大容量ファイルの索引*

池　野　信　一**

1. はしがき

大量の情報のなかから，必要なものだけをいかにして能率よくとり出すかということは，現代生活のあらゆる面において，重要な問題となっている．このために，ファイリングシステムその他，情報を能率よく整理・保管する色々な手段が活用されているが，電子計算機の記憶装置の進歩に伴い，情報の検索を自動化する試みが各所で行なわれるようになった．

一口に情報検索といっても，これには色々の種類と段階がある．複雑なものでは，たとえば，世界中で出版される学術論文を記憶させておき，ある特定の問題に関する世界の研究状勢の要約を要求する場合のように，単なる表の内容をそのままとり出すのでなく，ある種の推論を必要とするものもある．このような高級な操作を行なうためには，まず，項目の内容を表わす見出しをどのようにつけるかという問題から始まって，色々困難な問題があり，まだ十分実用の段階に達してはいないが，その将来が期待されている[5),9)]．

本稿では，推論を必要とするような高級な処理の問題は他にゆずり，見出しのついた項目をどのような形に記憶させ，どのようにして索引するのがよいかという基本的な問題についてのべていくことにしよう．

2. 見出しと内容

索引のために与えられる入力を要求という．個々の要求に関係した情報の単位を項目と呼び，項目の集まりをファイルと呼ぶ．項目は見出しと内容とからなり立っている．ファイルを索引するというのは，与えられた要求に一致する見出しをさがし，この見出しに対応する項目の内容をとり出すことである．

見出しは（したがって要求も）普通，数字や文字のならんだもので表わされるが，計算機のなかでは，数値に変換されるから，ここでは始めから数値，しかも整数値で表わされているものとしよう．見出しの最大値から最小値までの区間をその変域と呼び，変域内の整数の個数をその幅と呼ぶことにする．

見出しはすべて互に異るものとすれば，一つの変域内には，その幅に等しい個数の見出しまでとることができる．いいかえれば，見出しの密度は最大1までである．一方，見出しの桁数は，場合場合により様々で，見出しを互に区別するのに必要な桁数よりかなり長い場合もあり，また，一つのファイルの中でも長短の見出しがまじっていることもある．

内容の桁数もまた様々で，多数の語にわたる長いものから，短いものでは1ビットのものまである．とくに1ビットの場合は，1か0かの区別を，ファイル中に見出しがあるかないかで表わすことにすれば，内容の桁数は0ですむことになる．一般には，内容の長さは数語にわたり，しかも長さがまちまちのことが多いが，簡単のため，長さはすべて一定とし，1項目の記憶の単位を1語ということにする．したがって本稿でいう1語は，実際には数語のブロックを表わすこともあるわけである．

3. 記憶装置の形式と索引方式

記憶装置には，大きくわけて，

（i） 逐次形

（ii） 即時呼び出し形（ランダムアクセス形）

の二つの形がある．前者は磁気テープのように，端から順次に読んでさがしていかねばならないもの，後者はコア記憶装置のように，任意の番地から即時に呼び出せるものである．ドラム，ディスクなども後者に入れてよいであろう．

要求の一つ一つに対して，即時に応答が要求される実時間処理方式に対しては，当然即時呼び出し形が適しているが，要求をいくつかまとめて適当な前処理をしてから索引することのゆるされる束処理方式では，逐次形を使用することもできる．本稿では，即時呼び出し形による実時間処理の場合だけを考えることにする．

4. ファイル索引に要求される条件

基本的な条件は次の三つである．

（i） 索引に要する時間のなるべく少ないこと．

（ii） 必要な記憶容量のなるべく少ないこと．

---

* Searching of Large File, by Nobuichi Ikeno (Electrical Communication Laboratory)

** 電気通信研究所

(iii) 追加項目の挿入と，不要項目の削除が容易に行なえること．

これらは両立しない関係を含んでいるから，場合により，重点をどこにおくか，適当に選ぶ必要がある．

5. 番地指定による索引と内容指定による索引

普通の記憶装置では，番地を指定することによって，その番地の1語の内容が読み出される構造になっている．そこで，見出しをそのまま番地と考え，見出しが100なら100番地に，101なら101番地にというように各項目を記憶させておけば，要求に対して，直ちにその番地を指定して読み出すことができる．ところで，この方法では，見出しの密度がそのまま記憶装置の利用率を表わすことになり，密度が低いと利用されない語が多数残され，能率が悪くなる．

この方法の特徴は，番地がそのまま見出しであるから，記憶装置のなかに見出しを蓄わえる必要がないことと，また，読出しがきわめて速くできることである．密度が低い場合には，後述の伊吹の方法を使えば，密度に無関係に，かなりの利用率にすることができる．

見出しと番地の対応がばらばらになっているファイルでは，各語に，見出しを記憶させておき，要求が出されたときには，これと等しい見出しをもつ語を読み出さなければならない．つまり内容の指定による読み出しが必要である．これをハードウェアで実現したアソシアチブ記憶装置を使えば，簡単に，しかも高速に読み出しができるが，普通使われているのは，番地指定により読み出される記憶装置であるから，これによって内容指定による索引を能率よく行なうためには，以下のべるような色々な工夫が必要になるのである．

6. Table Look Up 法

項目をそのまま記憶装置のなかへならべて入れておいて，片っぱしからその見出しを読み出していき，要求と一致するものが，見つかるまでさがす方法である．これは，記憶装置をすき間なく使うことができるが，索引時間がかなり長くなる欠点をもっている．計算機によっては，table look up を行なうマクロ命令をもち，プログラムでやるよりかなり速くできるものもあるが，それでも，桁違いに速くできるわけではない．

いま，項目数をNとすれば，要求項目を見つけるまでに，平均 $(N+1)/2$ 回の探索が必要である．要求されるものがファイルに入っていないこともあるような場合には，入っていないことを知るために，ファイルの最後までさがさなければならないから，平均探索回数はもっとふえることになる．このような場合には，あらかじめファイルを項目の見出しによって分類し，少さい方（または大きい方）から順にならべておけば要求値より大きいところまでさがす必要がなくなり，平均回数は $(N+1)/2$ のままとなる．

以上の考察は，各項目の利用頻度は一様であると暗黙のうちに仮定した．項目の利用頻度が一様でない場合には，頻度の高いものから順にならべることにより，比較回数を減らすことができる．次に頻度分布のいくつかの形について，平均探索回数を求める式をあげておこう[1]．

r 番目の項目が利用される確率を $g(r)$ とすれば，平均探索回数 $\bar{h}$ は

$$\bar{h}=\sum_{r=1}^{N} r g(r), \quad \left(\sum_{r=1}^{N} g(r)=1\right) \qquad (1)$$

で表わされる．そこで，

(i) 一様分布の場合は $g(r)=1/N$ であるから，
$\bar{h}=(N+1)/2$

(ii) 双曲形分布の場合，$g(r)=c/r$
$\bar{h}\fallingdotseq N/\log_e N$

(iii) 逆2乗分布の場合　$g(r)=c/r^2$
$\bar{h}\fallingdotseq \log_e N/(1-1/N)$

(iv) 指数分布の場合　$g(r)=c\alpha^r$ $(\alpha<1)$

$$\bar{h}=\frac{1-(N+1)\alpha^N+N\alpha^{N+1}}{(1-\alpha)(1-\alpha^N)}$$

$$\to 1/(1-\alpha) \quad (N\to\infty)$$

などのようになる．

7. 分割法

見出しの大きさの順にならべられたファイルでは，まず，全体をいくつかの区間に分割し，その境界にある項目だけをしらべて，要求項目がどの区間に入っているかを定め，次にその区間の中だけをさがすようにすれば，比較回数を大きく減らすことができる．この考えを進めたものが，いわゆる2分割法である．与えられた要求は，まずファイルの中央の項目と比較され，その前半にあるか，後半にあるか，が定められる．次に，その半分の区間について同様に，中央の項目と比較され，以下これが繰返されて，区間が半分，半分と縮められる方法であって，その平均探索回数は

$$\bar{h}\fallingdotseq \log_2 N \qquad (2)$$

で表わされる．これを前節のものと比較すると，ほぼ

逆2乗分布と同程度であることがわかる．いいかえれば，逆2乗分布よりも一様分布に近いような分布の場合には，頻度による分類よりは，見出しの大きさによる分類をして2分割法を使った方がよいということになる．

ドラム，ディスクなどのように，ブロックの選択がランダムアクセスで，ブロック内は逐次式であるものでは，ブロックの選択に分割法を，ブロック内は table look up 法を用いるのがよい．

以上，索引の能率だけを考えて来たが，項目の追加挿入と削除とを考えると，分類されたファイルはきわめて手数がかかる．カードのようなものであれば，その中に挿入したり削除したりすることは何でもないことであるが，記憶装置で，ある語以下のすべての語を，1語ずつ繰り下げるということは，読み出し，書き込みをその語数だけ繰り返えさなければならないのである．このような，ならべかえの手数を減らすには，別に小さな補助ファイルを設け，新項目は一応この中に入れておき，要求項目が主ファイルに見当らないときはこの補助ファイルをさがすようにし，適当個数たまったなら，補助ファイルだけをまず分類し，次にこれを主ファイルに挿入するようにすればよい．

8. 直線内挿法

見出し密度が小さいとき，これをそのまま番地に対応させたのでは，記憶装置のむだが多くなることは，すでにのべた．しかし，密度は小さくても，見出しのきざみ Δ が一定であれば

$$y=y_0+(x-x_0)/\Delta \qquad (3)$$

（x_0：最小見出し，y_0：最初の番地）

という線形変換で，見出し x を新しい見出し y に変換して，変域を圧縮し，密度を1にすることができる．

見出しのきざみが一定でない場合にも同様にして

$$y=y_0+[(x-x_0)/\Delta] \qquad (4)$$

（[ ] は整数部分，Δ は平均きざみ）

という圧縮をおこなえば，密度を1にすることはできるが，この対応はおそらく1対1ではなく，y の一つの値に x のいくつかの値が対応する部分がでてくるであろう．しかし，分布が一様である場合には，見出しの小さい方からならべたファイルを作れば，要求された項目は，（4）で計算される番地の近くにあるであろうから，そのまわりを少しさがせば見つけられるであろう．

9. 見出しの無作為化

見出しが，その変域の間に一様に分布しているということは，前節の方法ばかりでなく，以後，のべる方法にとっても非常に望ましい条件である．ところが，実際の見出しの分布は，一般にかなりかたよりがある．たとえば，人名簿の場合，鈴木という姓には，多くの名前が集中しているのに，木鈴という姓をもつ名前は恐らく一つもないであろう．また，名前の最後の字についても，子とか夫とかいうものはきわめて多い．このような見出しに対して，線形な圧縮とか，あるいは，見出しのなかの適当な部分から必要な桁数だけとり出すといった簡単な圧縮をすると，同一の値に多数のものが集中する危険がある．このように，いくつかの桁が共通であるような見出しは，“むれ”を作るという．良い圧縮法は，このようなむれをばらばらにし，見出しを無作為化するものでなければならない．

このためには，各桁を，そのまますてることなく，最後の値に影響をのこすようにすることが必要である．その一つの方法は，たとえば7桁を4桁に圧縮するのに，上3桁をとり，これを下4桁にたして4桁の数とするのである．（加えた結果が5桁になったら，もう一度繰返す）一般的にいえば

$$\mathrm{mod}(10^r-1)$$

の計算をして r 桁にする方法である．これは操作が簡単であるからときどき用いられる．

第2の方法は，平方してその中央付近の必要な桁数をとり出すやり方で，これは各桁の効果がいくつかの桁の範囲にひろげられるから，第1の方法より，一様化にかなり有効で，多くの場合，満足すべき結果を与える．

さらに進んだ方法は，“radix”の変換によるものである[6]．いま，見出しの2進数表示を適当に区切って，p 進法の数と考え，これを q 進法に変換し，その下位の m 桁をとる．つまり

$$A=\sum d_i p^i \quad (\mathrm{mod}\ q^m)$$

を求めるのである．すると，同じ数に変換されるものは，q^m の整数倍の差をもつものであるから，p, q を互に素に選んでおけば，これは p 進法の数としては多くの桁で異る値をもった数となる．つまり，同一のむれには属さないものである．p, q は $p=q+1$ と選ぶのが便利で，ハードウェアで実現するのも容易である．たとえば $p=11$, $q=10$ とすれば，結果は 10 進

m桁となって都合がよいであろう．

このような無作為化圧縮を行なうと，重複のおこる割合は，その密度σを平均値とするポアソン分布で決定される．つまり，一つの値にi個の見出しが重複して対応する確率は

$$p_i=\sigma^i e^{-\sigma}/i!$$

で与えられる．

10. 開放番地方式[8)]

まず，見出しを無作為化圧縮する．次に，各項目をこの圧縮見出しで指定される番地に一つずつ格納していく．もし重複が一つもなければ，全部無事に格納されて，番地指定の索引ができるわけであるが，一般には前節の終りに示したような重複があるから，ある項目を格納しようとすると，すでにその番地に別の項目が入っている事態がおこる．この場合は，この番地に続く番地を順次しらべていって，最初に発見された空き番地に格納する．このようにして全項目を逐次格納する．最後の番地まで行って空きがないときには，最初の番地にもどってくり返す．つまりサイクリックに使用させる．索引するのも同様で，要求から同一の圧縮変換で番地を計算し，その番地の内容を読み出して，その見出しが要求と一致するかどうかしらべる．一致していればよいが，していなければこれに続く番地を順次しらべて，見つかるまでさがせばよい．これからわかるように，この方式は番地指定法と違って，その見出しを省略することはできない．

この方式で問題になるのは，その平均探索回数がどれくらいになるかということである．これは，その索引の仕方からわかるように，項目数がある程度以上になれば密度だけで決定され，全項目数には関係しない．Petersonは，これを計算機によるシミュレーションで実験的に求めた**第1表**はその結果の一部である．（バケット容量1の欄）Petersonは実は最初からもっと一般的な形について考察している．それは，記憶装置全体を，バケットと名づける一定語数のブロックに分割し，バケット番号の選択を上記のように圧縮見出しで行ない，バケットの中は頭から順次さがす方式をもちいて，そのバケットになければ次のバケットに進む．バケットの容量が大きくなれば，一つのバケットに割り当てられる項目の数が平均化してくるから，溢れて次のバケットに送られるものが少なくなる．第1表に示したものは，バケットの探索回数である．これは，ドラム，ディスクなどの場合便利である．しかし完全なランダムアクセス記憶の場合は，バケットの中をさがす速度と，バケットをさがす速度は同じであるから，容量が1の場合，つまり，バケットに分割しない場合が1番能率がよい．

第1表　平均探索回数

密度% ＼ バケット容量	1	5	20
20	1.137		
40	1.366	1.015	1.000
60	1.823	1.072	1.002
80	3.223	1.280	1.043
90	5.526	1.762	1.126
100	16.914	6.870	3.319

ところで，この方法を解析し，探索バケット数の分布や平均値を求めるのはやっかいである．この問題は最近になって Schay, Tainiter らによって解かれた[10)]．この解析のために彼らは次の定理を利用した．

（定理）開放番地方式では，平均探索バケット数は，項目を格納する順序に無関係である．

証明はむずかしくない．いま，a, b がこのファイルの二つの項目で，bがaに続いて格納されたとする．a, b の順序を変えたらどうなるであろうか．もし，a, b がそれぞれもとと同じ場所にいれられるなら，回数は不変である．そこでもし，bがもとのaのあった場所に入ったとすると，aはもはやそこに入ることができないから，当然もとbの入った所に入ることになる．したがって，探索回数は，bについて減った分だけaで増すから，全体としての回数は不変である．項目のどのような入れかえも，相続く二つの項目のいれかえを組み合わせて作られるから，探索回数は格納順序に無関係であることが結論される．

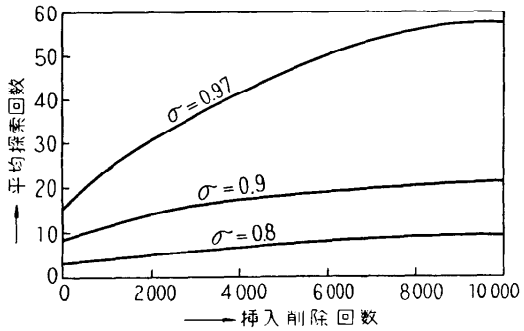
この定理によれば，項目をあらかじめ圧縮見出しに関して分類し，小さいものから順に格納しても，その平均探索回数は変らないことになる．こうしてできたファイルでは各項目は圧縮見出しの順にならんでいる．（サイクリックに使うときは，始端のところに工夫が必要である）あるいは，あらかじめ分類しなくても，空き番地をさがすとき，自分よりも後位の項目にぶつかったら，それ以後のものを1語ずつ押し下げてその前に格納するようにしても，順序どおりのファイルがえられる．ファイルの形をこのようにしたものは変形開放番地方式と呼ばれる．解析がおこなわれたのは，このようなファイルの場合であった．それにして

も，結果は，バケット容量1の場合の平均比較回数を表わす式

$$n=1+\frac{\sigma}{2(1-\sigma)}$$

のほかは簡単な式では表わせず，超越方程式や多元連立方程式を含んだ計算になり，相当な計算量となる．結果は Peterson の実験にかなりよく一致している．

以上，ファイルはあらかじめ作られたものと考え，途中での挿入，削除は考えなかった，これをゆるしたとすると，一般に，削除されるところと，挿入されるところとは一致しないから，ファイルにむだなすき間や，順序の反転がおこって，探索回数が増すはずである．**第1図**は，Peterson の実験結果の一部である．



第1図　挿入削除の影響

このように開放番地方式は，挿入するとき，記憶内容を1語ずつ順次押し下げるという必要はないが，探索回数がふえてくるという欠点がある．これを除くには，変形開放番地方式のような押し下げを行なわなければならない．

開放番地方式と似た方式として，番地の重複がおこったら，その番地には目印と，その項目を格納するための補助ファイル（これには主ファイルが書き終ったあとでその空き番地を使ってもよい）中の番地を記録しておく方式など，色々な工夫がある[8)3)]．

11. 伊吹の方法[4)]

開放番地方式では，各項目の見出しを省略することはできない．ところが，前節の最後にのべた方式では重複した項目は，別のファイルに移されるから，主ファイルの方では，各項目は重複のないものばかりとなり，見出しを省略することができる．この考えを進めたのが伊吹の方法で，重複のために，主ファイルから除かれた項目の見出しに対し，前の変換と独立な変換で無作為化し，これに従って補助ファイルに格納する．再び重複がおこれば，さらに独立な変換をおこなって，第2の補助ファイルに格納する，というようにして，すべてつきるまで補助ファイルを増設するのである．索引も同様であって，まず第1の変換で番地を計算してその内容を読み出す．これに重複のマークがついていなければ，これが求めるものである．またもしマークがついていれば，第2の変換を行なって，第2のファイルを索引する，というようにして見つかるまで探していく．したがって見出しは完全に省略できる．

この方法では，ファイル中に入れようとする項目密度が高ければ，重複の数が増し，重複マークのための余分の記憶容量が必要である．逆に密度をさげれば，使用されないスペースが増して，これまた不経済になるから，最も記憶容量をへらす密度が存在する．

入れようとする項目密度を σ とすると，重複なく格納される項目数は $e^{-\sigma}N$ で表わされる．そこで，各補助ファイルの項目密度がすべて σ であるとすれば，必要な全記憶容量は

$$(N/\sigma)\sum_{i=0}^{\infty}(1-e^{-\sigma})_i=Ne^{\sigma}/\sigma$$

したがって $\sigma=1$ のとき記憶容量は最小で，Ne 語となる．

一方，平均探索回数も容量に求められて，e^{σ} 回となり，$\sigma=1$ なら e 回となる．このように，この方法でも探索回数は，ファイルの大きさに無関係で，しかもかなり少ない．

とにかく，この方法の特徴は，見出しを記憶する必要がないことであるから，内容に比べて見出しが長いようなファイルに適している．

12. Push Down Store

中にバネが装置してあって，上から皿を積んでいくと，その分だけ下に沈み，逆に皿をとり出せばその分だけ上にあがって，いつも最上位の皿が口のところまできている装置がある．これが push down store である．この特徴は，最後に格納されたものが最初にとり出され，途中からとり出すことはできないこと，出し入れの順が一定しているから，番地をもつ必要がないことである．

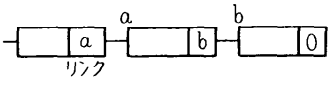
実際の記憶装置で実現するときには，新しい項目が入ったとき，それ以前のものを“押し下げる”ことはせずに，逆に出し入れ口の目印を一つ上にずらせるの

が便利である．

13. リスト構造[7]

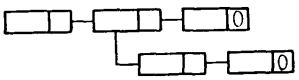
普通の記憶装置では，各語は番地の順に一つの連結関係でつながれた構造になっている．これは，固定した1次元の array の形の情報を蓄えるには便利であるが，途中に新項目を挿入したり，削除したり，あるいは，枝分れした連結関係を記憶させるには不便である．リスト構造は，これらの処理を容易にする記憶構造である．

まず，各語は，他の任意の語と連結することができるようになっている．**第2図**は3語を連結した状態を示したもので，このように，いくつかの語を連結したものをリストと呼ぶ．

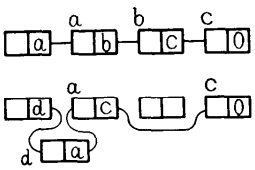


第2図　リスト構造

リスト構造といっても，実際の記憶装置のなかでは**第2図**のように線を引っぱって連結するわけにはいかないから，各語の1部にリンクを設け，ここに，次の語の所在番地を記録しておくようにし，特に最後の語のリンクは0にしてリストの終端マークとする．このようにすれば，リンクを見ながら，いもづる式にリストをたどっていくことができる．このように，連結関係がリンクで表示されるから，記憶装置内の語の順序はどうなっていてもかまわない．各語のリンクを除いた残りの部分には色々情報を蓄えることができるが，この部分で別のリストを指定することもできる．これを普通の情報と区別するために，1ビットの目印が必要である．**第3図**は，このような枝分れを含んだリストの例である．



第3図　枝分れのあるリスト



第4図　挿入削除

リスト構造では，途中に新項目を挿入したり不要項目を削除したりするのはきわめて容易である．**第4図**はその様子を示したもので，語を次々と送って，間をあけたり，つめたりする必要はなく，図のように，リンクの番地を書きかえるだけでよい．

さらに，空いた語が，各所に1個ずつばらばらにあっても，これを連結して一つのリストとして使えるから，記憶装置を能率よく使うことができる．リストの処理は，挿入，削除が多いから，このような空いた番地がどこにあるかを知る必要がある．このために，空いた語を全部つないで一つのリストとし，一種の push down store のように，必要に応じてその頭から使用し，不要になった語は，この空き語リストの頭に連結していくようにすればよい．

リスト構造では，2個所以上に同じ構造が表われたとき，これを一つにして共用することもある．このような構造をゆるした場合は，ある一つのリストについて，ある部分が不要になったとしても，他で引用されているかも知れないから，すぐに空き語リストに移すわけにはいかない．そこで，常時は，空き語リストはとり出す一方とし，全部使い果したら，処理を一時中止して，全語をしらべ，どこにも連結されていないものをさがして空き語リストを作ればよい．つまり，原料がなくなったとき，“廃品回収”にまわるのである．

リスト構造は，数値をとりあつかうよりは，記号をとりあつかうプログラムを作るときに便利である．実際このような目的のために，リスト構造をもったプログラムシステムがいくつか作られている．たとえば，IPL，LISP などは有名である．これらのシステムでは，データはもちろん，プログラムもリスト構造で作られ，その実行は解釈ルーチンによって行なわれる．

リストの中に蓄えられた情報や連結関係は，頭から順にたどっていかなければ知ることができない．枝分れしたリストを残すところなくたどるには，枝分れ点に来たとき，そのリンクを一つの push down store へ入れ，枝の方をたどり，枝が終ったら push down store からリンクをとり出してその先をたどるようにすればよい．リストの終端マークを別に設け，そのリンクに0の代りに次にさがすべき枝のリンクを入れておけば push down store を使わなくてさがすことができる．

リスト構造の一つの変形にTrie Memory がある[2]．見出しのアルファベットの数を n とすると，各語は，

n個のリンクをもち，その一つ一つが各文字に対応している．一つの見出しは一つのリストで表わされる．内容がビットで，見出しがあるか否かで表わされる場合に便利である．

14. 間接番地法

以上，色々な方法をのべてきたが，見出しに比べて内容が長く，しかも長さが一定でない場合は，内容だけのファイルを作り，別に見出しからその内容の番地を索引するファイルを作るのが便利である．分類が必要なときは後者についてだけ行なえばよい．特に記憶装置が，低速と高速の二つからなるときは，後者を高速記憶に入れれば索引が速くできて，能率があげられる．

15. むすび

最初にことわったように，話を機械的な索引に限ったために，はなはだ常識的なことばかりになってしまった．実際の場合は，電話の番号案内のファイルをとってみても，単なる索引だけではすまない問題が多い．興味をもたれた方は，さらにその方面の文献を読まれるよう希望する．

参考文献

1) A.D. Booth: The efficiency of certain methods of information retrieval. Inf. & Cont. **1**, 2 (1958)p. 159
2) E. Fredkin: Trie memory. CACM **3**, 9 (1960)
3) W.P. Heising: Method of file organization for efficient use of IBM RAMAC files. WJCC (1958) p. 194
4) 伊吹公夫：見出しを記憶しないファイルの索引法 情報処理 **3**, 4 (1962) p. 184
5) 木沢誠：言語で表した情報の機械検索をめぐって 情報処理 **2**, 5 (1961) p. 277
6) A.D. Lin: Key addressing of random access memories by radix transformation. SJCC (1963) p. 355
7) J. McCarthry: Recursive functions of symbolic expressions and their computation by machine, Part I. CACM **3**, 4 (1960) p. 184
8) W.W. Peterson: Addressing for random-acces storage. IBM J. **1**, 2 (1957) p. 130
9) D.R. Swanson: Information retrieval: state of the art. WJCC (1961)p. 239
10) M. Tainiter: Addressing for randum-access storage with multiple bucket capacties. JACM **10**, 3 (1963) p. 307

（昭和38年11月4日受付）