コンピュータ・グラフィクスにおけるデータ構造の問題*

古川康一**

Abstract

Various kinds of data structures which have been so far developed in Computer Graphics to describe data with complex relations are investigated mainly in view of their abilities to represent the generalized directed graph. They are classified in terms of their capabilities. The author found out that the data structure based on the associative triple is equivalent in its ability to the most complex ring-based data structure such as ASP.

Thorough comparisons between the two structures are made with respect to memory space, search time, and some problems related to use of secondary storage.

1. はじめに

電子計算機を利用する立場から，そのシステムを分類すると，まず計算機に対して必要な指命なりデータなりを与える入力部分，与えられたプログラム，およびデータに従って計算を実行する演算部分，ならびに結果を人間にわかるような形で表現する出力部分とからなっている．

この場合，人間と計算機の情報のやりとりは，必ず入出力部分を介して行なわなければならず，入出力装置の能力によって，おのずからその利用形態が制約をうける．

現在，広くゆきわたっているバッチ処理システムでは，計算機を利用しようとする人は，まず問題を整理し，細かく分析して，それをプログラミング言語で記述し，カード（または紙テープ）に穿孔して計算機に読み込ませ，計算を実行させ，結果をラインプリンタに打ち出させるという手順を踏むのが，一般的な使い方である．

このように，非常に制約された利用形態を改善する目的で，いくつかの方向で研究が進められている．そのうちの1つは，計算機との情報のやりとりをもっと頻繁に行なって，人間-計算機系の効率を上げる研究で，on-line interactive system といわれている．もう一つの方向としては，入出力に図形を用いる研究がある．前者の例としては，TSS があり，後者の例には，XY プロッタ，CRT (Cathod Ray Tube) ディスプレイ装置，ST (Storage Tube) ディスプレイ装置などを入出力装置として使う研究がある．

On-line interactive graphic system の研究は，この2つの面を同時に持っている．これには，入出力装置としては，CRT ディスプレイ装置が最も一般的である．図形入力として，ライト・ペンを用いる場合と，Rand Tablet などを用いる場合がある．

コンピュータ・グラフィクスには，ハードウェアとソフトウェアの問題があり，それらは互いに関連しているが，ここでは現在実現されているハードウェアをもとにして，ソフトウェアの問題について考えることにする．

ディスプレイ上の図形を構成するデータは，計算機の記憶装置内にあると考えてよい．ディスプレイ上に図形を書かせるだけであれば，そのデータとしては，図形の各部分の座標を与えれば十分であるが，一般には図形の各部分は，いろいろな意味を持っており，また，その各部分どうしの関係も大切な意味を持っている．その図形に基づいて，ある種の計算をさせたり，図形の一部の変更・追加・削除などをしようと思えば，それらの意味までデータの一部分として，記憶装置内に持っていなければならない．このような複雑な関係を持ったデータを管理する形式を，一般に compound data structure[2] という．

このようなデータ構造 (data structure) は，もちろん，LISP 1.5, SLIP などのリスト処理言語を用いて作ることもできる[5]．しかし，その場合でも，論理的なレベルを1段上げて，各操作に新しい意味を持た

---

* The Problem of Data Structure in Computer Graphics, by Koichi Furukawa (Information System division, Electrotechnical Laboratory)

** 電子技術総合研究所情報システム研究室

せている.

また，このようなリスト処理言語の基本要素が，1語長あるいは2語長のセルであるのに対して，コンピュータ・グラフィクスを構成するデータ構造の基本要素は，もっと大きい単位のブロックを必要とする．そのために，不必要なリスト処理をも含むことになり，早い処理を要求するデータ構造に，これらのリスト処理言語を用いるのは不適当である.

2. 以降では，1つのセルが可変長のブロックで構成されるようなデータ構造だけを取り上げることにする.

データ構造の研究は，コンピュータ・グラフィクスのソフトウェアの研究の1つの眼目である．それによって，システムの効率，その能力などが，大きな影響をうける．ここでは，いままでに提案されてきた各種のデータ構造を紹介し，その能力・効率などの比較検討を行なうことにする．そのために，最も一般的なモデルとして，一般有向グラフ (generalized directed graph) の概念を導入し，その表現を通して，話しを進めていくことにする.

2. 一般有向グラフの計算機内部での表現 ——1——

有向グラフは，node の集合と，node の対を結ぶ方向を持った arc の集合として定義される．だが，各種の問題を有向グラフでモデル化したいとき，上記の定義だけでは表わせない部分がでてくる．たとえば，電子回路を考えてみると，素子を node で表現する場合，その素子の種類とか，その値などを記述できなければならない．ゆえに各 node は，任意のデータを持つことが許されなければならない．また，arc についても，それがただ単にある node から出発して，他の node に達する道であるということのほかに，その2つの node の結合の意味内容を示す情報（これを arc のタイプということにする）をも表わせることが必要である.

有向グラフにこの2つの性質 (node がデータを持つことと，arc がタイプを持つこと）をつけ加えたものを，一般有向グラフということにしよう.

有向グラフの一般化は，もちろん，もっと進めることができる．それは arc 自身がデータを持つ場合である．しかしこの場合は，その arc を2分し，その間に新しい node を1つ付け加え，その node に，もとの arc のデータを吸収させてしまえば，上のモデルに帰着させて考えることができる.

このようなモデルを考えると，node と arc の役割の区別がはっきりする．いいかえれば，情報の大部分を node に集中化し，arc は，node 間の関係に関する情報だけを持たせることにする.

このようにモデルを制限したのは，その方がより計算機内部での表現に適しているからであり，上述のように，モデルとしての一般性を欠いているものではない.

データ間に構造 (structure) を持ったデータを表現する手段として開発された種々のデータ構造は，この一般有向グラフの，計算機の記憶装置上での表現を与える．異なったデータ構造の比較を行なう手段として，同一の一般有向グラフのモデルを表現することを考えてみよう．そうすることによって，各データ構造の特徴を容易に引き出すことができるであろう．そのモデ

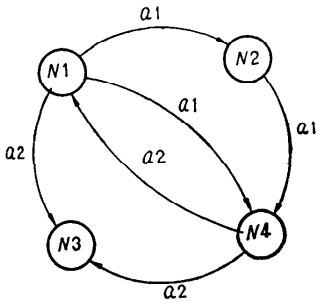

図 1 一般有向グラフによるモデルの表現

ルとしては，**図1**のようなものを考える．ここで $N1$〜$N4$ は，各 node の名前で，a_1〜a_2 は arc のタイプを表わす.

データを持ったトリー構造を表わすものとして plex[1] があるが，それを拡張して，つぎのようなデータ構造が考えられた．それは，node に対して一定の記憶領域を割りあて（それを element と呼ぶ)，node

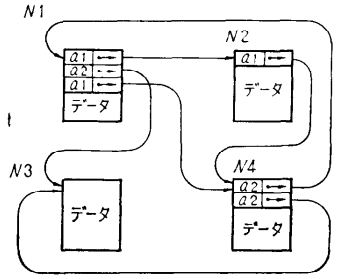

図 2 第1のデータ構造によるモデルの表現

間の結合を表わす arc は，それらの element 間をポインタで結合する方法である（**図2**）．

この方法では node $N1$ を見ればわかるように，1つの node から同じタイプの arc が出ているとき，その情報を，その node を表わす element 上に重複して持たなければならないことになる．これは，各 element でのリンク・ポインタ用の領域を，その node から出ていく arc の数だけ用意しなければならないことを意味する．また，1つの node から同じタイプの arc がたくさんでているときには，記憶領域はその分だけむだになる．

この点を解決するために，node あたりのリンク用の領域として，その node を始点とする arc の種類の分だけを用意すればよいようなデータ構造が考えられた．それは，1つの node から出ていく同じタイプの arc の終点となっている node の集合を，1つの ring (circular list) として表現する方法である．

この ring で出発点のある node だけは，他の node と性質が違うことに注意しなければならない．また，この ring は，node の部分集合の表現と見ることもできる．これは，多くの集合演算に対して能率のよい，わかりやすいアルゴリズムのあることを暗示している．

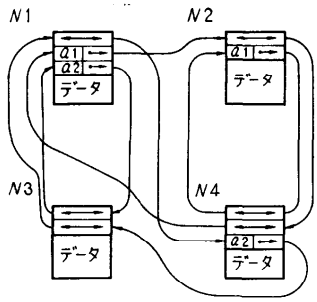

図 3　第2のデータ構造によるモデルの表現

前のモデルをこのデータ構造で表現したのが**図3**である．この図を前の図2と比べるとわかるように，リンク・ポインタの占める領域は，かえってふえてしまっている．これは，node にはいってくる arc に対しても，必ずポインタが必要となるからである．ただし，arc のタイプの情報は，図2よりも少なくなっている．

このデータ構造を基礎とした言語に，PL/I の拡張である APL[2),3)] (Associative Programming Language), CORAL[2)] (Class Oriented Ring Assoicative Language), それに SKETCHPAD[2)] などがある．APL では，この ring を set といい，この ring の出発点にあたるリンク・ポインタ (arc の始点の node を表わす element 上にある) を subset reference link といい，ring 上の他のリンク・ポインタを associative set reference link という．また SKETCHPAD, CORAL では，前者を ring start,後者を ring tie という．

この第1および第2のデータ構造の最大の欠点は，element を表わす記憶領域のうち，リンク・ポインタとして用意すべき領域の大きさが，モデルによって変わってくることである．つまり，初めにその大きさを決めてしまうと，それによって表現できるモデルが，そのリンク・ポインタの最大数によって制限されてしまい，モデルを変更したときに1つの node から出るリンク・ポインタの数が，許された最大数を越えてしまうときは，初めからすべてを作り直さなければならなくなってしまう．

そこで，つぎにこの点を改めて，モデルをどのように変更しても，容易にそれを表わせるようにするデータ構造が考え出された．それはつぎのような構造を持っている．すなわち，第2のデータ構造に出てきた element 上にある ring-start の集合を考え，その要素をまとめて管理する1段レベルの高い ring（集合の集合）をつくり，その ring の ring start を，始点となる node におくのである．同様に各 node を終点とする arc を表わす ring tie も，その集合を考えて ring とし，その ring の ring start のみその node が持てばよい．これはまた，element 上にあった ring start および ring tie のアレイを circular list で表わしていることと同じで，それによって構造の柔軟性を得ているのである．

いままでのモデルと同じものをこのデータ構造で表

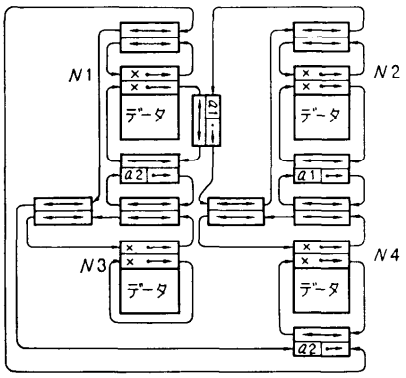

図 4　第3のデータ構造によるモデルの表現

現したのが**図4**である.

以上の考察でわかるとうり，この第3のデータ構造は，一般有向グラフの表現を与える最も高い能力を備えた方法であるといえる．このデータ構造の例は，ASP[2),4),5)] (Associative Structure Package) がある．ASP では，ring start の ring を lower ring (L ring) といい，ring tie の ring のことを upper ring (U ring) という．これら2つの ring の ring start は，element 上にある．

U ring の要素は，もう一つの ring に属している．その ring の ring start は，他の node の L ring の要素である．この2つの ring に同時に属している要素を associater という．つまり，ASP では，3種類の ring があって，それらがくさりのようにつながって node の結合を表現しているわけである (図 13 参照)．

ASP では，各要素 (element, ring start, associater) を**図5**のような記号を用いて表わす．L ring および U ring の ring start はとくに明記せず，element から2本の線を出して表わす．これらの記号を用いることによって簡単にその構造を図に書くことができる．図1のモデルは，**図6**のように表わせる．

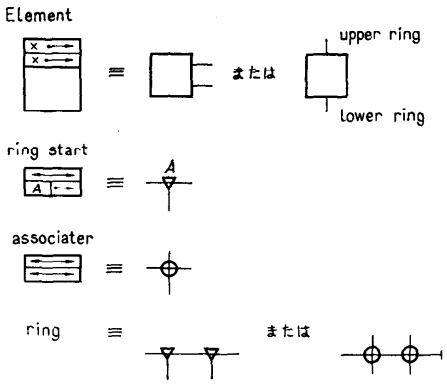

図 5 ASP の構成要素の図式表現

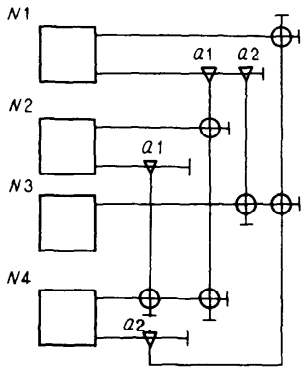

図 6 モデルの図式表現

また ASP では，element, ring start および associater の各セルのフォーマットが，それぞれ一通りに決まるのも特徴である．ただし，element のデータを格納する部分の大きさは，可変である．

上述した ASP の完全な構造柔軟性は，しかしその反面，操作の点からみれば，その複雑な構造を反映して時間がかかり，非能率的である．また，必要とする記憶容量も，前二者と比べると非常に大きくなる．同じモデルを表現するのに，第2のデータ構造の約2倍のリンク・ポインタ領域を要する (**4.** 参照)．

これまでの3つのデータ構造は，arc の表現にリンク・ポインタを使うという点で共通している．また，第2および第3の方法は，集合を ring で表わすという点で同じ性質のものである．第3のデータ構造は，集合の集合にも ring を用いて表現しているが，本質的に意味のある ring は，第2のそれに出てきた ring である．つまり，ASP でいえば，3重の ring の真中のものである．

一般有向グラフの表現として，いままで述べたものと全く異なった観点から開発されたデータ構造がある．それは上記の本質的な意味のある ring を，ポインタによらずに，他の方法で管理する方法である．それは，連想記憶装置のソフトウェア・シミュレーションの技法をもとにした方法で，データのインデックス・テーブルを拡張したものと考えることができる．以下次節で，その詳細を述べる．

3. 一般有向グラフの計算機内部での表現 ——2——

情報 (information) あるいは事実 (fact) を表わす単位として，対象となる物 (Object) とその属性 (Attribute) およびその値 (Value) の3つ組 (triple) を考え，この3つ組を連想記憶装置上に置くというアイデアが，J.A. Feldman によって提案された．それはつぎの式で表わされる．

Attribute (Object)＝Value

$A(O)=V$

例をあげると，ディスプレイの画面上の線分 LINE 1 の始点が POINT 3 である場合

STARTPOINT (LINE 1)＝POINT 3

と表わされる．Feldman はこの3つ組をデータ構造の基本的構成要素として，集合演算の機能を持つ言語 AL を作った．$\langle A, O, V\rangle$ の各成分は，空でない1つ以上の要素からできており，集合と考えることができる．その基本的演算は，$\langle A, O, V\rangle$ のうちの任意の成分を指定して，他の成分を引き出すことである．その演算の形式 (form) は，つぎの8つである．ここで x, y, z は変数（その値として集合をとる）を表わす．

$F0: A(O)=V$
$F1: A(O)=x$
$F2: A(x)=V$
$F3: A(x)=y$
$F4: x(O)=V$
$F5: x(O)=y$
$F6: x(y)=V$
$F7: x(y)=z$

ここで $F0$ は，演算結果として，'真' または '偽' の値をとる．他の $F1$〜$F7$ に対しては，その式を満足する item (A, O, V を構成する要素) の集合が答えである．

いま $F1: A(O)=x$ を考える．この問に対して有効に答えを引き出すために，"2重ハッシュ" 技法を用いる．それは，ある演算（ハッシュ演算)[9] を (A, O) の対にほどこして，一定のアドレス空間内（ハッシュ・テーブル領域）の一つのアドレスを得て，$A(O)=V$ を満足する V (Value) の集合をそのアドレス部分におくわけである．その場合，集合の表現は，前節に出てきた ring を用いる．こうすることによって，(A, O) を与えれば，直接 $A(O)=V$ を満足する Value の集合が得られるので，この機構は，連想記憶装置のソフトウェア・シミュレーションになっていることがわかる．この意味で，3つ組 $\langle A, O, V\rangle$ を連想3つ組 (associative triple) という．"2重ハッシュ" の方法はいろいろ考えられるが，最も簡単なものは，たとえば，A の名前と O の名前の exclusive or をとり，それをハッシュ・テーブルの容量で割った余りを $\langle A, O, V\rangle$ に対するアドレスとすればよい．

これだけでは，2つの変数を含む演算形式，たとえば $F3: A(x)=y$ に対する答えを引き出すのは大変である．そこで，同じ Attribute を持つ3つ組を1つの ring にまとめておく．ここで $F1$ と $F3$ に対する連想3つ組は，同じハッシュ・テーブル上に構成される．これを A-page という．演算形式全体は，A, O, V に対して対称であるから，この他に，O-page, V-page がある．これらはそれぞれ，$F4$, $F5$ に対するページおよび $F2$, $F6$ に対するページである．つまり，1つの連想3つ組 $\langle A, O, V\rangle$ は，3つのページ，A-page, O-page, V-page のすべてにおかれることになる．

実際に作製された例では，この他に，A, O, V を構成する要素 (item) の集合，およびそれらのデータを管理するテーブルを持っている．そのテーブルは，インデクスとデータを分離して管理してもよいし，一緒に一つのテーブルで管理してもよい．テーブル・サーチの効率を考えて，ハッシュ・テーブルにしてある．

また，$\langle A, O, V\rangle$ の3つ組の各対に対する "2重ハッシュ" は，この item のテーブルで変換された internal symbol を使って行なわれる．それは，連想領域におく情報量をできるだけ少なくするためである．

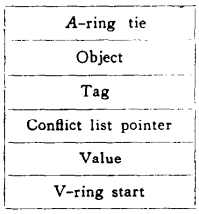

図 7　A-page を構成する連想3つ組のセル

連想3つ組の1つのセルの例を図7に示す．この図で Tag は，セルの種類を表わす．また，conflict list pointer というのは，異なった A, O のペアに対してハッシュ演算が同一の番地を与える場合（これを，コンフリクトという）それを解除するために，他にセルを割り当てるためのポインタである．

この連想3つ組を基礎にしたデータ構造を持っている言語に AL を拡張して作られた LEAP[6]，マクロ言語 TRAC を用いる TRAMP[7] および A. J. Symonds による PL/I を拡張した仕事[8]などが報告されている．それらは，言語の形態，機能など多少の差異はあるが，基本的な考えはすべて同じである．

さて，ここで紹介したデータ構造で，前節に述べた一般有向グラフを表わすにはどうすればよいであろうか．それは，つぎのように考えればよい．すなわち，1つの arc の始点に対して Object を対応させ，arc のタイプに Attributeを対応させ，終点の node の集合に Value を対応させる．与えられた A と O に対しては，連想3つ組によるデータ構造は，たしかに $A(O)=V$ を満足する node の集合 V を与えるので，

これによって，一般有向グラフが表わせたことになる．前節で扱ったモデルをここであげたデータ構造で表現すると**図8**のようになる．

ここでとり上げられたデータ構造は，一見，ASPのデータ構造と全く異なっているように見えるが，ある対応を与えれば，一方から他方への変換を行なうことができる．それは，ASP の element に "Object" を対応させ，ring start に "Attribute" を対応させ，その ring 上の associater を U ring 上に持つ element の集合に "Value" を対応させればよい．その対応関係の例を**図9**に示す．

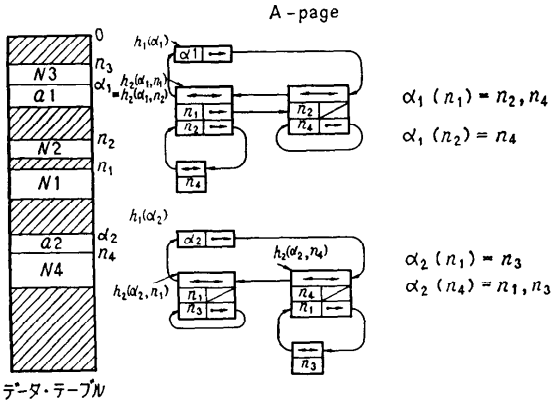

図8　連想3つ組データ構造によるモデルの表現

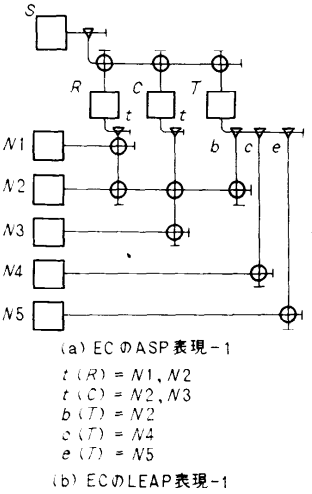

図 11

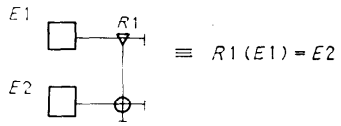

図 9　ASP による表現と，LEAP による表現の対応

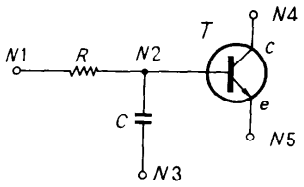

図 10　電子回路 EC

また，この対応によって，ASP での双対表現の意味が，それを LEAP 表現に写像することによって，一層明確になる．

いま，ASP で，**図10**のような電子回路を表現しようとすると，それは，**図11 (a)**，**図12 (a)** のように，2つの表わし方ができる．それに対応する LEAP 表現はそれぞれ**図11 (b)**，**図12 (b)** となり ASP における双対表現が，LEAP では逆関数による表現となることがわかる．ここで注意しなければいけないのは，図11(a) で ring start ▽ に与えたタイプを表わすラベル t は，図12 (a) では，それに相当する ring start ▽ に，逆関数 t^{-1} の形をとって表われるということである．

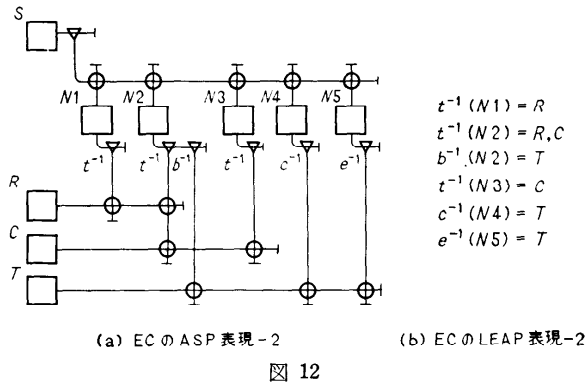

図 12

4.　各データ構造の比較

以上で，いままでこの分野でなされてきた仕事についての説明は終わるが，各データ構造の比較検討は不充分である．それらはつぎの点について調べられなければならない．

(1) 表現能力．

(2) 操作能力．

(3) 必要とする記憶容量．

(4) 操作にかかる時間.
(5) 使いやすさ.
(6) 2次記憶装置を使ったときの問題，たとえば，ブロックの作り方，操作時間など.

前節まででは，おもに (1) について述べ，(3)，(4) などについても若干述べた．(2)，(5) の問題は，データ構造の問題というよりも，むしろ，それを操作する言語の問題といってもよいであろう．これも重要な問題をいろいろ含んでいるが，ここでは触れないことにする．(6) の2次記憶装置を組み入れたデータ構造の問題も，大変重要な問題で，データ構造の研究における解決しなければならないテーマの1つとなっている．その1つの解が，連想3つ組によるデータ構造であるということもいえるであろう[5),8)]．リング構造に，2次記憶装置を組み入れることは容易なことではない．これは，リンク・ポインタを基礎にしているすべてのデータ構造に対していえることである．というのは，第1に，ポインタに含ませる情報として，主記憶領域の番地をそのまま利用することができなくなり，その取り扱いが複雑になること（ページ・セグメント方式でアドレス付けしなければならない）で，第2には，上手なブロッキングをしないと，ポインタをたどるたびに，2次記憶装置からのデータの読み出しが必要となるからである．またこのことは，データの更新のアルゴリズムにも影響をおよぼす．

それに比べて，LEAP では，連想3つ組を収容する領域のブロック化が行なわれている．同一のブロックになければならないのは，A ページを例にとると，1つの attribute に関連するデータすべてである．これは，その attribute に関して，1つの ring を構成する ($A(x)=y$ を満足する連想3つ組の集合)．$A(O)=x$ を満足する連想3つ組の集合は，上記の集合の部分集合である．もしこの集合が，前もって決められた容量のブロックにはいりきらないときは，他のブロックにまたがってしまうが，そうでなければ，前述の7つの FORM に対しておのおの1回の読み込みで答えをうる．LEAP ではこのアフレの処理を，可変長ブロックを用いて解決しているが，固定長ブロックだけを用いるような環境で，この問題をうまく解決することは困難である．それは，与えられた attribute に対して（A-ページを例にとると）どのブロックを割り当てるかを決めるアルゴリズムとも関係してくる．一般には，1つのブロックに複数個の attribute を割り当てるが，アフレが生じたときに，ある attribute を持った連想3つ組の集合を他のブロックへはき出させるためには，attribute によるページの指定が変更できなければならない．これは，各 item がブロック番号を持っていなければいけないことを意味する[8)]．これは，記憶容量をその分だけ要するので，望ましくない．もっとうまい方法を見つけるのが今後の課題であろう．

連想3つ組を基礎にしたデータ構造の代表として LEAP を考え，LEAP と ASP の比較をもう少し押しすすめてみよう．

LEAP で，A, O, V の任意の2つから残りの1つを引き出すということは，ASP では，2つのエレメントおよび arc のタイプのうちの任意の2つから，残りの1つを引き出すことになる．図6で $N1$, $a1$, を与えると，まず $N1$ の L ring をたどって $a1$ を値とする ring start をみつけ，つぎにこの ring をたどって，その上にある associater をすべて引き出して，その associater を要素とする U ring をたどることよって，2つの node $N2$, $N4$ が引き出せる．

逆に $N2$, $a1$ を与えて $N1$ を引き出すこともできるが，これが有効にできるためには，arc のタイプの情報 $a1$ を $N2$ の上にある associater に持たせておかなければならない．そうすれば，3つの ring をたどる順序を逆にすれば，$N1$ をみつけることができる．

この場合，不要なリストのたぐりをさけるために，各 ring tie に，その ring の ring start へのポインタを持たせることも有意義であろう．そうすれば，最初の例で，($N1$, $a1$) から $N2$, $N4$ を求めるときに，associater を見つけてから，それが含まれている U ring をたどる必要がなくなるからである．この場合の ring のたどり方を図13に示す．

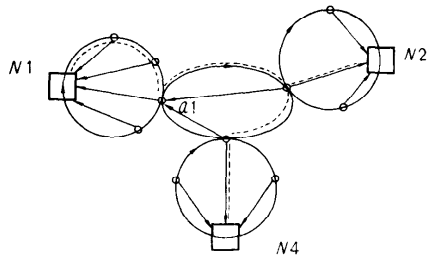

図 13　ASP で ($N1$, $a1$) から $N2$, $N4$ を求める方法

そうしたとしても，第1の ring をたどって与えられたタイプ $a1$ を含むような ring tie を見い出すという操作は残る．

最後に残った問題は，$N1$ と $N2$ を与えて $a1$ を引き出すことであるが，これは，上の2つの場合よりも複雑な，集合演算を要する．それは，$N1$ の L ring 上の ring start の集合と，$N2$ の U ring 上の associater を要素とする ring の ring start の集合との共通集合を求める演算である．

図13でわかるとおり，$N1$ から出発して $N2$ に至るまでのポインタをたぐる操作は，$k+2$ となる．ここで k は，$N1$ の L ring 上での $a1$ の ring start が何番目にあるかを示すものである．$N1$ の L ring 上にある ring start の数を n とすると，平均の操作回数は $(n/2)+2$ となる．それに対して LEAP では，コンフリクトの処理を考慮すると，平均2回以下の操作で上と同じことができる[9]．ただし，1回の操作に要する手間は，ASP でのそれに比べて，多少長くなると思われる．

また LEAP では，上にあげた3つの問題に対して同じ方法で処理でき，その平均操作回数も同じになるが，ASP では前述のように $(N1, N2)\rightarrow a1$ という問題に対しては，他の2つよりも余分の操作を必要とする．

また LEAP で簡単にできる A, O, V の1つを与えて他の2つを引き出す問題では，ASP では，前述した A, O, V の2つを与えて，他の1つを引き出す方法を組み合わせて行なわなければならないので，大変厄介である．

これまでは，おもにサーチの効率の比較を論じたが，操作時間に関しては，この他にモデルの変更の手間についても考えなければならない．それは，おもに新しいエレメントの追加，あるいは，すでにあるエレメントの削除である．追加あるいは削除に伴うサーチの手間は，すでに述べたとおりであるが，LEAP の場合，追加に対してはサーチを要しない．ASP では，エレメントの追加または削除に伴って，自由記憶領域の管理を要する．これには L^6 のようなブロック処理言語を使うことができる[10]．circular list へのエレメントの追加・削除の効率は，それが，片方向か両方向かによって異なる．片方向の circular list では，その list 上の与えられたエレメントを削除する場合でも，その list の頭からたどる必要がある．それは，その前のエレメントのリンク・ポインタの内容を書き換えなければならないからである．モデルの構造によらない一定の操作時間を保証するためには，両方向のリンクを持たなければならない．

一方，LEAP では，1つの連想3つ組 (A, O, V) の追加・削除は，各ページについて行なわなければならず，一ぺんに3箇所の変更を要する．

以上は操作時間の比較であるが，必要とする記憶容量の比較をしてみよう．ただしこれは，2次記憶装置をシステムに組み込むかどうかによって，その意味するところの重要性が全く変わってしまう．いいかえれば，2次記憶装置の使用の問題の方がずっと大切であるということである．

集合 S の要素の数 (cardinarity) を $|S|$ で表わすことにする．一般有向グラフについて，つぎの集合を定義する．

N: node の集合．
N_s: 始点となる node の集合，$N_s \subset N$.
N_d: 終点となる node の集合，$N_d \subset N$, $N_s \cup N_d = N$.
A_r: arc の集合．
$T_{s(ni)}$: $n_i \in N_s$ から出る arc のタイプの集合．
$T_{d(ni)}$: $n_i \in N_d$ にはいる arc のタイプの集合．
T_t: arc のタイプの集合．

また $T_{s(ni)}$, $T_{d(ni)}$ から，μ_s, μ_d をつぎのように定義する．

$$\mu_s \equiv \left(\sum_{n_i \in N_d} |T_{ni}|\right)/|N_s|$$

$$\mu_d \equiv \left(\sum_{n_i \in N_d} |T_{ni}|\right)/|N_d|$$

これらはそれぞれ始点および終点での，arc のタイプの node 当りの平均である．

(1) ASP (APL) での pointer 領域の容量は，
各 node にある2つの ring-start: $2|N|$
L ring 上の ring-start: $4|N_s|\mu_s$ $(2|N_s|\mu_s)$
associater: $3|A_r|$ $(2|A_r|)$
となり，合計で $2|N|+4|N_s|\mu_s+3|A_r|$ $(2|N_s|\mu_s+2|A_r|)$ となる．

(2) LEAP での pointer 領域の容量は
・A-page について
(A, O) のペアの総数 : $5|N_s|\mu_s$
連想3つ組のうちの上の残り: $2(|A_r|-|N_s|\mu_s)$
A のリストのヘッド : $2|T_t|$
・O-page について
(O, V) のペアの総数 : $5|A_r|$
連想3つ組のうちの上の残り: 0
O のリストのヘッド : $2|N_s|$
・V-page について
(A, V) のペアの総数 ・ $5|N_d|\mu_d$

連想3つ組のうちの上の残り：$2(|A_r|-|N_d|\mu_d)$
V のリストのヘッド　　：$2|N_d|$

以上の合計は

$$3|N_s|\mu_s+3|N_d|\mu_d+9|A_r|+2|N_s|+2|N_d|+2|T_t|$$

となる．$|N_s|=|N_d|=|N|$，$\mu_s=\mu_d$ とするとこれは

$$6|N|\mu_s+4|N|+9|A_r|+2|T_t|$$

となる．

LEAP で，2つの変数を含む問に答えるためのリンク・ポインタの領域の容量を差し引くと，各ページでの容量が

A-page: $4|N_s|\mu_s+2(|A_r|-|N_s|\mu_s)$
O-page: $4|A_r|$
V-page: $4|N_d|\mu_d+2(|A_r|-|N_d|\mu_d)$

となり，合計で

$$2|N_s|\mu_s+2|N_d|\mu_d+8|A_r|$$

となる．これを ASP の容量と比べると，$|N_s|$, $|N_d|$, μ_s, μ_d, $|A_r|$ の比によって異なるが，たとえば，$|N|=|N_s|\mu_s=|N_d|\mu_d=|A_r|/2$ とすると，10：6 となり，1.67倍の容量を要する．APL の pointer 領域の容量と比較すると，この場合3.3倍となる．この1.67倍を要する原因は，コンフリクトを処理するための，2-word の領域である．その分 $(2|N_s|\mu_s+2|N_d|\mu_d+2|A_r|)$ を差し引くと，上の例では，APL と LEAP の比は1となる．

図1の例では，$|N_s|=3$, $|T_{s(N1)}|=2$, $|T_{s(N2)}|=1$, $|T_{s(N4)}|=1$, $|N_s|\mu_s=4$, $|N_d|=4$, $|N_d|\mu_d=4$, $|A_r|=6$, $|N|=4$, $|T_6|=2$ となるので

APL: 20
ASP: 42
LEAP: 64 (A-ring, V-ring, O-ring なし)
　　: 96 (A-ring, V-ring, O-ring あり)

となる．

5. むすび

前節で詳しく述べたように，LEAP および ASP は，それぞれの特徴を持っていることがわかる．LEAP は ASP に比べて，とくにサーチの効率がよいことがわかる．モデルの変更の手間は，どちらが速いかは，判断がつきかねる．また，記憶容量の点では，ASP が LEAP の約半分以下であることがわかる．また，モデルが簡単な場合には，もっと簡単なデータ構造，たとえば APL，CORAL などの方が，効率，記憶容量の両方で ASP よりもすぐれていることがわかる．

データ構造の研究には，この他に，集合論を基礎にした研究や[12),13)]，IR (Information Retrieval) の質問の形に対する組合論的考察を基礎にした研究[14)]などの，基礎的な研究が発表されている．集合論による接近は，連想3つ組 $A(O)=V$ を，もっと一般的な関係 $R(x)=y$ としてとらえる．ここで R は，x と y の関係 (relation) である．これを，x と y の順序対の集合 $\{\langle x, y\rangle\}$ で表わす．また，順序対を順序 n-組 (ordered n-tuple) にまで拡張して，その集合を一般的関係と定義し，そのような関係に対して，一般の集合論の演算ができるようなデータ構造を作る研究がなされている．

組合論的考察では，質問の形に応じて，どのようなインデクスの集合をつくっておけばよいかということを，質問の形にある制限を設けて求めている．

これらの考察から出てきたものが，実際のデータ構造の作製に役立つには，いろいろな問題があると思われるが，統一した視点でシステムをいかに作り上げるか，興味のあるところである．

一方，これらのソフトウェア的考察とならんで，ハードウェア的に，データ構造に適した機能を追求するという研究も進められている．連想記憶装置の研究，ハードウェアによるリスト処理[15)]などがその例である．

ハードウェアの持つ能力によって，そこに実現されるデータ構造の能力も当然違ってくる．今後の研究で必要なことは，ハードウェア，ソフトウェアの両面を考えて，ハードウェアの持つべき基本的な機能は何かということを考え，それに対してソフトウェア・システムを考えることであろう．

末筆ながら，日頃ご指導いただいているソフトウェア部長西野博二博士，電子計算機部長野田克彦博士，計算機方式研究室長和田磯秀夫博士，加藤雄士主任研究官，淵　一博主任研究官および討論してくださった研究室のみなさまに感謝する次第である．

参考文献

1) D. T. Ross & J. E. Rodrigues: Theoretical foundations for the computer-aided design system. SJCC, 1963.
2) J. C. Gray: Compound data structure for computer aided design; a survey. Proc. ACM National Conference, 1967.
3) G. G. Dodd: APL-a language for associative data handling in PL/I. FJCC, 1966.

4) C. A. Lang & J. C. Gray: ASP-a ring implemented associative structure package. CACM Vol. 11, No. 8, 1968.
5) N. E. Wiseman & J. O. Hiles: A ring structure processor for a small computer. Computer Journal, Vol. 10, No. 4, 1968.
6) J. A. Feldman & P. D. Rovner: An algol-based associative language. CACM Vol. 12, No. 8, August, 1969.
7) W. L. Ash & E. H. Sibley: Tramp: An interpretive associative processor with deductive capalilities. Proc. ACM National Conference, 1968.
8) A. J. Symonds: Auxiliary-storage associative data structure for PL/I. IBM. SYST. J. Vol. 7, No. 3 & 4, 1968.
9) R. Morris: Scatter storage technique. CACM Vol. 11, No. 1, Jan. 1968.
10) K. C. Knowlton: A programmer's Description of L^6. CACM Vol. 9, No. 8, August, 1966.
11) D. E. Knuth: The art of computer programming, Vol. 1/Fundamental Algorithm. 2. 2 Linear Lists. ADDISON-WESLEY, 1968.
12) D. L. Childs: Description of a set-theoretic data structure. FJCC, 1968.
13) D. L. Childs: Feasibility of a set-theoretic data structure. IFIP Congress 68.
14) D. K. Ray-Chaudhuri: Combinatorial information retrieval systems for files. SIAM J. Appl. Math. Vol. 16, No. 5, September 1968.
15) J. K. Iliffe: Basic machine Principles. MACDONALD, 1968.