自然言語処理 85－3
（1991. 9. 19）

トライ構造による形態素辞書と共起辞書の統合法

森本勝士　　青江順一

徳島大学工学

自然言語処理で扱われる語彙は膨大な数に上る．さらにそれらの単語の組み合せによる共起関係や複合語は無限に生じる．しかし，同じ短単位の単語を含む共起語彙や複合語は多数あり，それらをそのままの形で記憶するのは記憶効率が悪い．そこで，本手法はこれらの２単語の関係を記憶する際の記憶量の節減を実現するものである．

本稿では二つの基本単語からなる共起関係の記憶を二つのトライ構造を用いて実現する方法を提案する．そして，種々の複合語に対するシミュレーション結果により本手法の有効性を実証する．

A Method for Building Morphological and Co-occurrence Dictionaries by Trie Structures

Katsushi Morimoto　　Jun-ichi Aoe

University of Tokushima

Minami-josanjima-Cho,Tokushima-Shi,770 Japan

Vocabulary used in natural language processing systems is very extensive. The number of compound and co-occurrence words which can be obtained from basic words is infinite. Furthermore, there are many compound words which contain the same basic words, so it is not efficient try to memorize compound words without a formal treatment.

This paper discusses a new method for the treatment of compound words which are made from only two basic words, and its linkage. This method aims to decrease the amount of memory required to from compound words. Memorization of compound words is made by use of two trie-structures, and the dictionary built serves for semi-static searching. Performance of this approach is shown by various simulations.

1．　はじめに

自然言語処理システムにおける単語の共起関係の利用は有用であり，形態素解析や構文解析における曖昧性の解決手段[13]，単語の概念分類，かな漢字変換の同音異義語の決定[19]，音声認識における候補文字の制約[16]などに利用されている．但し，実用システムとして有用な共起情報は膨大な数となり，それを格納する辞書の記憶倹約が必要不可欠となる．

この辞書サイズの問題は，短単位[3,5]の単語の組み合わせにより無限に生じる複合語に関しても同様に対応する．一般に，複合語を基本単語の組み合わせとしてとらえ，各基本単語ごとに分けて記憶し，少ない語数で多くの複合語を扱う研究がなされてきた．この方法では解析の際，複合語を辞書にある基本単語の組み合わせに，意味的にも正しく分割する必要がある[2]．しかし，複雑な複合語は多くの場合，得られる分割のパターンは複数あり，このことが原因となって誤った分割がなされる場合が多かった．また複合語を分割することができても，分割された各基本単語から，元の複合語の意味を合成，復元することが困難な場合がしばしばあり，そのためそのような語は複合語のまま辞書に登録せざるを得ないことも少なくなかった[4]．

そこで，本研究は二つの基本単語からなる共起単語（複合語も含める）をリンク情報と共に記憶して形態素辞書を実現する手法を提案する．本手法では，共起単語の二つの単語に対応する二つのトライ構造を利用し，二つのトライをリンクする効率的データ構造を，青江[8,9,10]の提案したダブル配列法で実現する．また，実際に３，４漢字複合語に対して実験を行い本手法の有効性を実証する．

2．　共起単語の記憶

はじめにも述べたように，共起関係は種々の品詞に対して定義されるが，ここでは説明を簡単にするために，２単語で構成される複合語を対象として，共起関係を説明する．例えば，”情報を処理する”は，２単語”情報”と”処理”による共起関係より，複合語”情報処理”を構成する．

2．1　２単語リンクによる構成

図１に本手法による複合語の記憶の概念図を示す．

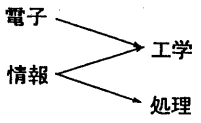


図１
電子工学，情報工学，情報処理に対する記憶の概念図

例えば「情報工学」の検索では，「情報」までマッチングが成功した後，さらに矢印で示されるリンクをたどって残りの「工学」の検索を続行できる．従来の基本単語による検索法では「情報」がマッチングした時点で一度検索を終了し，あらためて「工学」のマッチングを行う．つまり「情報工学」の検索は本手法では「情報工学」という語を検索するが，基本単語による検索法では「情報」，「工学」の二つの語の検索ということになる．このことは，本手法において，複合語の意味の復元・合成を行う必要の無いことを表しており，形態素解析のオーバーヘッドを減らすことにも役立つ．

また，新たに複合語を登録する場合，複合語をそのままの形で辞書に登録する手法と比べて，新しい複合語をそのままの形で登録しないで，既に登録してある基本単語間（未登録の場合は基本単語の登録を行う必要はあるが）にリンクを引けばよいので記憶量の節減にもなる．「情報…」や「…工学」といった語を系統だてて多数登録してやれば，多数の複合語で少数の基本単語を共有するようになるのでその効果は大きなものとなる．

2．2　トライ構造

形態素解析用の辞書は一文字ごとにマッチングを行っていくトライ構造化された辞書が使用される[4]．図２にキー集合K’＝｛電気工学#，電子工学#，情報工学#，情報処理#｝に対するトライの例を示す．トライによる検索は一文字ごとに進められるので，最悪の検索時間はキーの長さに比例し，キーの登録数には無関係であるので，非常に多くのキーを検索対象とする場合，非常に高速な検索手法となる．また，先頭の何文字かが同じ綴である語が記憶されている場合，先頭からの綴の同じ部分はそれらの語全体で共有されており記憶量を少なくしている（接頭辞の圧縮）．

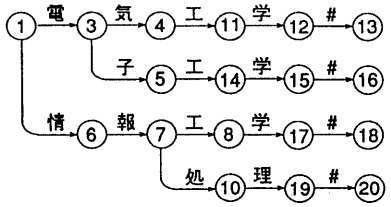


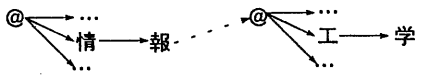
図２　K’に対するトライ

2．3　トライ構造による実現

トライは一つの根を持つ木構造を成しており，一般に根から葉へ向かって検索を進める．複合語の後半の基本単語へのリンクをトライの根に引くと，求める複

合語単語を一意に検索できない．

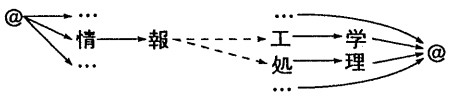
例えば，図３のように「情報」からのリンクは「工学」へのものか「処理」へのものかを区別できない上にその他の無関係な語へのリンクとも成り得てしまう．



（@はトライの根，破線は基本単語間のリンクを表す）

図３

これを解決するために，本手法では二つのトライを用い，一つは複合語の前半部の基本単語を，もう一つに後半部の基本単語（前半部との重複を許す）を記憶し，図４のように後半部のトライは葉から根の向きにマッチングを進めるようにする．



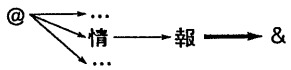
（@はトライの根，破線は基本単語間のリンクを表す）

図４　二つのトライ構造による複合語記憶の概念図

従って登録の際，後半部のトライは通常の逆向きに作る必要がある．これにより２．２で述べた接頭辞の圧縮と同様に，接尾辞の圧縮も可能となる．

２．４　レコード情報の検索方法

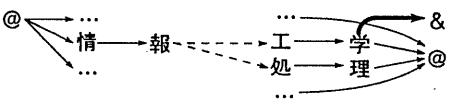
従来はキーに対応したレコード情報を格納するのに図５のように，トライ上でマッチングが成功した最後の文字に付け足す形の手法が取られてきた．



（@はトライの根，太線はレコード情報へのポインタ，&はレコード情報を表す。）

図５

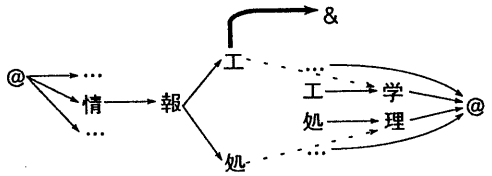
しかし，本研究の手法にこのようなレコード記憶法を用いると図６のようになり，&で示されるレコードが「情報工学」のものなのか「…工学（例えば電子工学）」のものなのか区別できない．



（@はトライの根，破線は基本単語間のリンク，太線はレコード情報へのポインタ，&はレコード情報を表す。）

図６

これを解決するために，基本単語間のリンクにレコード情報へのポインタを持たせる．また，前半の基本単語から複数の語へのリンクがある場合，どのリンクをたどれば目的の語へたどりつくのかを調べるのに計算コストがかかるので，複合語の前半部のトライに後半部が一意に決まるまでの文字を記憶しておく．この概念図を図７に示す．



（@はトライの根，破線は基本単語間のリンク，太線はレコード情報へのポインタ，&はレコード情報を表す。）

図７

”情報工学”に対するレコードには，複合語ならば”情報工学”に関する情報を，またそれが”情報に関係する工学”の共起関係ならばその情報を共有して格納できるので，一般的な共起関係もこのデータ構造で格納可能である．

３．ダブル配列によるトライの実現

本章では，トライ構造をインプリメントするためのデータ構造，ダブル配列法[8,9,10]を説明する．

３．１　データ構造

トライを形式的に説明するために，ノードrからtにラベルaのアークが引かれているならg(r,a)=tと書き，そうでないときg(r,a)=failと書く．ダブル配列法は二つの１次元配列BASEとCHECKの対（DAと呼ぶ）で，アークg(r,a)=tを次の手順で確認できる．

t←BASE[r]+a;

if(CHECK[t]=r) then 次のノード番号はｔである．

else アークは未定義．

ここで，aは記号aに対する内部表現値を意味する．また，アークの逆向き走査は次の関係で実現される．

r←CHECK[t];

if(BASE[r]+a=t) then 前のノード番号はrである．

else アークは未定義．

ダブル配列で一つのアークを確認する最大時間計算量は常にO(1)となり非常に高速となる．但し，未使用要素が容易に解放できないので，頻繁な追加・削除のあるキー集合の動的検索よりは，本手法で想定するような，あらかじめ基本的なキー集合を構築しておき，後に使用者がキーを適宜追加して拡充させるような準静的検索に適している．

３．２．　トライ検索への応用

図２を見ても判るように，トライには分岐のないパス(path)がある（例えばg(4,'工')=11，g(11,'学')=1

2, g(12,'#')=13で表されるパス）．このパスの始まるノードをセパレートノード（ＳＰノード）とし，ＳＰノードrより後のアークラベルより構成されるストリングをＳＰストリングSTR[r]（先の例では"工学#"となる）と定義し，トライからこのＳＰストリングを除いたものを圧縮トライと呼ぶ．図２に対する圧縮トライとＳＰストリングを図８に示す．

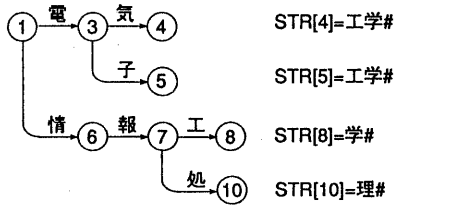


図８　K'に対する圧縮トライとＳＰストリング

文献[9]では圧縮トライをダブル配列で表現し，ＳＰストリングはTAILに記憶する．ＳＰノードrに対して，TAIL中のSTR[r]のアドレスをpとするとき，BASE[r]には-pが格納される．以下にダブル配列法の検索アルゴリズムを示す．但し，特別な記号'#'をキーの最後尾に付加した集合をＫとする．

【検索アルゴリズム】

入力キーをx#とするとき，x#∈ＫならばTRUE，x#∉ＫならばFALSEを出力する．但し，DA-SIZEはCHECK[q]≠0なる最大のインデックスとする．

［方法］

ステップ１：　rにルート1をセットする．

ステップ２：｛ダブル配列上の検索｝

```
BASE[r]<0（ＳＰノード）になるまで次を繰り返す．
INに次の入力記号をセットする；
BASE[r]+INを計算して，変数tにセットする；
if(t>DA-SIZE or CHECK[t]≠r)
then return(FALSE)
else rにtをセットする；
```

ステップ３：｛TAIL上の検索｝

```
if(INが#に一致する)
then return(TRUE)
else TAIL上の-BASE[r]の位置より#までのストリング
     を取り出し，S_TEMPにセットする；
if(残りの入力ストリングとS_TEMPが一致する)
then return(TRUE)
else return(FALSE)
```

キー集合K'に対するダブル配列を図９に示す．但し，文字と内部コードの対応は以下のとおりとする．

内部コード	1	2	3	4	5	6	7	8	9	10	11
文字	#	電	気	子	情	報	工	学	処	理	科

	1	3	4	5	6	7	8	10
BASE	1	1	-1	-4	1	1	-7	-9
CHECK	10	1	3	3	1	6	7	7

	1	2	3	4	5	6	7	8	9	10
TAIL	工	学	#	工	学	#	学	#	理	#

図９　K'に対するダブル配列

キー"電子工学#"の検索例を示す．

ステップ１：r=1

ステップ２：t=BASE[1]+'電'=3;　CHECK[3]=1;

r=3;　｛'電'のマッチング完了｝

t=BASE[3]+'子'=5;　CHECK[5]=3;

r=5;　｛'子'のマッチング完了｝

ステップ３：INは'子''で''#'に一致しない．

-BASE[5]=4なるTAILのインデックスよりS_TEMPは"工学#"とセットされる．

残りの入力ストリング"工学#"とS_TEMP"工学#"が一致する．

以上よりキー"電子工学#"∈K'が判定される．

４．　複合語検索への応用

３．の手法は同じ語尾を持つキーが多数ある場合，TAILの記憶は冗長になるので，本章では，この改良を圧縮トライとＳＰストリングを二組のダブル配列（P_DAとS_DAと呼ぶ）で表現する手法として提案する．

４．１　２組のダブル配列の利用

圧縮トライのP_DAへの格納は３．と同じ（ＳＰノードのBASEの要素のみ異なる）だが，ＳＰストリングのS_DAへの格納は，各ＳＰストリングの文字列を逆順にしたストリングの集合K_{rev}に対するトライ（圧縮トライでない）に対して行う．以後，BASE,CHECK,DA-SIZEはそれぞれP_BASEとS_BASE，P_CHECKとS_CHECK，P_DA-SIZEとS_DA-SIZEを指すものとする．

しかし，トライを二つのダブル配列で実現することでS_DAのどこからＳＰストリングをアクセスするかという問題が生じる．従来のダブル配列法のように，STR[r]をアクセスするノード番号をpとしたとき-pをP_BASE[r]に格納すると追加の際，S_DA側のノード番号が変わるとP_BASEの全ての要素をチェックしなければな

らず非効率的である．そこで本手法では，一次元配列Sを用いて以下のようにP_DAとS_DAをリンクする手法を提案する．

［定義］　トライに対して，二つのダブル配列P_DAとS_DAは，次の条件を満足する．

(A-1)　アークg(r,a)=tに対して，次が成立する．

t=BASE[r]+a;　　CHECK[t]=r;

(A-2)　ＳＰノードｒに対して，次が成立する．

(1) P_BASE[r]<0

(2) STR[r]はq=S[p]なるS_DAのノード番号qからアクセスできる（但し，p=-P_BASE[r]）．

(A-1)はダブル配列の条件そのものであるが，(A-2)はＳＰノードの判定し，しかもP_DA上のＳＰストリングのアドレスを（間接的に）指定する条件である．また，ダブル配列の大きさDA-SIZEをCHECK[q]≠0なる最大のインデックスとし，S-SIZEをS[i]≠0となる最大のインデックスとするとき，CHECK[1]にDA-SIZEを，S[1]にS-SIZEを格納する．

４．２　検索アルゴリズム

【検索アルゴリズム】

入力キーをx#とするとき，x#∈ＫならばTRUE，x#∉ＫならばFALSEを出力する．

［方法］

ステップ１：　rにルート1をセットする．

ステップ２：　P_BASE[r]<0（ＳＰノード）になるまで次を繰り返す．

```
INに次の入力記号をセットする；
P_BASE[r]+INを計算して，変数tにセットする；
if(t>P_DA-SIZE or P_CHECK[t]≠r)
then return(FALSE)
else begin
        rを次のノード番号tに変更する；
        セパレートノード候補tをSPNにセットする；
        { SPNは追加・削除アルゴリズムで使用 }
     end
```

ステップ３：{ 後半のトライの検索準備 }

```
if(INが#に一致する)then return(TRUE)
else { 二つのトライのリンク }
        tにS[-P_BASE[r]]をセットする；
```

ステップ４：{ 後半のトライの検索 }

```
  if(tが1に一致する) then return(FALSE)
  次を繰り返し，tが1になればreturn(TRUE).
      rにS_CHECK[t]をセットする
      if(r>S_DA-SIZE or t≠S_BASE[r]+IN)
      then return(FALSE)
        else t←r;
      INに次の入力記号をセットする；
```

キー集合Ｋ’に対する二つトライと対応するダブル配列をそれぞれ図１０と図１１に示す．

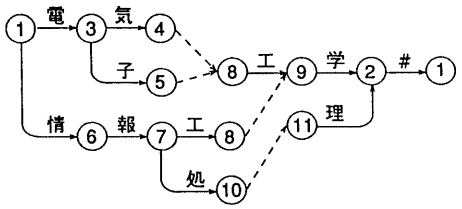


図１０　Ｋ’に対する二つのトライ

	1	3	4	5	6	7	8	10
P_BASE	1	1	-2	-2	1	1	-3	-4
P_CHECK	10	1	3	3	1	6	7	7

	1	2	3	4
S	4	8	9	11

	1	2	8	9	11
S_BASE	1	1	0	1	0
S_CHECK	11	1	9	2	2

図１１　Ｋ’に対する二つのダブル配列

キー"情報工学#"の検索例を示す．

```
ステップ１：r=1
ステップ２：t=P_BASE[1]+'情'=6;  P_CHECK[6]=1;
      r=6;  SPN=6;    { '情'のマッチング完了 }
          t=P_BASE[6]+'報'=7;  P_CHECK[7]=6;
      r=7;  SPN=7;    { '報'のマッチング完了 }
          t=P_BASE[7]+'工'=8;  P_CHECK[8]=7;
      r=8;  SPN=8;    { '工'のマッチング完了 }
ステップ３：INは'工'で'#'に一致しない．
          -P_BASE[8]=3よりS[3]=9をtにセットする．
ステップ４：t=9;  r=S_CHECK[9]=2;
    t=S_BASE[2]+'学';  { '学'のマッチング完了 }
          t=2;  r=S_CHECK[2]=1;
    t=S_BASE[1]+'#';  { '#'のマッチング完了 }
```

以上よりt=1となりキー"情報工学#"∈Ｋ’が判かる．

４．３　追加アルゴリズム

P_DA,S_DAの初期状況はそれぞれCHECK[1]とBASE[1]が共に1であって，DA-SIZEを超える要素は0とするとき，追加アルゴリズムは検索アルゴリズムreturn(FALSE)を次のように変更することで得られる．

a)ステップ２のreturn(FALSE)の変更：

```
begin  { 残りの入力ストリングをaxとする }
      A_INSERT(r, ax);  return(FALSE)
end;
```

b)ステップ４のreturn(FALSE)（２ヶ所）の変更：

```
begin { SPNから検索が失敗するまでの文字列をx,
残りの入力ストリングをxcy;対応するS_DAのストリン
グをxdzとする}
        B_INSERT(x, cy, dz);  return(FALSE)
end;
```

主要な手続きと関数を以下に説明する．

P_X_CHECK(LIST):c∈LISTなるすべてのcがP_CHECK[q+c]=0を満足する最小のインデックスqを返す．

GET_S_ADD(p):S[i]=pなるiがあればそれを返し，なければS-SIZEを1増やし，S[S-SIZE]にpをセットしてS-SIZEを返す関数．

S_DA_ADD(tail):文字列tailを逆順にしたものをS_DAに記憶し，最後の文字（逆順にされる前のtailの先頭の文字）に対するノード番号を返す関数．

```
procedure A_INSERT(r, ax);
{ ダブル配列P_DA上でのミスマッチの場合}
 begin
   t←P_BASE[r]+a;
   if(P_CHECK[t]が空いていない) then
      ノードrとh=P_CHECK[t]なるノードhについて，
      それぞれのノードからでるアークの数が少ない
      方のノードのノード番号を未使用のものに変更
      する；
   INS_STR(r,ax);
 end;

procedure B_INSERT(x, cy, dz);
{ ダブル配列S_DA上でのミスマッチの場合 }
 begin
   r←SPN;
   loop{ x=b1,b2,…,bk }
     begin  { 連続アークの定義 }
       P_BASE[r]←P_X_CHECK({b1});
       P_CHECK[P_BASE[r]+b1]←r;
       r←P_BASE[r]+b1;
     end;
   P_BASE[r]←P_X_CHECK({c,d});
   { 新しいP_BASE[r]の決定 }
   INS_STR(r,dz);  { S_DAへの再格納 }
   INS_STR(r,cy);  { S_DAへの新規格納 }
 end;

procedure INS_STR(r,ey);
{ 残り文字列のS_DAへの格納とP_DAへのリンク }
 begin
   t←P_BASE[r]+e; P_CHECK[t]←r; {g(r,e)=tを定義}
   S_DAにyを登録し，その検索位置をt'にセットする；
   P_BASE[t]←(-1)*GET_S-ADD(t'); { P_DAとS_DAのリ
ンク }
 end;
```

S_DAへの追加

文字列tailを逆順にし，S_DAへ登録，追加を行い，逆順にされたtailの最後の文字に対するノード番号を返すアルゴリズムを以下のようにまとめておく．

【S_DAへのtailの登録アルゴリズム】

［方法］

ステップ１：入力キーtailの反転を行う．

ステップ２：INに最初の入力記号をセット

ステップ３：r，tにルート１をセット

ステップ４：次を繰り返し，INが空記号列εになれば tを返す．

```
    if(S_BASE[r]が0である)then
       S_BASE[r]の要素をS_CHECK[q+IN]=0
       を満たす最小のインデックスqに変更;
    t←S_BASE[r]+IN;
    if(t>S_DA-SIZE or S_CHECK[t]≠r)
    then return(BRANCH_INSERT(r, ax))
         { 残りの入力ストリングをaxとする }
    else
      begin
        r←t;
        INに次の入力記号をセットする;
      end
```

このアルゴリズムを関数S_DA_ADD(tail)とし，関連する関数を以下に説明する．

S_X_CHECK(LIST)は，S_DAに対するP_X_CHECKと同じ働きをする関数．

S_CHANGE(old,new):S[i]=oldなるS[i]をS[i]=newに変更する．

```
function BRANCH_INSERT(r, ax);
{ ダブル配列上でキーがミスマッチしたときにキーを
追加し，axの最後の文字に対するノード番号を返す }
    begin
     t←S_BASE[r]+a;
     if(S_CHECK[t]が空いていない) then
       begin
         ノードrとh=P_CHECK[t]なるノードhについ
         て，それぞれのノードからでるアークの数
         が少ない方のノードのノード番号を未使用
         のものに変更する；
```

```
        変更されたノード番号を持つSの要素を変更
        後のノード番号に変更する；
      end
    t←S_BASE[r]+a;
    S_CHECK[t]←r;
    return(STRING_INSERT(t, x));
  end;

function STRING_INSERT(r, ax);
{ axをノードrに追加し，axの最後の文字に対するノード番号を返す }
 begin
   if(aがεに一致する) then return(r);
   if(S_CHECK[S_BASE[r]+a]が空いていない)
     then S_BASE[r]←S_X_CHECK[{a}];
   t←S_BASE[r]+a;
   S_CHECK[t]←r;
   return(STRING_INSERT(t, x))
 end;
```

ここで図１１で示される二つのダブル配列にキー"情報科学#"を登録する場合を考える．「情報」の検索は先のキー"情報工学#"と同じなので，その次から説明する．

t=P_BASE[7]+'科'=12;
P_CHECK[12]=0(≠7)となり
A_INSERT(7,"科学#")が実行される．

A_INSERTでは

t=P_BASE[7]+'科'=12;
P_CHECK[12]=0となるのでINS_SRT(7,"科学#")が実行される．

INS_STRでは

t=P_BASE[7]+'科'=12;　P_CHECK[12]=7;
{ g(7,'科')=12なるアークの定義 }
S_DA_ADD("学#")=9となる（詳細は後で説明する）のでt'に9をセットする．
GET_S_ADD(9)=3よりP_BASE[12]の要素を-3とする．

これでキー"情報科学#"が登録されたことになる．最後に，先程のS_DA_ADDでは

ステップ１：逆順にされたtailは"#学"となる．
ステップ２，３：IN←'#',r←1,t←1;
ステップ４：

```
S_BASE[1]=1;  t=S_BASE[1]+'#'=2;  S_CHECK[2]=1;
          r=2;  { '#'のマッチング完了 }
S_BASE[2]=1;  t=S_BASE[2]+'学'=9;  S_CHECK[9]=2;
          r=9;  { '学'のマッチング完了 }
```

ここでINがεとなるので9を返す．

４．４　削除アルゴリズム

削除アルゴリズムは，検索アルゴリズムのreturn(TRUE)を次のように変更することで得られる．

```
begin
  P_BASE[SPN]←0;
  P_CHECK[SPN]←0;
  return(TRUE)
end
```

この手順は，ＳＰノードSPNに至るアークをダブル配列P_DA上から削除することを意味する．

５．評価

５．１　理論的評価

二つのダブル配列についてそれぞれ，入力記号の総数をe，トライのノード数をn，ダブル配列の大きさをn+m，Sの大きさをaとして説明する．

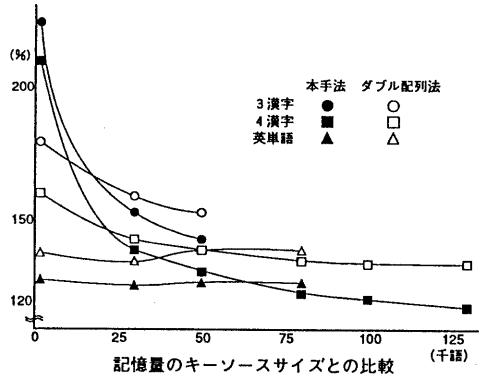
追加の計算量は関数BRANCH_INSERTに依存する．ここでダブル配列へ追加の最悪の時間計算量は，文献[9]よりＯ（$m \cdot e + e^2$）となり，この関数で実行される表Sの検索の計算量はＯ(a･e)となるので，Ｏ（$m \cdot e + a \cdot e + e^2$）が得られる．

５．２　具体的評価

従来のダブル配列法と本研究で提案した記憶法とでキー登録を行い記憶量と，追加，検索に要する時間の比較を行った結果を以下に示す．双方とも記憶量はキーの表記のみ（#を含む）ものであり，レコード情報の記憶は行っていない．検索，追加時間は最初に30,000語を登録しておき，登録済みの30,000語の検索と，未登録の300語の追加の１語あたりに要する時間である．

本手法の構成システムは約1,500行のＣ言語で記述されておりSparc Station2上で稼働している．実験結果より検索時間の高速性が判る．追加時間は，本研究では準静的検索な辞書を対象としていることを考えると実用的な値となっている．記憶量については従来のダブル配列法と比較した場合，３漢字，４漢字のキー及び英単語のキーについては本手法の有効性が現れている．長いキーの集合を対象とすると記憶量，追加時間のコストは悪化するがその理由は以下のように考えられる．長いキーを登録した場合，前半部のダブル配列P_DAにはには圧縮されたトライを格納するのでノード数はさほど多くならないが，後半部のダブル配列S_DAに格納すべきＳＰストリングの長さが増加する．S_DAはトライを逆向きにたどる性格上，トライ（圧縮トライではない）をそのまま記憶しているのでノード数が爆発的に増える．前半部のダブル配列の記憶量はどちらの手法も同じだが，ＳＰストリングの集合をストリングとして持つ従来の手法に比べ，本手法ではS_DAな

るダブル配列で持つので長いキーを記憶するのには向いていない．



記憶量のキーソースサイズとの比較

表1　追加，検索に要する時間

キー集合	3漢字	4漢字	英単語
検索時間	0.08	0.08	0.06
（ミリ秒）	0.08	0.10	0.06
追加時間	56.7	80.0	173.3
（ミリ秒）	73.3	136.0	176.7

（上段：従来のダブル配列法　下段：本手法）

6．　まとめ

以上，本研究では形態素辞書に複合語も含めた共起関係を効率的に実現するために，二つのトライ構造を用いた辞書構造を提案し，それをダブル配列で実現する手法を提案した．また，二つの短単位の基本単語からなる複合語の共起関係に対しては，シミュレーション結果により有効性が実証された．

但し，2単語間の一般的共起関係に対する実験結果や，三つ以上の基本単語からなる共起関係の検索法に付いては今後更に検討を進める予定である．

謝辞　本研究の一部は，文部省科学研究費一般研究(C)，電気通信財団，民間企業共同研究（（株）ジャストシステム）の援助を受けた．

参　考　文　献

[1]長尾真　辻井潤一　山上明　建部周二：「国語辞書の記憶と日本語文の自動分割」情報処理，Vol19，No6(1978-06)

[2]宮崎正弘：「係り受け解析を用いた複合語の自動分割法」情処学論，25，6，pp.970-979(1984-11)

[3]中野洋　野村雅昭：「日本語の形態素分析」情報処理，Vol.20，No.10(1989-10)

[4]田中穂積：「自然言語解析の基礎」産業図書，1989

[5]武田浩一　藤崎哲之助：「統計的手法を用いた漢字複合語の短単位分割」情処自然言語処理研究会資料48-2(1985)

[6]吉村賢治　渡辺美津乃　津田健蔵　首藤公昭：「日本語文の形態素解析アルゴリズム」情処研報，86-JDP-8-1，(1986)

[7]石崎雅人：「2名詞漢字複合名詞内の意味の多義解消アルゴリズム」情処論，31，11，pp.1696-1699(1990-11)

[8]青江順一：「ダブル配列による有限機械の効率的インプリメンテーション」信学論(D)，J70-D，4，pp.653-662(1987-04)

[9]青江順一：「ダブル配列による高速ディジタル検索アルゴリズム」信学論(D)，J71-D，9，pp.1592-1600(1988-09)

[10]青江順一：「自然言語辞書の検索－ダブル配列による高速検索アルゴリズム」bit，21，6，pp.776-784(1990-5)

[11]中嶋章子　杉村領一：「ＴＲＩＥ構造とグラフスタックを用いた日本語形態素解析」情処第３９回全国大会，1F-4，pp.589-590(1989)

[12]田中康仁　吉田将：「自然言語の基礎知識獲得－語と語の関係について，朝日新聞記事データの分析－'が'について」情処研報，88-NL-69-3，(1988)

[13]高橋直人　板橋秀一：「単語共起頻度を利用した形態素解析」情処研報88-NL-69-5，(1988)

[14]松本一則　黒岩真吾　鈴木雅実　榊博史：「共起関係データの蓄積と利用のための基礎実験」情処研報89-NL-73-8，(1989)

[15]松川智義　中村順一　長尾真：「共起関係に注目したＤＭ分解と確率的推定による単語のクラスタリング」情処研報89-NL-72-8，(1989)

[16]淵潟健一　荒木健治　宮永喜一　栃内香次「表層より得られる単語共起関係とその評価」情処研報90-NL-80-2，(1990)

[17]松本一則　榊博史　野垣内出：「単語間の共起関係情報の一利用手法とその効果」1991年電子情報通信学会春季全国大会，SD-5-3(1991)

[18]宮崎正弘　池原悟　横尾昭男：「単語結合型辞書引きを用いた日英機械翻訳用辞書の構成」情処研報，91-NL-84-12，(1991)

[19]長崎秀紀　唐崎幸弘　宮間俊雄　宮本恒雄　岩木雅汎：「表層格を用いた共起変換の一実用方法」情処第39回全国大会，4F7，pp.632-633，(1989)