詞細分類が組織名で，二つ目の形態素は品詞が助詞で品詞細分類が接続助詞で単語が「の」で，三つ目の形態素は品詞が名詞で品詞細分類が普通名詞であるものである．

その場合は，先頭の形態素は右から抽象度の高い順にデータを並べた「：組織名：名詞」となる．真中の形態素は抽象度の低いデータを抽象度の高いデータが囲むように整形した「助詞：接続助詞：の：接続助詞：助詞」となる．末尾の形態素は左から抽象度の高い順にデータを並べた「名詞：普通名詞：」となる．そして，整形した検索キーはそれぞれをつなげた以下のものとなる．

：組織名：名詞／助詞：接続助詞：の：接続助詞：助詞
／名詞：普通名詞：

そして，この表現を suffix array を利用してデータベースを検索する．

suffix array はテキスト中の文字列を高速に検索するアルゴリズムとして広く知られている．検索速度はテキストの大きさを n とすると，log(n) のオーダである．

本稿のアルゴリズムでは上記の複雑な形式のデータも suffix array の一回の検索で実現することができるのである．

また，例えば，検索したい表現が以下のものだったとする．

組織名：名詞
助詞

つまり，一つ目の形態素が品詞が名詞で品詞細分類が組織名で，二つ目の形態素が助詞であるものである．この場合先頭の形態素は右から抽象度の高い順にデータを並べた「組織名：名詞」となり，末尾の形態素は左から抽象度の高い順にデータを並べた「助詞」となるので，整形した検索キーはそれぞれをつなげた以下のものとなる．

組織名：名詞／助詞

この例も，本稿のアルゴリズムでは suffix array の一回の検索で検索を実現することができる．

2.3 補　　足

本稿では抽象度の低いデータを抽象度の高いデータが囲むように整形したが，これを単純に抽象度の低いデータから順に並べたものだけでデータベースを構成した場合は本稿のような検索は実現できない．例えば，この方法だと先の表 1 のデータは，

／郵政省：組織名：名詞／
の：接続助詞：助詞／
技術：普通名詞：名詞／
。：句点：記号／

の形式でデータベースに格納される．この場合は，

組織名：名詞
助詞

のように一つ目の形態素が品詞が名詞で品詞細分類が組織名で，二つ目の形態素が助詞であるものを一回で検索することができない．これは，データベースで，「組織名：名詞」と「助詞」のデータの間に「の：接続助詞」が存在し，連続した一つの表現で検索キーを表現できないためである．これに対して本稿のアルゴリズムでは，

／名詞：組織名：郵政省：組織名：名詞／
助詞：接続助詞：の：接続助詞：助詞／
名詞：普通名詞：技術：普通名詞：名詞／
記号：句点：。：句点：記号／

のようにデータベースに格納しているため，「組織名：名詞」と「助詞」の間には区切り記号「／」しかないため，

組織名：名詞
助詞

というものを検索したい場合は，「組織名：名詞／助詞」という整形した検索キーを作れば一回の検索だけで検索を実現できるのである．

2.4 アルゴリズムの拡張

上述ではデータベースの構築には単純に各データの階層のデータを「：」の区切り記号でつなげて表現したが，より丁寧に，階層データが何個目の階層のものかを示す数字もつけて階層のデータを記述してもよい．例えば，品詞の階層を 1，品詞細分類の階層を 2，単語の階層を 3 とすると，表 1 の形態素列データは以下のデータの形で格納される．

／1；名詞：2；組織名：3；郵政省：2；組織名：1；
名詞／
1；助詞：2；接続助詞：3；の：2；接続助詞：1；
助詞／
1；名詞：2；普通名詞：3；技術：2；普通名詞：1；
名詞／
1；記号：2；句点：3；。：2；句点：1；記号／

ここでは階層の番号とデータの間には「；」の区切り記号を利用している．

また，この場合は検索キーでもこの階層を示す数字をつけて検索する．例えば，

組織名：名詞
助詞

というものを検索したい場合は,「2；組織名：1；名詞／1；助詞」という整形した検索キーを作って検索する.

2.5 アルゴリズムの注意

本稿のアルゴリズムでは一回の検索で検索できるデータ列には条件がある. 検索したいデータ列の先頭と末尾のデータ以外のデータは, 抽象度の最も低いデータから高いデータまですべてを検索キー内で指定すること, また, 検索したいデータ列の先頭と末尾のデータは, 抽象度の最も高いデータから順に省略することなくデータを検索キー内で指定することである. そうでなければ, 検索キーを一つの連続した文字列で表現できない.

例えば,

組織名
助詞

という, 一つ目の形態素の品詞細分類が「組織名」で二つ目の形態素の品詞が「助詞」であるものは一回の検索では検索できない. データベースでは,

／名詞：組織名：郵政省：組織名：名詞／
助詞：接続助詞：の：接続助詞：助詞／

のようにデータが格納されており,「組織名」と「助詞」の間に「名詞」が存在しており, 一つの連続した文字列に検索キーを表現することができず, 一回の検索では検索できないのである.

この場合は, 階層構造データに関する知識を使い, 上位の階層のデータを補ってやることで検索するとよい. 例えば, 先の例だと,「組織名」の上位のデータは「名詞」である. このため,

組織名
助詞

という検索が与えられたときも,「組織名」に対してそれの上位のデータの「名詞」を補ってやり(注2),

組織名 名詞
助詞

という検索が与えられたと解釈して, アルゴリズムにしたがって「組織名：名詞／助詞」という検索キーを作成する. こうすれば一回の検索で検索できる.

3. 実　　験

本稿のアルゴリズムは, 高速検索の suffix array で一回の検索で実現できるので, 高速であるのは明らかではあるが, 普通の検索に比べて本稿のアルゴリズムが具体的にどのぐらい高速であるのかを確認するために実験を行なった. この実験では本稿のアルゴリズムの手法と以下の比較手法の二つを利用した.

(注2)：階層構造データによって下位のデータに対して上位のデータが複数割り当てうる場合がある. そのような場合は, その複数のもので検索キーを作り, そこでできた複数の検索キーで OR 検索をするか, 複数割り当てうるものから実際に使いたいものをユーザーが選択しその選択したもので検索キーを作りそれで検索するか, 多義性解消の技術を使って現時点で使われている意味がどの意味であるかを推定しその推定した意味に基づいて検索キーを作るなどをする必要がある.

表 2　平均検索時間

比較手法の検索時間	本手法の検索時間	検索時間の比
0.410 秒	0.017 秒	24.3

比較手法としては一回の検索では検索できないために検索キーを二つにわけて AND 検索を利用する. AND 検索としては, 一つ目の検索キーを suffix array で検索し, そしてその検索結果に対して二つ目の検索キーがあるかないかを調べてあったものを検索結果として取り出すという方法を使った.

実験では SUN UE420R (450MHz) の計算機を利用した. プログラムには C を利用した.

データベースに格納するデータとしては京都テキストコーパス Version 2.0 [6](約 2 万文) を利用した. データベースでは, 品詞を最上位の階層に, 品詞細分類を二番目に上位の階層に (ただし, 動詞や形容詞など変化する単語については品詞細分類でなく変化形を二番目に上位の階層のデータとして用いた), コーパスにあった全形態素情報を一つにまとめて三番目に上位の階層に用いた. 例えば,「首相　しゅしょう　＊　名詞　普通名詞　＊　＊」という形態素であれば, 最上位の階層は「名詞」であり, 二番目に上位の階層は「普通名詞」であるので,「名詞　普通名詞　首相　しゅしょう　＊　名詞　普通名詞　＊　＊　普通名詞　名詞」となる. ここでは区切り記号には全角空白“　”を利用した. また全データは検索しやすいようにすべて全角に変換してからデータベース化した.

実験結果を表 2, 表 3, 表 4 に示す. 実験では表 3 の第一列目に示す 24 個を第一検索キー, 表 4 の第一列目に示す 51 個を第二検索キーとして用いるすべての組み合わせの 1224 通り (= 24 × 51) の検索の実験を行なった. 探すデータは上記第一検索キーと第二検索キーがこの順で連接するデータである. 本稿のアルゴリズムはこれら二つの検索キーを一つの検索キーで表現して一回の検索を行なう. 比較手法では, 第一検索キーでまず検索し, その検索結果から第二検索キーに合致するものを取り出す.

表 2 には比較手法と本手法の平均検索時間と, 本手法に対する比較手法の平均検索時間の比を示している. 本手法は平均 0.017 秒と極めて速いが, 比較手法では平均 0.410 秒かかっており, 本手法は比較手法よりも 24 倍速いことがわかる. このことから本手法の有効性がわかる.

また, 表 3 には, 第一検索キーごとに, 第一検索キーの検索結果の総数 (すなわち, データベース中の第一検索キーに合致するものの個数), 第一検索キーがそのデータの場合の比較手法の検索時間の平均, 本アルゴリズムの検索時間の平均, 本アルゴリズムに対する比較手法の検索時間の平均の比を示している. 表 4 には, 第二検索キーごとに, 第二検索キーの検索結果の総数, 第二検索キーがそのデータの場合の比較手法の検索時間の平均, 本アルゴリズムの検索時間の平均, 本アルゴリズムに対する比較手法の検索時間の平均の比を示している.

表 3 の結果を見てまずわかることは, 第一検索キーの検索結果総数が大きい時に, 本アルゴリズムに対する比較手法の検索時間の平均の比も大きくなることである. すなわち, 第一検索

表 3　第一検索キーごとの検索時間

第一検索キー	第一検索キーの検索結果総数	比較手法の検索時間	本手法の検索時間	検索時間の比
「　動詞｜」	48692	1.123 秒	0.017 秒	65.1
「　形容詞｜」	11213	0.274 秒	0.018 秒	15.5
「　接頭辞｜」	3150	0.082 秒	0.017 秒	4.7
「　連体詞｜」	532	0.025 秒	0.016 秒	1.5
「　指示詞｜」	4513	0.120 秒	0.017 秒	7.1
「　名詞｜」	178487	3.955 秒	0.020 秒	194.0
「　サ変名詞　名詞｜」	45110	1.103 秒	0.019 秒	58.6
「　普通名詞　名詞｜」	87130	2.085 秒	0.017 秒	120.8
「　形式名詞　名詞｜」	4704	0.130 秒	0.015 秒	8.5
「　人名　名詞｜」	6845	0.172 秒	0.015 秒	11.7
「　地名　名詞｜」	10196	0.257 秒	0.016 秒	16.0
「　組織名　名詞｜」	3249	0.091 秒	0.019 秒	4.9
「　固有名詞　名詞｜」	97	0.020 秒	0.018 秒	1.1
「首相　しゅしょう　＊　名詞　普通名詞　＊　＊　普通名詞　名詞｜」	447	0.030 秒	0.018 秒	1.7
「政府　せいふ　＊　名詞　普通名詞　＊　＊　普通名詞　名詞｜」	529	0.033 秒	0.019 秒	1.7
「期待　きたい　＊　名詞　サ変名詞　＊　＊　サ変名詞　名詞｜」	152	0.018 秒	0.016 秒	1.1
「人　ひと　＊　名詞　普通名詞　＊　＊　普通名詞　名詞｜」	542	0.030 秒	0.015 秒	2.0
「村山　むらやま　＊　名詞　人名　＊　＊　人名　名詞｜」	232	0.021 秒	0.016 秒	1.3
「東京　とうきょう　＊　名詞　地名　＊　＊　地名　名詞｜」	390	0.023 秒	0.014 秒	1.7
「もの　もの　＊　名詞　形式名詞　＊　＊　形式名詞　名詞｜」	681	0.035 秒	0.017 秒	2.0
「こと　こと　＊　名詞　形式名詞　＊　＊　形式名詞　名詞｜」	2255	0.088 秒	0.019 秒	4.7
「の　の　＊　名詞　形式名詞　＊　＊　形式名詞　名詞｜」	1350	0.061 秒	0.015 秒	4.1
「思う　おもう　思う　動詞　＊　子音動詞ワ行　基本形　基本形　動詞｜」	120	0.020 秒	0.015 秒	1.4
「ある　ある　ある　動詞　＊　子音動詞ラ行　基本形　基本形　動詞｜」	1228	0.054 秒	0.018 秒	3.0

キーの検索結果総数が大きい場合は，本アルゴリズムに比べて比較手法の検索時間が圧倒的に大きくなる．例えば，第一検索キーが名詞の場合は，第一検索キーの検索結果総数が 178487 個と非常に大きく，このため，比較手法では検索時間の平均は 4 秒もかかっている．それに比べて本アルゴリズムは 0.020 秒であり極めて高速である．検索時間の比は，194.0 であり，194 倍も本手法の方が速くなるのである．このことから本アルゴリズムの有効性がわかる．

逆に第一検索キーの検索結果総数が小さい時は，本アルゴリズムに対する比較手法の検索時間の平均の比はそれほど大きくない．例えば，「　固有名詞　名詞｜」の場合は，第一検索キーの検索結果総数が 97 と小さいがそのときは比較手法と本アルゴリズムとの検索時間はそれぞれ 0.020 秒と 0.018 秒でそれほど大きくは変わらない．

次に第二検索キーごとに調査した表 4 の結果を見てみる．こちらの方はどの第二検索キーに対しても比較手法はほぼ同じ検索時間である．つまり比較手法の速度は第二検索キーにあまり依存しないことがわかる．

より詳細な調査を行なうために，検索キーの検索結果総数と比較手法の検索時間の関係，検索キーの検索結果総数と本手法の検索時間の関係，比較手法と本手法の検索時間の関係の散布図を描いてみた．これは表 3 と表 4 の両方のデータで行なった．その結果を図 1 と図 2 に示す．

図 1(a) を見ると，第一検索キーの検索結果総数と比較手法の検索時間がほぼ比例していることがわかる．このことから，比較手法の検索時間は，第一検索キーの検索結果総数にほぼ比例することがわかる．比較手法は，第一検索キーの検索結果すべてに対して第二検索キーが第一検索キーの直後に連接しているかどうかを調べる手法であるので，第一検索キーの検索結果の数だけ，第二検索キーが第一検索キーの直後に連接しているかどうかを調べる必要がある．このため，検索時間が第一検索キーの検索結果の数に比例するものと思われる．

次に図 2(a) を見ると，第二検索キーの検索結果総数に対して，比較手法の検索時間がほぼ一定であることがわかる．比較手法は，第一検索キーの検索結果すべてに対して第二検索キーが第一検索キーの直後に連接しているかどうかを調べる手法であるため，第二検索キーに相当するものが実際のデータベースにどのくらいあるかは実際の検索時間に影響を与えない．このため，第二検索キーの検索結果総数に対して，比較手法の検索時間がほぼ一定であるのだと思われる．

図 1(b) と図 2(b) を見ると，だいたい本手法の検索時間は，第一検索キーの検索結果総数，または，第二検索キーの検索結果総数の大きさに応じて大きくなることがわかる．

次に本手法に対する比較手法の検索時間の比について，図 1(c) と図 2(c) を使って考察する．本手法に対する比較手法の検索時間の比は，比較手法の検索時間を本手法の検索時間で割ったものであるので，図 1(a), 図 2(a) の値を図 1(b), 図 2(b) の値で割ったものが図 1(c), 図 2(c) になる．図を見ると，実際にそうなっている．図 1 だと，図 1(a) で比較手法はほぼ左下から右上への直線であったが，少し右上がりの図 1(b) の値で割るため，図 1(c) の値は，左下から右上へのびているが少し直線に比べて右下がりに伸びている．また，図 2 だと，図 2(a) で比較手法はほぼ横一直線の直線であったが，少し右上がりの図 1(b) の値で割るため，図 1(c) の値は，少し右下がりのデータになっている．つまり，本手法に対する比較手法の検索時間の比は，第一検索キーの検索結果総数に対しては比例関係に比べ

表 4　第二検索キーごとの検索時間

第二検索キー	第二検索キーの検索結果総数	比較手法の検索時間	本手法の検索時間	検索時間の比
「動詞　」	48692	0.396 秒	0.015 秒	27.2
「助動詞　」	4074	0.417 秒	0.018 秒	23.3
「形容詞　」	11213	0.406 秒	0.015 秒	26.4
「接尾辞　」	37322	0.420 秒	0.020 秒	20.6
「判定詞　」	4616	0.403 秒	0.018 秒	22.5
「名詞　」	178487	0.408 秒	0.020 秒	20.4
「名詞　サ変名詞　」	45110	0.400 秒	0.016 秒	24.6
「名詞　普通名詞　」	87130	0.425 秒	0.018 秒	24.3
「名詞　形式名詞　」	4704	0.407 秒	0.015 秒	27.1
「名詞　人名　」	6845	0.425 秒	0.014 秒	30.9
「名詞　地名　」	10196	0.405 秒	0.017 秒	23.7
「名詞　組織名　」	3249	0.420 秒	0.017 秒	25.2
「名詞　固有名詞　」	97	0.392 秒	0.018 秒	21.4
「助詞　」	125877	0.417 秒	0.022 秒	18.5
「助詞　格助詞　」	68951	0.410 秒	0.018 秒	22.3
「助詞　終助詞　」	860	0.422 秒	0.015 秒	27.4
「助詞　副助詞　」	24565	0.394 秒	0.019 秒	21.0
「助詞　接続助詞　」	31501	0.410 秒	0.020 秒	20.9
「助詞　格助詞　から　」	2286	0.419 秒	0.017 秒	24.5
「助詞　格助詞　が　」	12240	0.404 秒	0.014 秒	28.5
「助詞　格助詞　で　」	6901	0.400 秒	0.014 秒	29.1
「助詞　格助詞　と　」	11248	0.402 秒	0.021 秒	18.9
「助詞　格助詞　に　」	15697	0.416 秒	0.018 秒	23.8
「助詞　格助詞　の　」	1601	0.421 秒	0.018 秒	24.1
「助詞　格助詞　は　」	1	0.394 秒	0.015 秒	25.5
「助詞　格助詞　へ　」	883	0.426 秒	0.018 秒	23.3
「助詞　格助詞　まで　」	795	0.395 秒	0.016 秒	24.9
「助詞　格助詞　より　」	232	0.426 秒	0.020 秒	20.9
「助詞　格助詞　を　」	17064	0.411 秒	0.016 秒	25.3
「助詞　終助詞　か　」	699	0.407 秒	0.013 秒	31.5
「助詞　終助詞　ね　」	47	0.405 秒	0.019 秒	21.6
「助詞　終助詞　よ　」	41	0.406 秒	0.019 秒	21.2
「助詞　終助詞　わ　」	4	0.407 秒	0.017 秒	24.4
「助詞　副助詞　くらい　」	19	0.412 秒	0.014 秒	29.9
「助詞　副助詞　ぐらい　」	17	0.422 秒	0.022 秒	19.5
「助詞　副助詞　こそ　」	87	0.417 秒	0.016 秒	26.3
「助詞　副助詞　さえ　」	52	0.403 秒	0.015 秒	27.7
「助詞　副助詞　しか　」	127	0.404 秒	0.015 秒	26.9
「助詞　副助詞　すら　」	41	0.404 秒	0.015 秒	27.7
「助詞　副助詞　だけ　」	580	0.415 秒	0.016 秒	25.5
「助詞　副助詞　だって　」	11	0.417 秒	0.016 秒	25.7
「助詞　副助詞　でも　」	482	0.407 秒	0.017 秒	24.4
「助詞　副助詞　など　」	1555	0.417 秒	0.018 秒	23.3
「助詞　副助詞　のみ　」	24	0.412 秒	0.018 秒	23.5
「助詞　副助詞　ばかり　」	49	0.408 秒	0.016 秒	25.1
「助詞　副助詞　まで　」	6	0.408 秒	0.015 秒	26.5
「助詞　副助詞　も　」	4663	0.422 秒	0.016 秒	26.6
「助詞　接続助詞　と　」	15	0.417 秒	0.013 秒	32.3
「助詞　接続助詞　の　」	25739	0.400 秒	0.015 秒	25.9
「助詞　接続助詞　や　」	1530	0.409 秒	0.019 秒	21.8
「助詞　接続助詞　も　」	7	0.419 秒	0.017 秒	24.5

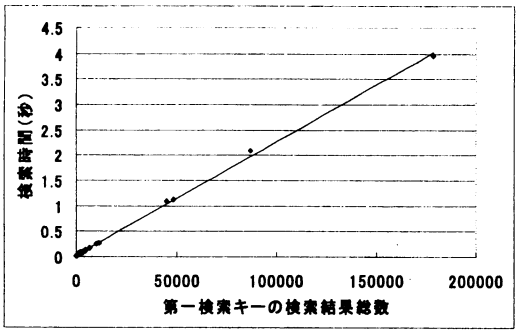
ると少し右下がりの関係であり，第二検索キーの検索結果総数に対しては少し右下がりの関係になっている．

このことから，第一検索キーの検索結果総数が大きい場合は特に本手法が有効であることがわかる．第二検索キーの検索結果総数が大きい場合は本手法の速度も少し低下し本手法と比較手法の速度の差が少し小さくなることがわかる．

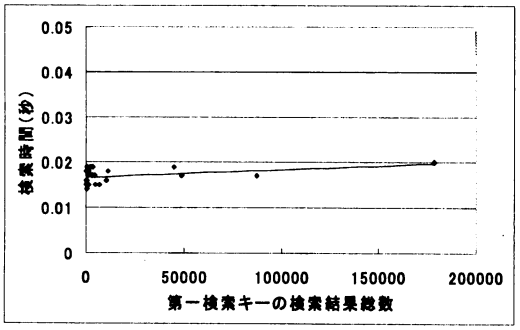
本稿の比較手法では，データの整理がしやすいように常に第一検索キーで検索した後でその検索結果に第二検索キーがあるかを調べる方法を利用した．しかし，一般には検索結果の総数の少ない方の検索キーでまず検索し，その後でその検索結果にもう一方の検索キーがあるかを調べることが行なわれる．これは，一つ目に検索するキーの検索結果の総数が少ない方が速度が速くなるためである．そこで，常に第一検索キーで検索した後でその検索結果に第二検索キーがあるかを調べるのではなく，

表 5　平均検索時間

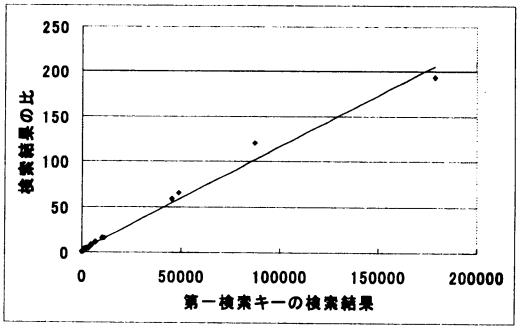
比較手法 2 の検索時間	本手法の検索時間	検索時間の比
0.112 秒	0.017 秒	6.65



(a) 比較手法の検索時間

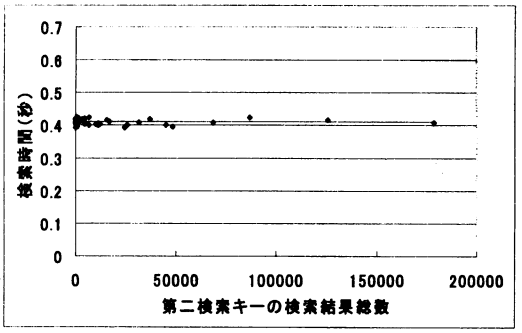


(b) 本手法の検索時間

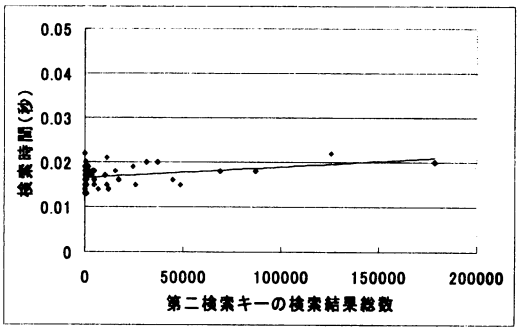


(c) 本手法に対する比較手法の検索時間の比

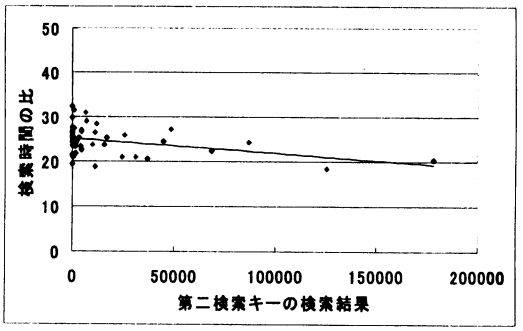
図 1　第一検索キーごとの調査



(a) 比較手法の検索時間



(b) 本手法の検索時間



(c) 本手法に対する比較手法の検索時間の比

図 2　第二検索キーごとの調査

検索結果の総数の少ない方の検索キーでまず検索し，その後でもう一方の検索キーで検索することを行なった．ここではこの手法を比較手法 2 と呼ぶこととする．この場合の実験結果を表 5 に示す．この場合，比較手法 2 の検索時間は 0.112 秒であり，比較手法の検索時間の 0.409 秒よりは速くなったがそれでも本手法の 0.018 秒よりははるかに遅く，6.2 倍遅い．

ところで，本稿で扱ったような階層構造をもったデータを扱うのに SQL などのデータベースを利用する方法も考えられる．しかし，MySQL を利用した場合，一つの形態素について検索するだけでも数秒前後時間がかかった (例えば「名詞」の検索で 3.59 秒,「動詞：基本形」の検索で 0.59 秒がかかった)．また MySQL で各形態素を一つのレコードに格納して，二つの形態素の連接を検索する場合，数分から数時間かかった (「名詞：普通名詞」と「助詞：格助詞：で」の連接の検索で 3 時間 34 分,「名詞：首相」と「助詞：接続助詞：の」の連接で 4 分かかった)．このため，SQL などを利用するよりも本稿の方法は，一つの形態素の検索でもはるかに速く (本手法が 0.0 数秒で SQL で数秒かかるのでだいたい 100 倍速い)，形態素の連接の検索にいたっては極めてはるかに速い (本手法が 0.0 数秒で SQL が数分から数時間かかるのでだいたい 6000 から 360000 倍くらい速い)．

4. 応 用 先

本稿のアルゴリズムは京都テキストコーパスに限らず，階層構造の形式をとったデータ列に対してすべて適用できる．

例えば，表 6 のように階層シソーラスの意味情報を利用した

表 6　階層シソーラスの意味情報を利用したデータ列

単語	意味レベル 3	意味レベル 2	意味レベル 1
郵政省	行政機関	役所	組織
の	なし	なし	なし
技術	技法	手段	抽象物
。	なし	なし	なし

表 7　分類語彙表の意味情報を利用したデータ列

単語	分類番号
郵政省	1271003100
の	0000000000
技術	1342105030
。	0000000000

データ列に対しても同様の処理が可能である．表 6 のデータだと，データベースには，

／S1_組織：S2_役所：S3_行政機関：S4_郵政省：S3_行政機関：S2_役所：S1_組織／
S1_なし：S2_なし：S3_なし：S4_の：S3_なし：S2_なし：S1_なし／
S1_抽象物：S2_手段：S3_技法：S4_技術：S3_技法：S2_手段：S1_抽象物／
S1_なし：S2_なし：S3_なし：S4_の：S3_なし：S2_なし：S1_なし／

のように整形して格納される．ここで，“S 数字_” は階層のレベルを意味する記号である．このデータだと，例えば，「S3_行政機関 の S2_手段」を検索したいと思えば，「S3_行政機関」「S2_手段」の上位概念を階層シソーラスの知識情報を使って補って，「：S3_行政機関：S2_役所：S1_組織／S1_なし：S2_なし：S3_なし：S4_の：S3_なし：S2_なし：S1_なし／S1_抽象物：S2_手段：」という検索キーを作成することで一回の検索で検索できるようになる．

また，表 7 のように分類語彙表 [7] の分類番号を意味情報としてもったデータ列を考えてみる．ここでは分類番号を持たない単語には “0000000000” をふっている．分類語彙表の分類番号は上位の桁が上位の意味レベルを表現し，下位の桁が下位の意味レベルを表現する(注3)．ここではこの分類語彙表の分類番号の性質を利用し，上位の桁を上位の階層，下位の桁を下位の階層ととらえ，上位の桁の数字を外側に下位の桁の数字を内側に配置し整形した以下のようなものをデータベースとして用いると，本稿のアルゴリズムを利用できる．

／1271003100：郵政省：0013001721／
0000000000：の：0000000000／
1342105030：技術：0305012431／
0000000000：。：0000000000／

(注3)：分類語彙表では各単語は 10 桁の分類番号を与えられている．この 10 桁の分類番号は 7 レベルの階層構造を示している．上位 5 レベルは分類番号の最初の 5 桁で表現され 6 レベル目は次の 2 桁，最下層のレベルは最後の 3 桁で表現されている．

例えば，分類語彙表の意味レベル「政府機関」(分類番号：1271) と単語「の」と分類語彙表の意味レベル「才能」(分類番号：13421) の連接のデータを検索した場合は，「1721／0000000000：の：0000000000／13421」が検索キーとなりこの検索キーにより一回で上記の検索を実現できる．

5. おわりに

本稿では，京大コーパスの形態素情報の品詞，品詞細分類，単語などの階層構造をもったデータの列を簡易に高速検索するアルゴリズムに記述した．実際に本手法の有効性を確かめるために比較手法と速度を比較する実験を行なった．比較手法としては，検索キーを二つに分割し，そのうちの一つの検索キーで検索しその検索結果でもう一つの検索キーがあるかを調べる方法を利用した．実験を行なった結果，本稿で提案する手法は比較手法に比べて，速いときで 194 倍速く，平均でも 24 倍速かった．また，本手法が比較手法に比べてどのような場合に速くなるかを調べたところ，一つ目の検索キーの検索結果総数が大きい場合は特に本手法が有効であることがわかった．また，二つ目の検索キーの検索結果総数が大きい場合は本手法の速度も少し低下し本手法と比較手法の速度の差が少し小さくなることがわかった．

また，本稿のアルゴリズムは他の形態の階層構造にも利用できる．本稿では，他の形態として，テキスト中の各単語の情報が，単語，下位の意味ラベル，中位の意味ラベル，上位の意味ラベルの階層構造を持ったデータを示した．また，テキスト中の各単語が，分類語彙表の分類番号を持った場合のデータも示した．これらのデータの場合に本アルゴリズムをどのように適用すればよいかも示した．

このテキストに意味的な情報を組み込む技術は WEB のテキストに対しても適用できる．WEB のテキストに対して本アルゴリズムにより意味的な情報を組み込めば，例えば，「意味ラベル：役所」と「単語：の」と「意味ラベル：技術」が連接した表現を検索するということも可能になる．このような検索が一般の WEB テキストに対して使えれば，従来の WEB 検索よりもより汎用的で便利な検索が実現できるようになる．WEB の検索は一般的に需要が大きく，本稿のアルゴリズムはそういう需要の大きな課題にも利用できるものである．

文　献

[1] 黒橋，長尾：“京都大学テキストコーパス・プロジェクト”，言語処理学会第 3 回年次大会，pp. 115–118 (1997).
[2] 村田，内山，井佐原：“階層構造データ検索システム，階層構造データ検索処理方法およびそのプログラム記録媒体”，特許広報 (特許第 3538636 号) (2004).
[3] U. Manber and G. Myers: “Suffix Array: a newmethod for on-line string searches”, SIAM Journal on Computing, **22**, 5, pp. 935–948 (1993).
[4] 長尾，森：“大規模日本語テキストの n グラム統計の作り方と語句の自動抽出”，情報処理学会 自然言語処理研究報告，No. 96-1, pp. 1–8 (1993).
[5] 山下：“用語解説「Suffix Array」”，人工知能学会誌，**15**, 6, p. 1142 (2000).
[6] 黒橋：“京都大学テキストコーパス version 2.0” (1998).
[7] 国立国語研究所：“分類語彙表”，秀英出版 (1964).