アルゴリズムとデータ構造

第5回探索のためのデータ構造（1）

付録：二分探索木の C言語による実装のソースコード

付録：二分探索木のC言語による実装のソースコード

この付録は，演習・自己学習の参考資料にしてください

- ソースコードは，インクルード文から最後のmain文までを，一つのファイルに打ち込めば実行できます．
- テストコード（main文）は二種類添付しています．テストしたい関数に応じて，どちらか一つだけ使ってください．（main文は一つのプログラムに一つしか入れられないので，注意してください）
- 下記OS上のC言語/C++言語のコンパイラであるccまたはgccなどでコンパイル・実行可能です．
  - ✓ Windows (MinGW)
  - ✓ Mac (コンソール/Xcode)
  - ✓ Linux（ターミナル）
- Mac上のcc/gccで動作確認しています．

演習・自己学習の参考資料

付録：C言語のソースコード1


```
/* bsearch.c */
#include <stdio.h>
#include <stdlib.h>

/* ノードの定義 */
typedef struct _node {
  int key;              /* 要素（キー） */
  struct _node *right;  /* 右の子へのポインタ */
  struct _node *left;   /* 左の子へのポインタ */
} node;

/* 二分探索木の定義 */
typedef struct _tree {
  node *root; /* 木の根へのポインタ */
} tree;

/* 空の二分探索木を作成する関数 */
tree* create() {
  tree *T = (tree *)malloc(sizeof(tree));
  T->root = NULL;
  return T;
}
```

ノードのnode型は次をもつ
- 要素（整数）keyと，
- 右の子へのポインタ right，
- 左の子へのポインタ left

二分探索木 T 自身は，根のノードを指すポインタrootをもつ

復習：C言語のメモリ関係の命令
- **struct** cell {・・・}：構造体cellを・・・と定義
- **typedef** ・・・ cell：・・・をデータタイプcell型として定義
- **sizeof**(cell)：cell型のデータサイズ（バイト）
- **malloc**(n)：nバイトのメモリーを確保
- **(cell *)**malloc(n)：確保したnバイトの領域をcell型データ格納領域とみなす。

空の二分探索木を新しく作る関数
tree *T = create()
のようにして使用する

演習・自己学習の参考資料

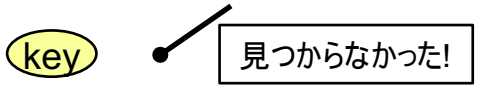
付録：C言語のソースコード２

```
/* 木の探索 */
/* 根 x をもつ二分探索木でキーkeyを要素にもつノードを
 * 探索する再帰関数（大事）．もしキーkeyを要素にもつノードzが
 * あればそれを返し，なければNULLを返す．*/
node *search_rec(node *x, int key) {
  if (x == NULL) /* (1) 空のとき */
    return x; /* NULL */                    /*見つからなかった */
  else if (key == x->key) /* (2) キーと等しいとき */
    return x;                                /* 見つけた! */
  else if (key < x->key) {  /* (3) キーが小さいとき */
    return search_rec(x->left, key);   /* 再帰的に探す*/
  }
  else {  /* key > x->key  */   /* (4) キーが大きいとき *
    return search_rec(x->right, key); /* 再帰的に探す*/
  }
}

/* 二分探索木 T で，キー keyを探索する演算 */
void search(tree *T, int key) {
  node *x = search_rec(T->root, key);
  if (x == NULL)  printf("Not Found¥n");    /* 見つからなかった */
  else  ("Found¥n");                          /* 見つけた! */
}
```

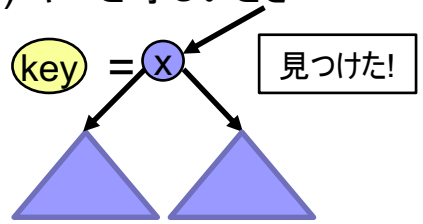
二分探索木の探索

根の値と比べて，場合分けする

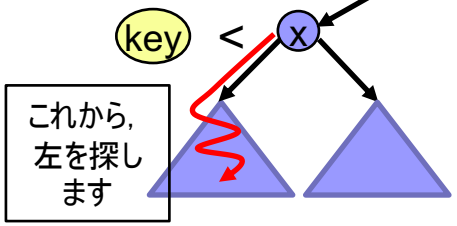
(1) 空のとき



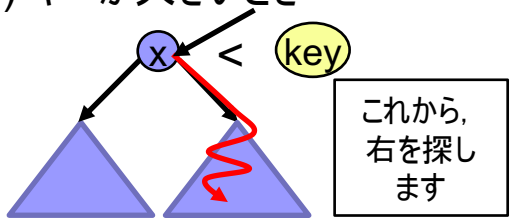
(2) キーと等しいとき



(3) キーが小さいとき



(4) キーが大きいとき



付録：C言語のソースコード３

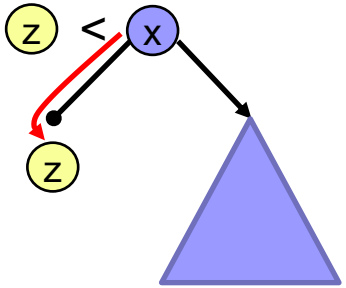
```
/* 根 x をもつ木にノード z を追加する再帰関数*/
void insert_rec(node *x, node *z) {
 if (z->key < x->key) {                /* (1) 左の子に行く */
  if (x->left == NULL)          /* (1.a) 左の子が空のとき*/
   x->left = z;  /*   直にzを挿入する */
  else                                  /* (1.b) それ以外の場合 */
   insert_rec(x->left, z); /*右の部分木に再帰的に挿入 */
 }
 else {  /* z->key >= x->key */       /* (2) 右の子に行く */
  if (x->right == NULL)         /* (2.a) 右の子が空のとき */
   x->right = z; /*直ちにzを挿入する*/
  else                                  /* (2.b) それ以外の場合 */
   insert_rec(x->right, z); /*左の部分木に再帰的に挿入 */
 }
}
                        /* 注意：上の実装では，同じ値の複数回の挿入を許している */

/* 二分探索木 T に，キー keyを挿入する演算 */
void insert(tree *T, int key) {
  node *z = (node *)malloc(sizeof(node)); /* 新しく挿入するノード */
  z->key   = key; z->right = NULL;  z->left  = NULL;
  if (T->root == NULL) { T->root = z; }
  else { insert_rec(T->root, z);
}
```

二分探索木の挿入

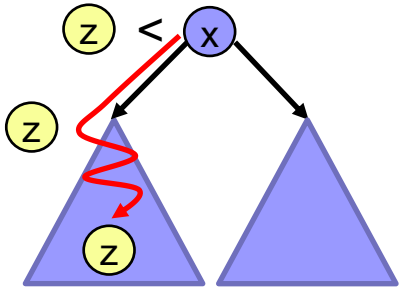
ケース1）挿入するzの値が，根xの値より小さい時（左にあるとき）

z->key < x->key

ケース1.a）左の子が空のとき



ケース1.b）左の子が空でないとき



ケース2）zの値が根xの値より大きいとき（右にあるとき）も同様（省略）

z->key >= x->key

付録：C言語のソースコード４

二分探索木

```
/* 通りがけ順（inorder）の印刷 */

/* 根 x をもつ二分木の要素を，通りがけ順（inorder）で
 * 印刷する再帰関数（大事） */
void print_inorder_rec(node *x) {
  if(x == NULL)                    /* 葉で，子供がないので戻る */
    return;
  else {
    print_inorder_rec(x->left);    /* 左の木で再帰 */
    printf("%d ", x->key);
    print_inorder_rec(x->right);   /* 右の木で再帰 */
    return;
  }
}
```

出力関数（printf）の[印刷]位置

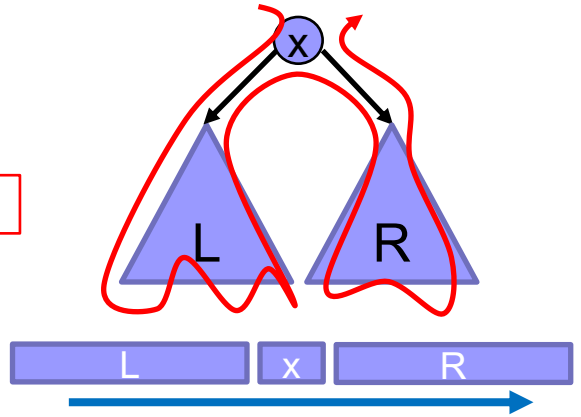
```
/* 二分木 T の要素を，通りがけ順で印刷する演算 */
void print_inorder(tree *T) {
  print_inorder_rec(T->root);
  printf("¥n");
  return;
}
```

二分木の通りがけ順（inorder）の印刷

(a) 葉のとき（NULL）

(b) 中間ノードのとき（非NULL）

- 左の木 Lを再帰的に印刷
- ノードxの値を印刷
- 右の木 R を再帰的に印刷



通りがけ順（inorder）の印刷

発展（ヒント）：行きがけ順（preorder）と帰りがけ順（postorder）の印刷は，通りがけ順（inorder）から，出力関数（printf）の[印刷]位置を変えるだけで実現できる（第6回授業予定）

- 行きがけ順は，[印刷]，左，右
- 通りがけ順は，左，[印刷]，右
- 帰りがけ順は，左，右，[印刷]

演習・自己学習の参考資料

付録: C言語のソースコード 5

二分探索木

```
/* 3_1.c
 * 挿入と印刷のテストコード
 */

/* ここに，二分探索木の定義と関数create, insert,
print_inorderのソースコードを貼り付ける
*/

int main() {
  /* 空の二分探索木Tを生成する  */
  tree *T = create();

  /* T に整数 7，3，10，1，4，8，6，5 を挿入 */
  insert(T,7);
  insert(T,3);
  insert(T,10);
  insert(T,1);
  insert(T,4);
  insert(T,8);
  insert(T,6);
  insert(T,5);

  print_inorder(T); /* とおりがけ順で印刷する */
  return 0;
}
```

```
/* 3_3.c
 * 探索のテストコード: 起動して，標準入力から，
5個の整数を改行または空白で区切って与えて，
検索結果に応じて”Found”または”Not Found”を
返す．  */

/* ここに，二分探索木の定義と関数create,
insert, searchのソースコードを貼り付ける
*/
int main() {
  tree *T = create();

  /* T に整数 7，3，10，1，4，8，6，5 を挿入 */
  insert(T,7);  insert(T,3); insert(T,10);
  insert(T,1);  insert(T,4);  insert(T,8);
  insert(T,6);  insert(T,5);

  int N = 5, key;  /* N=5個の入力を読み込む */
  for (int i = 0; i < N; i++) {
    /* 入力．各行に1つの整数を読み込む．  */
    scanf("%d", &key);
    printf("input:%d¥n", key);
    search(T, key);
  }
  return 0;
}
```

ソースコードは以上です