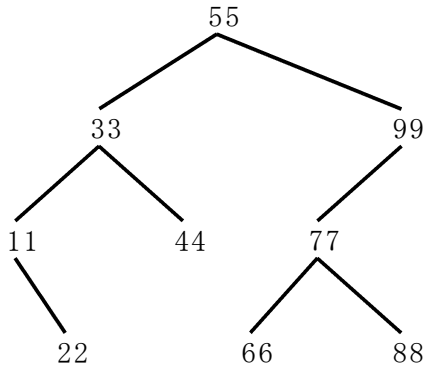
２分探索木Ⅱ

０．目次

５．２分探索木の操作

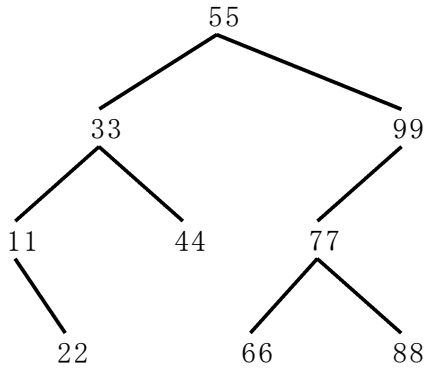
５．１　要素の探索

●要素44の探索



(1)要素55と44を比較して、左部分木をたどる。
(2)要素33と44を比較して、右部分木をたどる。
(3)要素44を見つけた。

●要素68の探索



(1)要素55と68を比較して、右部分木をたどる。
(2)要素99と68を比較して、左部分木をたどる。
(3)要素77と68を比較して、左部分木をたどる。
(4)要素66と68を比較して、右部分木をたどる。
(5)右部分木は空なので、要素68を見つけられなかった。

●プログラム・再帰版（bt511.c）

```
1  /* << bt511.c >> */
2  #include <stdio.h>
3  #include <stdlib.h>
4  struct NODE {
5    int info;           /* 節点がもつデータ。*/
6    struct NODE *left;  /* 左の子節点を指すポインタ。*/
7    struct NODE *right; /* 右の子節点を指すポインタ。*/
8  };
9  struct NODE *mktree();
10 struct NODE *search(struct NODE *p, int x);
11
12 int main() {
13   int x;
14   struct NODE *root,
15               *pos;
16
17   /* ２分探索木の作成。*/
18   root = mktree();
19
20   /* 探索。*/
21   while( 1 ) {
22     /* 探索データxの読み込み。*/
23     scanf("%d",&x);
24     if( x <= 0 ) { break; }
25     printf("探索データ：%d  ",x);
26
27    /* データxを指定し探索。pos:要素xの位置。*/
28     pos = search(root->left,x);
29     if( pos != NULL ) {
30       printf("%d 見つかりました¥n",pos->info);
31     } else {
32       printf("見つかりません¥n");
33     }
34   }
35 }
36
37 /* 探索。*/
38 struct NODE *search(struct NODE *p, int x) {
39   struct NODE *q; /* q:要素xの位置。*/
40
41   if( p == NULL ) { return NULL; }
42
43   /* 要素xを見つけた場合。*/
44   if( x == p->info ) { return p; }
45
46   /* 要素xが見つからない場合。*/
47   if( x < p->info ) {
48     /* 左部分木を探索する。*/
```

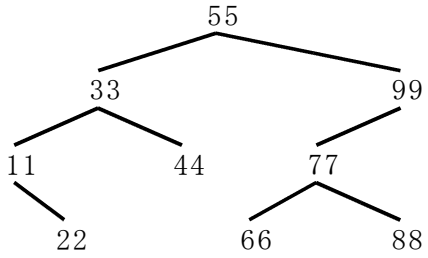
```
49      q = search(p->left,x);
50    } else {
51      /* 右部分木を探索する。*/
52      q = search(p->right,x);
53    }
54    return q;
55  }
```

実行結果

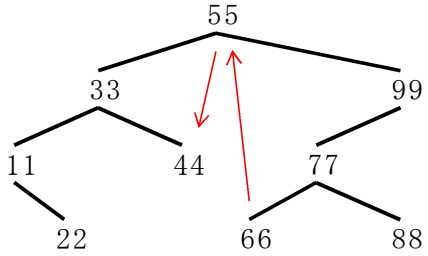
```
% cc bt511.c
% a.out
55 33 99 11 44 77 22 66 88 0
44
探索データ：44   44 見つかりました
68
探索データ：68   見つかりません
0
```

５．２　直前の要素の探索

直前の要素の要素の探索を行う。２分探索木において、直前の要素は、離れた位置にある。たとえば、

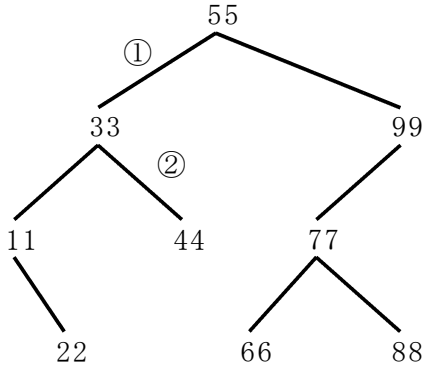


において、55の直前の要素44は、55の左部分木にあり、66の直前の要素55は、親をたどった先にある。



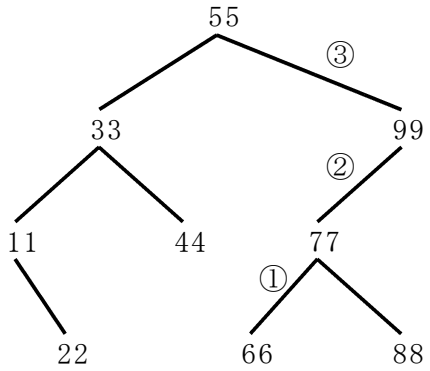
すなわち、直前の要素は、指定した要素に左部分木がある場合、左部分木の中で最大の要素、左部分木がない場合、親をたどっていき、最初に右部分木へ分岐した辺の親節点である。

●要素55の直前の要素44の探索



(1)要素55を見つけるまで２分探索木をたどる。
(2)要素55を見つけたら、左部分木があるかないか調べ、
　ある場合、左部分木の節点に移動し、その木の右部分木を最後までたどる。最後の節点に求める要素がある。
　ない場合、親節点をたどり、最初に右部分木へ分岐した辺の親節点に求める要素がある。

●要素66の直前の要素55の探索



(1)要素66を見つけるまで２分探索木をたどる。
(2)要素66を見つけたら、左部分木があるかないか調べ、
ある場合、左部分木の節点に移動し、その木の右部分木を最後まで
たどる。最後の節点に求める要素がある。
ない場合、親節点をたどり、最初に右部分木へ分岐した辺の親節点に
求める要素がある。
手順は、①,②,③とたどり親節点を探す代わりに、要素66を
見つける途中で右部分木へ移動するときに、親節点の位置を
記憶しておく。

●プログラム・再帰版（bt521.c）

```
1  /* << bt521.c >> */
2  #include <stdio.h>
3  #include <stdlib.h>
4  struct NODE {
5    int info;           /* 節点がもつデータ。*/
6    struct NODE *left;  /* 左の子節点を指すポインタ。*/
7    struct NODE *right; /* 右の子節点を指すポインタ。*/
8  };
9  struct NODE *mktree();
10 struct NODE *before(struct NODE *root, int x,struct NODE *t);
11
12 int main() {
13   int x;
14   struct NODE *root,
15               *pos;
16
17   /* ２分探索木の作成。*/
18   root = mktree();
19
20   while( 1 ) {
21     /* データxの読み込み。*/
22     scanf("%d",&x);
23     if( x <= 0 ) { break; }
24     printf("データ：%d  ",x);
25
26     /* データxを指定し探索する。pos:直前の要素の位置。*/
27     pos = before(root->left,x,NULL);
28     if( pos != NULL ) {
29       printf("直前のデータ：%d¥n",pos->info);
30     } else {
31       printf("直前のデータなし¥n");
32     }
33   }
34 }
35
36 /* 直前のデータを求める。*/
37 struct NODE *before(struct NODE *p, int x, struct NODE *t) {
38   struct NODE *q; /* q:直前の要素の位置。*/
39
40   if( p == NULL ) { return NULL; }
41   /* 要素xを見つけた場合。*/
42   if( x == p->info ) {
43     if( p->left != NULL ) {
44       /* 左部分木がある場合。*/
45       p = p->left;
46       while( p->right != NULL ) { p = p->right; }
47       return p;
48     } else {
```

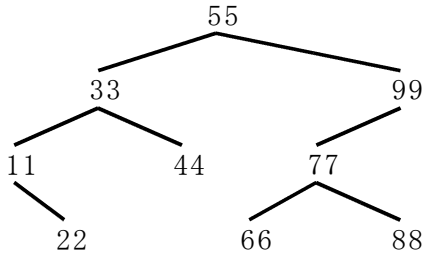
```
49        /* 左部分木がない場合。*/
50        return t;
51      }
52    }
53
54    /* 要素xが見つからない場合。*/
55    if( x < p->info ) {
56      /* 左部分木へ進む。*/
57      q = before(p->left,x,t);
58    } else {
59      /* 右部分木に進んだとき、節点pを保存する。*/
60      q = before(p->right,x,p);
61    }
62    return q;
63  }
```

実行結果

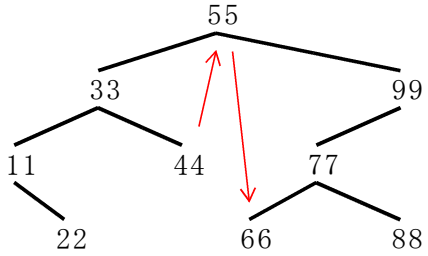
```
% cc bt521.c
% ./a.out
55 33 99 11 44 77 22 66 88 0
11
データ：11　直前のデータなし
22
データ：22　直前のデータ：11
33
データ：33　直前のデータ：22
44
データ：44　直前のデータ：33
55
データ：55　直前のデータ：44
66
データ：66　直前のデータ：55
77
データ：77　直前のデータ：66
88
データ：88　直前のデータ：77
99
データ：99　直前のデータ：88
0
```

５．３　直後の要素の探索

直後の要素の探索を行う。２分探索木において、直前及び直後の要素は、離れた位置にある。たとえば、

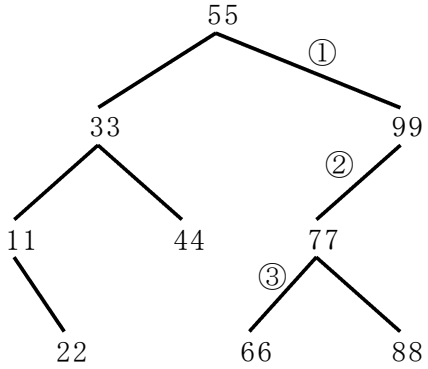


において、44の直後の要素55は、親をたどった先にあり、55の直後の要素66は、55の右部分木にある。



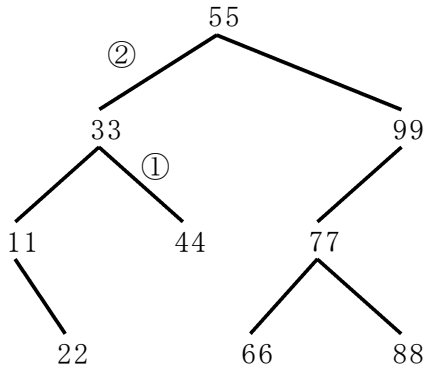
すなわち、直後の要素は、指定した要素に右部分木がある場合、右部分木の中で最小の要素、右部分木がない場合、親をたどり、最初に左部分木へ分岐した辺の親節点である。

●要素55の直後の要素66の探索



(1)要素55を見つけるまで２分探索木をたどる。
(2)要素55を見つけたら、右部分木があるかないか調べ、
　ある場合、右部分木の節点に移動し、その木の左部分木を最後までたどる。最後の節点に求める要素がある。
　ない場合、親節点をたどり、最初に左部分木へ分岐した辺の親節点に求める要素がある。

●要素44の直後の要素55の探索



(1)要素44を見つけるまで２分探索木をたどる。
(2)要素44を見つけたら、右部分木があるかないか調べ、
ある場合、右部分木の節点に移動し、その木の左部分木を最後まで
たどる。最後の節点に求める要素がある。
ない場合、親節点をたどり、最初に左部分木へ分岐した辺の親節点に
求める要素がある。
手順は、①,②とたどり親節点を探す代わりに、要素44を
見つける途中で左部分木へ移動するときに、親節点の位置を
記憶しておく。

●プログラム・再帰版（bt531.c）　直後のデータを求めるプログラム

```
1  /* << bt531.c >> */
2  #include <stdio.h>
3  #include <stdlib.h>
4  struct NODE {
5    int info;           /* 節点がもつデータ。*/
6    struct NODE *left;  /* 左の子節点を指すポインタ。*/
7    struct NODE *right; /* 右の子節点を指すポインタ。*/
8  };
9  struct NODE *mktree();
10 struct NODE *after(struct NODE *root, int x,struct NODE *t);
11
12 int main() {
13   int x;
14   struct NODE *root,
15               *pos;
16
17   /* ２分探索木の作成。*/
18   root = mktree();
19
20   while( 1 ) {
21     /* データの読み込み。*/
22     scanf("%d",&x);
23     if( x <= 0 ) { break; }
24     printf("データ：%d  ",x);
25
26     /* データxを指定し探索する。pos:直後の要素の位置。*/
27     pos = after(root->left,x,NULL);
28     if( pos != NULL ) {
29       printf("直後のデータ：%d¥n",pos->info);
30     } else {
31       printf("直後のデータなし¥n");
32     }
33   }
34 }
35
36 /* 直後のデータを求める。*/
37 struct NODE *after(struct NODE *p, int x, struct NODE *t) {
38   struct NODE *q;
39
40   if( p == NULL ) { return NULL; }
41
42   /* 要素xを見つけた場合。*/
43   if( x == p->info ) {
44     if( p->right != NULL ) {
45       /* 右部分木がある場合。*/
46       p = p->right;
47       while( p->left != NULL ) {
48         p = p->left;
```

```
49        }
50        return p;
51      } else {
52        /* 右部分木がない場合。*/
53        return t;
54      }
55    }
56
57    /* 要素xが見つからない場合。*/
58    if( x < p->info ) {
59      /* 左部分木に進んだとき、節点pを保存する。*/
60      q = after(p->left,x,p);
61    } else {
62      /* 右部分木に進む。*/
63      q = after(p->right,x,t);
64    }
65    return q;
66  }
```

実行結果

```
% cc bt531.c
% ./a.out
55 33 99 11 44 77 22 66 88 0
11
データ：11　直後のデータ：22
22
データ：22　直後のデータ：33
33
データ：33　直後のデータ：44
44
データ：44　直後のデータ：55
55
データ：55　直後のデータ：66
66
データ：66　直後のデータ：77
77
データ：77　直後のデータ：88
88
データ：88　直後のデータ：99
99
データ：99　直後のデータなし
0
```

５．４　要素の削除

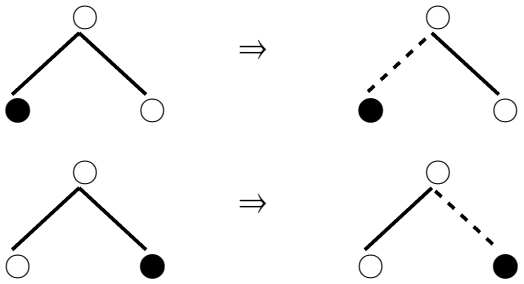
２分探索木から要素を削除する方法を考察する。

●考え方

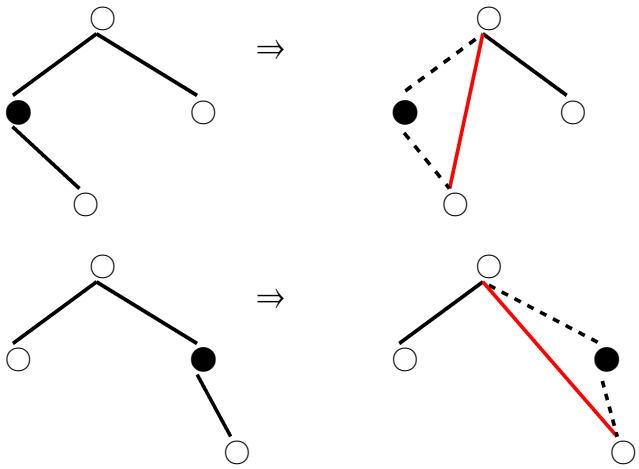
２分探索木から、指定した要素を削除する。次の４つの場合が考えられる。

（場合１）削除するデータ●が葉の場合。

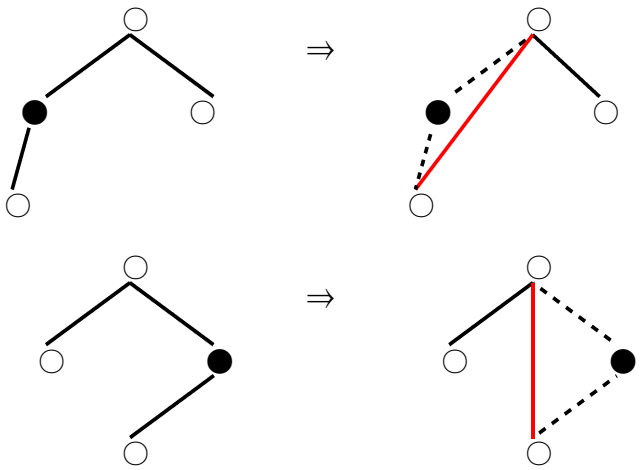
親節点からの辺を取り除けばよい。



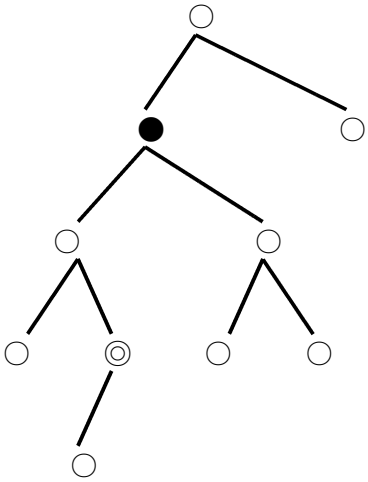
（場合２）削除するデータ●の左部分木が空、右部分木が空でない場合。



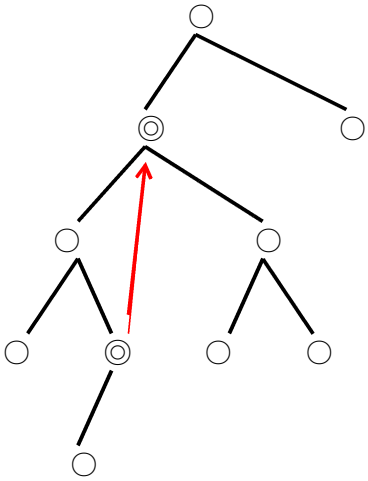
（場合３）削除するデータ●の左部分木が空でなく、右部分木が空の場合。



（場合４）削除するデータ●の左部分木、右部分木が空でない場合。

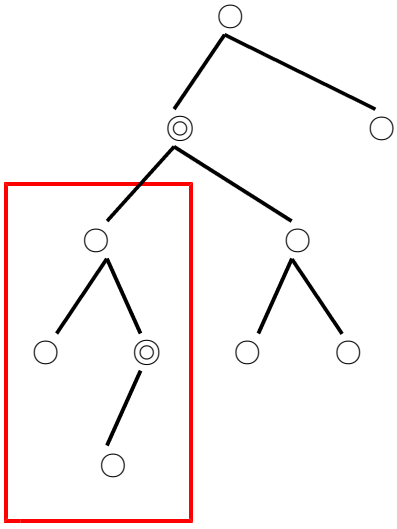


（４．１）データ●の直前のデータ◎を探し、●の位置に◎を代入する。



この位置ならば、全体の構成に
変更を生じない。

（４．２）●の位置の左部分木から◎を削除する操作を続ける。
最後には、場合１，２，３のいずれかになり終了する。



●プログラム・再帰版（bt541.c）　要素を削除するプログラム

```
1  /* << bt541.c >> */
2  #include <stdio.h>
3  #include <stdlib.h>
4  struct NODE {
5    int info;           /* 節点がもつデータ。*/
6    struct NODE *left;  /* 左の子節点を指すポインタ。*/
7    struct NODE *right; /* 右の子節点を指すポインタ。*/
8  };
9  void inorder(struct NODE *p);
10 struct NODE *mktree();
11 void del(struct NODE *root, int x);
12
13 int main() {
14   int x;
15   struct NODE *root;
16
17   /* ２分探索木の作成。*/
18   root = mktree();
19
20   while( 1 ) {
21     /* 削除データxの読み込み。*/
22     scanf("%d",&x);
23     if( x <= 0 ) { break; }
24     printf("削除前     : ");
25     inorder(root->left); printf("¥n");
26     printf("削除データ :  %d¥n",x);
27
28     /* 削除データxを指定する。*/
29     del(root,x);
30     printf("削除後     : ");
31     inorder(root->left); printf("¥n");
32   }
33 }
34
35 /* データの削除。*/
36 void del(struct NODE *root, int x) {
37   struct NODE *before(struct NODE *p, int x, struct NODE *t);
38   struct NODE *p,
39               *q,
40               *r;
41
42   p = root->left;
43   q = root;
44
45   /* 要素xを探す。*/
46   while( p != NULL ) {
47     if( x == p->info ) { break; }
48     q = p;
```

```
49      if( p->info < x ) { p = p->right; } else { p = p->left; }
50    }
51    if( p == NULL ) { return; }
52
53    /* 削除操作。*/
54
55    /* 場合１：削除節点が葉の場合。*/
56    if( (p->left == NULL)&&(p->right == NULL) ) {
57      if( p == q->left ) {
58        /* 削除データが左部分木にある。*/
59        q->left = NULL;
60      } else {
61        /* 削除データが右部分木にある。*/
62        q->right = NULL;
63      }
64      return;
65    }
66
67    /* 場合２：削除節点の左部分木が空の場合。*/
68    if( p->left == NULL ) {
69      if( p == q->left ) {
70        /* 削除データが左部分木にある。*/
71        q->left = p->right;
72      } else {
73        /* 削除データが右部分木にある。*/
74        q->right = p->right;
75      }
76      return;
77    }
78
79    /* 場合３：削除節点の右部分木が空の場合。*/
80    if( p->right == NULL ) {
81      if( p == q->left ) {
82        /* 削除データが左部分木にある。*/
83        q->left = p->left;
84      } else {
85        /* 削除データが右部分木にある。*/
86        q->right = p->left;
87      }
88      return;
89    }
90
91    /* 場合４ */
92    r = before(p,x,NULL);
93    p->info = r->info;
94    del(p,r->info);
  }
```

実行結果

```
% cc bt541.c
% ./a.out
55 22 44 11 33 66 0

55
削除前     :  11 22 33 44 55 66
削除データ :  55
削除後     :  11 22 33 44 66

22
削除前     :  11 22 33 44 66
削除データ :  22
削除後     :  11 33 44 66

66
削除前     :  11 33 44 66
削除データ :  66
削除後     :  11 33 44

11
削除前     :  11 33 44
削除データ :  11
削除後     :  33 44

33
削除前     :  33 44
削除データ :  33
削除後     :  44

44
削除前     :  44
削除データ :  44
削除後     :
0
```

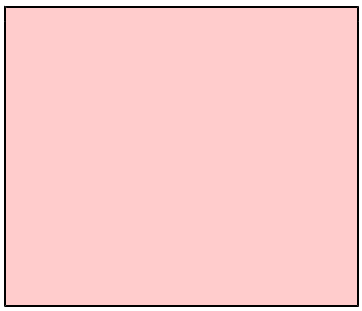
５．５　問題

問題１

データを読み込み、二分探索木を作成する。つぎに、指定された順に要素を削除した後の二分探索木を求めよ。ただし、講義で説明した方法を使うこと。

（１）入力データ：44,22,66,11,33,55,77
削除する要素：44,33,66

①



（２）入力データ：77,22,88,99,11,66,44,33,55
削除する要素：77,66,55

②