講　　座

データ構造 (I)

古　川　康　一*

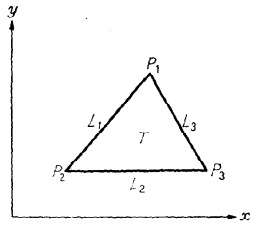
1. 序　　論

データ構造という言葉は，いろいろなところで，いろいろな意味で使われている．例えば，ALGOL とか FORTRAN, PL/I 等の高次言語中に出てくるArray, Dimension, Structure などは，データ構造と呼ばれる．また，リンクド・リストや，LISP などに出てくるリスト構造もデータ構造とよばれる．現在，データ構造が特に問題になっている分野は，CAD および，それに関連しているコンピュータ・グラフィックスなどであり，逆にこれらの研究にともなって，大きくクローズ・アップされた問題であるともいえよう．それらの分野では，データ間に存在する複雑な関係の表現，およびそれを計算機の中に実現する仕組をデータ構造と呼んでいる[1]-[3]．

このように，データ構造という言葉は，使われるところにより異った意味をもち，また人によって使う意味が異なるおそれもあるので，ここでは，まずその意味を明確に与えてから話を進めることにしたい．読者の理解を容易にするために，コンピュータ・グラフィックスの簡単な例をとり上げて考える．

いま，**図1 (a)** のような3角形をグラフィック端末（例えば CRT＋ライトペン）上に表示させて，それに対してある種の操作をほどこすことを考える．3角形を表示するには，3頂点の座標を与え，それが3角形をなすという情報を与えればよい．これを計算機のプログラムで実現するには，例えば，**図1 (b)** のような各点の x, y 座標の配列をデータとして与え，それが3角形の頂点であるという情報は，それらの各点を結ぶ線を発生させる procedure として与えるという方法が考えられる．

ところがこのようなプログラムでは，各辺の記述が（データとして）なされていないので，各辺を指定するには，その両端を与えなければならない．しかるに問題によっては，各辺をデータの形で表わしておいた方がよい場合も多い．例えば輸送問題のように，各辺が固有の容量をデータとして持つような場合，辺自身をデータの形で表わさなければ，この容量のデータは宙に浮いてしまう．



(a)

P_1	x_1, y_1
P_2	x_2, y_2
P_3	x_3, y_3

(b)

L_1	P_1, P_2
L_2	P_2, P_3
L_3	P_3, P_1

(c)

T	L_1, L_2, L_3

(d)

図1　3角形とそれを表わすデータ

これは，上のプログラムの例でいえば，procedure 内にふくまれていた情報をデータとして分離することを意味している．するとデータは，点の配列に，**図1 (c)** のような線分 L_1～L_3 の配列を加えたものとなる．ここで各線分の配列には，その両端の点の名前が置かれる．対応する procedure は，線分 L_1, L_2, L_3

* 電子技術総合研究所ソフトウェア部情報システム研究室

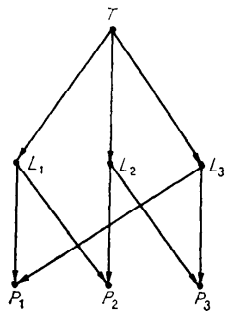


図2　3角形 T のもつ内在構造

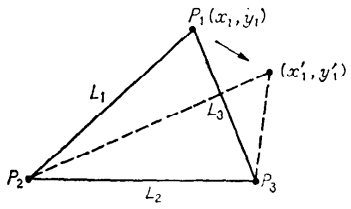


図3　3角形の一点の移動

を描く命令よりなるが，この procedure は，結果として3角形を書くが，そのプロセス自体は3角形を目指していない点に注意しなければならない．つまり，この procedure は，L_1, L_2, L_3 が3角形を構成するという“認識”なしに働くわけである．

3角形を“認識”するためには，点から線分を定義したように，3角形自身を識別するための名前 T を与えて，それを線分によって定義したものを，データとしてもたなければならない(**図1 (d)**)．

このように，データ構造を規定するものに，まず procedure によって“認識”されるデータの単位があり，これを**データ項目** (item) という．

次に，データ項目間に存在する最も基本的な関係によってできる構造を**内在構造**と呼ぶ．上記の例についていえば，3角形 T が3線分 L_1～L_3 により構成されるという関係，および各線分がその両端の点からできているという関係によって作られる**図2**のような構造が，この内在構造にあたる．

ここで，画面上の3角形に対して，簡単な操作をほどこすことを考えてみる．**図3**のように，3角形の1つの頂点 P_1 を，(x_1, y_1) から (x_1', y_1') に動かすとする．ライトペンにより点 P_1 を指示することにより，データ項目 P_1 がアクセスされるとする．点 P_1 を動かすためには，線分 L_1 と L_3 を動かさねばならず，それには，点 P_2, P_3 の座標が必要となる．つまり

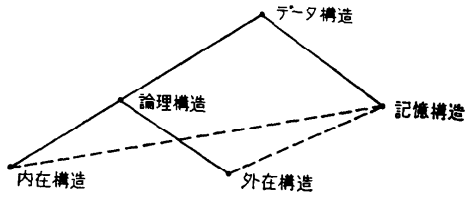


図4　データ構造の構成

$$P_1 \begin{cases} \nearrow L_1 \longrightarrow P_2 \\ \searrow L_3 \longrightarrow P_3 \end{cases}$$

という検索の路 (path) を要する．このような路を，起こり得るすべての操作について求めたものは，すべての操作に対する検索の路を含む．これは，前に与えた内在構造に対して，操作という外からの動作によって生じる構造という意味で，**外在構造**と呼ぶことにする．

データ構造は，これら2つの構造を包括するものと考えられる．各構造はデータ構造の部分構造と考えてもよいし，あるいは，もっと異なった形で，データ構造内に組み込まれることがあるかも知れない．

データ構造には，もう一つの側面がある．それは，その表現の手段からくる問題で，言葉による記述あるいは木構造のようなグラフによる表現を**論理構造** (logical structure) といい，計算機上での具体的な表現を**記憶構造** (strage structure) という．

これら諸概念の関係を**図4**にしめす．記憶構造は，内在構造および外在構造によって規定され，それらを反映して作られる．ゆえに，**図4**の破線は，内在構造，あるいは外在構造から記憶構造に向かう線と考えてもよい．木構造，リスト構造，リング構造などの言葉を，よく何のことわりもなしに使うが，それらは，あるときは論理構造，とくに内在構造の意味で使い，またあるときは，記憶構造として使うことがあるので注意を要する．

コンピュータ・グラフィックスという領域で発展したデータ構造の研究の成果は，他の分野，例えば，OS (Operating System)，言語プロセッサなどにもとり入れられるだけの価値がある．

より一般的に，計算機プログラムは procedure とデータから成っていると考えることができるので，データ構造の考え方は，プログラミング技術の基礎をなすものであるということもできるであろう．

この講座では，2回に分けてデータ構造についての解説を行う予定であるが，今回は，その内在構造がツ

リー図式（後述）で表現でき，最も簡単な外在構造を規定する操作のみを考慮に入れたデータ構造についてのべる．それらは，ほとんど皆周知の事実であるが，それらを上で与えたような観点により整理してみた．

次回は，リング構造，連想3つ組構造を中心に，より複雑なデータ構造について述べることにする．

2. ツリー図式による内在構造の表現

与えられた問題に適したデータ構造を求めることは，簡単なことではない．普通，それは一意的には定まらないであろう．そのため，最適な構造を求めることが問題となるが，与えられた問題が複雑になると，最適性の判定も困難になり，すっきりとした結論は得られにくい．例えば，検索の路を求める場合でも，頻繁に検索が行なわれるような路に対しては近路をつけた方がよいであろうし，不必要なまでに検索の路をつけるのは，その路を作る手間や，必要とする記憶量を考慮すると，かえって好ましくないであろう．

そこで，ここでは上記のように与えられた問題からデータ構造を求めるという過程をとらずに，逆に出来合のいろいろな構造の性質を調べておいて，その中から与えられた問題に最も適する構造を選ぶという立場をとることにする．

各構造は，その構造によって表わし得るモデルの集合，および検索の効率に差がある．これらの2つの特性は互いに相反する性質をもっているので，普通，簡単な構造は検索効率がよく，複雑なモデルまで表わし得る構造は検索効率がわるい．その点からみても，最適な構造を選ぶことの重要性が認識されよう．

この節では，比較的簡単な内在構造（配列，木構造等）を統一的に表現するためにツリー図式（tree diagram[3]）という概念を利用し*，各構造の記述を試みる．

2.1 ツリー図式に関する諸定義

ツリー図式とは，**図5**にみられるように，1つ以上の有向木構造を横に寝かせ，ルート（根）を左側にそろえて表わしたものである．（ただし，図上では方向を示す矢印は省略する．）以下に，ツリー図式を表わすための言葉の定義を与える．

m（モーメント）：ツリー図式中の点の総数．

r（基数）：ツリー図式中のルートの数．

点の次数：その点から出る弧の数．

リーフ：次数が0である点．

w（重み）：ツリー図式中のリーフの数．

レベル：ルートからの距離．（ルートのレベルを1とする．）

h（高さ）：ツリー図式中の最高のレベルの値．

s（次数和）：次数の総和．（弧の総数に等しい．）

ファミリィ：同じ点から出た弧の集合．（ルート全体も一つのファミリィをなすと考える．）

次元：ファミリィ中の点の数．

c（カウント）：同一レベル中の点の数．

ツリー図式中の点の関係を表わす言葉として，ある点から出ている弧でつながれている点をその点の**子**という．また，逆に，出発点の方を**親**という．また，ある点から到達できるすべての点よりなるツリー図式をもとのツリー図式に対して**サブツリー図式**という．

すべてのツリー図式について，次式が成り立つ．

$$m=r+s. \tag{1}$$

データ項目の集合の内在構造は階層構造になっていることが多いが，その場合には，ここにあげたツリー図式により表現できる．また階層構造の多くは，ある特定の性質をもったツリー図式のクラスによって表わされる．そこで，そのようなクラスを規定する代表的な構造をとりあげ，その性質をのべる．

r 進構造

すべてのファミリィが等しい次元rをもつ構造である．**図5 (a)** に例を示す．f をファミリィの数とすると，点の総数 m は，

$$m=f\times r \tag{2}$$

となる．ここで，リーフでない点は $(f-1)$ 個ある（ルートのファミリィを除く各ファミリィに，その親となるリーフでない点が対応する）ので，リーフの総数wは，

$$w=m-(f-1)=1+\frac{r-1}{r}\times m \tag{3}$$

となる．

底 $\boldsymbol{r}$ 構造

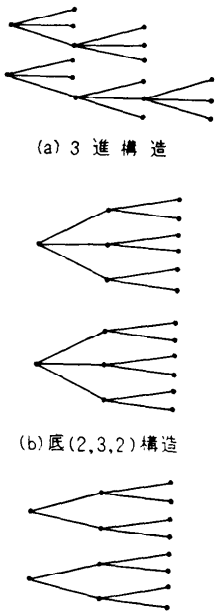
各レベル内でのファミリィの次元がすべて等しい構造である（**図5 (b)** 参照）．ここで$\boldsymbol{r}=(r_1, r_2, \cdots, r_h)$の各成分は，各レベルでの次元を表わす．各レベルのカウント $c_i(i=1, \cdots, h)$ は

$$c_i=r_1\times r_2\times\cdots\times r_i=\prod_{j=1}^{i} r_i \tag{4}$$

となる．重みwはレベルhでのカウントc_hに等しく，

---

* 2節から4節にかけてのツリー図式による統一的な記述は，文献3)によった．読者は，合わせて参照されたい．



(a) 3 進 構 造

(b) 底(2,3,2)構造

(c) 底2,高さ3構造

図5 ツリー図式の特別なクラスの例

$$w=c_h=\prod_{j=1}^{h} r_j \qquad (5)$$

で与えられる．点の総数 m は，次のようになる．

$$m=c_1+c_2+\cdots+c_h=\sum_{i=1}^{h}\prod_{j=1}^{i} r_j. \qquad (6)$$

底 r，高さ h 構造

r 進構造で，かつリーフのレベルがすべて h となる構造である．**図5 (c)** に底2，高さ3構造の例を示す．重み w は，r と h により次式で表わされる．

$$w=r^h \quad \text{for } h\geq 1. \qquad (7)$$

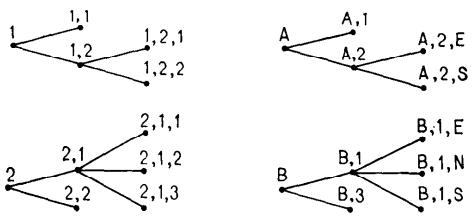
また r 進構造であるから（3）式が成り立ち，

$$m=\frac{r}{r-1}(r^h-1) \quad \text{for } r>1 \qquad (8)$$

となる．

この他に，リーフのレベルがすべて等しい構造を，**一様構造**という．底 r，高さ h 構造は，もちろん底 r 構造である．またこれらは共に一様構造である．

ツリー図式は，構造を図示するので，人間の直観的な洞察に適している．しかし，この構造を計算機に格納したり，通信回線で送ったりする場合，一度それらを文字または数字の列で表わさねばならない．そのためには，各点の名前付け，および各弧の識別が必要不可欠である．これはツリー図式の1次元的記述法を与える．次に，2つの代表的な名前の付け方をあげる．



(a) インデックス・タグ (b) ダブ・タグ

図6 ツリー図式の点の名前のつけ方

インデックス・タグ

1. ルートのファミリィに1(0の場合もある．そのときは0-origin という）から通し番号をつける．

2. 親の点のインデックス・タグの次に，（コンマ）をおき，そのファミリィ内の1からの通し番号をつける．

図6 (a) に，インデックス・タグの例を示す．

ダブ・タグ

インデックス・タグでの通し番号を，任意の単調増加数列または（ある意味で順序のある）文字列におきかえたもので，**図6 (b)** にその例をあげる．

2.2 内在構造の表現

内在構造をツリー図式により表現するには，ツリー図式中の各点にデータ項目を対応させ，点のつながりをデータ項目間の関係に対応させればよい．

データ項目の中には，それに付随するデータを持つものと持たないものとがある．1節で述べた3角形の記述の例では，3角形 T と線分 $L_1\sim L_3$ は，データを持っていないが，点 $P_1\sim P_3$ は，座標 (x_i, y_i) をデータとして持っている．

多くの問題は，そのデータの持ち方で，2つの極端な場合のいずれかになっている．それは，その内在構造をツリー図式で表わしたときにすべての点がデータを持つ場合と，すべてのリーフだけがデータを持つ場合で，前者をツリー (tree) といい，後者をツリム (trim) という．

3. ツリー図式の計算機内での表現

1節でも述べたように，データ構造の計算機内での表現，すなわち記憶構造は，モデルの内在構造だけでは決まらず，外在構造も考慮されねばならない．しかし，この節ではまず内在構造を完全に表わすことを目標にして話をすすめる．

計算機の記憶は番地によって順序付けられているので，ベクトルや行列の表現は容易である．ゆえに，ツリー図式をベクトルや行列で表わすことができれば，それからただちに，対応する記憶構造が得られる．そのような各表現を与える前に，基本となる2つの代表的な，ツリー図式中の点の順序付けを与える．

右インデックス

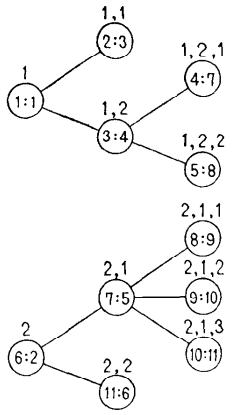
第1レベルから始まり，各レベル内で上から番号をふり，次のレベルにいく方法である．(**図7** (**a**) 参照.)

左インデックス

第1レベルの第1点から始まり，必ずその点のサブツリー図式中のすべての点に番号をふってから，そのファミリィの次の点にいく方法である．(**図7** (a) 参照.)

3.1　タグ行列

インデックス・タグを行列状にならべる方法である．行列の大きさは $m \times h$ で，各行が各点に対応する．



(p:q で，p は左インデックス，q は右インデックスを表わす)

(a)

L

p:q			
1:1	1	∘	∘
2:3	1	1	∘
3:4	1	2	∘
4:7	1	2	1
5:8	1	2	2
6:2	2	∘	∘
7:5	2	1	∘
8:9	2	1	1
9:10	2	1	2
10:11	2	1	3
11:6	2	2	∘

左タグ行列

R

			p:q
∘	∘	1	1:1
∘	∘	2	6:2
∘	1	1	2:3
∘	1	2	3:4
∘	2	1	7:5
∘	2	2	11:6
1	2	1	4:7
1	2	2	5:8
2	1	1	8:9
2	1	2	9:10
2	1	3	10:11

右タグ行列

(b)

図7　左，右インデックスと左，右タグ行列

各タグをベクトルと考えると，レベルが h より小さい点のタグ・ベクトルの次元（成分の数）は h 未満となる．そのベクトルに，空要素 (1-origin であれば 0 でよい) を加えて，すべて h 次元とする．そのとき，タグを右にそろえて，左側に空要素を加え，行に関してソートして得られる行列を右タグ行列という．このとき，空要素は，他のどの要素よりも小さいとする．また，タグを左にそろえて，右側に空要素を加えたものを行に関してソートして得られる行列を左タグ行列という．そのとき，その行の順序は，それぞれ，右インデックスおよび左インデックスの順序になっている．(**第7図**(**b**) 参照.)

タグ行列は，ツリー図式を完全に表わしている．さらに，リーフでない点に対する行はとりのぞくことができ，後で再現させることもできる．

右タグ行列は，レベルの順にならんでいるので，これを一様構造に適用するとそのリーフは全て行列の下部に集まるので都合が良い．また，タグ・ベクトルの各成分をタグ・アルファベットと呼ぶことにすると，右タグ行列の最右列に，すべてのタグ・アルファベットが出てくるのがわかる．

左タグ行列は，各点のサブツリー図式がすぐ下に出てくるので，サブツリーの解析に適している．

タグ行列が順序づけられているということは，構造の変化に対する適応性があまり良くないことを示している．

3.2　ポインタ行列

$m \times 2$ 行列で，その要素は空または行番号である．各行は，ツリー図式の各点に対応している．第1列の要素はそのファミリィ内の次の点に対応する行の番号である．もしその点が，ファミリィ内の最後の点ならば空である．第2列は，その点の子のファミリィ内の第1要素に対応する行の番号である．もしその点がリーフであれば，その要素は空である．ここで，ツリー図式の各点の順序づけは，任意である．(**図8**参照.)

このポインタ行列は，ツリー図式を2進木構造で表わしたときの各セルを行列状にならべたものである[4]．

この表現は，順序によらないので，構造の変化に簡単に追従し得る．任意の点に弧をつけ加えるには，その行列の次にその点に対応する欄をとり（あるいは自由領域からセルを確保し），その行へのポインタを，

L				P	
1	∘	∘	1	6	2
1	1	∘	2	3	∘
1	2	∘	3	∘	4
1	2	1	4	5	∘
1	2	2	5	∘	∘
2	∘	∘	6	∘	7
2	1	∘	7	11	8
2	1	1	8	9	∘
2	1	2	9	10	∘
2	1	3	10	∘	∘
2	2	∘	11	∘	∘

左タグ行列　　ポインタ行列

図8　図7のツリー図式に対応するポインタ行列（左インデックス順）

適当につければよい．任意の点の削除も，同様にポインタを書き替えることによってなされる．

3.3 次数ベクトル

点の次数を並べたもので，右インデックスの順序に並べたものを右次数ベクトル，左インデックスの順序に並べたものを左次数ベクトルという．

右次数ベクトルからもとのツリー図式を得るには，まずルートの数を求めなければならない．右次数ベクトルを$(d_1, d_2, \cdots, d_m)$で表わすと，次数和sは，

$$s=\sum_{i=1}^{m} d_i \tag{9}$$

となる．これと（1）式より，

$$r=m-s=m-\sum_{i=1}^{m} d_i \tag{10}$$

により，ルートの数rが決まる．rが求まれば，ツリー図式を求めるのは容易である．この右次数ベクトルは，右タグ行列と同様，各レベルにそって解析を行なう場合に適している．

左次数ベクトルは，左タグ行列と同様，サブツリーの解析に有効である．左次数ベクトルから，各点のサブツリーを見出すには，サブツリーに対して成りたつ次の関係を利用すればよい．それは，

$$m-s=1 \tag{11}$$

なる関係である．これは，（1）式に$r=1$を入れて求まる．任意の点のサブツリーを見出すには，左次数ベクトルをその点から右へ走査し，m個走査した時点で初めてその合計が$m-1$になっている点を求めればよい．

3.4 フラグ行列

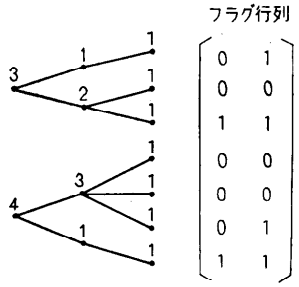


図9　一様構造とフラグ行列
（各点の数字は，その点の重みを表わす．）

これは，一様構造に対してのみに有効な方法で，$w \times (h-1)$の(0, 1)行列から成る．各行はリーフの各点を表わし，各列はリーフ以外の各レベルを表わす．各レベルで点を上からたどるとき，その点の重みをwとすると，$(w-1)$個の0につづいて1をおく．（**図9**参照．）この方法は，右次数ベクトルに比べて簡潔さの点では劣るが，一様ツリムを表わす場合，リーフのデータの一部として，各行をおくことができる点で有利である．

3.5 次元ベクトル

底r構造の表現にのみ用いられる方法で，h次の次元ベクトル$\boldsymbol{r}=(r_1, r_2, \cdots, r_h)$から成る．リーフを配列としてならべたとき，その番号は，0-originインデックス・タグと，$\boldsymbol{r}$から容易に求めることができる．まず重みベクトル$\boldsymbol{w}$は$\boldsymbol{r}$の成分により次式で求められる．

$$\begin{pmatrix} w_1 \\ w_2 \\ \vdots \\ w_h \end{pmatrix} = \begin{pmatrix} r_2\times r_3\times\cdots\times r_h\times 1 & & \\ & r_3\times\cdots\times r_h\times 1 & \\ & & \ddots \\ & & r_h\times 1 \end{pmatrix}. \tag{12}$$

任意のリーフのインデックス・タグ・ベクトルを$\boldsymbol{i}=(i_1, i_2, \cdots, i_h)$で表わすと，そのリーフの配列上での番号$k$は，

$$k=\sum_{j=1}^{h} w_j i_j \tag{13}$$

で求まる．

3.6 基数と高さ

底r，高さh構造は，rとhによりその構造が完全に決まる．重みベクトル$\boldsymbol{w}$は，

$$w_i=r^{h-i} \qquad \text{for } i=1, 2, \cdots, h \tag{14}$$

により与えられ，これを(13)式に代入して任意のリーフの配列上での番号をうる．

4. 表 検 索

この節では，内在構造に加えて，表検索(table search)という操作により規定される外在構造を反映した種々の記憶構造を与えて，その検索効率について論じる．表検索とは，タグを指定して，そのタグを持ったレコードをとり出す操作をいうが，これは，最も簡単な外在構造を与える．それは，各項目の間には全く関係を定めず，すべての項に外から直接アクセスする路を与えるだけである．

4.1 指向走査検索

配列状に並んだ表を1レコードずつ順々にとり出して比較する方法で，検索表は，タグ行列の各行にデータをつけ加えて作られる．比較は，＝，キの2つであり，表はソートされていなくてもよい．走査を始める点は，表の先頭あるいは前の検索で一致した行の次などが考えられる．m をレコードの総数とすると，平均探索回数は $(m+1)/2$ である．

この場合の外在構造は，**図10**のような線型グラフ，または環状グラフで表わされる．この方法は，平均探索回数が他の方法に比べて最も大きいが，外在構造を反映して，カード・デッキや磁気テープなどの sequential access 記憶装置を使う場合には適している．

4.2 制御走査検索

順序付けられた表の検索を，その順序を利用して行なう方法で，検索表は指向走査検索表をソートしたものである．比較は ＞，＝，＜ の3つの結果を生み出す．3進制御走査検索の外在構造を**図11**にしめす．この図で各ファミリィ（この図を3進ツリー図式の2進表現とみたときのファミリィ）の最後の頂点はダミーである．また各頂点の円の中の数字は，その頂点に到達するまでの探索回数を表わし，外の数字は検索表中での小さい方からの番号を表わす．

一般に h レベル r 進制御走査検索の外在構造は，底 r，高さ h 構造になる．(正確にいえば，その2進木構造表現.) 次にその平均探索回数を求めよう．

第1レベルで検索の終わるレコードの数 n_1 は，$(r-1)$ 個で，その平均探索回数 t_1 は $r/2$ 回である．一般に，第 i レベルで検索の終わるレコードの数 n_i は $r^{i-1}(r-1)$ で，それらの平均探索回数 t_i は，そのレベルに達するまでの平均回数に，そのレベルでの平均回数を加えたものである．それは，

$$t_i=\frac{r}{2}+\frac{(r-1)(r+2)}{2r}(i-1) \qquad (15)$$

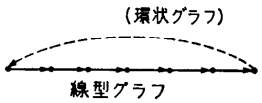


図 10 指向走査検索に対する外在構造

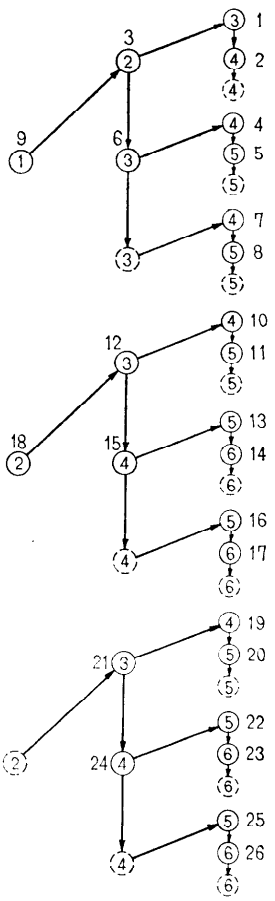


図 11 3進制御走査検索に対する外在構造

で与えられる．ここで後の項の $(i-1)$ の係数は，各レベルを通りぬける回数の平均で，

$$\frac{1}{r}\{1+2+\cdots+(r-2)+(r-1)+(r-1)\}$$

に等しい．全体の平均探索回数 $E(r,h)$ は，

$$E(r,h)=\frac{\sum_{i=1}^{h} n_i t_i}{\sum_{i=1}^{h} n_i}$$

$$=\frac{r^h-1}{2}\sum_{i=1}^{h} r^{i-1}(r-1)\left\{\frac{r}{2}\right.$$

$$+\frac{(r-1)(r+2)}{2r}(i-1)\Big\}$$

$$=\frac{(r-1)(r+2)hr^{h-1}}{2(r^h-1)}-1 \qquad (16)$$

となる．2進検索での平均探索回数は，

$$E(2,h)=h\left(\frac{2^h}{2^h-1}\right)-1 \qquad (17)$$

となる，h が適当に大きければ，これは $h-1$ で近似される．

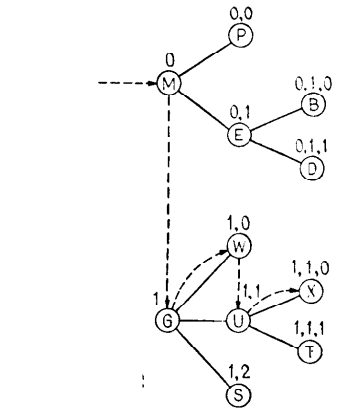
4.3 構造検索

外在構造として，内在構造を表わすツリー図式の等価的表現である2進木構造を考えたとき，その検索法を，構造検索と呼ぶ．その中には，ツリー図式中での各ファミリィ内のたどり方によって3つの異なった方法がある．それらを**ランク・パス検索** (rank path search)，**インデックス・パス検索**，**直接パス検索**という．

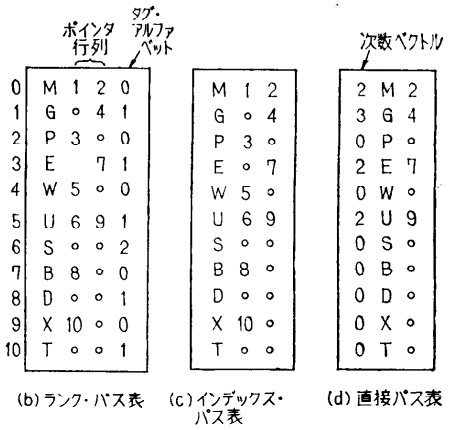
ランク・パス検索では，検索表としては，ポインタ行列の各行にデータと，タグ・アルファベットをつけ加えたものを用いる．いま，**図12 (a)** のようなツリーで，タグ (1, 1, 0) をもつ点を検索することを考える．検索はかならずランク・パス表 (**図12 (b)**) の先頭から始める．まず，タグ (1, 1, 0) の第1成分を，その欄のタグ・アルファベットと比較し，等しくないので，ポインタ行列の第1列のポインタをたどってそのファミリィの次の点に対応する行 (# 1行) をアクセスし，そこをしらべる．そこでは，タグ (1, 1, 0) の第1成分とタグ・アルファベットが一致するので，次の成分をしらべるために，その子のファミリィを，ポインタ行列の第2列のポインタをたどってアクセスする (# 4行)．以上の操作を，タグが調べおわるまでつづける．この場合，タグはダブ・タグでもよい．また，各ファミリィ内の連鎖はソートされなくてもよい．各成分の検索はランク演算[3]と呼ばれている演算に相当するので，これをランク・パス検索という．

インデックス・パス検索は，ランク・パス検索と，次の2点で異なる．(1) タグはインデックス・タグしか許されない．(2) 各ファミリィ内はインデックスの順につながれている．このため，検索は比較を要せず，計数するだけでよい．インデックス・パス表は，各ファミリィ内でソートされたポインタ行列の各行にデータをつけ加えて得られる．(**図12 (c)**)

直接パス検索は，インデックス・パス検索でファミリィ内をポインタでたどる代りに，各ファミリィを連続領域におき，ファミリィ内のレコードの行番号を指定タグから計算によって得る方法である．直接パス表には，ファミリィ内のポインタは不要である．その代りに指定変数の誤りが検出できるように，次数の情報をおいた方がよい．(**図12 (d)**)



(a) ツリーの一例



(b) ランク・パス表　(c) インデックス・パス表　(d) 直接パス表

図12　構造検索に用いる3つの検索表

次に，これらの方法での平均探索回数を求めよう．一般のツリー図式について求めるのは困難であるので，底 r，高さ h 構造について求める．直接パス検索では，

$$E(\text{direct path}, r, h)=\frac{\sum_{i=1}^{h} ir^i}{\sum_{i=1}^{h} r^i}=\frac{hr^h}{r^h-1}-\frac{1}{r-1} \qquad (18)$$

となる．ランク・パス検索では，探索されるファミリィの数が直接パス検索での探索回数に等しいので，

$$E(\text{rank path}, r, h)=\frac{r+1}{2}[E(\text{direct path}, r, h)]$$

$$=\frac{r+1}{2}\left(\frac{hr^h}{r^h-1}-\frac{1}{r-1}\right) \quad (19)$$

で与えられる．ツリムに対するランク・パス検索（リーフ・ランク・パス検索という）の探索回数は，

$$E(\text{leaf rank path}, r, h)=\frac{r+1}{2}h=\frac{r+1}{2}\log_r w \quad (20)$$

となる．(20) 式は，

$$\log_e r-\frac{r+1}{r}=0$$

のとき最小となる．これは $r\simeq 3.6$ で，そのときの探索回数は，

$$E(\text{leaf rank path}, 3.6, h)=\frac{3.6+1}{2\log_e 3.6}\log_e w$$

$$\simeq 1.25\log_2 w \quad (21)$$

で与えられる．これから，リーフ・ランク・パス検索では，2進検索の，少なくとも1.25倍の探索回数を要することがわかる．しかし，ランク・パス検索は次のような利点をもつ．（1）検索分布が一様でないとき，それを利用して検索効率を上げることができる．（2）2進検索では比較の結果が3通り（>，=，<）であるが，ランク・パス検索では2通り（=，≠）である．（3）2進検索では比較がタグ・ベクトル全体であるが，ランク・パス検索では1成分だけである．

4.4 ハッシュ検索

この節の最初にのべた外在構造の性質を最も直接的に反映している検索法にハッシュ検索[5]がある．それは，タグに一定の計算をほどこして，そのタグに相当するレコードの行番号を求める．ゆえに，検索の路は，外から各レコードに直接ついていると考えられる．しかし，一般にはタグから行番号への変換（これをハッシュ変換という）は1対1ではないので，異なったタグが同じ整数へ変換されることがある．この現象をコンフリクト（conflict）という．検索にとって，良いハッシュ変換は，このコンフリクトが統計的に少ない方法であり，それは各整数が等確率で発生されるような変換である．

また，コンフリクトの処理方法はいくつか考えられるが，その仕方によって平均探索回数および必要とする記憶量が異なってくる．ここでは，代表的な3つの方法をあげ，それらの平均探索回数を与える．

直接連鎖法

同じ整数に変換されたすべてのレコードをポインタでつないで連鎖として管理する方法を直接連鎖法（direct chaining method）という．コンフリクトを起した項目はハッシュ表の外にとられた領域におかれる．

平均探索回数は，他のどんな方法よりも少ない．ハッシュ表の容量（コンフリクト用の領域を含まない）に対する，登録されたレコード数の比率（これを占有率という）を α とすると，平均探索回数 $E(\text{direct chain}, \alpha)$ は，

$$E(\text{direct chain}, \alpha)=1+\frac{\alpha}{2} \quad (22)$$

で与えられる．

必要とする記憶領域は，ポインタの分だけ他の方法よりも多い．

線形法

コンフリクトを起こした項目を，その欄から走査して最初に見つかった空欄に置く方法を線型法という．

この方法は，平均探索回数が最も大きく，

$$E(\text{linear}, \alpha)\simeq 1+\frac{1}{2}\left(\frac{\alpha}{1-\alpha}\right) \quad (23)$$

で与えられる．

ランダム法

コンフリクトを起こした項目を処理するために，項目を次々に別の整数に変換し，その項目を最初に見つかった空欄に置く方法で，各項目に対する整数列は擬似乱数列から求める．

この方法の平均探索回数は，上記の2つの方法の中間で，

$$E(\text{random}, \alpha)\simeq -\frac{1}{\alpha}\log(1-\alpha) \quad (24)$$

で与えられる．

参 考 文 献

1) Gray, J. C.: "Compound data structure for computer aided design ; a survey," Proc. ACM National Conference, 1967.
2) 大須賀節雄："コンピュータ・グラフィックス[2]"，情報処理，Vol. 12, No. 7, 1971, pp. 422～434.
3) Johnson, L. R.: "System structure in data, programs, and computers," Prentice-Hall, 1970.
4) Knuth, D. E.: "The art of Computer Programming," Vol. 1 Fundamental algorithms, pp. 332～347.
5) Morris, R.: "Scatter storage techniques," CACM 11, 1 (Jan. 1968), pp. 38～44.

(昭和47年2月12日受付)