ツリー構造 (その 1)

山本昌志*

2006 年 1 月 16 日

1　本日の学習内容

1.1　前回の復習

前回の講義では，再帰呼び出しというアルゴリズムを学習した．リスト 1 は階乗の計算を再帰呼び出しで行っている．このプログラムは，階乗の漸化式

$$n! = n \times (n-1)! \qquad 0! = 1 \tag{1}$$

をそのままプログラミングした形になっている．

リスト 1: 再帰呼び出しを使った階乗を計算するプログラム

```
1  #include <stdio.h>
2
3  /*=============================================================*/
4  /*  階乗を計算する関数                                         */
5  /*=============================================================*/
6  int kaijyo(int n){
7
8    if(n==0){
9      return 1;
10   }else{
11     return n*kaijyo(n-1);
12   }
13
14 }
15
16 /*=============================================================*/
17 /*  メイン関数                                                 */
18 /*=============================================================*/
19 int main(){
20   int n;
21
22   printf("階乗を計算します．値を入れてください\n");
23   scanf("%d",&n);
24
25   printf("%d!=%d\n",n,kaijyo(n));
26
27   return 0;
```

*独立行政法人　秋田工業高等専門学校　電気情報工学科

```
28 }
```

1.2 本日の学習内容

本日はデータ構造のツリー構造の学習を行う．本日のゴールは，以下の通りである．

- リストとツリー構造の違いが分かる．
- リストを実現する方法が理解できる．これは復習であるが，これが分からないとツリー構造のプログラムはできない．
- 2 分探索木のデータの追加と削除，探索の方法が理解できる．

実際のツリー構造のプログラムは，来週の講義とする．

2 ツリー構造とは

2.1 ツリー構造とは

同じような複数のデータを表す場合，配列やリスト，スタック，キューのようなデータ構造を学習してきた．これらのデータ構造で教科書の Fig.6-1 のような，階層をもつデータの集まりを表すことは，困難である．このように階層構造をもつデータを処理するときに威力を発揮するのがツリー構造である．ツリーとは，tree(木) のことである．

ツリー構造の例として，教科書では生き物や Windos のファイルシステムのツリー構造が書かれている．諸君が学習で使っている UNIX のファイルシステムは，図 1 のようになっている．

[**練習 1**] ツリー構造をしたデータは，他に何があるか?

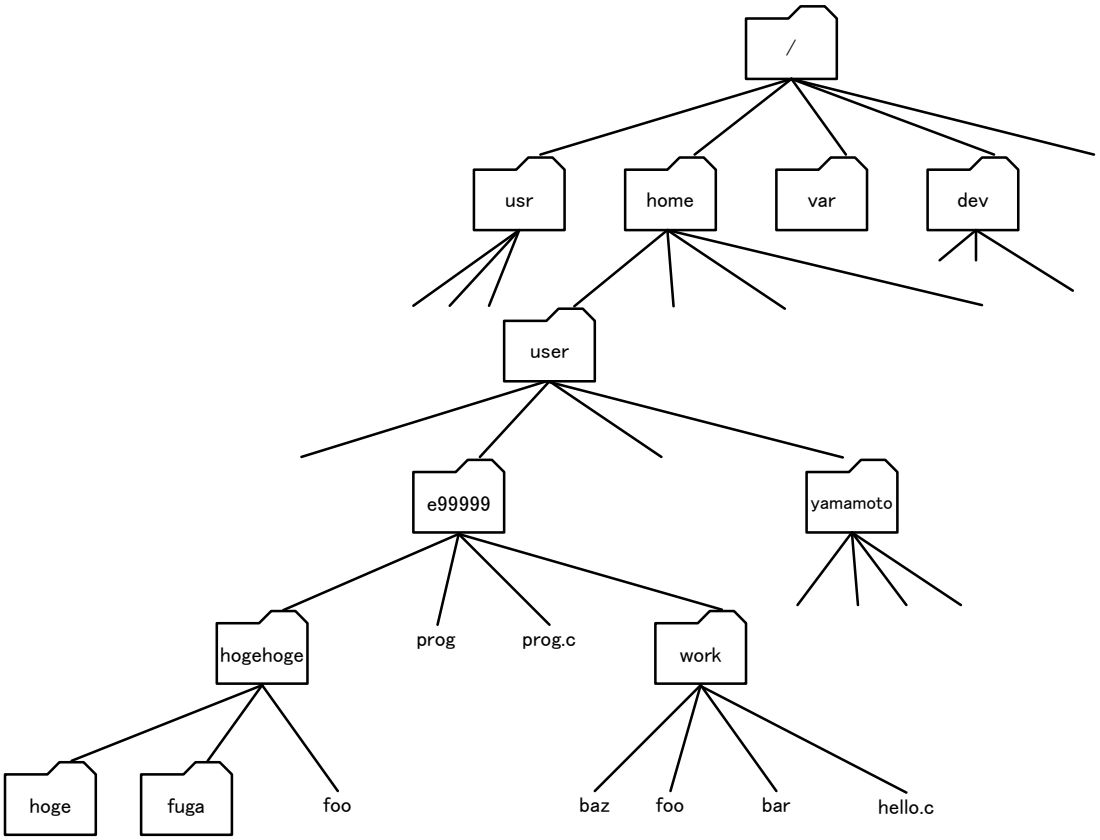


図 1: UNIX のファイル構造

2.2 ツリー構造を実現する方法

教科書 [1] に書いてあるように，ツリー構造のデータ構造はリストを少し変えるだけで実現できる．そこで，まずはリストのおさらいをしてから，ツリー構造を実現する方法を示す．

2.2.1 単方向リスト

リストは，図 2 のようなデータ構造である．一つの要素 (element) はデータとつながりを表すポインターからなり，構造体で表すことができる．リストは，各々の要素のつながりしか分からないので，最初の要素を示すポインターが必要である．このようなリストを使う場合，次のような宣言を行えば良い．

```
/* --- 単方向リストを表す構造体 ---*/
typedef struct tag_list_element{
  struct tag_list_element *next;
  int data;
}list_element;
```

```
list_element *top;       /* 先頭を示すポインター */
```

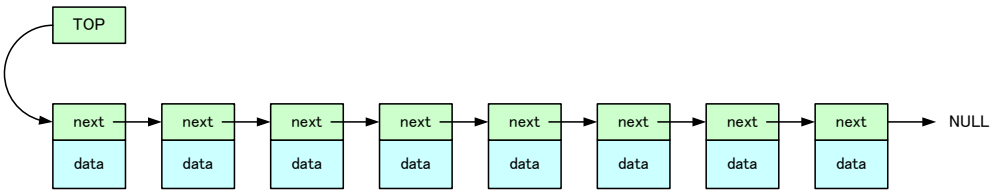


図 2: 単方向リスト

配列に比べ，リストはデータの追加と削除が容易である．データの追加と削除の方法は，教科書 [1] の Fig.6-3(p.163) に示してあるとおりである．

[**練習 1**] 配列のデータの追加と削除の方法を述べよ．

リストを使って，実際にデータの追加と削除を行うプログラムをリスト 2 に示す．このプログラムの動作は以下の通りである．

- 入力した整数は，リストの形で記憶される．
- エレメント作成の後，直ちに，データが昇順に並ぶように，接続のポインター (`next`) を設定する．
- 負の整数で，データ入力が終了する．
- 入力されたデータを昇順に表示する．
- データを削除する．
- 残っているデータを昇順に表示する．

リスト 2: リストを使ったプログラム例

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <memory.h>
4
5  /* ---- 単方向リストを表す構造体 ----*/
6  typedef struct tag_list_element{
7    struct tag_list_element *next;
8    int data;
9  }list_element;
10
11
12 list_element *top;       /* 先頭を示すポインター */
13
14 /*=============================================================*/
15 /* リストを作る関数                                            */
16 /*=============================================================*/
```

```
17 list_element *creat_new_element(int num){
18   list_element *new_list;
19
20   new_list=(list_element *)malloc(sizeof(list_element));
21   new_list->data = num;
22   new_list->next = NULL;
23
24   return new_list;
25 }
26
27 /*=============================================================*/
28 /* リストを挿入する関数                                        */
29 /*=============================================================*/
30 void insert_list(int num){
31
32   list_element *new,*loop;
33
34   /*--- データがひとつもないとき ---*/
35   if(top==NULL){
36     top = creat_new_element(num);
37     return;
38   }
39
40   /*--- 先頭のデータよりも小さいとき ---*/
41   if(num < top->data){
42     new = creat_new_element(num);
43     new -> next = top;
44     top = new;
45     return;
46   }
47
48   loop=top;
49   while(1){
50     if(loop->next==NULL){
51       loop->next = creat_new_element(num);
52       break;
53     }
54
55     if(num<loop->next->data){
56       new=creat_new_element(num);
57       new->next=loop->next;
58       loop->next=new;
59       break;
60     }
61     loop=loop->next;
62   }
63 }
64
65
66 /*=============================================================*/
67 /* リストを削除する関数                                        */
68 /*=============================================================*/
69 void delete_list(int num){
70     list_element *loop;
71
72     if(top->data==num){
73       top=top->next;
74       return;
75     }
76
77     loop=top;
78
```

```
79      while(loop->next != NULL){
80        if(loop->next->data == num){
81          loop->next=loop->next->next;
82          return;
83        }
84        loop=loop->next;
85      }
86  }
87
88  /*=================================================================*/
89  /* リストを表示する関数                                            */
90  /*=================================================================*/
91  void print_data(){
92    list_element *elm;
93
94    elm=top;
95
96    while(elm!=NULL){
97      printf("%d ",elm->data);
98      elm=elm->next;
99    }
100
101   printf("\n");
102
103 }
104
105
106 /*=================================================================*/
107 /* メイン関数                                                      */
108 /*=================================================================*/
109 int main(){
110   int in;
111
112   top=NULL;
113
114   /*--- キーボード入力とリストへの追加 ---*/
115   printf("正の整数のデータを追加します(負:入力終了).\n");
116   do{
117     scanf("%d",&in);
118     if(in<0)break;
119     insert_list(in);
120   }while(1);
121
122
123   print_data();
124
125   /*--- リストからでーたの削除 ---*/
126   printf("整数のデータを削除します(負:入力終了).\n");
127   do{
128     scanf("%d",&in);
129     if(in<0)break;
130     delete_list(in);
131   }while(1);
132
133   print_data();
134
135   return 0;
136 }
```

2.3　ツリー構造

ツリー構造は，図 3 のように表すことができる．一つのノードは，リスト同様，構造体で表すことができる．構造体は，データと子を示すポインターを持つことになる．リストでは先頭の要素を表すポインターが必要であったが，ツリー構造ではルートを示すポインターが必要である．構造体の宣言は，以下のようにすればよいだろう．

```
/* --- ツリーのノードを表す構造体 ---*/
typedef struct tag_tree_node{
  struct tag_tree_node *ko_1, *ko_2, *ko_3;
  int data;
}tree_node;

tree_node *root;       /* ルートを示すポインター */
```

このツリーを実現させるための構造体には，ひとつ問題がある．子の数が 3 と決められている．また，多くのデータでは子の数が不定であることが多い．そのため，必要なポインターの数が分からない．この解決方法については，次週の講義で説明する．ここでは，リストとほとんど同じ構造体で，ツリー構造が実現できることを理解して欲しい．

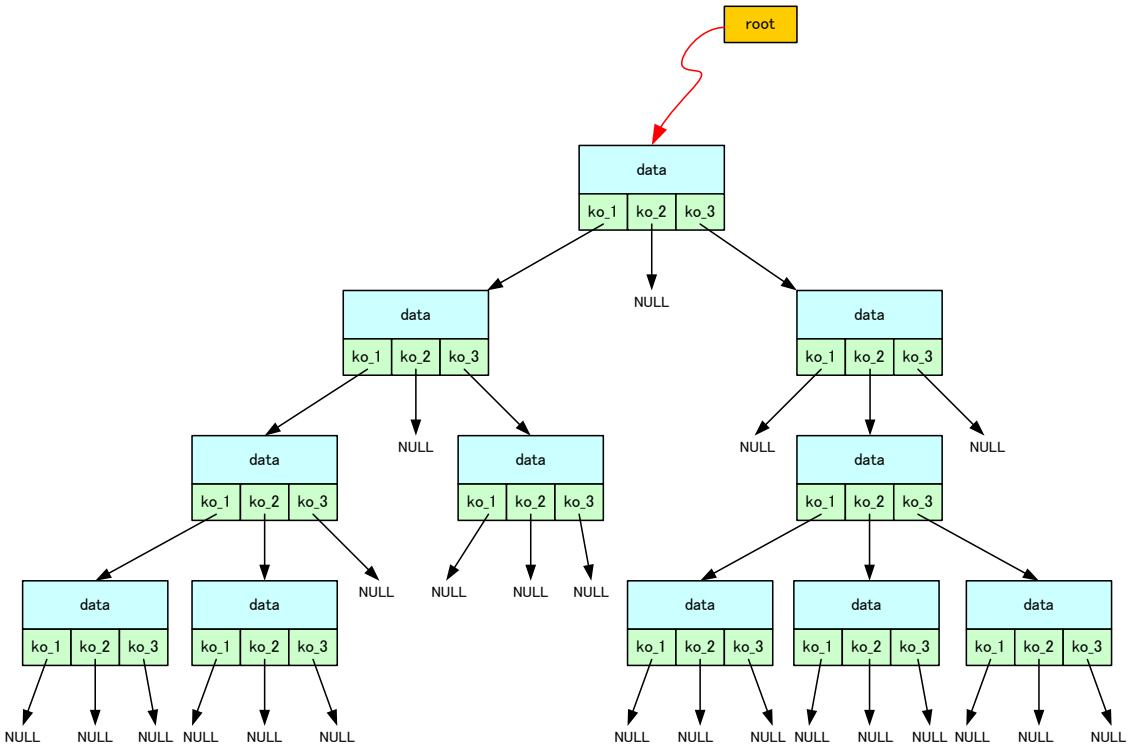


図 3: ツリー構造

2.4　ツリー構造の各部の名称

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 3 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．

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

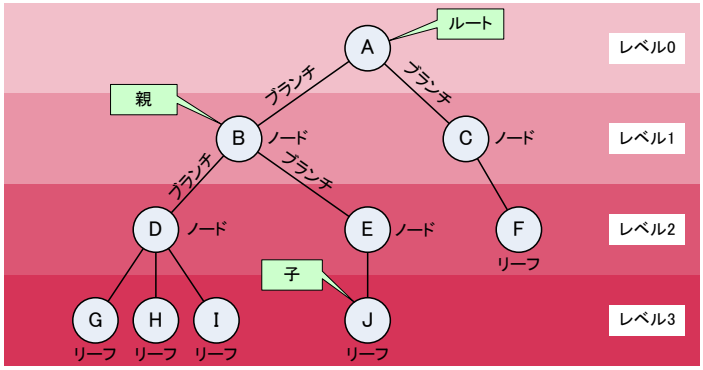


図 4: ツリー構造．図中の親子関係はノード E を基準にしている．

2.5　ツリー構造の要素の追加と削除

教科書 [1] の p.166-167 の通り．

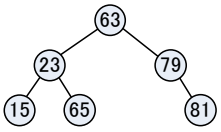
3　2 分木 (binary tree)

ツリー構造のうち，子の数が最大 2 となるものを 2 分木と言う．そして，ある規準に従ってその 2 分木が作られている場合，2 分探索木 (binary search tree) と呼ばれる．ここでは，その 2 分探索木のデータの追加と削除，探索方法を示す．

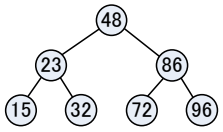
残りは，教科書 [1] に沿って説明する．

4 練習問題

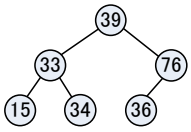
[**問 1**] 図の中で 2 分木になっているものはどれか?



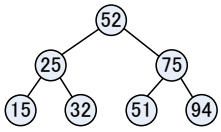
(ア)



(イ)



(ウ)



(エ)

[**問 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 分木を作成せよ．

(ア) 45,64,28,90,63,24,98,23,42,95

(イ) 69,26,36,45,89,65,11,12,14,23,44

(ウ) 15,18,16,23,44,55,63,72,84,95,10

(エ) 85,76,71,65,61,53,45,41,33

(オ) 57,23,75,15,32,77,86,11,31,55,70,95

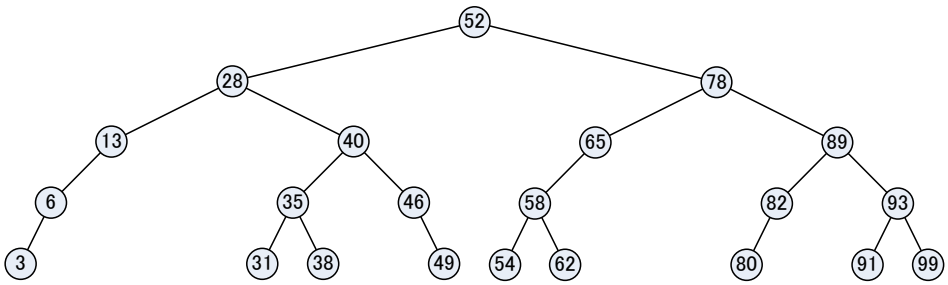
[**問 3**] 図の 2 分木にデータを追加する．以下の問いに答えよ．

(ア) 18 を追加する．

(イ) 53,68 と順に追加する．

(ウ) 70,73,67,75,77,66 と順に追加する．

(エ) 66,77,75,67,73,70 と順に追加する．



データを追加する 2 分木

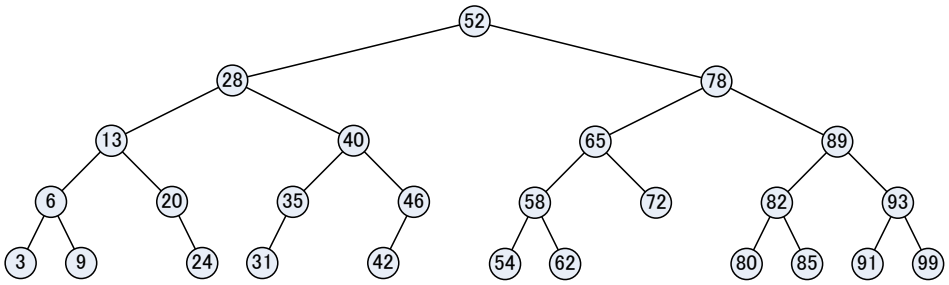
[**問 4**] 図の 2 分木にデータを削除する．以下の問いに答えよ．

(ア) 72 を削除する．

(イ) 46 を削除する．

(ウ) 28 を削除する．

(エ) 52,46,54 と順に削除する.



データを削除する 2 分木

4.1 レポート提出要領

提出方法は，次の通りとする.

期限	1 月 23 日 (月) AM 10:40
用紙	A4
提出場所	山本研究室の入口のポスト
表紙	表紙を 1 枚つけて，以下の項目を分かりやすく記述すること.
	授業科目名「情報工学」
	課題名「課題　ツリー構造 (2 分木)」
	2E　学籍番号　氏名
	提出日
内容	2 ページ以降に問いに対する答えを分かりやすく記述すること.

参考文献

[1] 紀平拓男, 春日伸弥. プログラミングの宝箱　アルゴリズムとデータ構造. ソフトバンクパブリッシング (株), 2004 年.