第9章 木構造

木とは／木の表現／2分探索木

第8章では線形リストの解説をしました．線形リストというのは1次元の連結リストでしたが，この章で解説する木の構造は2次元あるいは多次元の連結です．基本的な考え方は同じですが，使い道がさらに拡大します．

9.1 木とは

我々の世界の木は地面に根を生やし，幹が地表に出て，枝分かれし，末端は葉となる上方向に伸びた木です．下が元で，上が広がった形です．

コンピュータの世界でも木構造があります．もちろんこれは，論理的なつながりをもったデータ群をイメージしたものです．コンピュータの世界で木を想定するときは，根や幹が上方，枝や葉を下方に配置して考えます．別にどちらが上でも下でもかまわないのですが，根幹という言葉があるように，基軸となるところ，中心となるところを上方にする習慣からです．

企業や団体の組織図というのがあります．これもやはり××長という基幹となる人が上方にいて，各部門に下方に枝分かれして，ちょうどピラミッドといわれるような形に構成されています．これと同じで，そのほうが考えやすかったから，そういう形態にしているだけのことです．

9.1.1 木要素の名称

論理的にデータ群を関連付けたイメージを**図9.1.1**に示します．丸印にABCと表記した部分が個々のデータ（オブジェクト）になります．

木構造については用語が多いので，明確にしておきます．根，枝，葉など実際の木を想定すれば，どこの部分かを予想はできますが，**表9.1.1**にまとめて示します．家系図に似せて，親，子，兄弟，孫などと呼ぶこともあります．そして子が二つある場合に，左の子（left child）や右の子（right child）とも呼びます．また，部分的に見て，同様に親子の関係が成り立っている部分を部分木という呼び方もします．

また，根から一番遠い葉までの枝の数を木の高さ（height）といいます．同図の木の高さは3です．根から

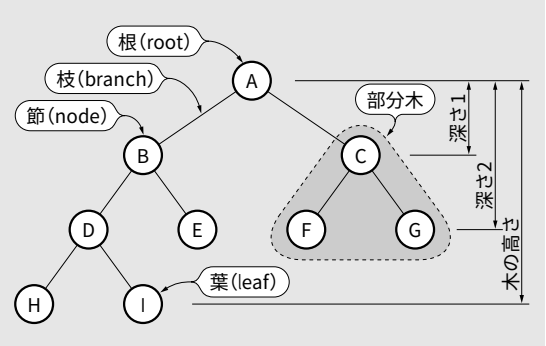

図9.1.1　木の構造と名称

表9.1.1　木の要素の名称

木の要素		意　味
木	tree	定められた条件を満たす節と枝の空でない集合
節(節点)	node	単一のオブジェクトで枝が集まっているところ．頂点(vertex)ともいう
枝	branch	二つの節の結びつき(関連)を示す．辺(arc，edge，branch edge)ともいう
葉	leaf	子のない節．終端点(terminal node)，外点(external node)ともいう
根	root	枝が入ってこない最初の節
親	parent	根を除く節から上位のレベル
子	children	自分のすぐ下にある節点
兄弟	sibling	同じレベルの子
祖父母	grandparent	二つ上位の親．反対に孫と呼ぶこともある
内点	internal node	子をもつ節．非終端頂点(non-terminal node)ともいう
外点	external node	子をもたない節．終端頂点(terminal node)ともいう

表9.1.2　木の種類

木の種類		内　容
部分木	subtree	任意の節点とその下にある節点
順序木	ordered tree	節が複数の子(兄弟)をもつとき，子の間に順序がある木
2分木	binary tree	子の数が二つ以下の順序木
完全2分木	complete binary tree	節の左右にある枝の数が等しい2分木．根から葉までの距離はどこでも等しい
2分探索木	binary search tree (BST)	節の配置にルールをもたせ，探索できるようにした木
M分木	M-array tree	節の子の数がM(M>2)個までの木．多進木(multi-way tree)ともいう
バランス木	balanced tree	木の深さがほぼ等しい木．平衡木．代表的なのはAVL木
AVL木	AVL tree	どの節でも左右の部分木の高さが1しか違わない木
B木	B tree	すべての葉の深さが同じ，バランス多分木．平衡木の一種

葉までの枝の数を深さ(depth)という呼び方をし，根は深さ0，その子は深さ1で孫は2となります．この例では，G節は深さ2の位置ということになります．

9.1.2　木の種類

木はその構成内容により，**表9.1.2**に示すようなさまざな種類，呼び名があります．代表的なものについて解説をしておきます．また名称だけではわかりづらいので，これらの相関を**図9.1.2**に示します．このうちヒープについてはデータの整列(11.9節)で解説します．

■ 完全2分木

各節の左右にある枝の数が等しい2分木を完全2分木と呼びます．もし充足されていない場合でも，深さの浅いほう，かつ左のほうから詰まっている2分木のことです．言い換えれば，一番下のレベルを除く全てのレベルに節が詰まっていて，一番下のレベルでは左から順に節が詰まっている2分木です．**図9.1.3**に示したように，節Fには左の子しかありませんし，節Gには子がなく，いわゆる葉になっています．しかし深さ2までは完全に詰まっていますし，深さ3のレベルでは左から順に充足されています．

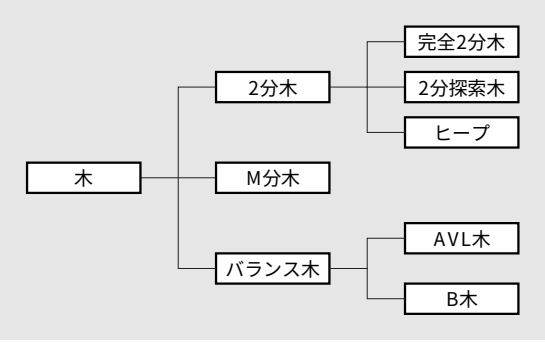

図9.1.2　木の関連

n個の節をもつ完全2分木の高さは一般には，

$\log_2(n+1)$

で表されます．正確には，

$\log_2(n+1)$ -1 以上の最小の整数

です．トーナメント戦の優勝チームの最大試合数と同

じです．

■ 2分探索木

左の子，親，右の子の，大小関係などの序列に一定のルールを持たせた配置にした木のことをいいます．探索に利用する構造で，詳細は9.3で解説します．

■ AVL木

1962年，Adel'son-Verl'skiiとLandisにより考案されたものです．**図9.1.4**に示すように全ての節において左部分木と右部分木の高さの差が1以内になっている2分木をAVL木と呼びます．

節EやGには一つの子しかありません．しかし節B以下の部分木を見たときに，BDEの高さは3で，BEでは2です．差は1です．C以下を見ても同様です．

■ B木

多進木の一種です．1972年，BayerとMcCreightによって考案されたものです．次のような条件を満たす木をm階のB木と呼びます．

・根は2個以上，m個以下の子をもつ
・根，葉以外の節は，$m/2$個以上m個以下の子をもつ
・全ての葉が同じ深さである

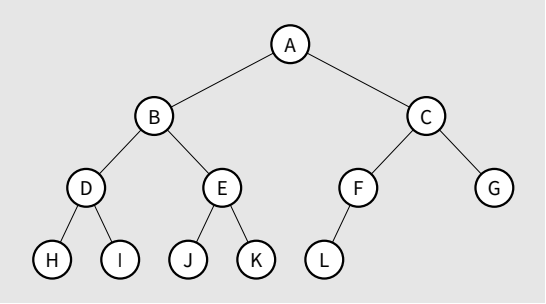

図9.1.3　完全2分木

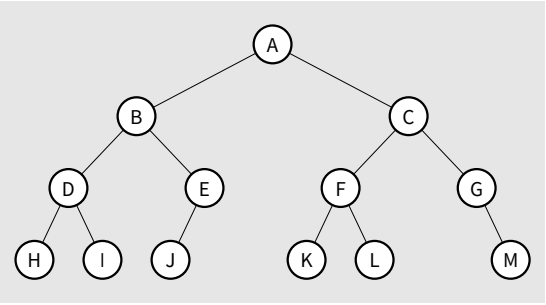

図9.1.4　AVL木

少々わかりづらいのですが，2個以上のm個の節を持つm分木をもとにしたものです．そしてB木は(末端の)葉だけにデータを持っていて，葉以外の節(内部節という)はキーだけを持つことになります．具体的な例を**図9.1.5**に示します．

各節はキー・データの部分とポインタの部分から構成されます．一般的にはキー部$2n$個に対して，ポインタは$2n+1$個あります．そしてキー・データというのは，格納されたデータそのものではなく，以下の節，あるいは葉のデータ群の境界値を示すものです．図ではポインタ部，境界値部としてあります．

ポインタ部は，他の木構造でもそうであるように，次の節または葉へのポインタを格納しています．境界値部は，次の節または葉が含む値の境界値です．境界値の左をAグループ，右をBグループと呼ぶことにすると，Aグループに含まれるデータは境界値未満であり，Bグループに含まれるデータは境界値以上であることを示しています．

このような構成の節が何階層かあり，実際のデータは葉の部分に格納されています．これで前述の，全ての葉が同

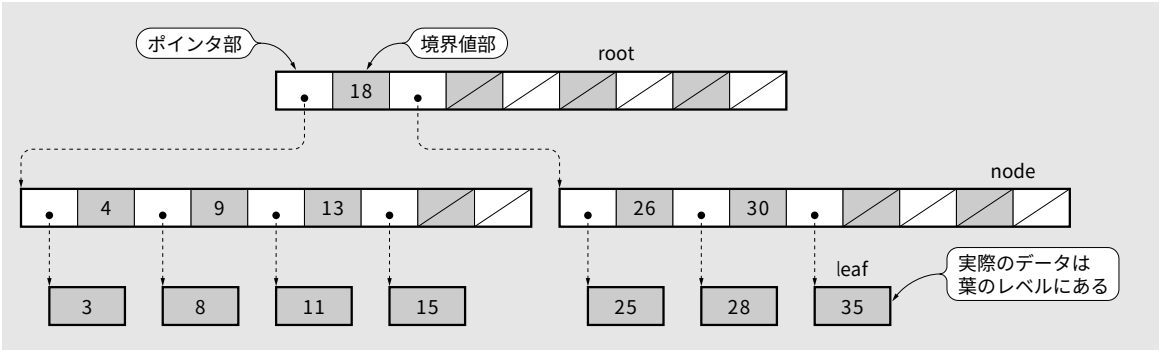

図9.1.5　B木

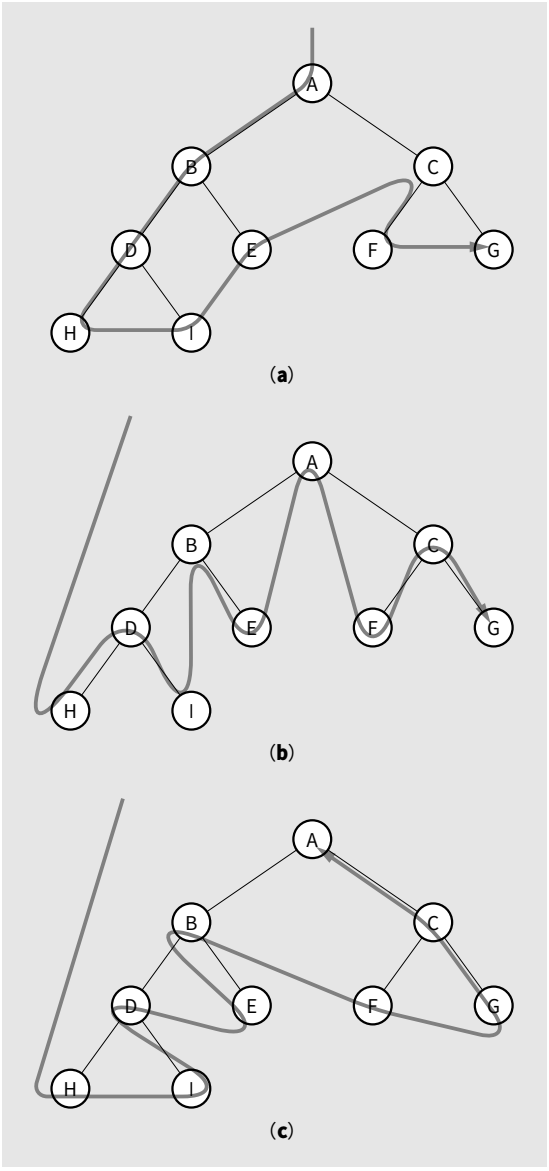

図9.1.6　木の走査

じ深さであるという木が構成されています．B木は挿入や削除の際の，動的な再編成が効率よく行えること，データ総数の割に深さを浅くでき，検索の効率がよいことなどから利用頻度の高い木構成方法です．リレーショナル・データ・ベース(relational data base)の骨格として多用されています．

9.1.3　2分木の走査

2分木の節を順にたどる(traverse)ことにより，データが読み出せることになります．これを木の走査(tree traversal)といいます．たどり方には，先行(preorder)順，中央(inorder)順，後行(postorder)順などがあります．データの配置方法と走査の順により，見かけ上データの並びを変えることができます．

① 先行順走査

先行順とは，まず節に立ち寄り，左部分木，その次に右部分木に行く，という順にたどる方法です．**図9.1.6**(**a**)のように，ABDHIECFGというルートです．行きがけ順とも呼ばれます．

② 中央順走査

通りがけ順ともいいます．先ず左部分木を走査，次に節点，続いて右部分木の順に走査する方法です．**図9.1.6**(**b**)のように，HDIBEAFCGの順になります．

③ 後行順走査

帰りがけ順ともいいます．左部分木を走査し，次に右部分木を，そして節を走査するという順にたどる方法です．**図9.1.6**(**c**)のように，HIDEBFGCA の順に走査する方法です．

いずれの走査も，プログラムで実現しようとすると，木の構造は部分木の集まりが階層的になっていることから，再帰の構造で構成できます．具体例は後述します．

9.2　木の表現

このように論理的にリンクされた木構造の表現をC言語ではどのようにあつかうのかを考えます．前述リスト構造と同様，ここでも自己参照構造体が威力を発揮します．

9.2.1　節の構造体

節を作るのは自己参照構造体です．自分自身のデータのほかに，子を指すポインタを含んでいます．いま2分木について考えます．**図9.2.1**(**a**)のような論理的にリンクされている木があるとき，その物理的な繋がりは**図9.2.1**(**b**)のようなものです．

節Aについては，データAを収容していると同時に，その子B，Cに対するポインタを内蔵しています．他の節に