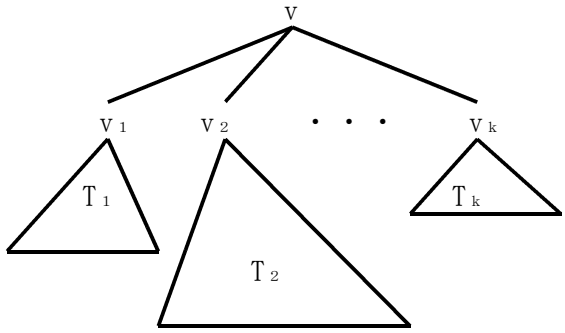
２分探索木Ｉ

０．目次

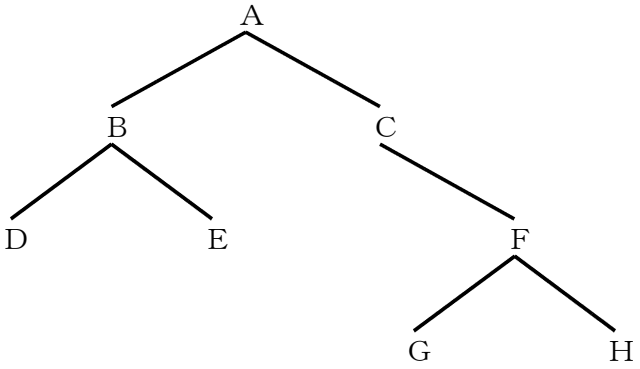
１．２分探索木とは

木はいくつかの節点と節点同士を結ぶ辺から構成される。
２つの節点u,vが直接辺で結ばれているとき、一方を親節点、他方を子節点という。ある節点の親節点は高々１つであり、子節点は０個以上である。節点の中には、親節点をもたない節点がただ１つあり、根という。また、子節点をもたない節点を葉という。木は、つぎのように定義される。

（１）ひとつの節点は、木である。
（２）vが節点で、$T_1, T_2, \cdots, T_k$が木で、それぞれの木の根が$v_1, v_2, \cdots, v_k$とする。このとき、vを$v_1, v_2, \cdots, v_k$の親とするのは、木である。
節点$v_1, v_2, \cdots, v_k$は節点vの子と呼ばれる。



２分木は、どの節点も高々２個の子節点をもつ。２個の子節点の内、左に書かれる子節点を根とする部分木を**左部分木**、右に書かれる子節点を根とする部分木を**右部分木**という。



節点：Ａ，Ｂ，Ｃ，Ｄ，Ｅ，Ｆ，Ｇ，Ｈ
（Ａは根でもある）
辺：ＡＢ，ＡＣ，ＢＤ，ＢＥ，ＣＦ，ＦＧ，ＦＨ
（ＸＹのとき、Ｘが親、Ｙが子を意味する）

２分木の節点に集合Ｓの要素を割り当てる。各節点において、割り当てられた要素の大きさが、左部分木のどの節点の要素よりも大きく、右部分木のどの節点の要素よりも小さいとき、**２分探索木**という。

２．２分探索木の作成

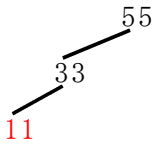
空の２分探索木から始める。挿入する要素と節点の要素を比較し、前者が小さい場合、左部分木に追加し、前者が大きい場合、右部分木に追加していく。

●手順　9個のデータ(55, 33, 11, 99, 44, 77, 22, 66, 88)で手順を示す。

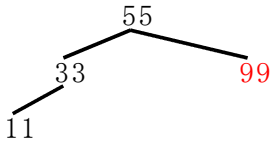
（１）

（２）

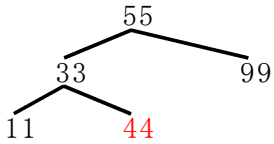
（３）



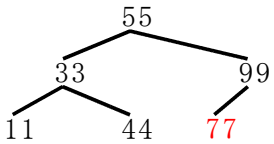
（４）



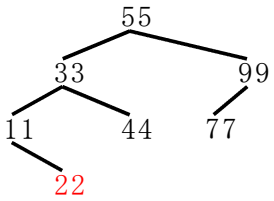
（５）



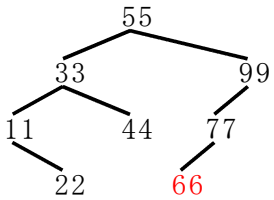
（６）



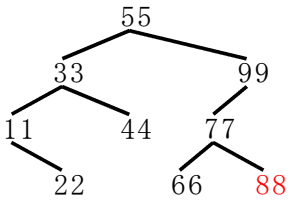
（７）



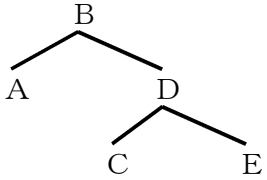
（８）

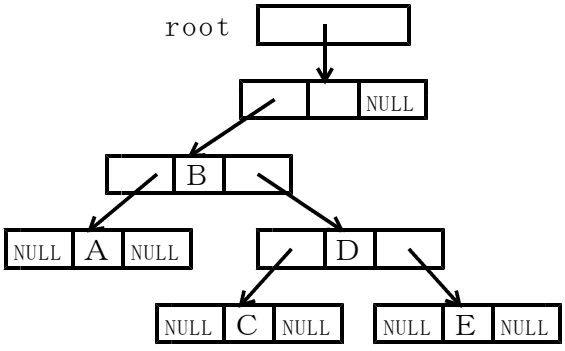


（９）



２分探索木を表現するために、データを保存する変数info、左部分木を指すポインタleft、右部分木を指すポインタrightをもつ構造体を用意する。根を指すポインタ変数をrootとする。ポインタ変数rootは空の同じ構造体を指す。
このようにすると、プログラムが簡潔になる。





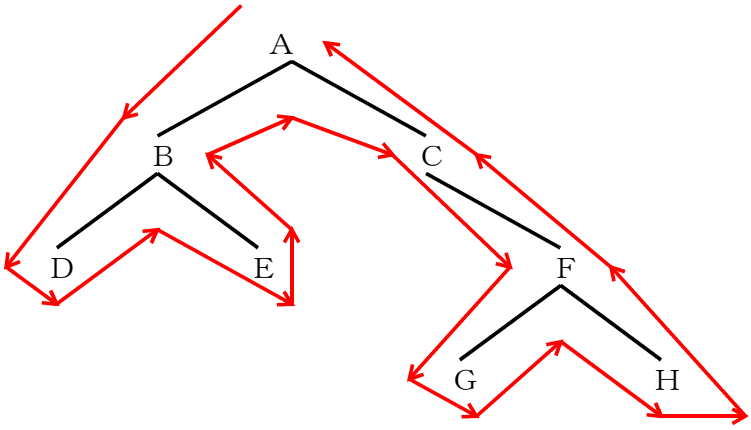
●プログラム・非再帰版（bt211.c）

```
/* << bt211.c >> */
/* ２分探索木の作成。*/
#include <stdio.h>
#include <stdlib.h>
struct NODE {
  int info;            /* 節点がもつデータ。*/
  struct NODE *left;   /* 左の子節点を指すポインタ。*/
  struct NODE *right;  /* 右の子節点を指すポインタ。*/
};
struct NODE *mktree();

int main() {
  struct NODE *root;

  /* ２分探索木の作成。*/
  root = mktree();
}

/* ２分探索木の作成。*/
struct NODE *mktree() {
  int data;
  struct NODE *p,     /* 現節点を指すポインタ変数。*/
              *q,     /* 現節点の親節点を指すポインタ変数。*/
              *r,     /* 作業用ポインタ変数。*/
              *root; /* 根を指すポインタ変数。*/

  /* 根を指すポインタ変数と空の構造体の作成。*/
  root = (struct NODE *)malloc(sizeof(struct NODE));
  root->left = NULL;
  root->right = NULL;

  while( 1 ) {
    /* データの読み込み。*/
    scanf("%d",&data);
    if( data <= 0 ) { break; }
    r = (struct NODE *)malloc(sizeof(struct NODE));
    r->info = data;
    r->left = NULL;
    r->right = NULL;

    root->info = data; /* ２分探索木にデータを追加するとき、
                          空の構造体の左部分木に追加できるように
                          root->info=data としておく。*/
    /* 初期設定。*/
    p = root->left;
    q = root;

    /* 追加する場所を探す。*/
    while( p != NULL ) {
      /* 重複データは追加しない。*/
      if( data == p->info ) { goto next; }

```

```
53        /* データdataが現在の節点のデータ(p->info)以下のとき
54           左部分木へ、大きいとき右部分木へ移動する。*/
55        q = p;
56        if( data <= p->info ) {
57          p = p->left;
58        } else {
59          p = p->right;
60        }
61      }
62
63      /* データの追加。*/
64      if( data <= q->info ) {
65        /* 左部分木として追加。*/
66        q->left = r;
67      } else {
68        /* 右部分木として追加。*/
69        q->right = r;
70      }
71    next:;
72    }
73    return root;
74  }
```

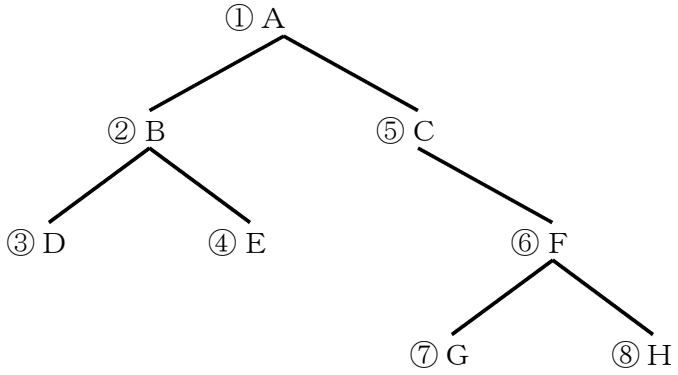
３．２分探索木の走査

２分探索木の辺を反時計回りに（赤い線に沿って）なぞっていくと２分探索木のすべての節点を訪問できる。



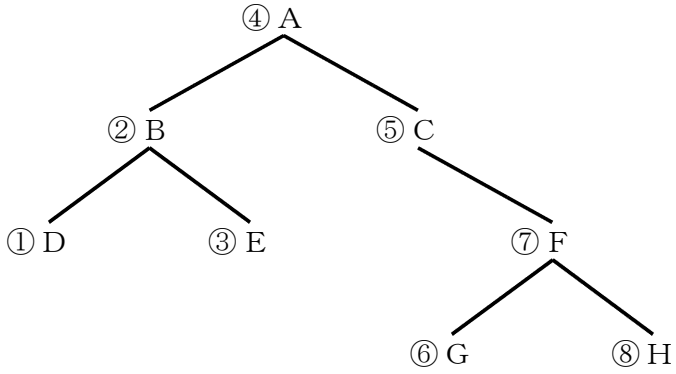
２分探索木のすべての節点をたどる方法として、前走査、中走査、後走査の３通り考えられる。

●**前走査**：まず、親節点を訪問する。
つぎに、左部分木のすべての節点を訪問する。
最後に、右部分木のすべての節点を訪問する。



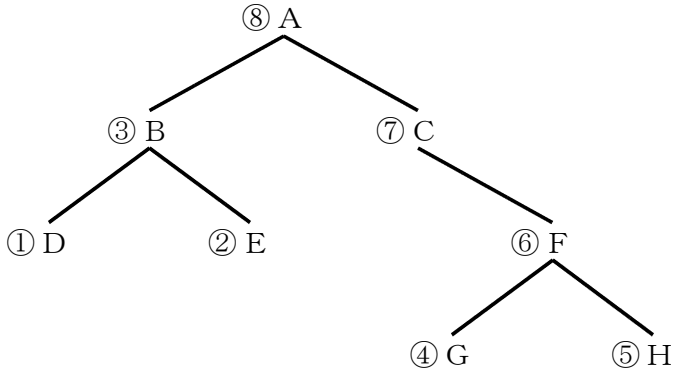
訪問順：Ａ Ｂ Ｄ Ｅ Ｃ Ｆ Ｇ Ｈ

●**中走査**：まず、左部分木のすべての節点を訪問する。
つぎに、親節点を訪問する。
最後に、右部分木のすべての節点を訪問する。



訪問順：Ｄ Ｂ Ｅ Ａ Ｃ Ｇ Ｆ Ｈ

●**後走査**：まず、左部分木のすべての節点を訪問する。
つぎに、右部分木のすべての節点を訪問する。
最後に、親節点を訪問する。



訪問順：Ｄ Ｅ Ｂ Ｇ Ｈ Ｆ Ｃ Ａ

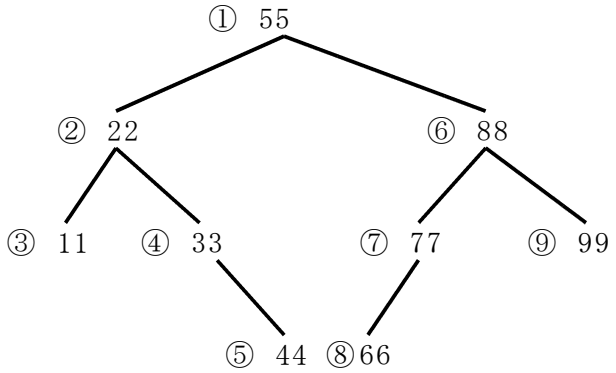
３．１　前走査

●プログラム・再帰版(bt311.c)

```
1  /* << bt311.c >> */
2  /* ２分探索木の走査（前走査）。*/
3  #include <stdio.h>
4  #include <stdlib.h>
5  struct NODE {
6    int info;           /* 節点がもつデータ。*/
7    struct NODE *left;  /* 左の子節点を指すポインタ。*/
8    struct NODE *right; /* 右の子節点を指すポインタ。*/
9  };
10 void preorder(struct NODE *p);
11 struct NODE *mktree();
12
13 int main() {
14   struct NODE *root;
15
16   /* ２分探索木の作成。*/
17   root = mktree();
18
19   /* 前走査。*/
20   preorder(root->left); printf("¥n");
21 }
22
23 /* 前走査。*/
24 void preorder(struct NODE *p) {
25   if( p == NULL ) { return; }
26   printf(" %d",p->info);
27   preorder(p->left);
28   preorder(p->right);
29 }
```

実行結果

```
% cc bt311.c
% ./a.out
55 22 88 11 33 77 99 44 66 0
 55 22 11 33 44 88 77 66 99
```



①から⑨の順に訪問する。

```
% ./a.out
11 22 33 44 55 66 77 88 99 0
 11 22 33 44 55 66 77 88 99

% ./a.out
99 88 77 66 55 44 33 22 11 0
 99 88 77 66 55 44 33 22 11

% ./a.out
22 11 55 44 33 99 88 77 66 0
 22 11 55 44 33 99 88 77 66
```

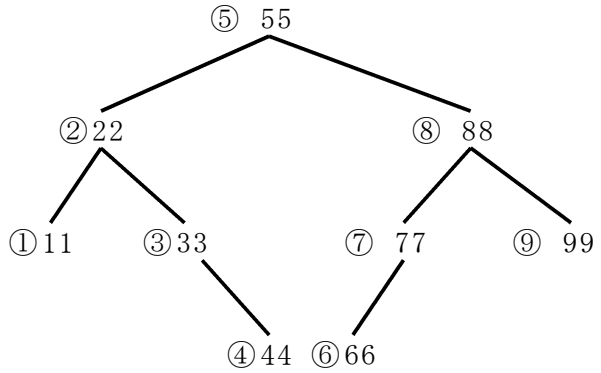
３．２　中走査

●プログラム・再帰版（bt321.c）

```
1  /* << bt321.c >> */
2  /* ２分探索木の走査（中走査）。*/
3  #include <stdio.h>
4  #include <stdlib.h>
5  struct NODE {
6    int info;           /* 節点がもつデータ。*/
7    struct NODE *left;  /* 左の子節点を指すポインタ。*/
8    struct NODE *right; /* 右の子節点を指すポインタ。*/
9  };
10 void inorder(struct NODE *p);
11 struct NODE *mktree();
12
13 int main() {
14   struct NODE *root;
15
16   /* ２分探索木の作成。*/
17   root = mktree();
18
19   /* 中走査。*/
20   inorder(root->left); printf("¥n");
21 }
22
23 /* 中走査。*/
24 void inorder(struct NODE *p) {
25   if( p == NULL ) { return; }
26   inorder(p->left);
27   printf(" %d",p->info);
28   inorder(p->right);
29 }
```

実行結果

```
% cc bt321.c
% ./a.out
55 22 88 11 33 77 99 44 66 0
 11 22 33 44 55 66 77 88 99
```



①から⑨の順に訪問する。

```
% ./a.out
11 22 33 44 55 66 77 88 99 0
 11 22 33 44 55 66 77 88 99

% ./a.out
99 88 77 66 55 44 33 22 11 0
 11 22 33 44 55 66 77 88 99

% ./a.out
22 11 55 44 33 99 88 77 66 0
 11 22 33 44 55 66 77 88 99
```

３．３　問題

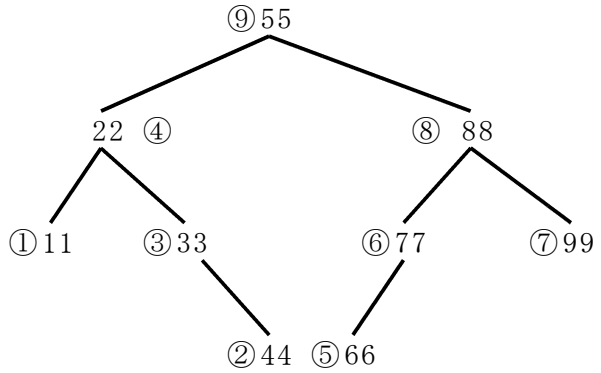
問題１　後走査

●プログラム・再帰版（bt331.c）

```
1  /* << bt331.c >> */
2  /* ２分探索木の走査（後走査）。*/
3  #include <stdio.h>
4  #include <stdlib.h>
5  struct NODE {
6    int info;           /* 節点がもつデータ。*/
7    struct NODE *left;  /* 左の子節点を指すポインタ。*/
8    struct NODE *right; /* 右の子節点を指すポインタ。*/
9  };
10 void postorder(struct NODE *p);
11 struct NODE *mktree();
12
13 int main() {
14   struct NODE *root;
15
16   /* ２分探索木の作成。*/
17   root = mktree();
18
19   /* 後走査。*/
20   postorder(①              ); printf("¥n");
21 }
22
23 /* 後走査。*/
24 void postorder(struct NODE *p) {
25   if( p == ②          ) { return; }
26   postorder(③            );
27   postorder(④            );
28   printf(" %d",p->info);
29 }
```

実行結果

```
% cc bt331.c
% ./a.out
55 22 88 11 33 77 99 44 66 0
 11 44 33 22 66 77 99 88 55
```



①から⑨の順に訪問する。

```
% ./a.out
11 22 33 44 55 66 77 88 99 0
 99 88 77 66 55 44 33 22 11

% ./a.out
99 88 77 66 55 44 33 22 11 0
 11 22 33 44 55 66 77 88 99

% ./a.out
22 11 55 44 33 99 88 77 66 0
 11 33 44 66 77 88 99 55 22
```

問題２

読み込んだデータから２分探索木を構成し、前走査、中走査、後走査でたどる順を示せ。

例　入力データ：55 22 88 11 33 77 99 44 66

前走査：55 22 11 33 44 88 77 66 99
中走査：11 22 33 44 55 66 77 88 99
後走査：11 44 33 22 66 77 99 88 55

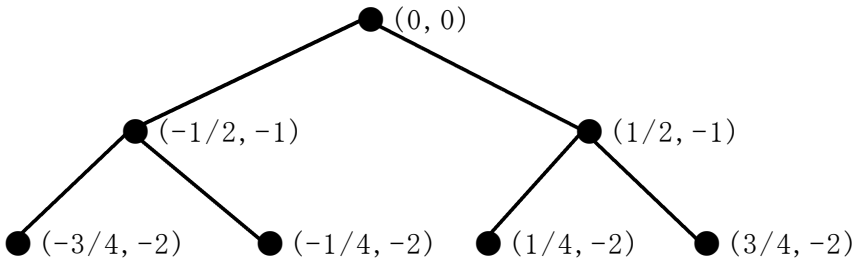
（１）入力データ：44 11 33 77 66 55 99 88 22

前走査	⑤
中走査	⑥
後走査	⑦

４．２分探索木の表示

２分探索木をExcelのグラフ機能を使って表示する。

まず、各節点に座標を割り当てる。節点が交わらないように下図のように座標を決める。根の座標は(0,0)とする。



すなわち、２分探索木の深さが一つ増えるごとに、節点間の幅を1/2倍にする。

このプログラムで、節点の座標を求める。ただし、Excelのグラフ機能（散布図）を使って表示するため、前走査で節点を通過するたびに座標を出力する。

●プログラム・再帰版（bt411.c）

```
1  /* << bt411.c >> */
2  /* ２分探索木の走査（前走査）。*/
3  /* 節点の座標を出力する。*/
4  #include <stdio.h>
5  #include <stdlib.h>
6  struct NODE {
7    int info;           /* 節点がもつデータ。*/
8    struct NODE *left;  /* 左の子節点を指すポインタ。*/
9    struct NODE *right; /* 右の子節点を指すポインタ。*/
10 };
11 float width; /* 幅。*/
12 void preorder(struct NODE *p, float v, float h);
13 struct NODE *mktree();
14
15 int main() {
16   struct NODE *root;
17
18   /* ２分探索木の作成。*/
19   root = mktree();
20
21   /* 初期設定。*/
22   width = 1;
23
24   /* 前走査。*/
25   preorder(root->left,0,0);
26   printf("¥n");
```

```
27 }
28
29 /* 前走査に従って、節点のx座標v,y座標hを出力する。*/
30 void preorder(struct NODE *p, float v, float h) {
31   if( p == NULL ) { return; }
32
33   /* 座標(v,-h)の出力。*/
34   printf("%8.5f,%8.5f¥n",v,-h);
35   width = width/2;
36   preorder(p->left,v-width,h+1);
37
38   /* 座標(v,-h)の出力。*/
39   printf("%8.5f,%8.5f¥n",v,-h);
40   preorder(p->right,v+width,h+1);
41   width = width*2;
42
43   /* 座標(v,-h)の出力。*/
44   printf("%8.5f,%8.5f¥n",v,-h);
45 }
```

実行結果

```
% cc bt411.c
% ./a.out
44 22 11 33 66 55 77 0
 0.00000,-0.00000
-0.50000,-1.00000
-0.75000,-2.00000
-0.75000,-2.00000
-0.75000,-2.00000
-0.50000,-1.00000
-0.25000,-2.00000
-0.25000,-2.00000
-0.25000,-2.00000
-0.50000,-1.00000
 0.00000,-0.00000
 0.50000,-1.00000
 0.25000,-2.00000
 0.25000,-2.00000
 0.25000,-2.00000
 0.50000,-1.00000
 0.75000,-2.00000
 0.75000,-2.00000
 0.75000,-2.00000
 0.50000,-1.00000
 0.00000,-0.00000
```

● ２分探索木の表示

①節点の座標をcsvファイル（拡張子をcsvとする）にして保存する。

②Excelを起動し、csvファイルを読み込む。

③グラフ機能（散布図）使ってグラフを表示する。
余分な座標軸等は削除する。

x座標	y座標
0	0
−0.5	−1
−0.75	−2
−0.75	−2
−0.75	−2
−0.5	−1
−0.25	−2
−0.25	−2
−0.25	−2
−0.5	−1
0	0
0.5	−1
0.25	−2
0.25	−2
0.25	−2
0.5	−1
0.75	−2
0.75	−2
0.75	−2
0.5	−1
0	0