ファイルの蓄積と探索*

大駒誠一** 西村恕彦*** 昆野誠司**

0. はしがき

情報検索では，大量のファイルの中から必要なレコードをいかに早く正確にとり出すかということが重要な問題である．

そしてこのファイルの作り方，探し方は使用する機械，ファイルの大きさ，変更の多少，実時間処理で検索するかバッチ処理で行なうかなどによっていろいろ異なってくる．

ここでは特に情報検索用に作ったものでない通常の電子計算機を用いたファイルのつくり方と探索方法およびそれにもとづく木やグラフの利用について述べ，カードや本，冊子などのファイルを人間の目と手を使って探す方法，情報検索用の特殊機械を用いるものはあげない．

以下“ファイル”はレコードの集まりで，内部記憶装置には入りきらないくらい大きいとし，“レコード”（または“記録”）は文献とか個人の明細といった1個のまとまったデータで，著者名とか社員番号のような“キー”（または“見出し”）で識別することができるものとする．

1. ファイルのしかた

1.1 Search 方式（Document 方式）

ファイルをレコード単位に整理しておく方法で，たとえば図書館の索引カードのように一つの文献について1個のレコードを並べておく．この中から必要なものを探すときはレコードを一つ一つはじめから調べていく，これだと検索に要する手間はかかるが，ファイルの大きさは文献の数だけですむ．

1.2 Look up 方式

ファイルのレコードをキー単位に整理する方法で，たとえば一つの文献に複数個の見出し語があるときはその見出し語の数だけレコードをつくり，見出し語の順に並べる．これはキーからすぐに必要なものが探し出せるので検索時間は早いが，ファイルが膨大になる可能性がある．

1.3 電子計算機によるファイル方式

Search 方式とか Look up 方式というのは元来，人間の目と手を使って検索する際のファイル方式で，計算機を使う場合には必ずしもこのいずれかになるとはかぎらない．

ファイルの格納のしかたは記憶装置によってだいぶ異なり，カードや磁気テープのように逐次読み出ししかできないものではファイルをキーの順に並べておき，質問をたくさんためてこれをファイルと同じ順に分類してバッチ処理をするというくらいしか考えられない．

ドラムとかディスクのようなランダムアクセスの記憶装置ではファイルの各種の格納の方法，探索法が考えられ，磁気テープとちがって1ヵ所だけ書き直すこともできるし，実時間処理も可能なので，今後特殊装置を用いない情報検索用の記憶はほとんど大容量のドラムかディスクになり，磁気テープは記憶保護程度にしか使わなくなるであろう．

ランダムアクセスの記憶装置を使う場合にはファイルの大きさ，検索の頻度，追加削除の多少などをよく吟味して，ランダムアクセスの特長を生かしたリスト構造にするとか，キーを変換して格納アドレスとするような方法を考えるべきで，ファイル全体をはじから調べていくとか，頻繁にレコードを並べなおさなければならないようなファイルのしかたはさけるべきである．

2. 探索法

2.1 逐次探索法

文字どおりファイルを最初のレコードから順に見つけるべきキーと一つ一つ比較して探していく方法で，ファイルが N レコードからなっていれば平均 $(N+1)/2$ 回（キーが一様分布の場合）の比較をしなければならない．したがって，大きなファイルでは非常に時間がかかるが，この方法は順序を全く問題としないので，ランダムアクセスの記憶装置での追加削除はファイル

* Information Storage and Retrieval by Seiich Ohkoma (Keio Gijuku University), Hirohiko Nisimura (Electrotechnical Laboratory) and Seishi Konno (Keio Gijuku University)
** 慶応義塾大学
*** 通産省電気試験所

の最後の番地さえ，知っていれば非常に簡単にでき，この逐次探索法はあまり大きくないファイルで変更の頻繁に行なわれるものに向いている．

磁気テープの場合以下に述べる2分探索法とかリスト構造とかいったものは物理的に不向きなのでこの逐次探索法によるしかない．したがってファイルが磁気テープのときはファイルをキーについてアルファベット順か何かにそろえておき，質問をある程度ためてからファイルと同じ順序にそろえてまとめて処理をしてもかまわないときに向いている．磁気テープ1巻には普通数百万字の情報を収容できるので，スペースのわりに入る情報量が多く，原則的にはリールさえ増やせばいくらでも入れることができる．しかしながら磁気テープ上の情報の変更は全く不便で1ヵ所でも追加削除しようと思えば，そのテープ1巻全部別のテープに書き変えなければならない．またもし順序のそろっているテープで中間のテープに追加が必要になるとそれ以後のテープ全部書き変えなくてはならなくなることがある．

2.2　2分探索法

ファイルをキーの collating sequence の大きい順または小さい順にそろえておき，まず与えられたキーとファイルの中央のレコードのキーと比較し，その大小によって次に比較すべき上半分か下半分をきめ，さらにその半分の中央と比較するというように，範囲を次々に半分にせばめていって目的のレコードを探す方法である．このとき比較の回数は最大

$$\log_2 N$$

でファイルの順序がそろっていればこの2分探索法が最も早いし，目的のものがファイル中にないことをたしかめるのも同じ回数の比較ですむ．

しかし，この方法はレコードの追加がある場合には一つ追加するたびに平均 $N/2$ 個，一つずつずらさなければならないので追加が多い場合は良いファイル方法とはいえない．削除の場合は必ずしもその後へ下から順につめていかないでもキーに不要という目印をつけておいて探すときに見ないようにしておけばそれほど時間がかからないですみ，追加で動かすときついでに削除すればよい．

また，この変型として追加部分についてはすぐもとのファイルには加えず，追加用に別に作ったファイルに入ってきた順に並べておく．探すときにはまずもとのファイルを2分探索法で探す．もし見つからないときは追加用のファイルを頭から逐次探すようにする．もとの表からあふれたレコードの数を M とすれば $M/(N+M)$ の確率で平均 $\frac{1}{2}(M+1)$ 回追加用の表を探すことになる．

$N \gg M$ であるようなファイルでは比較の回数もそれ程増えないですみ，逐次型と同様に追加が非常に簡単である．あふれ用のファイルがある程度大きくなってきたら適当な時期にまとめてもとのファイルと一つに作り直すようにする．

この方式はレコードにキーが一つしかなくてしかも変更の少ないファイルには向いているが，キーが複数個になるともはや有効には使えない．たとえば人事ファイルで社員番号順に並んでいるものをこの方法で氏名とか学歴で検索することはできない．

3.　キーの変換

キーを何らかの方法で必要なレコードの入っている番地に変換できればほとんど外部記憶装置のアクセスタイムだけで読み出すことができ非常に効率が良い．

3.1　線形変換

キーの分布が一様分布かあるいはそれに非常に近いときは，キーをそのまま番地と考え，キーが600なら600番地に1966なら1966番地にというように各レコードを記録しておけばキーを見て直ちに必要なレコードを読み出すことができる．これだと，いわば番地がキーになっているので，キーをレコードの一部として蓄える必要がない．

また，キーの密度があらいとき，これをそのまま番地に対応させたのでは記憶装置の無駄が多くなる．しかし，この場合キーのきざみ h が一定であれば

$$y=y_0+(x-x_0)/h$$

（x_0：最小のキー，y_0：最初の番地）

という線形変換で，キー x の変域を圧縮した y を新しいキーとすれば密度を濃くすることができる．キーのきざみが一定でない場合でも h をきざみの平均またはそれよりやや小さい値ぐらいにとって同様の変換を行なえば，記憶装置の無駄は少なくなる．しかし，この場合には同じ番地に複数個のレコードが対応することがあり，その場合には y 番地の前後をいくつか探すことになる．そして，もはやキーも必ずレコードの中に入れておかなければならない．

3.2　キーの一様化

一般にキーの分布は一様分布でないのが普通でこの場合は前のような線形変換をしたのでは，かたよりが多くて効率が悪い．そこで何らかの方法でキーの分布

を一様化（randomize）し，それに比例した番地にレコードを蓄えるようにしてかたよりを少なくする方法が考えられる．

3.2.1 中央2乗法

キーを2乗してその中央部分を番地に都合がよいようにとり出してそれを格納番地とする．この他にも数学的に擬似乱数を作り出す手法はたいていつかえる（**第1図**）．

元のキー	2 乗 し た も の	交換されたキー
43429	188 6078 041	6078
23025	053 0150 625	0150
31415	098 6902 225	6902
27182	073 8861 124	8861
57721	333 1713 841	1713

第1図 事央2乗法

3.2.2 除 算 法

いまキーを K とし M を利用できる番地の範囲より大きくない数として

$$y=K(\mathrm{mod}\,M)$$

の変換を行なって y を格納番地とするのである（**第2図**）．

元のキー	変換されたキー	元のキー	変換されたキー
0230	25	4300	36
0360	32	5020	18*
0645	30	5100	16
1440	05	6600	40
1530	13	7070	18*
2200	27	7090	38

第2図 除 算 法 $M=41$

（* 同じキーになっている）

キーが n 進数の場合，M を n^k の形にすれば下 k 桁だけ残して上を切り捨てることになり除算をせずにできるが，一般にはいい結果が得られない．また M は必ずしも素数が良いとはかぎらない．この方法はキーの長さが不定だったり，かたよりが大きい場合のときに良い結果が得られることがある（第2図）．

3.2.3 おりたたみ法

前の二つの方法のように乗除算を使わず加算とシフトだけでキーを圧縮する方法である．上位半分の桁と下位半分の桁を加算して新しいキーとする（**第3図**）．

元のキー	変換されたキー	元のキー	変換されたキー
141421	562	264575	839
173205	378	282842	124
223606	829	316228	544
244949	193		

第3図 折りたたみ法

桁数の非常に長いキーのときなど除算が簡単にできない場合があるが，除算法でモジュールをとるときキーを適当に切り捨ててから除算をしたのではキーの重要な部分の情報を捨てることになるので，この方法でキーを短くしてから除算を行なえば，一応キーの全部の情報が結果に反映していることになる．

これらの方法はいずれも元のキーいかんによって必ず一様分布になるという保障はないので実際に行なう場合には除数をいろいろ変えてみるとか，いくつかの方法を組み合わせるとかしてそのシステムに最も良い方法を見つけ出さなければならない．

また，これまではキーがいずれも数字の場合だけを仮定していたが，キーがアルファベットの場合でもそれをそのまま数字と見なしたり，ゾーンビットを無視したりして計算できることもあるし，適当な対応表を使って計算可能な数字にしてしまえばキーがアルファベットでも特別な問題はおこってこない．

4. あ ふ れ

キーを一様化して，ファイルを格納する際に必ず考えておかなければならないのは，キーが重復したときの処置である．元のキーがよほど特殊でそれを利用した好妙な変換をしないかぎり，一般には変換後のキーの重復はさけられない．いまキーが重復していたら，そのレコードは次の番地に格納するものとし，そこもふさがっていたら，さらに次々とあいている番地を探していくようなファイル方式と仮定し，この場合平均何回の比較で必要なものが見つかるかを計算してみる．

ファイルには N レコード入るものとし，すでに M 個入っているものとする．

$$P=M/N$$

として，平均比較回数を H とすれば

$$H=(1-P)\times 1+P\cdot(1-P)\times 2+P^2\cdot(1-P)\times 3+P^3(1-P)\times 4+\cdots\cdots$$
$$=1/(1-P)$$

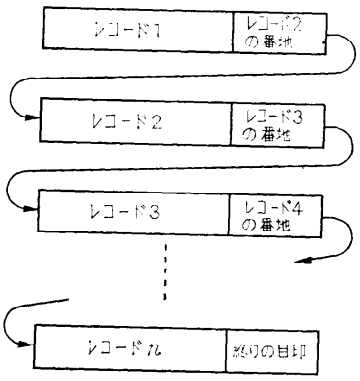
したがって，50% すでに入っていれば平均2回，75% 入っていれば平均4回比較することになる．これは他の2分法に比べても非常に少ない．しかし，記憶装置に余裕がないときには効率よい方法ではない．

また，一つのアドレスに対して複数個レコードが入るバケットとしておき，そのバケット内は逐次に調べることとして，バケットがいっぱいになったときにはじめてあふれとする方法がある．これだとあふれの回数は減り，1バケットは内部記憶に入るとすれば高速に探せるので，このバケット法もときに良い結果をあたえることがある．

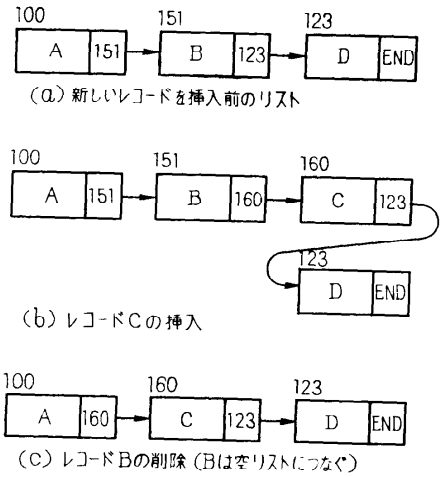
5. リスト構造

これは番地の順というものを全く無視して，各レコードの一部に次に続くレコードの番地（リンクアドレス）を入れておく方法である．この方法の特徴は記憶装置がいっぱいになるまではあふれの状態にならないこと，挿入削除がリンクアドレスを変更するだけでレコードを移動せずに簡単に行なえることである（**第4図**）．使っていない場所はすべて空リストとしてリンクでつないでおいて一つのリストとしておき，レコードを削除したときにはこの空リストに必ずつなげておく（**第5図**）．

しかし全ファイルを一つのリストにしておくのは探索の際あまり効率が良くないので，何らかの方法でファイルを分割しておくのが普通である．たとえば，アルファベットのキーの1桁でファイルを 26 に分割す



第4図　レコードをリスト構造にしたもの



第5図　レコードの挿入削除

るとか，主記憶装置の中にキーの大まかなテーブルを作り，まずこのテーブルを引いて探し始めるファイルの番地を定めるとかして目的のレコードに近ずく時間を短縮する．このような階層構造をもったファイルのしかたはリスト構造を使用していないときでも大きな効果がある．

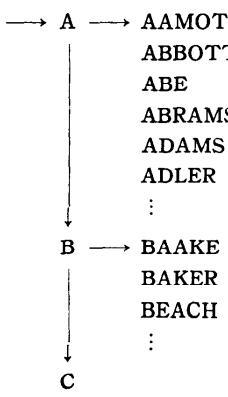
この方法のもう一つの大きな特徴は複数個のキーがあるとき，キーごとにリンクアドレスをもうけておき，関連のあるレコードをすべてつないでおく．たとえば，文献検索でファイルに関する文献はすべて同じリストとしてつないでおき，実時間処理に関する文献全部も一つのリストとしてつないでおく．こうしてファイルや実時間処理に関する文献の必要なときには何らかの方法でそのリストの一番最初の文献を見つけてくれば，あとはいもずる式に全部とり出すことが可能である．“実時間処理におけるデータのファイルのしかた”という文献があったとすれば，ファイルで引いても，実時間処理で引いても余計な所は見ないでとり出してくることができる．

6. 木とファイル

6.1 見出しの木

電話帳や辞書はファイルのある型としては典型的な例である．すなわち，一つのファイルは多くの記録の集合である．そして一つの記録は見出しと記述とからなり，見出しによって引いて記述をとりだす．このようなファイルのおもな問題はいかにしてはやく引く

か，つまり所要の見出しを有する記録に到達するかである．はやく引くのにいくつかの方法が提案されている．そのうち木（tree, list）による方法を電話帳の例で説明しよう[1]．

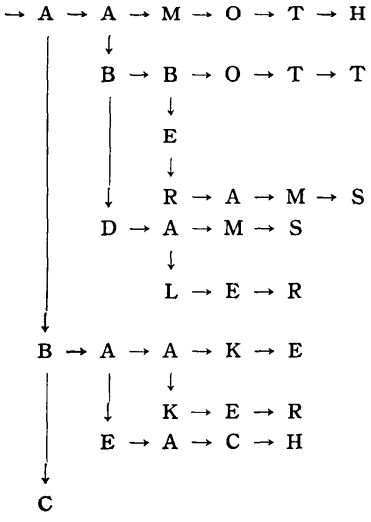


第6図 分冊

ファイルを，名前の頭文字の ABC 別の分冊に分割しておき，探したい名前の頭文字から所要の分冊を選択し，その分冊のなかだけを走査する仕方がある．

各分冊の大きさがそれでもまだかなり大きいときは名前の第2字目でさらに分割する．

われわれが電話帳や辞書を引くときには，ファイルが実際に分冊になっていないにしても，やはり大見出しなどによる分割をたよりにこれと近い手順をふむ．見出しの分割を頭文字から終りの字まですべての段階に適用すると木の形になる．



第7図 木

索表の対象となる見出しを木の形にすることの利点は，索表速度がはやくなることのほかに，不定長の見出しをあつかえること，最長一致による識別が容易なこと，構造の変更（記録の挿入や抹消）ができることなどがある．

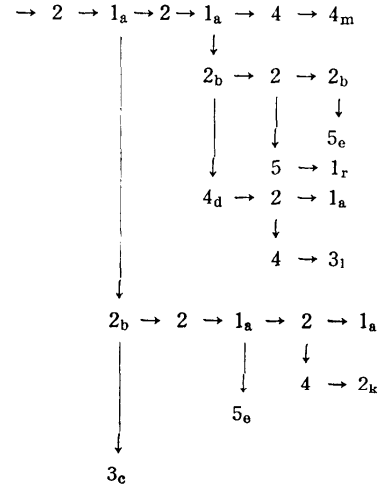
最長一致による識別（longest match identification）というのは，たとえば次のように，区切りのない文字列のなかから可能で最長の文字列を見出して区切りを入れることである．

もとの文字列 NORTHEASTWINDIS
可能な分割例 NOR|THE|AS|TWIN|DIS
最長一致分割 NORTH|EAST|WIND|IS

6.2 見出しの基数変換

ファイルの各段階を分冊化する，すなわち木を枝分かれさせる仕方にいろいろある．この議論にあたって各段階のサイズという言葉を説明しておく．たとえば1万人の記載されている電話帳を分割しないで走査するなら，第1段階のサイズは 10,000 である．この電話帳を頭文字のアルファベットで分割すると，第1段階のサイズは 26 で，第2段階の平均サイズは 400 である．日本名の電話帳でア行カ行と，行でまず分割し，そのなかでさらに段に分割すれば，第1段階は 10，第2段階は5というサイズになる．

各段階のサイズは2〜10という比較的小さな値におさえたほうが索表速度がはやくなる[2]．すなわち各段階での枝分かれの数が2〜10になるようにまとめてゆくのがよい．アルファベットの場合は枝分かれの平均サイズが 26 程度になり大きすぎるので数個ずつまとめて二段階に分けるのがよい．これを計算機でおこなうには基数変換またはビットの分割をする．すなわち多くの計算機ではアルファベットを2桁の8進数で表現できる．



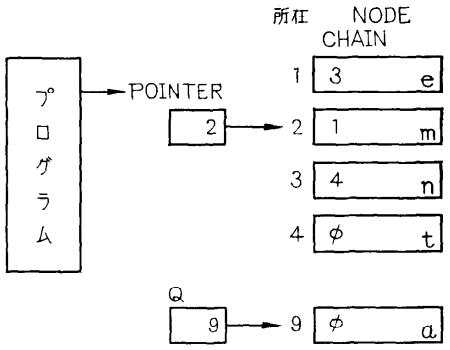
第8図 基数変換

A	21	J	41	S	62
B	22	K	42	T	63
C	23	L	43	U	64
⋮		⋮		⋮	
I	31	R	51	Z	71

この8進法の各桁を単位として見出しの木を組み立てると，索表速度がよりはやくなるものと期待される．

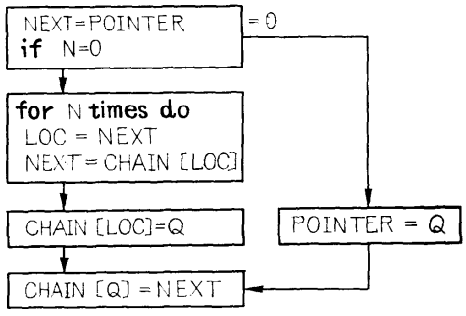
6.3 木のポインタの割付

木や列になっているデータを計算機のプログラムから参照するときに，その列の出発点の位置を指示する変数（pointer）をもちいる．列の要素（node）は，その要素に固有の値を記述する部分と，次に続く要素の所在を指示するつなぎ（chain）の部分とからなる．



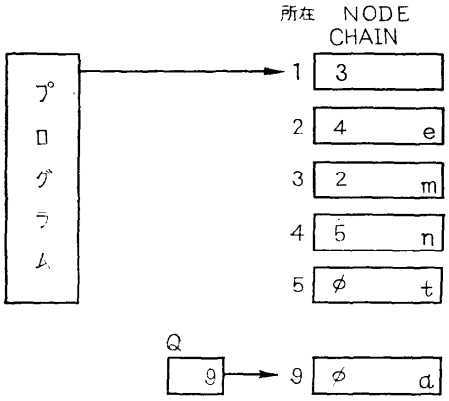
第9図　独立のポインタ

POINTER で指示される列は“ment”という値をもっていい，Qで指示される要素は“a”という値をもっている．このとき POINTER で指示された列の N 番目の要素の値のあとにQで指示された要素の値を挿入する手続きは次のとおりである．

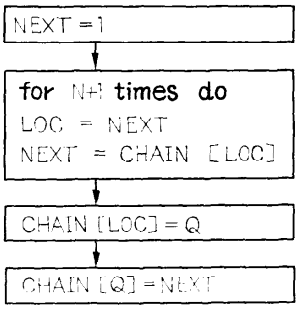


第10図　挿入手続き

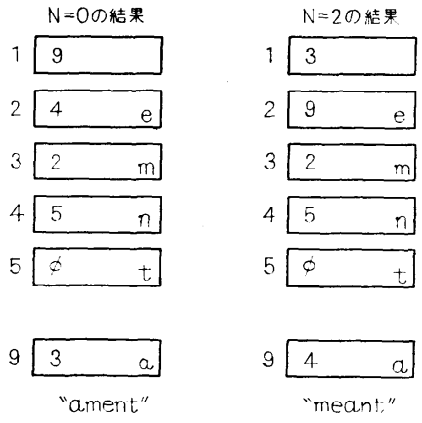
この手続きでは $N \geqq 1$，すなわち文字列の中間または後尾に値を挿入する場合には流れ図の左側を通り，$N=0$ すなわち文字列の左端に値を挿入する場合には



第11図　ベクトル中のポインタ



第12図　簡単化された手続き



第13図　挿入結果

流れ図の右側を通る．

ところが列の出発点を指示するポインタを特別な仕方で割付けておくとNの値によらずひとつの手続きですませられる[3]．

すなわちその変数を列のベクトル中の定位置（この例では添字1の位置）に固定して割付ける．この割付けによる手続きをしめす．

7. グラフの利用

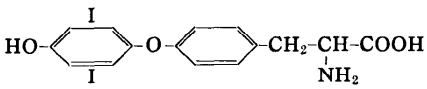
7.1 化合物の表現方法

化合物とくに有機化合物に関する情報の見出しとしてその化合物の構造式を利用する場合が多い．そこで1940年ごろから，PCS や電子計算機の発展に伴ない，大量の化合物情報の機械検索に関する研究が行なわれ，この構造式の表現方法が種々開発されてきた．化合物の構造式は本来，構成元素および元素間の結合関係を二次元的に表現したものであるが，これを一次元的な英数字の記号で表わす方法がいくつかある．その代表的なものに IUPAC 記号法がある．これは1949年に IUPAC（国際純粋・応用化学連合）のコード化，記号化，パンチカード技術委員会が国際的に通用し得る化合物記号法の条件として設定した次の11項目を満たすものとして Dyson 方式を改良し，1961年に勧告，出版したものである．

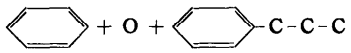
（1） 使用し易いこと
（2） 印刷・タイプが容易なこと
（3） 簡潔なこと
（4） 見てわかり易いこと
（5） 一元的な化学命名法となりうること
（6） 現行の無機化学命名法と両立すること
（7） 一義的なこと
（8） 番号のつけた方が明確で意義のあること
（9） 機械的方法，たとえばパンチカードによる操作が簡単なこと
（10） 記述的で化合物の相関性が明らかなこと
（11） どんな部分構造でも扱えること

この方法の概略を知るため，2，3の例をあげて説明する．

例 1.



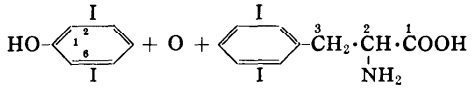
まず，この化合物を，炭素骨格とそれらを連結する連結部に分ける．すなわち



これらを符号で表わすと

$B_6 + Q + B6C_3$ となる．

Q は共有結合の酸素，B_6C_3 は3炭素側鎖をもって不飽和6員環を示す．各炭素骨格の符号に置換基の符号を挿入すると，



$B6Q_1I_{26} + Q + B6(3C_3XN_2)_1I35$

X はカルボキシル基の -OOH，N はアミノ基，I はヨード基，置換基の符号のあとの数字は炭素骨格に結合している位置を表わす．炭素側鎖はこれに結合している置換基の符号と共にかっこに入れる．側鎖の番号は別個につけ，優位の置換基（この場合は X）の位置番号を優先的に小さくする．かっこの中のはじめの数字は，環に結合している側鎖の位置番号を表わす．

置換基の符号は優先順に記載し，優位の基の位置番号をなるべく小さくする．左側の環はQの位置が1，右側の環は炭素側鎖の位置が1である．これらを結合して，優位（この場合，炭素側鎖の存在により右側）の環から記載して符号を完成する．

$B_6(3C_3X_1N_2)_1I_{35} : 4B_6Q_1I_{26}$

コロンの次は，連結部がその前の環に結合している位置番号とその符号，ストロークの後は次の環が連絡部に結合している位置番号とその符号を表わす．

これに対してフラグメントコードというものがある．これは化学構造を一定の規則に従って構成フラグメントに分解し，その各々にコード表から該当するコードを割り当て，それらのコードの組み合わせで一つの化合物を表示するものである．これの代表的なものとして，CBCC（Chemical Biological Coordination Center）で1950年出版した方式がある．これはパンチカードシステム用としてはコストが低く，よく利用されたが，コードの辞書に含まれていないフラグメントについての質問は処理できないこと，およびフラグメント間の結合関係が表わされていないため，化学構造とコードが多対1に対応することがあり，検索の際ノイズが生ずる場合があるという欠点がある．

そこで構造式をそのまま利用する方法としてグラフを利用するという方法が考えられた．

7.2 グラフについての定義

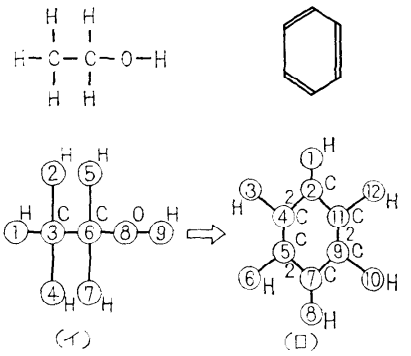
いくつかの項目と，それらの間の結合関係を平面的

に図示したものを「グラフ」と呼ぶ．すでに述べたトリー（tree）はいくつかの項目とそれの間の階層関係を平面的に図示したもので，一般には，この階層関係よりももっと入り組んだ複雑な結合関係を表わすのにグラフが用いられるが，このトリーもグラフの一種と考えてもさしつかえない．

グラフ上では，項目を小丸で表わし，これを「ノード（節）」と呼び，項目と項目の間の結合関係を実線で表わし，これを「ブランチ（枝）」と呼ぶ．項目の間の結合に方向がある場合には，矢印をつけた実線でこれを表わす．

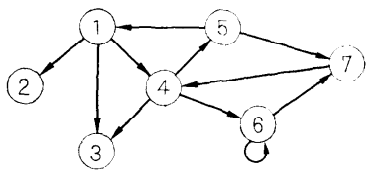
グラフでは一つの項目を一つのノードで表わしているわけであるが，この項目の内容を「ノードの値」と呼び，また結合の強さを「ブランチの値」と呼ぶことにする．ノードを表わす小丸の大きさ，およびブランチを表わす実線の長さや太さで，ノードやブランチの値を示すわけではないから，小丸の大きさや実線の長さは任意でよい．またノードの一つ一つに対して，任意の順番に続き番号を割り当てて，これをそのノードの「ノード番号」と呼ぶ．

例 1. 化学構造式をグラフで表現する場合を考えてみる．ノード番号を小丸の中に，ノードの値を小丸の近傍に，ブランチの値を給合を表わす実線の近傍に書き込むことにする．



第 14 図

例 2. 結合に方向がある場合のグラフ



第 15 図

グラフの各ノードを特徴づけるものは

（i） ノードのもつ値（項目の内容）

（ii） ノードからでているブランチの数（このノードが示す項目となんらかの関連をもつ項目の数），これをディグリーと呼ぶ

（iii） ノードの各ブランチのもつ値（項目と項目の間の結合の強さ）

（iv） そのノードからブランチによって結ばれているノードの集合

である．

7.3 グラフの照合方法

グラフGとHについて

（i） GとHが全く同じものを表わしているグラフであるか？

（ii） GがHの表わすものの一部分を表わしているか？

（iii） GとHは全く別のものを表わしているか？

といったことを判定するためにグラフの照合を行なわなければならない．

グラフのノードの特徴を表わすものは

（1） ノードの値

（2） ノードのディグリー

（3） ノードからでているブランチの値

（4） そのノードが他のどのノードとブランチで結ばれているかを示す．ブランチで結ばれているノードの部分集合

である．二つのグラフGとHについて，Gの各ノードとHの各ノードとの間に，この四つの特徴についてすべて1対1の対応がつくとき，この二つのグラフは「同形」であるという．またグラフHがグラフGから

（i） あるノードとそれに付随するブランチを取り除いたものであるか？

（ii） るあノードとそれに付随するブランチおよび，他の任意のブランチを取り除いたものであるか？

（iii） 任意のブランチだけを取り除いたものであるか？

のいずれかであるとき，グラフHはグラフGの「部分グラフ」であるという．

二つのグラフを照合する一つの方法として，次のような方法が考えられる．二つのグラフをHとGとし，それぞれのノードの集合を m, n とし，ノードの特徴の集合を C とし，C_i をもつノードの集合をそれぞれ

M^i, N とする.

（i） まずグラフHとGについてノードの数を比較する.

HとGが同形ならば $d[m]=d[n]$ でありHがGの部分グラフならば $d(m)\leqq d(n)$ となるはずである.

（ii） ある特徴 C_i についてノードの部分集合 M^i, N^i を作る. HとGが同形ならばすべての i について $d[M^i]=d[N^i]$, HがGの部分グラフならば $d[M^i]\geqq d[N^i]$ となるはずである. そこでノードの値, ブランチの値, ディグリー値, の種類の数だけ特徴をとり, それぞれについてノードの部分集合を作りそれに含まれるノードの数を比較し, 任意の i について $d[M^i]\neq d[N^i]$ ならばHとGは同形ではなく, また $d[M^i]>d[N^i]$ ならばHはGの部分グラフではないことになる. またこのようにして作りだした部分集合に含まれるノードの数がすべて1ならば, Hのノードと Gのノードの間に対応がつき, HがGと同形か部分グラフかが分る.

（iii） ノードの数が1でない部分集合がある場合にはHの各ノードがGの各ノードと対応がついているかどうか分らないから, 一つのノードが一つの部分集合にしか含まれないように, それぞれのグラフについて同じ方法でいくつかある部分集合から独立な部分集合を作る. この部分集合がまだ二つ以上のノードを含んでいる場合には, さらに別の特徴をとり, これに対するノードの部分集合を作り, これを加えて（ii）を繰り返えす.

参考文献

1) 多田和夫編: 経営科学シリーズ3「企業と情報」培風館, 昭和38年9月
2) 関根智明: 情報検索と電子計算機利用, 日本電子工業振興協会, 昭和40年9月
3) 西村恕彦: 木表現とリスト処理の算法（第7回プログラミング-シンポジウム報告集), 1966
4) E.H. Sussenguth: Use of Tree Structures for Processing Files (CACM 1963, 6-5, 272-279)
5) H.Nisimura: Allocations of Tree (IFIP Working Conference, PISA, 1966)
6) Survey of chemical notation systems; a report of the Committee of Modern Methods of Handling Chemical Information. U.S. National Academy of Science-National Research Council. xii, 467 p., 1964 (NAS-NRC publication 1150)
7) Sussenguth, E.H., Lr.,: Structure Matching in Information Processing, Information Storage and Retrieval. Report ISR-6 to the National Science Foundation, The Computation Laboratory of Harvard University (April 1964)
8) Sursenguth, E.H., Jr.: Automatic Structure-Matching Procedures, Information Storage and Retrieval. Report ISR-5 to the National Science Foundation, The Computation Laboratory of Harvard University (January 1964)

（昭和41年9月15日受付）