木構造

階層を持ったデータ構造

木構造（Tree Structure）は、階層を持ったデータ構造です。ある階層に属する一つのデータから、下位階層に位置する複数のデータが枝分かれした状態で配置されています。各階層は親子関係を持っており、親は複数の子を、子はただ一つの親を持ちます。

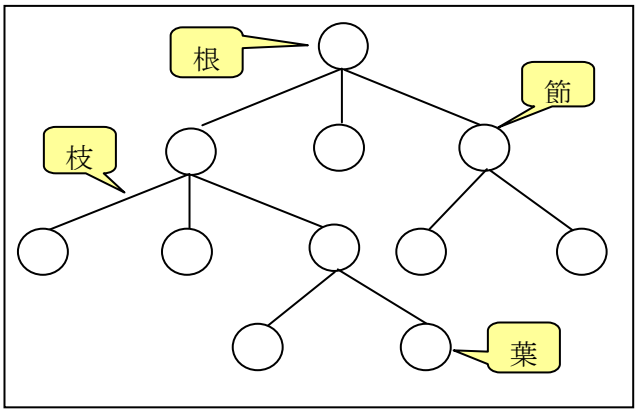

図 1. 木構造

木構造の、要素部分を**節**（ノード node）、子のない末端の節を**葉**（リーフ leaf）、親のない先頭の節を**根**（ルート root）といいます。要素と要素のつながりは、**枝**（ブランチ branch）です。ある節とその下の部分を取り出すと、それだけでも木構造となっています。これを**部分木**といいます。

木の種類

木構造には色々なものがありますが、最も重要なのは**２分木**（Binary Tree）です。２分木とは、各節が多くても２つの子しか持たない木のことです。２分木に対して、子が 3 つ以上ある節を持つ木を**多分木**といいます。

２分木の節の左側の部分木を左部分木、右側の部分木を右部分木といいます。

木の探索法

木構造で表されたデータを処理するためには、木の各節を順番にたどる必要があります。この節を巡回して探索する方法には、**幅優先探索**と**深さ優先探索**があります。

[幅優先探索]

根から開始し、同じレベルにある節を横方向に探索していく方法です。

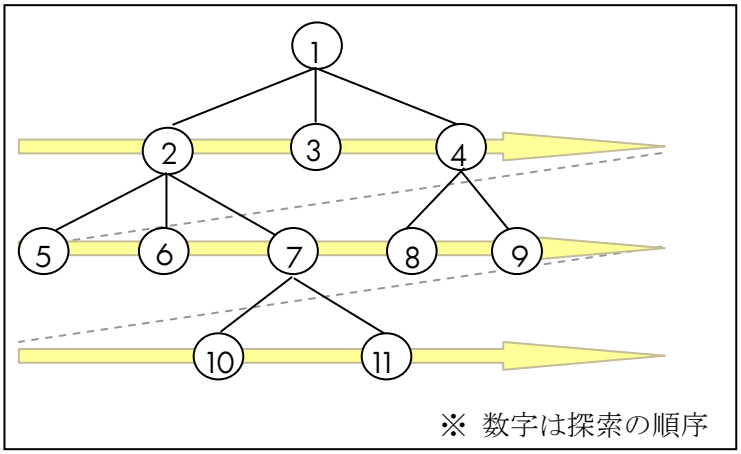

図 2. 幅優先探索

[深さ優先探索]

縦方向を優先して節を探索していく方法です。2 分木の深さ優先探索には、**先行順**（preorder）、**中間順**（inorder）、**後行順**（postorder）があります。先行順は親から左部分木、右部分木の順に探索します。中間順は左部分木から親、右部分木の順、。後行順は左部分木から右部分木、親の順に探索します。

表 1. 深さ優先探索

方法	説明
先行順	親→左部分木→右部分木の順
中間順	左部分木→親→右部分木の順
後行順	左部分木→右部分木→親の順

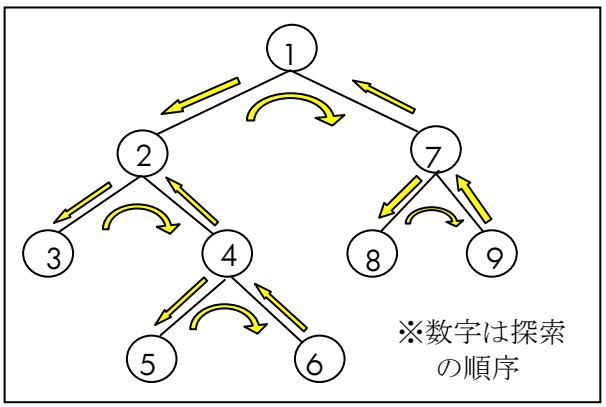

図 3. 先行順探索

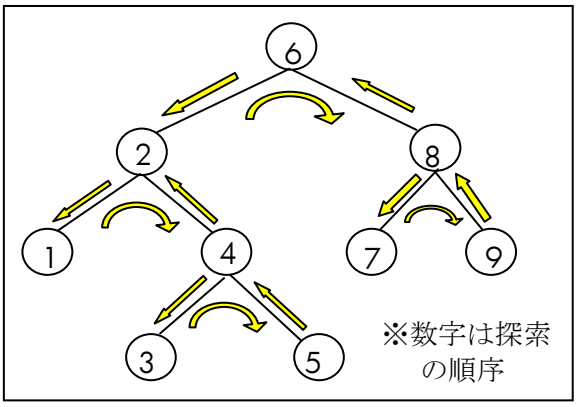

図 4. 中間順探索

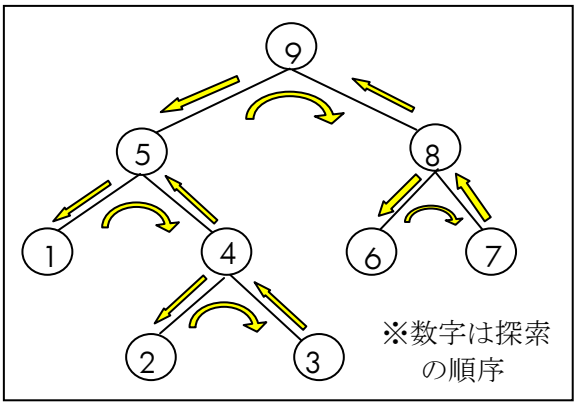

図 5. 後行順

数式を、木構造で表現することができます。次の図は、「A＊(B－C)＋D／E」という数式を表しています。

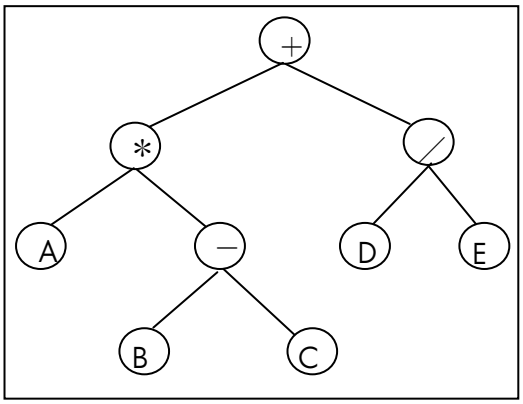

図 6 数式の木表現

この木を先行順にたどると前置記法（ポーランド表記法)、中間順にたどると中置記法（通常の記法)、後行順にたどると後置記法（逆ポーランド表記法）を得ることができます。

[再帰による探索]

木構造は、再帰的な構造をしています。したがって、その探索には再帰を用いると便利です。数式の木表現を先行順にたどって前置記法を求めるアルゴリズムは、次のようになります。

```
trace_preorder( TREE   *root )  {
      if   (root == NULL)
            return;
      print_node( root );                    // この部分木の根の記号を出力
      trace_preorder( root->left );          // 左部分木を探索
      trace_preorder( root->right );         // 右部分木を探索
}
```

★ trace_preorder の中で、自分自身（trace_preorder）を呼び出しています。

これを、中間順にたどって通常の記法を求めるアルゴリズムにしてみます。

```
trace_inorder( TREE   *root )  {
      if   (root == NULL)
            return;
      trace_inorder( root->left );           // 左部分木を探索
      print_node( root );                    // この部分木の根の記号を出力
      trace_inorder( root->right );          // 右部分木を探索
}
```

★ このアルゴリズムは、括弧を扱っていないため、不完全です。

後行順にたどれば、後置記法を求めることができます。

```
trace_postorder( TREE   *root )  {
      if   (root == NULL)
            return;
      trace_postorder( root->left );         // 左部分木を探索
      trace_postorder( root->right );        // 右部分木を探索
      print_node( root );                    // この部分木の根の記号を出力
}
```

3 つのアルゴリズムの違いを比べ、そのシンプルさを味わって下さい。

注 : 木構造を表すデータ構造 TREE を、C 言語で書くと次のような構造体になります。

```
struct   TREE {
   struct   NODEDATA      data:          // この節のデータ
   struct   TREE          *left;         // 左部分木へのポインタ
   struct   TREE          *right;        // 右部分木へのポインタ
}
```

（以上）