ツリー

山本昌志*

2007 年 7 月 17 日

概 要

ツリーと呼ばれるデータ構造の学習をする．特に 2 分木について，操作方法を学ぶ．

1 本日の学習内容

1.1 前回の復習

前回は，スタックとキューというデータ構造を学習した．

スタックの特徴は，最後に入れたデータが一番最初に取り出されることにある．取り出されるデータは，格納されている最新のデータで，最後に入れられたものが最初に取り出されることから，LIFO(last in first out, 後入れ先出し) と呼ばれる．図 1 がスタックのイメージである．

スタックではデータの挿入と取り出しが列の一方からのみに対して，キューは列の両端から行う．一方がデータの追加で一方がデータの取り出しとして使われる．キューでは，最初に入れたデータが一番最初に取り出されることにある．取り出されるデータは格納されている最古のデータで，最初に入れられたものが最初に取り出されることから，FIFO(first in first out, 先入れ先出し) と呼ばれる．図 2 がキューのイメージである．

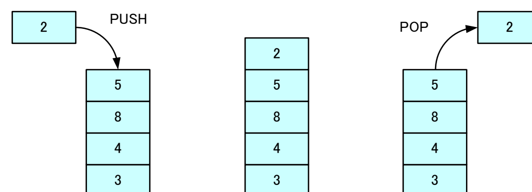


図 1: スタックのイメージ

*独立行政法人　秋田工業高等専門学校　電気情報工学科

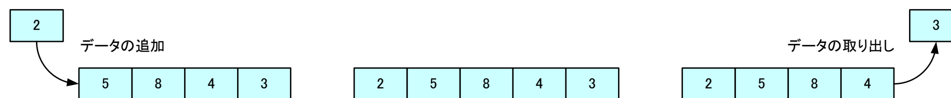


図 2: キューのイメージ

1.2　本日の学習内容

本日の学習内容は，ツリーと呼ばれるデータ構造の学習である．教科書 [1] の pp.199–217 が範囲である．ここでの学習のゴールは以下の通りである．

- ツリーのイメージができ，問題で与えられたデータの表現方法が分かる．
- 2 分木 (binary tree) によるデータの挿入追加が分かる．

2　ツリーの概要

2.1　ツリーの例

ツリー (three:木) と呼ばれるデータ構造は，階層構造を持ったデータの集まりを表すのに都合が良い．たとえば，組織図 (図 3)，コンピューターのファイルシステムなどである．情報科学では，データベースの中のデータの表現やコンパイラーでの原始プログラムの文法構造などでも使われる．

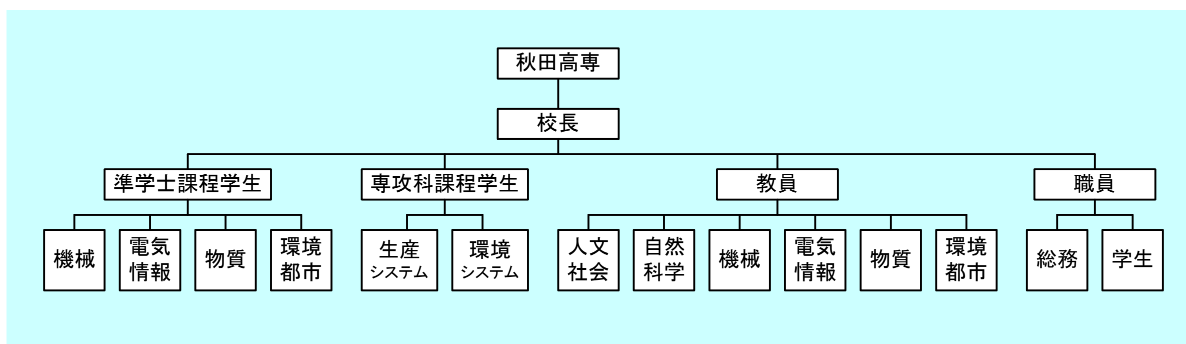


図 3: 秋田高専の組織図

このように階層構造をもつデータの集まりは，これまで学習してきたデータ構造—配列やリスト，スタック，キュー—で表すことは困難である．できないことはないが，プログラムが非常にわかりにくくなる．今後，諸君は階層構造を持つデータを取り扱う場合には，ツリーを用いよ! プログラムの内容が非常に分かりやすくなる．

2.2　ツリーの基本

2.2.1　基本用語

ツリー構造にはいろいろ名称があり，それを表1と図4に示す．なぜ，図1のようなデータ構造をツリー構造と呼ぶか?．それは，この図を反対にしてみるのである．すると，根が一番下にきて，枝や葉が上にあることが分かるであろう．まさに，木である．

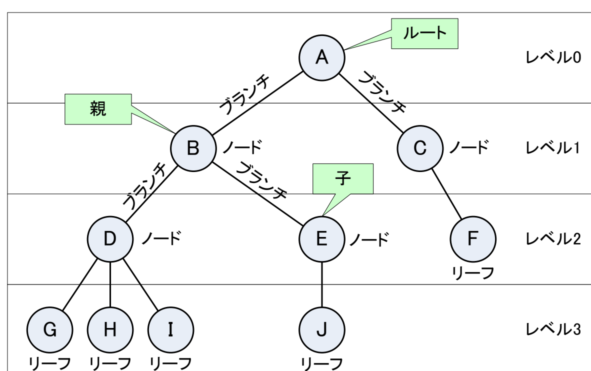


図 4: ツリー構造と名称．B と E は親子関係のの例である．

表 1: ツリー構造の名称

構成要素	内容
ルート (root:根)	最上位のノード
ノード (node:節)	枝が分かれるところ
リーフ (leaf:葉)	子がないノード
ブランチ (branch:枝)	親と子を結ぶ線
親 (parent)	上のノード
子 (child)	下のノード
兄弟 (brother)	同じ親を持つノード

2.2.2　特徴

子は一つの親を持つことがツリーの特徴である．二つの親を持ってはならないのである．そのため，任意の二つのノードあるいはリーフのつながりを表すパスは，一意に決まる．

2.2.3　データの順序

リーフあるいはノードにはデータが格納され，それらを順序づけることができる．順序づけの方法はいろいろあるが，次の3通りが重要である．

先行順 (preorder) 親のノード → 子の順に並べる．図5のようなツリー構造では次のような並びになる．

$$a \to b \to d \to g \to e \to h \to i \to c \to f \to j$$

となる．

中央順 (postorder) 子 → 親のノード → 子の順に並べる．これは，後で述べる2分木の他ではあまり使われない．2分木の場合，左の子 → 親のノード → 右の子のように順序づけることができる．図5のようなツリー構造では次のような並びになる．

$$g \to d \to b \to h \to e \to i \to a \to c \to f \to j$$

後行順 (inorder) 親のノード → 子の順に並べる．図 5 のようなツリー構造では次のような並びになる．

$$g \to d \to h \to i \to e \to b \to j \to f \to c \to a$$

このなかで，2 分木に使われることの多い中央順がもっとも重要である．

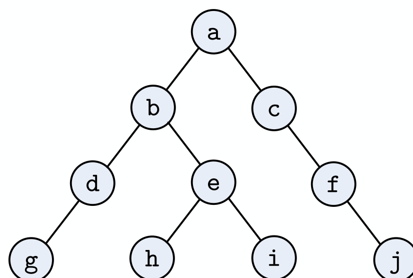


図 5: ツリー

3　2 分木

3.1　2 分木のイメージ

2 分木 (binary tree) は，各ノードの子の数がたかだか 2 であるようなツリーである．これは単純ではあるが，先に示した順序の中央順にデータを並べ，順序木を作ることで応用範囲が広がる．そのようなわけで，ツリー構造の中でもっとも単純ではあるが，アルゴリズムで使用される頻度はもっとも高い．単純なものほど応用範囲が広い—という一つの例である．

子と親の大小関係を決め，それに従って 2 分木を作ると順序木になる．たとえば，次のように大小関係を決めるのである．

- 左側の子は，その親より必ず小さい．
- 右側に子は，その親より必ず大きい．

図 6 にその例を示す．

図 6 の 2 分木を，中央順 (postorder) に並べると，

$$11 \to 15 \to 20 \to 23 \to 31 \to 37 \to 48 \to 55 \to 65 \to 73 \to 81 \to 88 \to 94$$

となる．昇順にデータが並んでいることが分かる．

この構造のあるノードの左側と右側はそれぞれの子を頂点とする木構造になる．この子孫に対しても，同じ規準が適用されれるので，再帰による処理が可能となる．

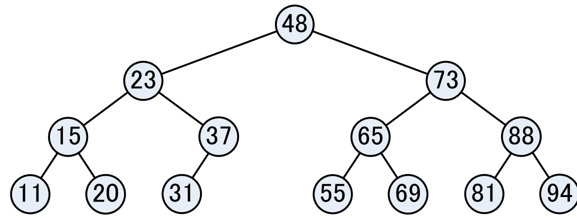


図 6: ツリー構造 (2 分探索木)

3.2 データの作成

以下の順序で 2 分木を作成する．ルートから大小関係をたどっていき，子が無いところへデータを挿入する．

```
6,10,2,8,12,4
```

作成されるツリーは，図 7 のようになる．

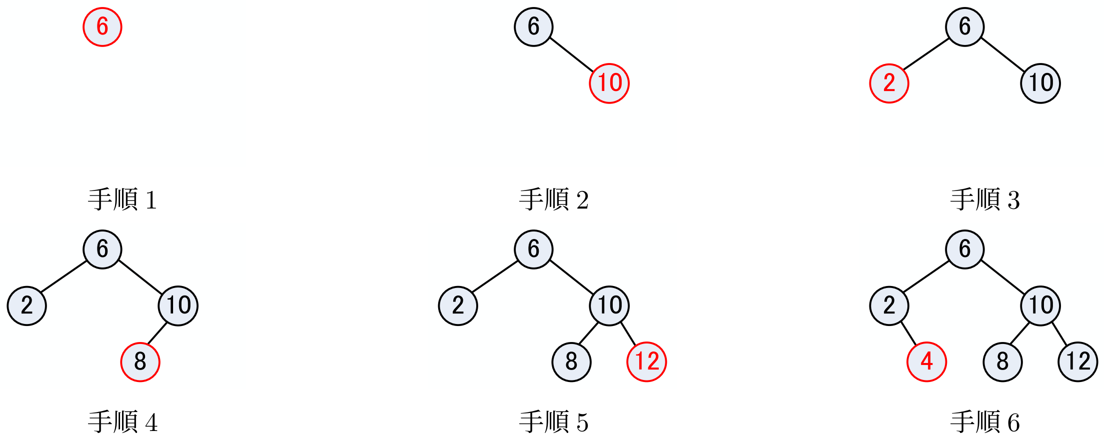


図 7: ツリーのノードの作成順序

3.3 データの追加

先ほど作成したツリーにデータ 9 を追加する．これも，ルートから大小関係をたどっていき，子が無いところへデータを挿入する．すると，図 8 のようになる．

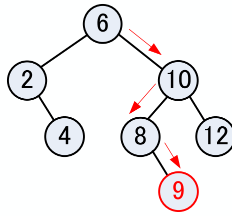


図 8: ツリーへのデータの追加

3.4 データの削除

元のツリー図 9 からデータを削除することを考える.

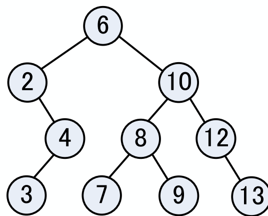


図 9: データを削除する前の 2 分木

データを削除する場合，子が無い場合，子がひとつの場合，子がふたつの場合にに分けて考えなくてはならない.

- 削除するデータに子が無い場合は，そのまま削除する (図 10).
- 削除するデータがひとつの子をもつ場合は，削除した場所にその子と子孫を入れる (図 11).
- 削除するデータがふたつの子を持つ場合．削除した場所に，左側の子孫の最大値 (あるいは右側の子孫の最少値) を移動させる (図 12).

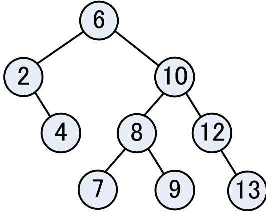


図 10: 元のツリー図 9 から 3 を削除．子が無い場合は，そのまま削除する．

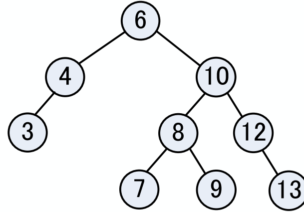


図 11: 元のツリー図 9 から 2 を削除．子がひとつの場合は，削除した部分にその子を移す．

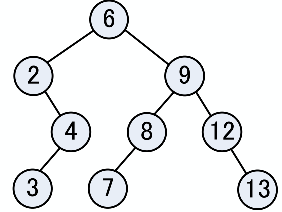


図 12: 元のツリー図 9 から 10 を削除．子が二つの場合は，削除した部分に，左側の子孫の最大の値をもつノードを移動させる．

4 プログラム例

4.1 二分木の実装

2 分木は，ノードとその接続状態を表すブランチからできている．C 言語でこれを表現するためには，ノードの中でデータと接続状態を示せばよい．リストを思い出せ，それとほとんど同じではないか! リストと同じように，ノードをあわす構造体を使えばよい．

```
typedef struct node_tag{
  int value;
  struct node_tag *left;
  struct node_tag *right;
}node;
```

このようなノードを使うと，図 13 のようにツリー構造を表すことができる．

ツリー全体を表すために，ルートを表す構造体

```
node *root_tree=NULL;
```

を宣言しておく．もちろん，これはノードを表す構造体のよりも後で宣言する必要がある．先に宣言すると，`tree_node` という型がコンパイラーが分からないからである．また，初期値は意味のないポインター (`NULL`) を入れておく．最初はルートがないため，それを表すポインターは意味が無いためである．

プログラムの詳細については，付録を見よ．

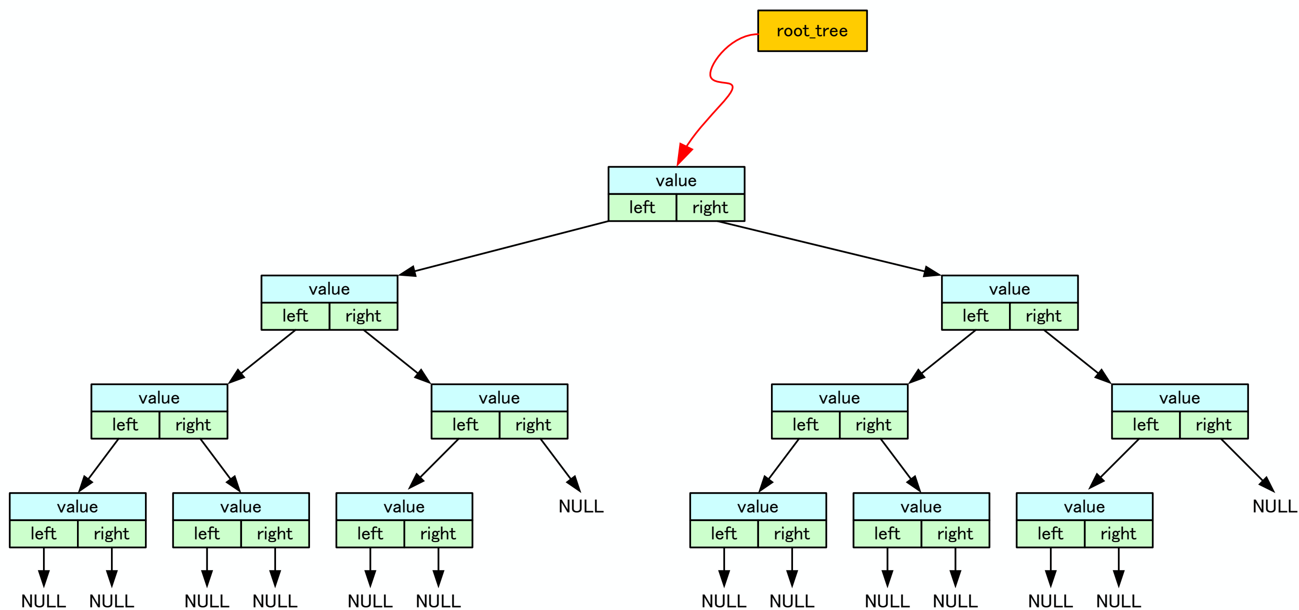
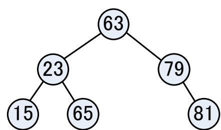


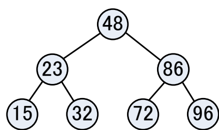
図 13: 構造体を使ったツリー構造

5　練習問題

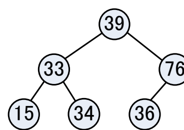
[練習 1]　図の中で 2 分探索木になっているものはどれか?



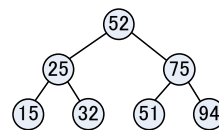
(ア)



(イ)



(ウ)



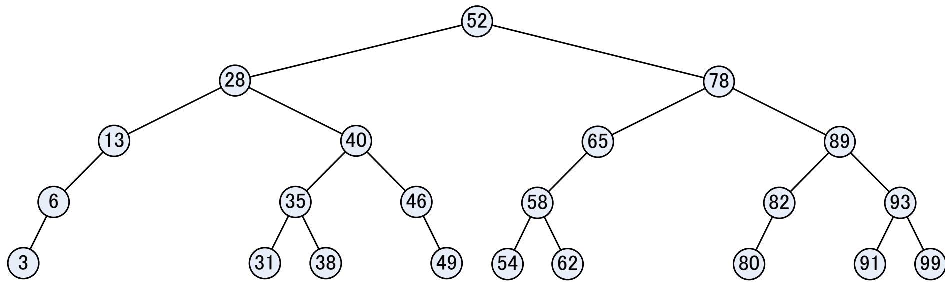
(エ)

[練習 2]　最初はデータが無く，以下の並びでデータが追加される．2 分探索木を作成せよ．

`69,26,36,45,89,65,11,12,14,23,44`

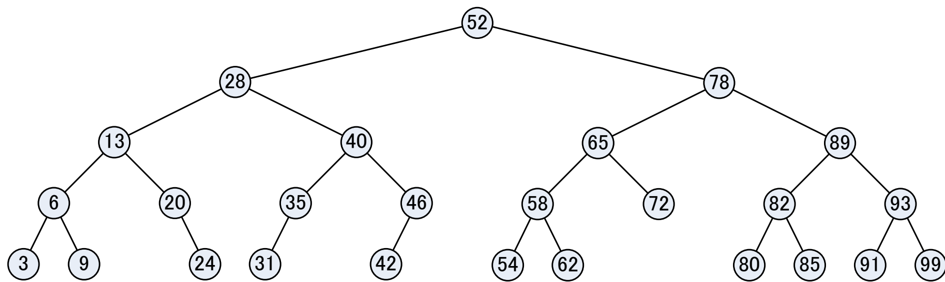
[練習 3]　図の 2 分探索木に `70,73,67,75,77,66` の順にデータを追加する．途中を含めて 2 分木を書け．

追加する．



データを追加する 2 分木

[練習 4] 図の 2 分探索木から，`52,46,54` の順にデータを削除する．



データを削除する 2 分木

[練習 5] 2 分木のプログラムを理解せよ．

6 課題

以下の課題を実施し，レポートとして提出すること．

[問 1] (復予) 教科書 [1] の pp.199–245 を 2 回読め．レポートには「2 回読んだ」と書け．

[問 2] (復) 本日配布したプリントを 2 回読め．レポートには「2 回読んだ」と書け．

[問 3] (復) 最初はデータが無く，以下の並びでデータが追加される．2 分探索木を作成せよ．途中の過程も記述すること．

$$43 \rightarrow 65 \rightarrow 51 \rightarrow 28 \rightarrow 73 \rightarrow 15 \rightarrow 17 \rightarrow 32 \rightarrow 45 \rightarrow 61 \rightarrow 70$$

[問 4] 図 14 の 2 分探索木に以下の順序でデータが追加せよ．途中の書いても記述すること．

$$62 \rightarrow 46 \rightarrow 82 \rightarrow 80 \rightarrow 38$$

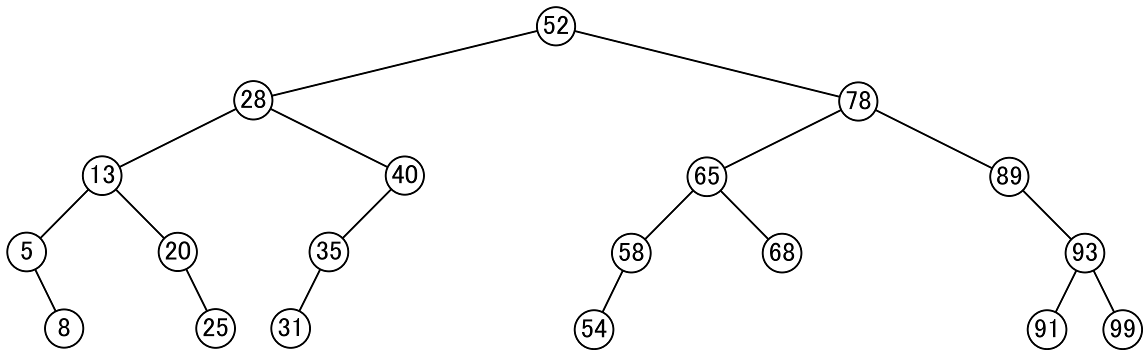


図 14: データを追加する 2 分探索木

[問 5] 以下の順序で，図 15 の 2 分探索木からデータが削除される．最後の値 52 が削除れたの後の 2 分木を書け．

$46 \rightarrow 53$

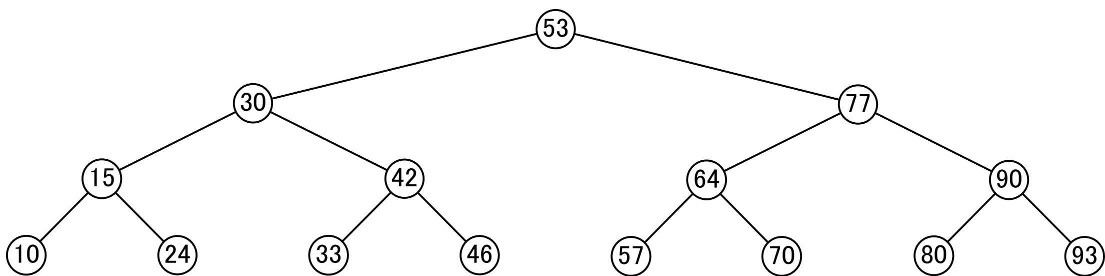


図 15: データを削除する 2 分探索木

[問 6] (復) ここでの学習内容でわからないところがあれば，具体的に記述せよ．

提出要領はいつものとおり．ただし，期限は 7 月 24 日 (火)AM 8:45，課題名は「課題　ツリー」とすること．

付録 A　プログラム例

以下にプログラム例を示す.

リスト 1: ヘッダーファイル (b_tree.h).

```
//============ ノードを表す構造体 ==============================
typedef struct node_tag{
  int value;
  struct node_tag *left;
  struct node_tag *right;
}node;

//============ プロトタイプ宣言 ==============================
extern node *creat_node(int new_data);
extern void insert_node(node *t_node, int new_data);
extern void delete_node(node *del_node, int del_data);
extern void print_tree(node *t_node, int depth);


extern node *root_tree;
```

リスト 2: ツリーを操作する関数 (b_tree.c).

```
#include <stdio.h>
#include <stdlib.h>
#include "b_tree.h"

node *root_tree=NULL;

//============ ノード作成関数 ==============================
node *creat_node(int new_data)
{
  node *new_node;

  new_node=(node *)malloc(sizeof(node));

  new_node->value = new_data;
  new_node->left = NULL;
  new_node->right = NULL;

  return new_node;
}


//============ ノードをツリーに追加する関数 ==============================
void insert_node(node *t_node, int new_data)
{

  if(t_node==NULL){            // rootが空の場合(最初のデータ)
    root_tree = creat_node(new_data);
    return;
  }

  if(t_node->value > new_data){
    if(t_node->left == NULL){
      t_node->left=creat_node(new_data);
    }else{
      insert_node(t_node->left, new_data);
    }
```

```
37     }else{
38       if(t_node->right == NULL){
39         t_node->right=creat_node(new_data);
40       }else{
41         insert_node(t_node->right, new_data);
42       }
43     }
44
45     return;
46   }
47
48   //============== ノードをツリーから削減する関数 ======================================
49   void delete_node(node *del_node, int del_data)
50   {
51     node *parent=NULL;        // 削除するノードの親
52     node *big=NULL;           // 左の最大ノード
53     node *pbig=NULL;          // 左の最大ノードの親
54     int flag=0;
55
56     // 削除するノードの探索
57     while(del_node!=NULL && del_node->value!=del_data){
58       if(del_node->value < del_data){
59         parent = del_node;
60         del_node = del_node -> right;
61       }else{
62         parent = del_node;
63         del_node = del_node -> left;
64       }
65     }
66
67     if(del_node==NULL){
68       printf("削除するデータがありません!\n");
69       return;
70     }
71
72     if(del_node->left == NULL){          // 削除するデータの左あるいは両方の子が無い
73       if(parent==NULL){                  // 削除するデータがrootのとき
74         root_tree = del_node->right;
75       }else if(parent->value > del_data){ // 削除するデータが親の左の子
76         parent->left=del_node->right;
77       }else{                              // 削除するデータが親の右の子
78         parent->right=del_node->right;
79       }
80       free(del_node);
81       return;
82     }
83
84     if(del_node->right == NULL){         // 削除するデータの右の子が無い
85       if(parent==NULL){                  // 削除するデータがrootのとき
86         root_tree = del_node->right;
87       }else if(parent->value > del_data){
88         parent->left=del_node->left;
89       }else{                              // 削除するデータが親の右の子
90         parent->right=del_node->left;
91       }
92       free(del_node);
93       return;
94     }
95
96     //--- 削除するデータに左右の子がある場合 ---
97
98     pbig=del_node;
```

```
99      big=del_node->left;
100
101     while(big->right != NULL){          // 左の最大値をもつノードの検索
102       pbig = big;
103       big = big->right;
104       flag = 1;
105     }
106
107     del_node->value = big->value;
108
109     if(flag == 0){
110       pbig->left = big->left;
111     }else{
112       pbig->right = big->left;
113     }
114
115     free(big);
116
117
118   }
119
120
121   //============== ええ加減にツリーを表示する関数 ==============================
122   void print_tree(node *t_node, int depth)
123   {
124     int i;
125
126     if(t_node==NULL)return;
127
128     print_tree(t_node->right,depth+1);
129
130     for(i=0; i<depth; i++)printf("   ");
131     printf("%d\n", t_node->value);
132
133     print_tree(t_node->left, depth+1);
134
135     return;
136   }
```

リスト 3: ツリーを使った例 (main.c).

```
1   #include <stdio.h>
2   #include "b_tree.h"
3
4   int main(void)
5   {
6
7     insert_node(root_tree,5);
8     insert_node(root_tree,9);
9     insert_node(root_tree,3);
10    insert_node(root_tree,4);
11    insert_node(root_tree,7);
12    insert_node(root_tree,6);
13    insert_node(root_tree,10);
14    insert_node(root_tree,12);
15    insert_node(root_tree,2);
16    insert_node(root_tree,1);
17
18    printf("\n\n");
19    print_tree(root_tree, 0);
20
```

```
21     delete_node(root_tree, 9);
22     printf("\n\n");
23     print_tree(root_tree, 0);
24     return 0;
25 }
```

参考文献

[1] 内田智史監修, (株) システム計画研究所編. C 言語によるプログラミング　応用編　第 2 版. (株) オーム社, 2006.