講　　座

データ構造 (II)

古川康一*

5. ルート・グラフの表現

データ構造 (I) では，内在構造がツリー図式で表現でき，外在構造が表検索によって規定される比較的単純なデータ構造について述べたが，今回は，内在構造を一般の有向グラフに拡張し，それにともなってより複雑な外在構造をもつデータ構造を考える．

ここで，有向グラフと内在構造の対応は，2.2 で与えたツリー図式と内在構造の対応と同様．有向グラフ上の各点がデータ項目を表わし，2点を結ぶ弧がデータ項目間の関係を表わす．

この節では，一般有向グラフとツリー図式との中間的な性質をもつクラスとしてルート・グラフを考え，その性質を調べ，対応する記憶構造を与える．

5.1 ルート・グラフの性質

ルート・グラフとは，ルートと呼ばれる特定の点を1つ持ち，他の各点は必ずそのルートから弧の方向に沿ってたどることができる有向グラフをいう．

ツリー図式では複数個の連結成分（その1つを“木”と呼ぶ）の集合を考えたが，ルート・グラフでは，1つの連結成分のみを考える．ルート・グラフと木の相違は，木でのルート以外の各点には必ずただ1つの弧が入ってくるが，ルート・グラフでは，それが複数個になってもよいことである．このことは，ルート・グラフでは，ツリー図式のようなタグによる各点の名前付けができないことを意味する．それは，各点の名前が一通りには決まらないからである．

この点を解決するために，ルート・グラフの性質を使い，各点をルートからのパスにより指定することを考える．ルートからの1つのパスを指定するためには，ある点まで到達したときにその点からどの弧に進むかが与えられればよい．そのために，弧に選択情報をおく．この選択情報を**セレクタ**と呼ぶ．セレクタは，ツリー図式において各点に与えられるタグに似ているが，タグそのものよりもむしろその点に新たに与えられたタグ・アルファベットに対応する．また，タグの場合と同様に，インデックス・セレクタと，ダブ・セレクタを定義することができる．

いま，**図 13** のようにルート・グラフのセレクタを定めると，各点は，ルートから出発していろいろなパスで到達できる．パスをセレクタの系列よりなるベクトルで表わすと，点7は，(1, 1, 1)，(2, 1, 1)，(3, 2) の3つのパスにより到達される．これらのセレクタのベクトルは，その点自身の名前と考えてもよい．つまり，各弧にセレクタを置くことにより，各点のもつ複数個の名前を表わしていることになる．

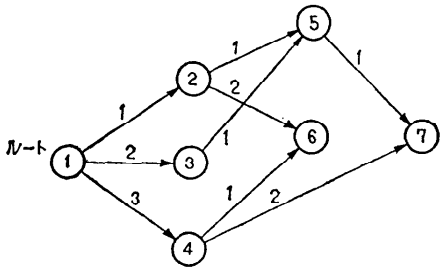


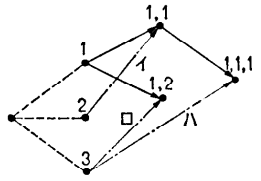
図 13　ルート・グラフGのセレクタによる点の指定

ここで注意を要するのは，ツリー図式の場合には各点のタグの集合を与えることによりツリー図式が完全に決まるが，ルート・グラフではルートとすべてのセレクタを与えてもそのグラフが求まらない点である．そこで次に，ツリー図式のタグの集合に匹敵するような，ルート・グラフの1次元的記述法を与えよう．

ルート・グラフを木の拡張とみなし，その中の極大木 (spanning tree) を求め，その極大木からルートおよびそこから出る弧をとり去って得られるツリー図式（これを極大森と呼ぶ）を考え，それをタグの集合によって表わす．

極大森上の各点のタグは，極大木上の各弧のセレク

* 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



	始点	終点
イ	2	1,1
ロ	3	1,2
ハ	3	1,1,1

（a）ルート・グラフGの極大森 G′ と各点のタグ　（b）極大木に含まれない弧を表わすタグの対

図 14　ルート・グラフの1次元的記述

タが，その弧の終点のタグ・アルファベットとなるように決める．極大木に含まれない弧は，その始点と終点に与えられたタグの対により表わす．極大森を表わすタグの集合と，それ以外の弧を表わすタグの対の集合によりルート・グラフが完全に決まるので，1次元的記述を与えたことになる．**図 14** に**図 13** のルート・グラフGに対する極大森 G′ とその各点のタグ，および極大木に含まれない弧を表わすタグの対を与える．

5.2 ルート・グラフの記憶構造

複合ポインタ行列

ルート・グラフの記憶構造は，ツリー図式を表わすポインタ行列を拡張して得られる．それを複合ポインタ行列という．ツリー図式の場合と同様に，s を次数和（弧の総数）とし，w をリーフの総数とすると，それは $(s+w)\times 2$ 行列で，その各要素は空または行番号である．各行は，ルート・グラフの各弧および各リーフを表わし，弧を表わす行のいくつかは，同時にリーフ以外の点を表わす．1つの点から出る弧の集合を**弧のファミリィ**ということにすると，第1列の要素は，その弧のファミリィ内の次の弧に対する行番号である．もしその弧がファミリィ内の最後の弧の場合，あるいはその行がリーフに対する行の場合，その欄は空である．第2列は，その弧の終点に対する行番号である．もしその行がリーフに対する行であれば，その欄は空である．そのとき，リーフ以外の各点は，その点を始点とする弧のファミリィの最初の弧を表わす行に対応させる．図13のルート・グラフGは，**図 15** のような複合ポインタ行列で表わされる．

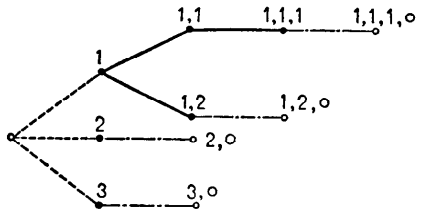
点	弧			行番号
1	1	2	4	1
	2	3	6	2
	3	o	7	3
2	1,1	5	9	4
	1,2	o	10	5
3	2,1	o	9	6
4	3,1	8	10	7
	3,2	o	11	8
5	1,1,1	o	11	9
6	(1,2,o)	o	o	10
7	(1,1,1,o)	o	o	11

図 15　ルート・グラフGの複合ポインタ行列による表現

この行列を，5.1 で与えたルート・グラフの1次元的記述から作るには，次のようにすればよい．

（1）極大森の各リーフにセレクタ。(空)をもった弧を1つずつ付け加え，その先を新たなリーフとする．

（2）それらの点に対するタグを求めてタグの集合に付け加える．



（a）リーフの付加された極大森 G″

タグ・ベクトル			行番号
1	2	4	1
2	3	6	2
3	o	7	3
1,1	5	9	4
1,2	o	10	5
2,o	o	o	6
3,o	o	o	7
——	——		8
1,1,1	o	11	9
1,2,o	o	o	10
1,1,1,o	o	o	11

（b）G″ のポインタ行列表現

図 16　ルート・グラフの1次元的記述から複合ポインタ行列を作る過程

（3） タグの集合から，対応するポインタ行列を作る．

（4） 極大木に含まれない弧を表わすタグの対の集合より，弧を追加する．

ここで，（4）の弧の追加は，その弧の始点がもとの極大木上でリーフである場合とそうでない場合でその方法が異なる．始点がリーフである場合，（1）によってそのリーフに追加した点に対応する行の第2列の要素を，弧の終点に対する行の第2列の要素に等しくする．始点がリーフでない場合，その弧に対する行を新たに確保し，その弧の属するファミリィの連鎖にその行をつなぎ，その行の第1列の要素は空とし，第2列は上の場合と同様に終点に対する行の第2列の要素をおく．**図 16** に，ルート・グラフGに対するリーフの付け加えられた極大森 G″ と，そのポインタ行列を示す．

複合ポインタ行列の各行は，3種の異ったもののどれかを表わしているが，それは行列の要素の性質によって分類することができる．すなわち，

i）点と弧を同時に表わす行：第1行（ルート），および第2列に表われた行番号が示す行のうち，その行の第2列の要素が空でないもの．

ii）弧のみを表わす行：第1列に表われた行番号に対する行．

iii）リーフを表わす行：第2列の要素が空である行．

ルート・グラフの表検索は，セレクタ・ベクトルを指定して行なわれる．その検索は複合ポインタ行列を用いるので，ツリー図式の場合の構造検索にあたる．この場合にも，弧のファミリィのたどり方により，ランク・パス検索，インデックス・パス検索，直接パス検索の3つを考えることができる．ここでは，最も一般的なランク・パス検索だけをとり上げる．検索表は，複合ポインタ行列の弧を表わす行にその弧のセレクタをおき，点を表わす行にその点のもつデータをおいて得られる．検索は必ずルートに相当する第1行から始められて，ツリー図式の場合のランク・パス検索と同様に与えられたセレクタ・ベクトルの成分を順に表中のセレクタと比較をして，ポインタをたどりながら行なう．そして最後のセレクタまで一致したら，その行の第2列にある行番号が指し示す行のデータを取り出す．この最後の操作はツリー図式の場合のランクパス検索と異なる．それは，タグとセレクタの違いから生ずる相違である．

この記憶構造は，弧の方向に沿った検索のみが可能で，逆方向にはたどれない．この構造でも有効に働く例としては，共有を許すファイルの構造がある．その場合，1つのファイルは，複数のパスによりアクセスされることが必要であるが，逆方向にたどれなくてもよい．これは，4. の表検索で与えた最も簡単な外在構造に相当する．

リング構造

より複雑な外在構造をもつ場合，当然逆方向にもたどる必要が生じてくる．このような機能をもつ記憶構造にリング構造がある．リング構造の基本的構成単位はリングであるが，それは閉じたリンクド・リスト（linked list）である．リングには**リング・スタート**（ring start）と呼ばれる特別な要素を1つもち，他の要素を**リング・タイ**（ring tie）と呼ぶ．1つのリングは集合を表わし，リング・スタートがその集合自身を表わし，リング・タイは，その集合の要素を表わしていると考えることができる，リング・スタートからすべてのリング・タイにアクセスすることも，逆に任意のリング・タイからリング・スタートをアクセスすることもリンク・ポインタをたどることにより可能である．

このリング構造によりルート・グラフを表現するには，ルート・グラフのもつ点の親子関係を，リング・スタート，リング・タイの関係で表わせばよい．各点は，ある他の点の子になっており，また一方自分自身の子の点をもつので，一つのリング・タイは，他のリングのリング・スタートとして働かなければならない．このような構造は，行列表現では考えにくいので，各点に対応して，**エレメント・ブロック**と呼ばれる主記憶の連続領域を考える．各エレメント・ブロックは，一つのリング・スタートと，いくつかのリング・タイ用のリンク・ポインタ領域をもつ．**図 17** に，ルート・グラフGに対するリング構造を示す．この記憶構造に

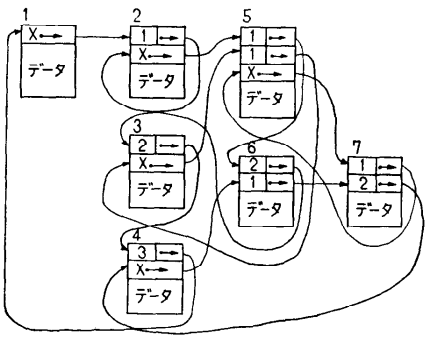


図 17　APL によるルート・グラフGの表現

は，APL[6] がある．

この構造で，点の親の集合をもリングで表わすと，各点のブロックの形式が一定となり便利である．この場合には，2つのリング（もとのリングと新しく作った親の集合を表わすリング）を結合するために，アソシエータ (associater) と呼ばれる補助的なブロックを要する．この構造は，ASP[7] を簡略化したものである．ここでは，これを simple-ASP (s-ASP) と呼ぶことにする．この構造による表現例を**図 18** に示す．この図は，ASP の図式表現(後述)を用いて表わした．

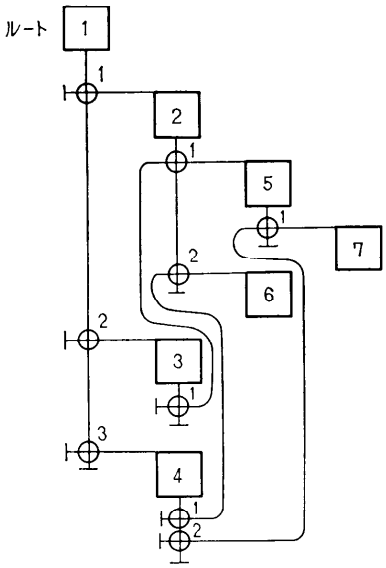


図 18　s-ASP によるルート・グラフGの表現

この構造では，エレメントが点を表わし，アソシエータが弧を表わしていると考えることができる．このことは，s-ASP が複合ポインタ行列よりもより整理された表現であることを示している．この構造で検索を行うためには，アソシエータにセレクタを追加し，エレメントにその点のデータをおけばよい．

s-ASP によく似たデータ構造に CYLINDERS[8] (CYL) がある．CYL は，2つの大きなリングと，そのリング上にのるシーム (seam) およびリンク (link) と呼ばれるセルから成る．**図 19** のようにシームはその2つのリング上に梯子状におかれる．上のリングと下のリングは方向が逆のリストになっていて，2つのシームにはさまれたループは，左のシームの下のリングから出発し，右のシームに達し，そこで上のリングに乗りかえることによって最初のシームにもどるので，サブリングを構成していると考えることができる．そのとき，左のシームをそのサブリングのリング・スタートと考える．リンクはシームと同様に，2語から成っていて，それらはそれぞれ2つのリング上におかれる．そして，シームによって作られたサブリングの任意の2つを結合する働きをする．図 19 では，リンク1は，サブリング1とサブリング3を結合している．シームとリンクは実際上は区別がつかないので，それを識別するためのフラグを要する．

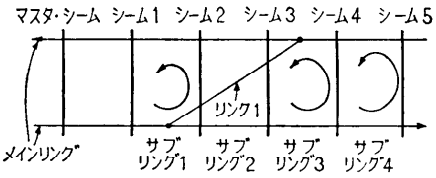


図 19　CYLINDERS の図式表現

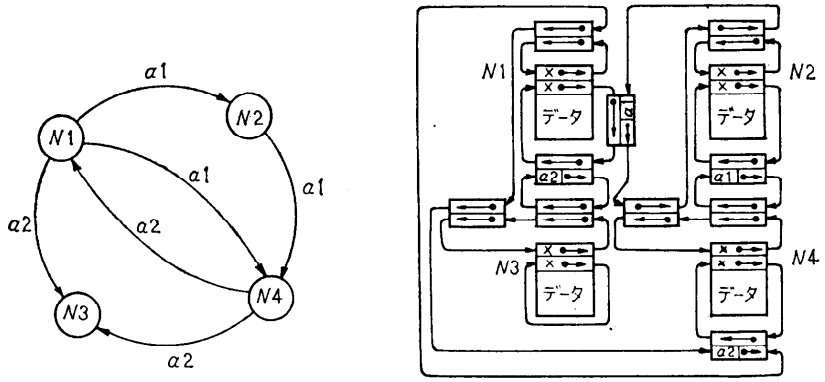
これを用いてルートグラフを表現するには，点にシームを対応させ，弧にリンクを対応させればよい．CYL によるルートグラフの表現を**図 20** に示す．この図で，上下の2つのリングをシームのところで切りはなし，個々のリングに作りなおすと，図 18 と全く等しい s-ASP が得られる．ここで，シームにはエレメントが対応し，リンクにはアソシエータが対応する．

図 20　CYLINDERS によるルート・グラフGの表現

6. 有向グラフの表現

6.1 有向グラフの性質

一般の有向グラフは，特定のルートをもたない．故に，各点へは，特定の点からのパスによってはアクセスできないことになり，ルートグラフのように，ルートとセレクタの系列によって各点を指定することができない．このため，各点の構造的命名法がなく，各点は，それぞれ個有の識別名 (id) が必要となる．この場合にも，セレクタという概念は有効であるが，特定のルートがないので，パスを選択する根拠も薄く，む



(a) (b)

図 21 一般有向グラフHとその ASP 表現

しろ各弧は，その始点と終点の2項関係を定めるものと考えた方が，より一般的である．この弧に与えられる2項関係の情報を**タイプ**と呼ぶことにする．セレクタとタイプの見かけ上の相違は，ある点から複数個の弧が出ている場合，セレクタは全て異なるが，タイプは同一のものがあってもよい点である．

このように，各点が id をもち，各弧がタイプ情報をもつモデルを一般有向グラフ[9]と呼ぶ．

一般有向グラフの1次元的記述は，各点のもつ id により容易に得られる．すなわち，すべての点の id の表を作り，各弧に対して与えられる〈タイプ，始点，終点〉の順序3つ組の表を作る．これら2つの表からもとの一般有向グラフを再現できるのは明らかである．

6.2 一般有向グラフの記憶構造

リング構造

ルート・グラフのリング構造による表現を拡張して，一般有向グラフに対するリング構造が得られる．ルート・グラフの場合には，各点に対応するエレメント・ブロックは，1つのリングを持ち，その点を親とする点のファミリィが，そのリング上におかれた．これに対して一般有向グラフでは，各点に対応するブロックに，その点を始点とする，異ったタイプを持った弧に対して1つずつのリング・スタートをおき，各リング上には，そのタイプをもつ弧の終点の集合をおく．

APL では，各点に対応するブロック内に複数個のリング・スタートをおくことにより，ルート・グラフから一般有向グラフへ容易に拡張される．

ASP では，各点のブロック内に複数個のリング・スタートをおくかわりに，それらのリング・スタートより成る集合を管理する一段レベルの高いリングを作り，そのリングのリング・スタートのみをもとのエレメント・ブロック内におく．また s-ASP と同様に各点の親の集合を1つのリングで表わし，2点間の接続関係は，始点と弧のタイプで決まるリングおよび終点を表わすリングを結合するアソシエータにより表わす．

以上の説明でも明らかなように，ASP は3つの異なった種類のリングにより構成されている．第1は，リング・スタートのリングで Lower ring (Lリング) といい，第2のリングは，各点の親の集合を表わすアソシエータのリングで Upper ring (Uリング) という．Lリングのリング・スタートは始点のブロック内にあり，Uリングのリング・スタートは終点のブロック内にある．各ブロックは，ある弧に対しては始点となり，また他のある弧に対しては終点となるので，LリングとUリングの両方のリング・スタートを同時に持つ．第3のリングは，始点と弧のタイプで決まるリングで，Lリング上にそのリング・スタートを持ち，そのリングの要素は，Uリングと結合する役割をもつアソシエータである．これを Center ring(Cリング) という．APL でのリングは，このCリングに相当している．

図 21 に一般有向グラフの例と，その ASP 表現を示す．

ASP では各要素を**図 22** のような記号を用いて表わす．LリングおよびUリングのリング・スタートはとくに明記せず，エレメントから2本の線を出して表わ

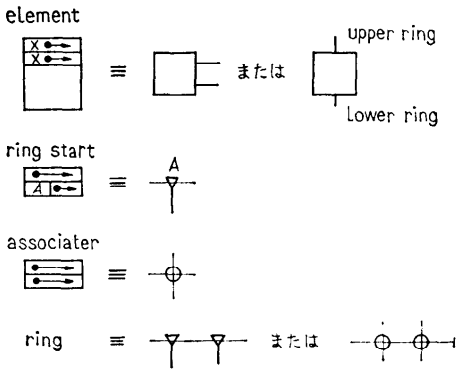


図 22　ASP の構成要素の図式表現

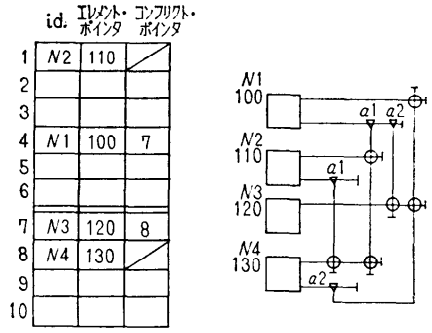


図 23　ハッシュ検索表をもった ASP (図中のエレメントにつけられた番号は，そのブロックの番地を示す.)

す.

ASP は，s-ASP と同様，各構成要素(エレメント，リング・スタートおよびアソシエータ) の各ブロックの形式が一定となり，計算機上での実現を容易にしている.

一般有向グラフ上での外在構造には，各点の id を指定してその点をアクセスする**直接検索**と，ある点から出発して関係をたどって他の点へアクセスする**局所検索**とが考えられる．ルート・グラフでは，直接検索の対象になる点はルートだけで，他の点はすべてルートからのパスによってアクセスされるが，一般有向グラフでは，すべての点を直接検索の対象と考える．この 2 つのモデルの間に，ある部分集合に属する点だけを直接検索の対象とするモデルを考えることもできる．これは，複数個のルートを持つグラフで，マルチルート・グラフと呼ぶ．実際に，多くの問題はこのモデルによって表わされる．マルチルート・グラフのデータ構造は，ルート・グラフの表現を拡張して容易に得ることができるが，ここではこれ以上ふれない.

各点の直接検索は，各点のもつ id によって行なわれるが，具体的な方法は，4. で与えた方法のどれを用いてもよい．その中で，点の数が増えても検索速度がほとんど遅くならない能率の良い方法にハッシュ検索がある．直接連鎖法を用いたハッシュ表を持ったリング構造の例を**図 23** に表わす.

直接検索によりある点をアクセスし，次にその点を出発点として局所検索を行う場合には，その点を含む関係を表わす 3 つ組 〈タイプ，始点，終点〉 が検索の基礎となる．それらの中には，始点とタイプを与えて終点を求める検索，終点とタイプより始点を求める検索などが考えられる.

弧の方向に沿っての検索は，ルート・グラフでの検索と大差ないので，逆方向の検索について考える．図 21 (a) の一般有向グラフで 〈$a1$, x, $N4$〉 を満足する点の集合 x を求めるには，まず図 21 (b) の $N4$ のエレメント・ブロックのUリングをたどり，アソシエータをアクセスし，そこでCリングに乗りかえる．次にCリングをたどってそのリング・スタートに達し，そこにおかれているタイプ情報が $a1$ であるので，そこでLリングに乗りかえて求めるべき点 $N2$ を得る．これを集合 x の要素とする．次にUリング上の次のアソシエータに対して全く同様の操作を行ない，点 $N1$ を得ることができる.

このように，逆方向の検索は図 21 (b) のような構造では，必ずしもすっきりしてはいない．これは，リング・スタートあるいはアソシエータに余分の情報(たとえば，各リングのリング・スタートへのポインタ) をおくことにより，検索操作を簡単化することができるが，それは一方では必要とする記憶量の増大と，構造を作り上げるときの手間の増加をひき起こすので，その得失の吟味が必要である[9].

連想 3 つ組構造

リング構造では，直接検索の対象が有向グラフ上の各点だけであるが，弧を表わす 3 つ組 〈タイプ，始点，終点〉 をも直接検索の対象とし，その上，その指定が不完全な場合にも連想機能により完全な 3 つ組をとり出すことのできるデータ構造に連想 3 つ組構造がある．有向グラフの場合に弧を表現する 3 つ組 〈タイプ，始点，終点〉 は，一般にはデータ項目間の関係の表現と考えられる．それを，対象となる物 (Object, O)，その属性 (Atribute, A) およびその値 (Value, V) の

3つ組と考え,

$A(O)-V$, $\langle A, O, V\rangle$

のように表わす．すると，上にのべた連想機能は，次のような7つの関数として考えることができる．

F0: $A(O)=V$

F1: $A(O)=x$

F2: $A(x)=V$

F3: $A(x)=y$

F4: $x(O)=V$

F5: $x(O)=y$

F6: $x(y)=V$

ここで，x，y は変数で，その値としては，その式を満足するようなデータ項目の集合をとる．またここでは，属性自身もデータ項目と考える．F0 は，他の F1～F6 と異なり，そのような3つ組がデータ・ベースの中にあるかどうかを確かめる論理関数で，答として‘真’または‘偽’の値をとる．この他に，関数全体の対称性を保つ意味で，

F7: $x(y)=z$

を考えることができる．これはデータ・ベース中にあるすべての3つ組をとり出す関数である．

以上のようなデータ構造をもつシステムに，LEAP[10], TRAMP[11], EDSP[12] 等がある．以下では，筆者等が開発した EDSP を参考にして，その連想処理機構を明らかにしよう．連想処理が実際に十分意味をもつためには，上記の7つの関数が高速に処理されることが必要であるが，それはハッシュ記憶法を用いて達成される．3つ組に対するハッシュ演算は，データ項目の id を直接その演算のオペランドとせず，一度データ項目用の検索表により検索を行い，そこで内部識別子（内部 id）に変換し，その内部 id をオペランドとする．内部 id としては，その検索表でのセルの番号を用いる．この検索表は，ASP で各点の直接検索のために必要とした表と同じで，この表自身もハッシュ記憶法を用いている．この表を Hash Dictionary Table (HDT) と呼ぶ．これに対して連想3つ組をおく表を Associative Triple Map (ATM) と呼ぶ．ATM は，A-page, O-page, V-page と呼ばれる A, O, V に対して対称な3つの部分表からできている．各表には，F1～F6 の各関数が，次のように対応する．

A-page: F1, F3

O-page: F4, F5

V-page: F2, F6

F0, F7 は，どの部分表を用いても効率に差はない．

いま，A-page の構成を考える．F1: $A(O)=x$ は，A と O を与えて，対応する集合 V をとり出す関数である．A, O, V に対応する内部識別子を a, o, v で表わす．F1, F3 に対して，a をアクセス・ワード，o をチェック・ワード，v をセマンティック・ワードと呼ぶ．各部分表は，等長のブロックにより細分化されていて，各ブロックは，機能の異なるいくつかのセルから成っている．各アクセス・ワードに対して，1つずつのブロックが割り当てられていて，アクセス・ワード a は，ブロック番地と考えることができる．このアクセス・ワード a を，チェック・ワード o により修飾し，新たなブロック番地を得る．この修飾がハッシュ演算に該当する．この修飾を，a と o を変数とするハッシュ関数 $h(a, o)$ と考えることができる．この関数としては，たとえば

$h(a, o)=a\oplus k\cdot o \bmod N$

などが考えられる．ここで N は部分表の大きさ（ブロック数），$\oplus$ はビット毎の排反和で，k は定整数である．こうして得られたブロックの特定セルに Value の集合を表わすリングのリング・スタートがおかれる．

このようにして作られた連想表は，F1: $A(O)=x$ に対しては，高速の検索が可能である．次に F3: $A(x)=y$ について考える．これは，$A(O_1)=x_1$, $A(O_2)=x_2$, …, $A(O_n)=x_n$ のように分解することができ，結局 F1 のために与えたリングのうち同じ A をもったものの集合を管理することに帰着される．この集合を表わすリングのリング・スタートは，a 番地のブロック中のセルに置けばよいことは明らかである．**図 24** に，図 21（a）の一般有向グラフ H に対する連想3つ組構造の表現を与える．この図は，いま述べた A-page だけを示しているが，O-page, V-page も A, O, V の役割を変えることにより容易に得られる．

EDSP におけるブロック内のセルの形式を**図 25** に示す．ハッシュのコンフリクトの処理は，直接連鎖法を用いている．また，セマンティック・ワードの集合は，もしその要素が1つだけのときには，リング表現によらず，リング・スタートのセル中にその要素をおく．また，ある3つ組を他の3つ組の要素として使う場合，それ自身をデータ項目として，HDT に登録しなければならないが，そのデータ項目の id (triple-id) は，その3つ組のセマンティック・ワードを表わすセル中におかれる．これらのセルの性質の区別をするためにフラグを置く．図中の自由セルは，複数個の

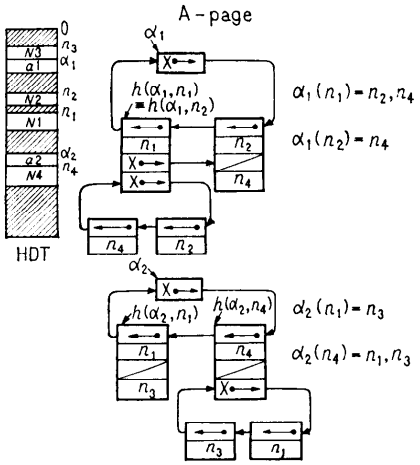


図 24 一般有向グラフHの連想3つ組データ構造による表現(図中で，$\langle\alpha_1, n_1\rangle$ と $\langle\alpha_1, n_2\rangle$ はコンフリクトを起している.)

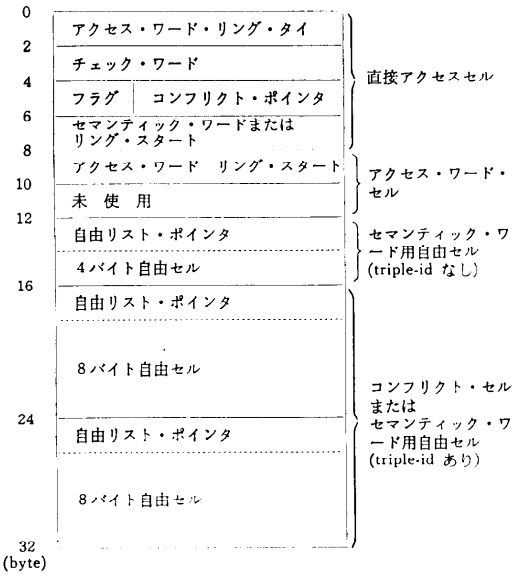


図 25 EDSP における ATM のブロックの形式

セマンティック・ワードを置くためと，コンフリクトを処理するためにある.

7. むすび

本講座では，データ構造を，内在構造と外在構造の複合体として考え，特に内在構造にグラフの各クラスを対応させて議論を進めてきた．外在構造については，表検索（直接検索）と局所検索をとり上げたが，

表 1 記憶構造の分類

	ツリー図式	ルート・グラフ	一般有向グラフ
sequential	タグ行列	——	——
simple list	ポインタ行列	複合ポインタ行列	——
ring	——	APL, s-ASP, SYLINDER	APL ASP
random	ハッシュ検索表	連 想 対	連想3つ組

より厳密な理論による展開が望まれる.

G. G. Dodd[13] は，記憶構造を sequential, random, list の3つに分類して論じているが，ここでは，list をさらに simple list と，ring に細分して，内在構造の各クラスにそれぞれあてはまる記憶構造を表にすると，表1のようになる．この表の中で (random, ルート・グラフ) の欄にある連想対については本講ではふれなかったが，連想3つ組との類推により，その機能および構造は推測されよう.

これまでにとり上げてきたデータ構造は，有向グラフを基礎にしているが，それは言い換えれば，2項関係あるいは3つ組をその対象としていることに他ならない．最近再び注目を集めてきた Artificial Intelligence (AI) の1つの分野に，Problem Solving あるいは，Theorem Proving などの研究があるが，そこでは，データ・ベースに，どのように“知識”をたくわえ，それをどのように利用するかが問題となる．その中で，C. Hewitt の開発した Problem Solving System を記述するのに適した問題向き言語 PLANNER[14] が特に注目を浴びているが，この言語の働きを支える基本的な能力の1つに，pattern directed matching があり，それを実現するには，一般の順序 n 組データの連想処理を必要とする．ここで n 組の各要素は，原子要素か，あるいは他の m 組である．このような，入れ子を許す n 組は，LISP 流のリスト構造によって表わすことができるが，n 組の集合よりなるデータ・ベースに対して連想処理をするには，連想3つ組での ATM のような連想表を作り，n 組の集合を管理しなければならない．これを連想3つ組のように，n 組のすべてのパターンに対して連想表を作れば連想処理自体は容易になるが，それには莫大な記憶量を要し，連想表の作成，変更にも長時間を要する．一方，n 組の各成分に対する連想表だけにすると，与えられたパターンをもった集合を得るには，連想表を検索した後にそこで得られた複数の集合間の集合演算が必要となる．PLANNER では，順序 n 組の各要素と，それが何番目の要素かを示す数字で対を作り，

そのような各対に対する連想表を作っている.

一般の順序 n 組を扱うデータ構造の研究には, Childs の Set Theoretic Data Structure[15)16)] (STDS) がある. これは, その名前の通り, 集合論の理論的考察より導き出されたデータ構造で, データ間の関係の操作はすべて集合演算で行なう. 順序 n 組に対しては, そのままでは集合演算ができないので, 'complex' という概念を導入してこれを解決している. 任意の順序 n 組は complex で表現でき, 集合演算は, 容易に complex 上の演算に拡張される. また, complex 上に順序を入れて, ソートを可能にしている. STDS の特徴の1つは, 強力なソートのサブ・ルーチンを持っていることである.

本講座では, データ構造をグラフ理論と関連づけながら説明してきたが, 実際にその処理システムを作ることを考えると, 本講ではふれなかったいくつかの重要な問題があるのを見落すことはできない.

最後にそれらの問題を列記しておく.

(1) 記憶領域の管理の問題[17)]. リストあるいはリング構造では, データ構造の更新にともなって必要とするブロックを確保したり, 不要になったブロックを後で再使用するために, 自由領域を管理しなければならない.

(2) 補助記憶の問題[18)]. 大きなデータ・ベースを作るためには補助記憶を用いる必要があるが, そのアクセスには時間がかかるので, その回数をできるだけ減らすための工夫が必要となる. また, セグメントの管理を要する.

(3) データ構造処理言語の問題[18)]. データ構造を操作する言語をどのように設計するかが問題となる. 使い易さを考えて, 他の高次言語に埋め込む方法がよく用いられる. また複雑な処理を自由に記述するためには, より低いレベルの言語が必要となる. この他, データ構造に対して recursive な処理を行う場合や, 処理のサブルーチン・コールを入れ子状に使いたい場合には, インタープリター形式の言語が有用になる.

参考文献

6) Dodd, G. G.: APL-a language for associative data handling in PL/I. FJCC, 1966, pp. 677～684.
7) Lang, C. A. & Gray, J. C.: APS-a ring implemented associative structure package. CACM Vol. 11, No. 8, Aug. 1968, pp. 550～555.
8) Weston, P. E. & Taylor, S. M.: CYLINDERS: a relational data structure. Proceedings of a Symposium on Data Structures in Programming Languages (ACM SIGPLAN) February, 1971, pp. 398～416.
9) 古川康一: コンピュータ・グラフィックスにおけるデータ構造の問題, 情報処理 Vol. 11, No. 9, 1970, pp. 523～532.
10) Feldman, J. A. & Rovner, P. D.: An algol-based associative language. CACM Vol. 12, No. 8, Aug. 1969, pp. 439～449.
11) Ash, W. L. & Sibley, E. H.: Tramp: An interpretive associative processor with deductive capabilities. Proc. ACM National Conference, 1968, pp. 143～156.
12) 古川康一, 山崎成美: 汎用データ構造処理システム EDSP について, 電総研彙報掲載予定.
13) Dodd, G. G.: Elements of Data Management Systems. Computing Surveys, Vol. 1, No. 2, June 1969, pp. 117～133.
14) Hewitt, C.: PLANNER: a language for manipulating models and proving theorems in a robot. Artificial Intelligence Memo No. 168, MIT (project MAC), Aug. 1970.
15) Childs, D. L.: Description of a set-theoretic data structure. FJCC, 1968, pp. 557～564.
16) Childs, D. L.: Feasibility of a set-theoretic data structure. IFIP Congress 68, pp. 420～430.
17) Knuth, D. E.: The art of Computer Programming, Vol. 1, Fundamental algorithms, pp. 406～451.
18) Williams, R.: A survey of data structure for computer graphics Csystems. omputing Surveys, Vol. 3, No. 1, March 1971, pp. 1～20.

(昭和47年6月2日受付)