情報構造 第十二回

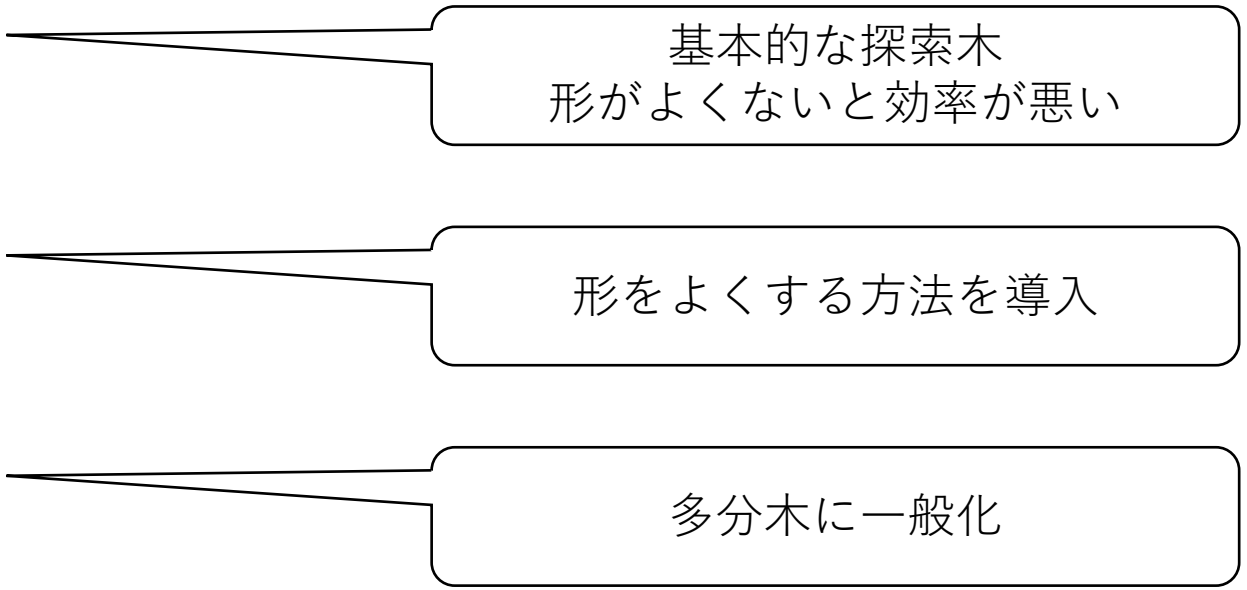
辞書の実現

2分探索木

AVL木

今日の予定

- 木による辞書の実現
  - 2分探索木
  - AVL木
  - B-木



- 続けて行うとごちゃごちゃするので，授業は3回に分けます

⇒今週は2分探索木

辞書の操作（前回）

辞書型 Dictionary　　要素型 Element

辞書型変数 D　　要素型データ x

- int Member(Element x, Dictionary D)
  - Post: 関数値は **x∈D** ならば真（1）さもなければ偽（0）
- Dictionary Insert (Element x, Dictionary D)
  - Post: 関数値は **D ∪ {x}**
- Dictionary Delete(Element x, Dictionary D)
  - Post: 関数値は **D - {x}**
- Dictionary Create (void)
  - Post: 関数値は，**空の辞書**
- Element Min(Dictionary D)
  - Pre: **D≠空の辞書**
  - Post: 関数値は**D**中の**最小要素の値**

辞書は集合
同じ型の集まり
要素の重複はない
並び順に意味はない

辞書の操作（改変）

辞書操作：辞書Dは木で表し，番地呼びに改変

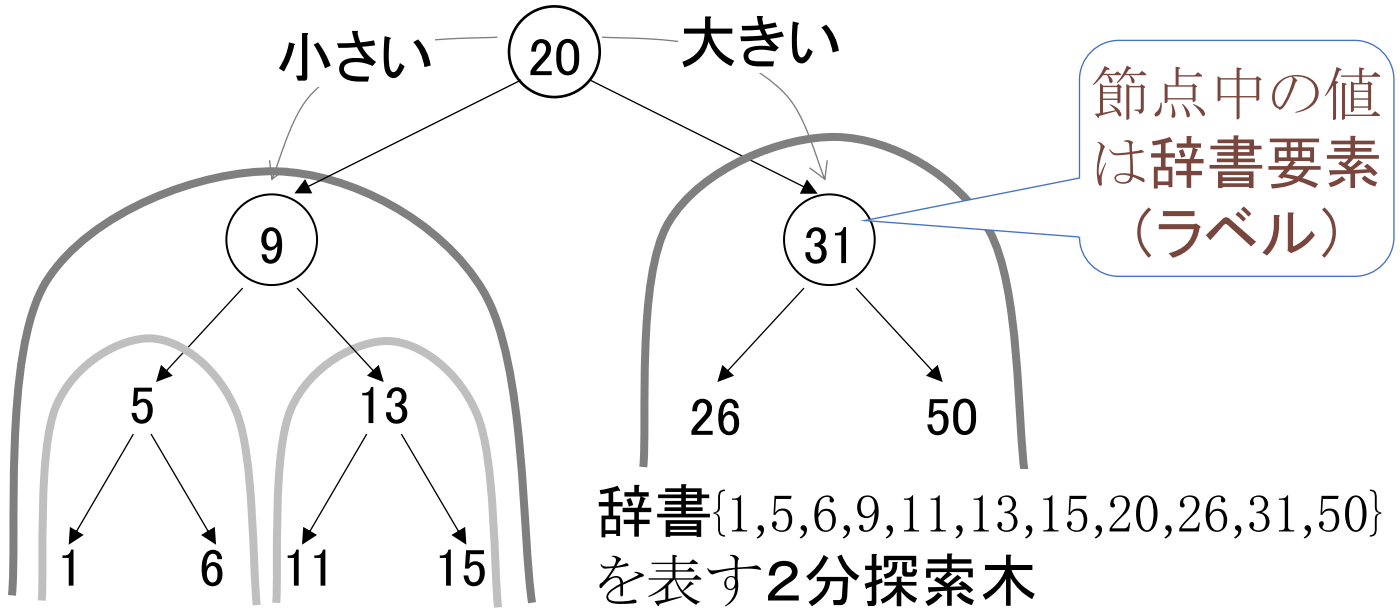
- int Member(Element x, Dictionary *D)
  - Post: 関数値は **x∈D** ならば真（1）さもなければ偽（0）
- void Insert (Element x, Dictionary *D)
  - Post: *Dは ***D** ∪ **{x}** に更新
- void Delete(Element x, Dictionary *D)
  - Post: *Dは **D - {x}**に更新
- void Create (Dictionary *D)
  - Post: *Dは **空の辞書**に更新
- Element Min(Dictionary *D)
  - Pre: ***D≠空の辞書**
  - Post: 関数値は**D**中の**最小要素の値**

木の特性を担保させる
↓
木の変更が必要になるため

2分探索木

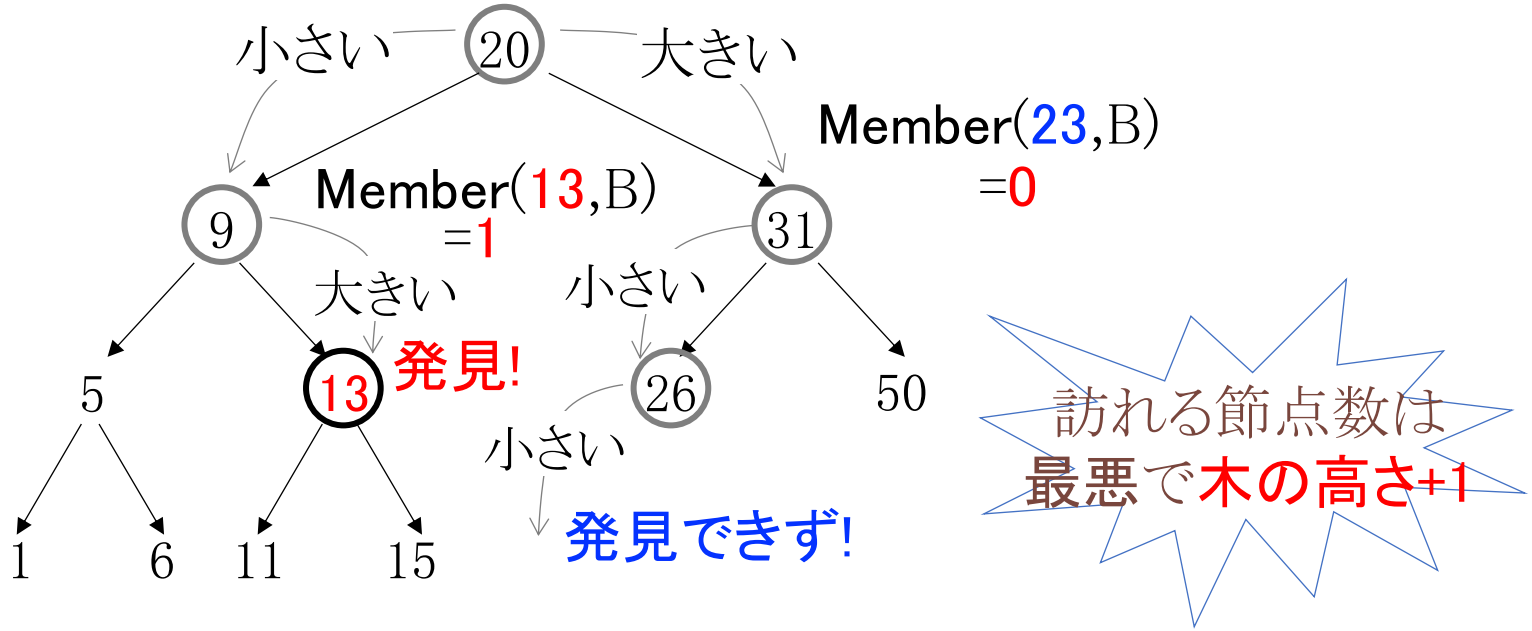
2分探索木

- 辞書の実現に適した2分木
- 2分木の節点のラベルとして，辞書要素を格納する
- 左部分木の辞書要素 < 任意節点の辞書要素 < 右部分木の辞書要素
- 操作は Member, Insert, Delete
- 通りがけ順にたどると，辞書要素のソート列が得られる



操作：Member(x, 2分探索木B)

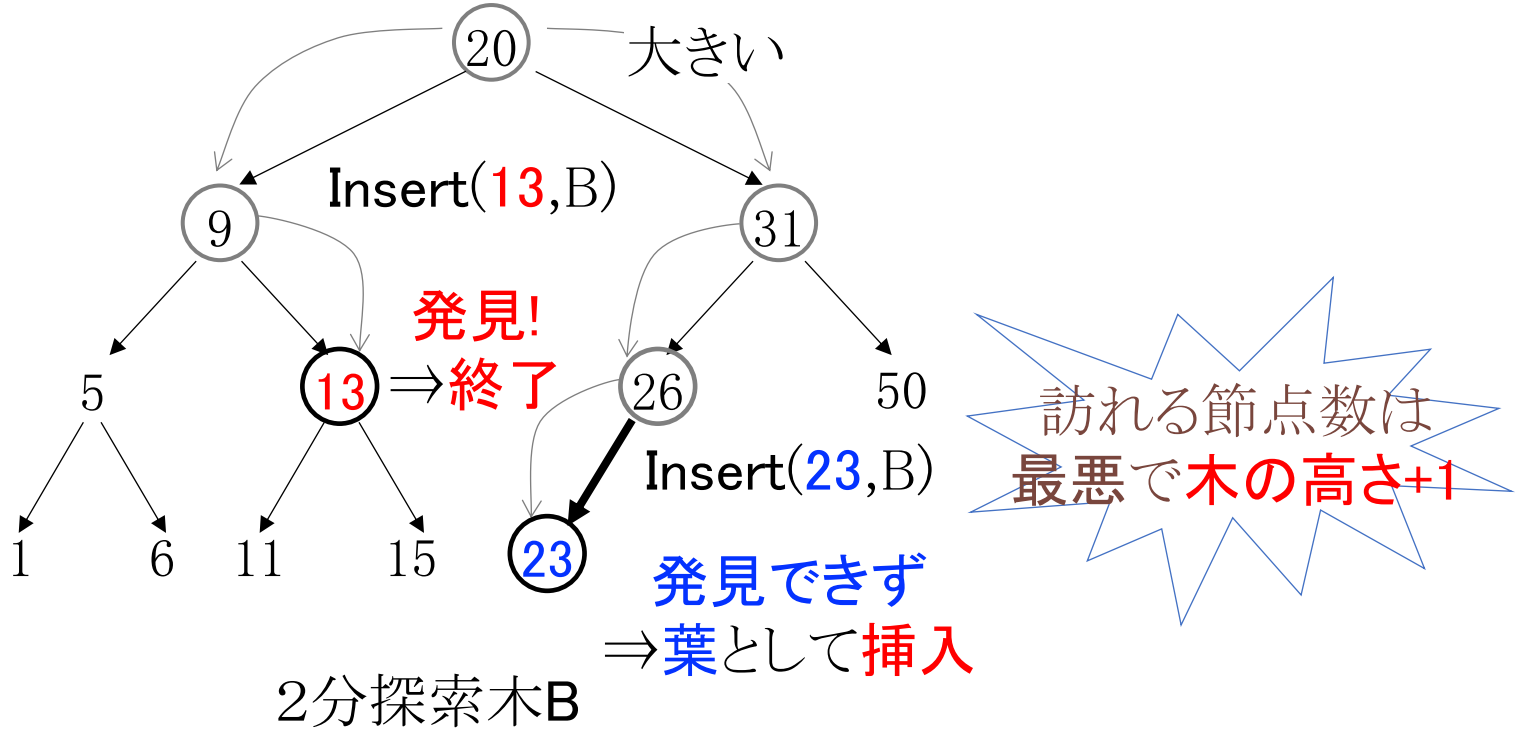
- **2分探索木B**のなかに，**x**の値があるか否か



2分探索木B

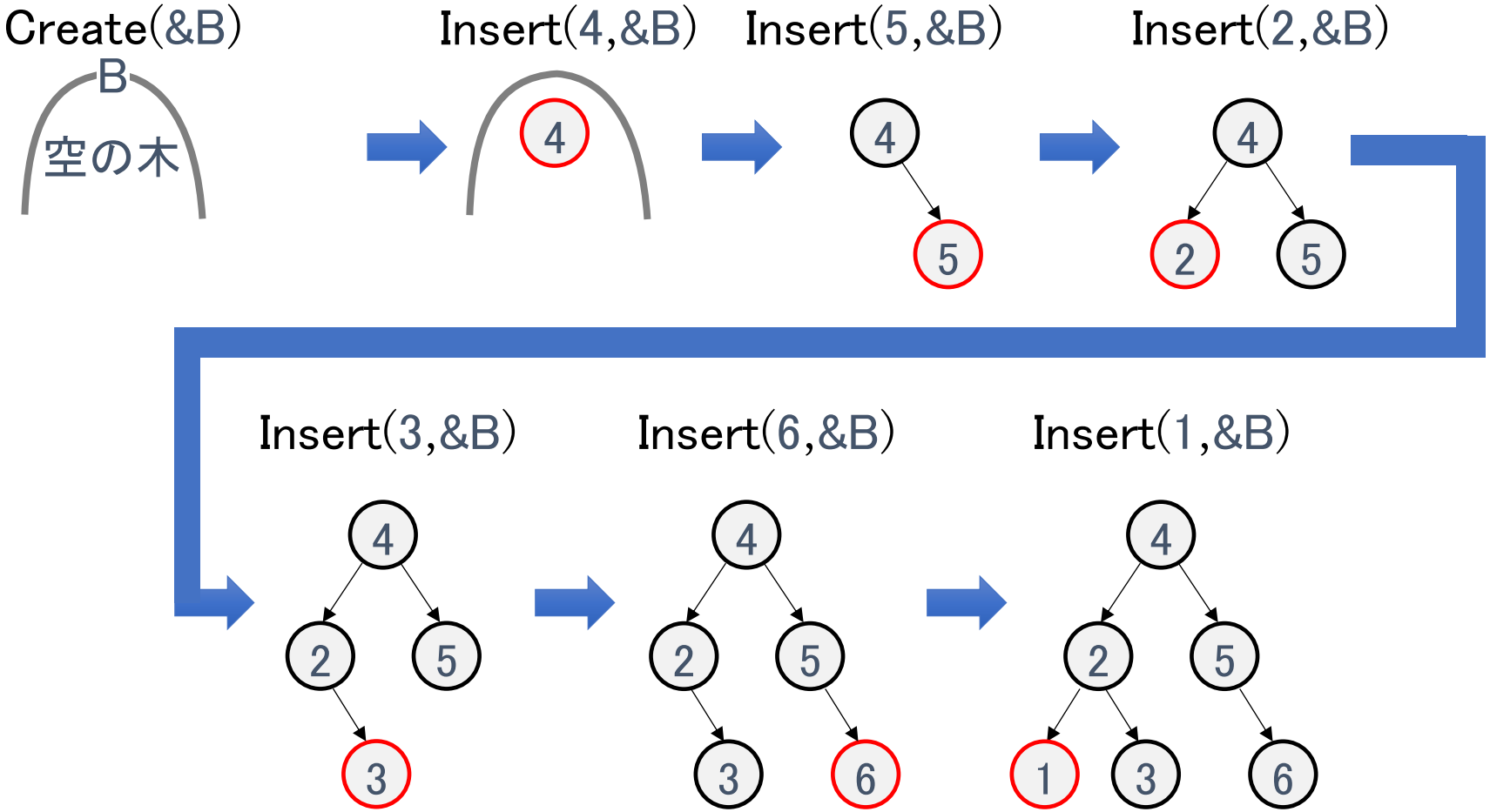
操作：Insert(x, 2分探索木B)

- **2分探索木B**に，性質を保ちつつ**x**を挿入



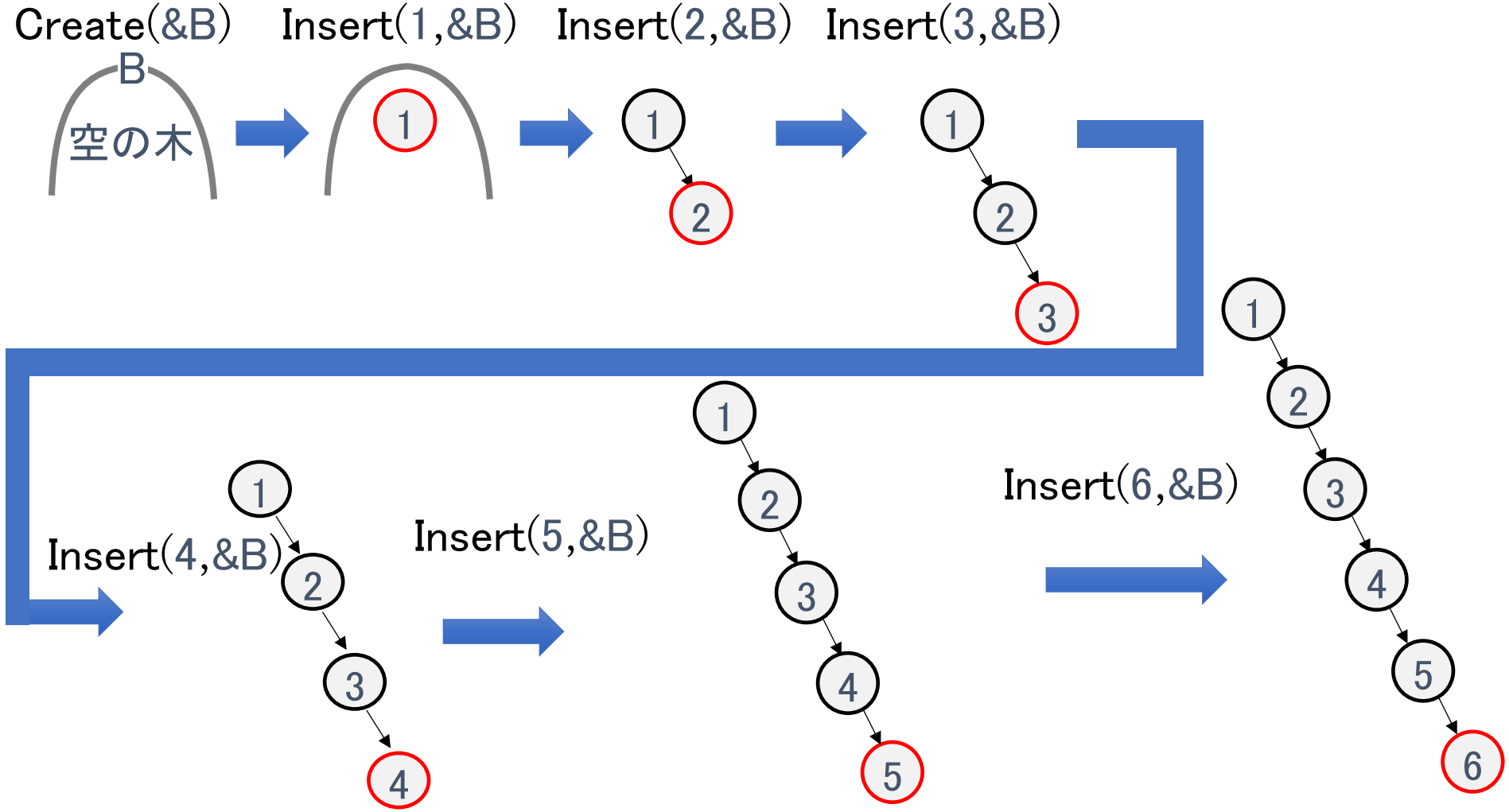
【例】辞書を表す2分探索木の構成①

- 値 4, 5, 2, 3, 6, 1をこの順で空の2分探索木Bに挿入する



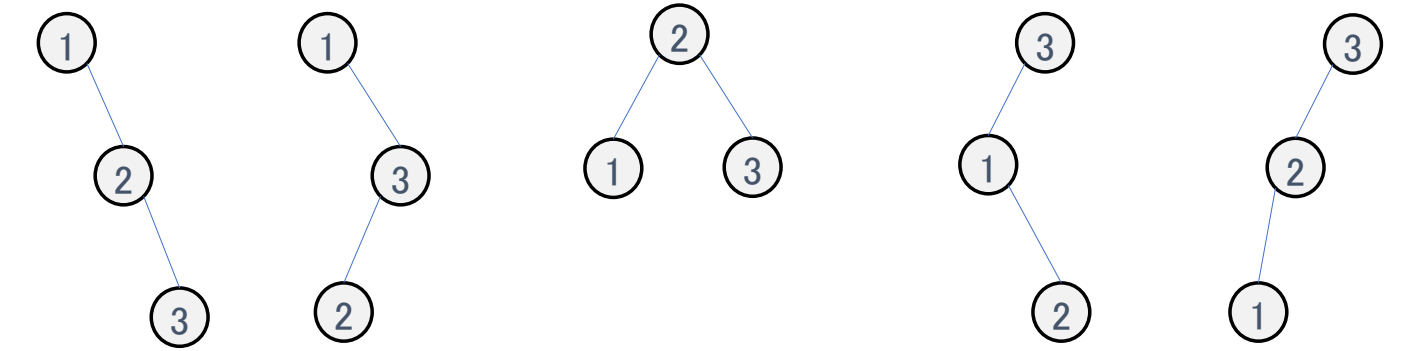
【例】辞書を表す2分探索木の構成②

- 値 1, 2, 3, 4, 5, 6をこの順で空の2分探索木Bに挿入する



【例】辞書を表す2分探索木の構成③

- 挿入する順序によって得られる木が変化する



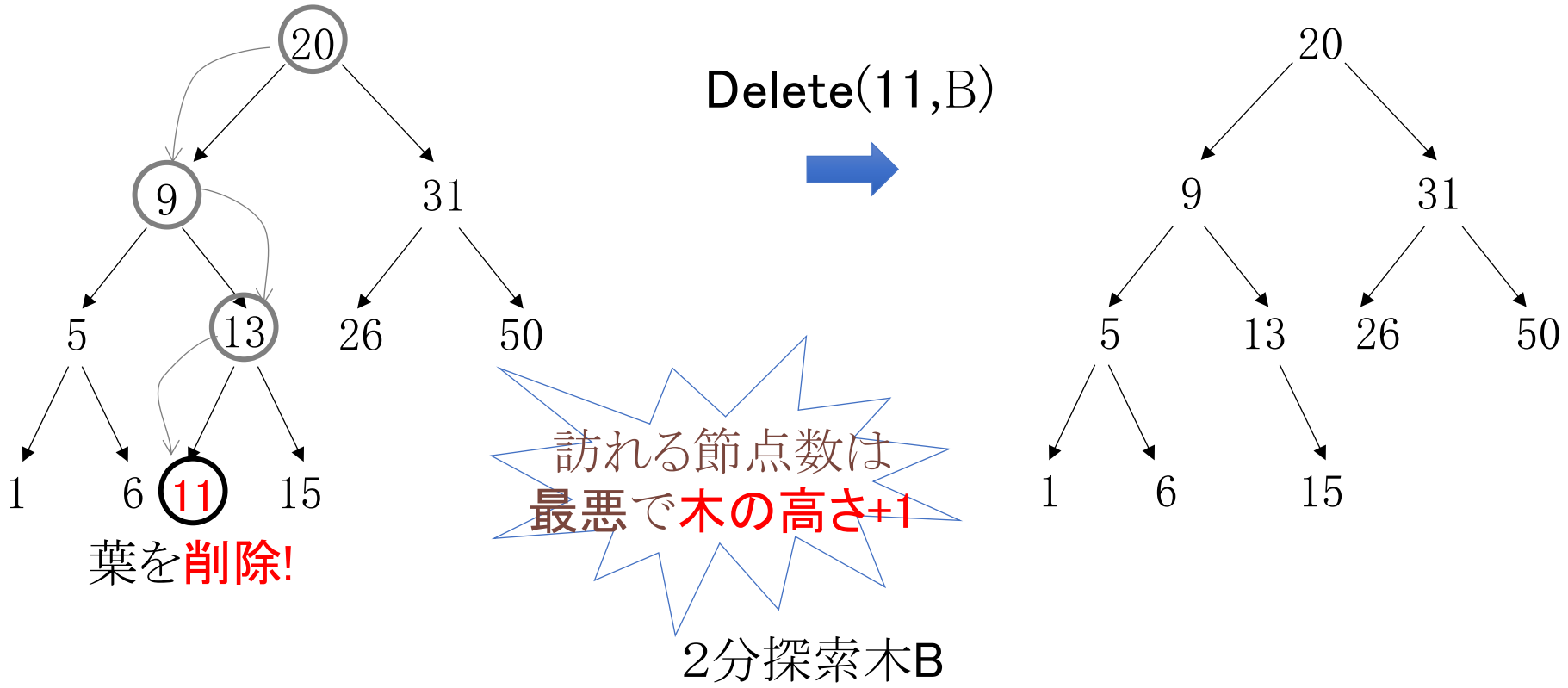
- データ数をNとすると考えられる順序はN!存在する

操作：Delete(x, 2分探索木B)

- 2分探索木Bから，性質を保ちつつxの値の節点を削除

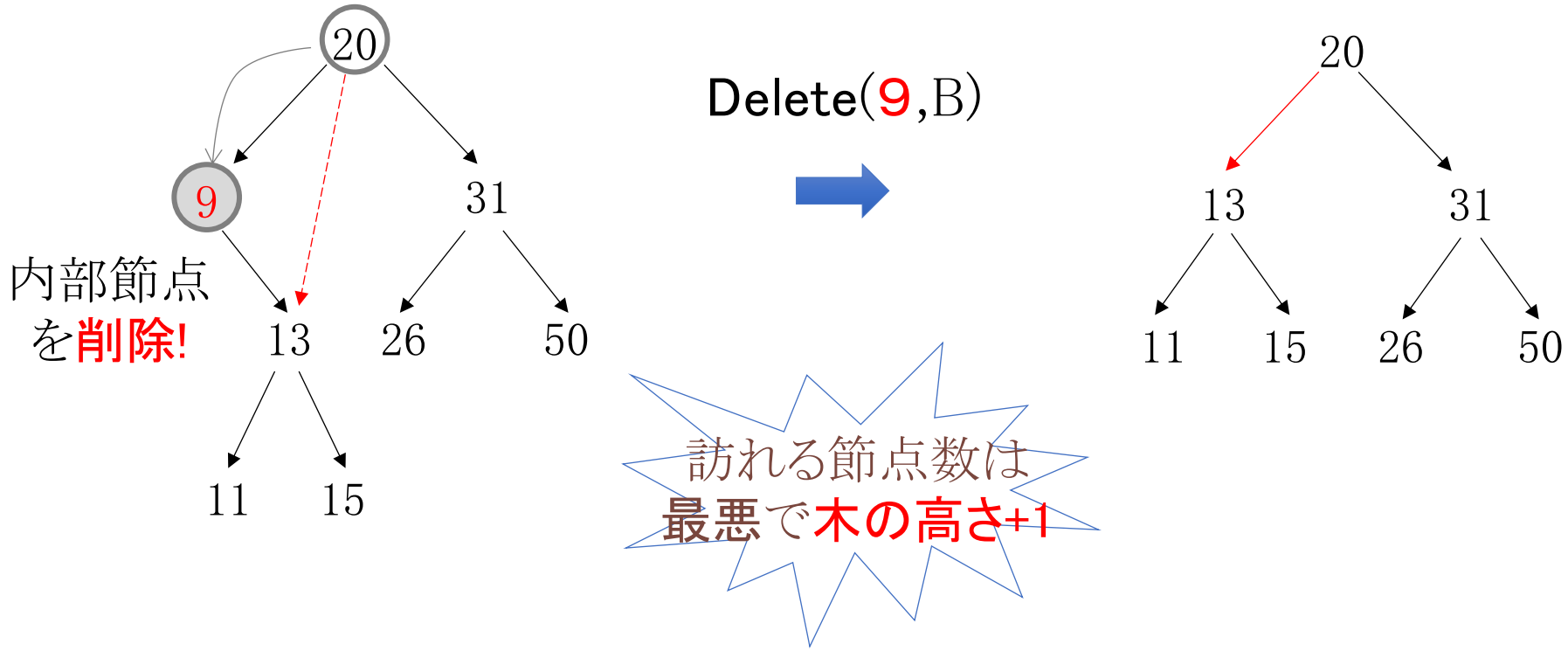
1. Bのなかにxがないとき，終了
2. Bのなかにxがあるとき，その節点を削除節点とし場合分けをする
   - 削除節点が葉のとき（子がいないとき）
   - 削除節点が子を1人もつとき
   - 削除節点が子を2人もつとき

Delete場合分け: 削除節点が葉のとき



Delete場合分け: 削除節点の子が1人

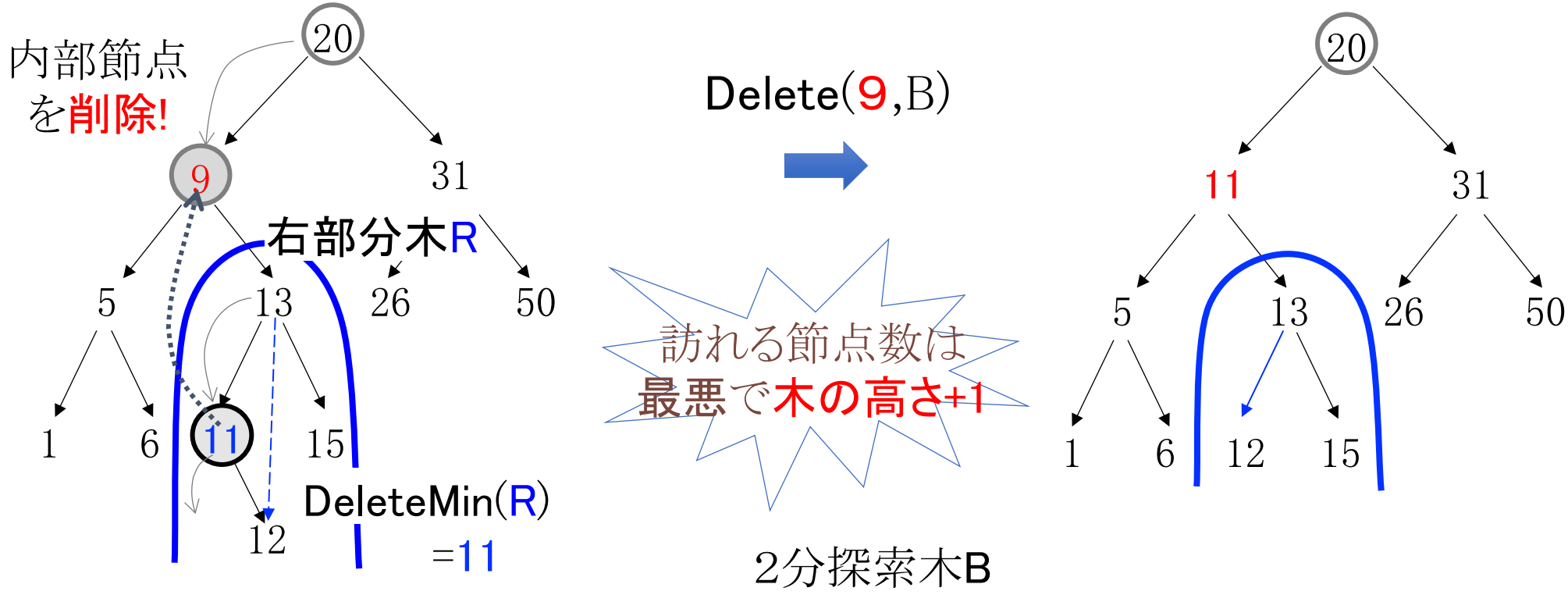
- 削除節点の子を削除節点の場所にする



2分探索木B

Delete場合分け: 削除節点の子が2人

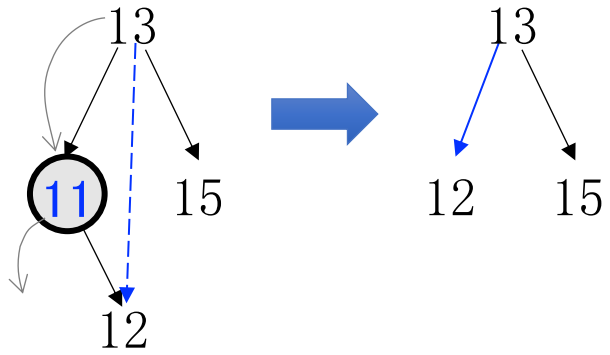
- 削除節点の右部分木の最小値を，削除節点の場所にする



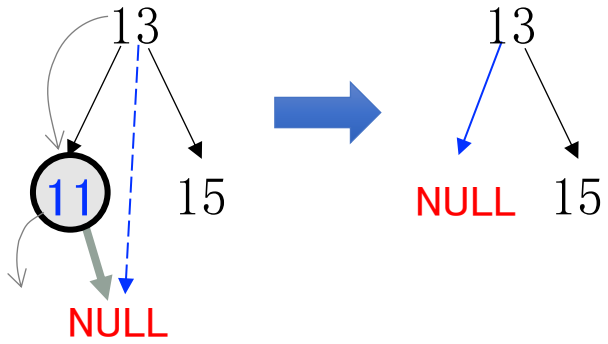
補助関数：DeleteMin(S)

- （空でない）2分探索木Sから，最小値の節点を削除
- 関数値は削除する最小値

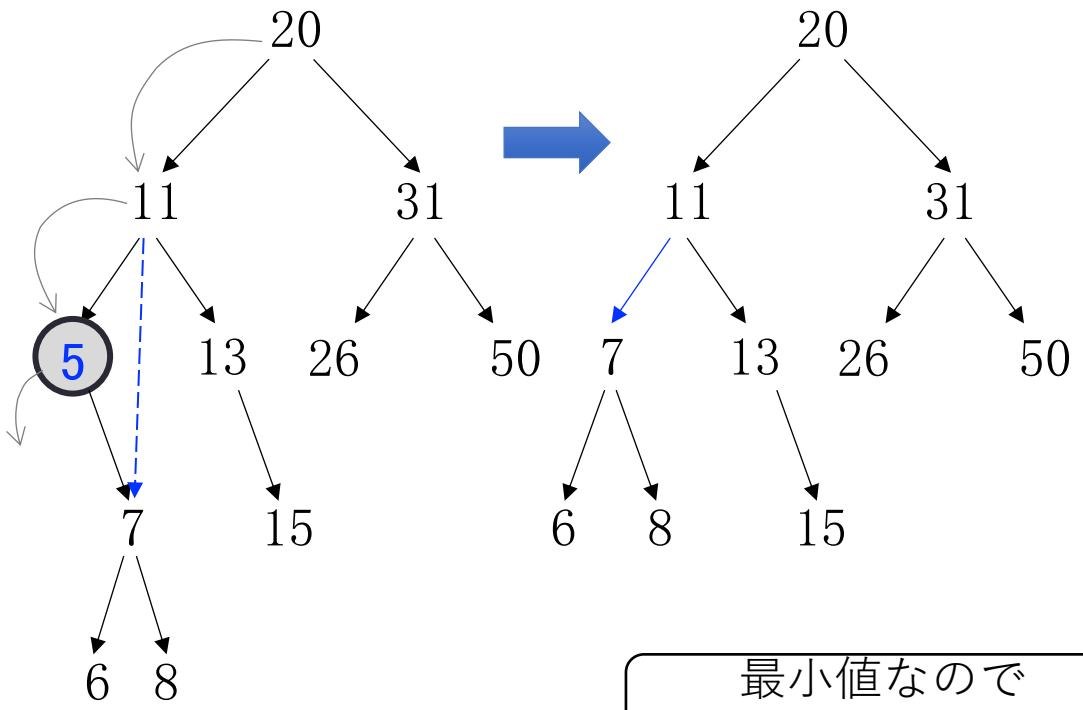
DeleteMin(S)=11



DeleteMin(S)=11



DeleteMin(S)=5



最小値なので
左の子はいない

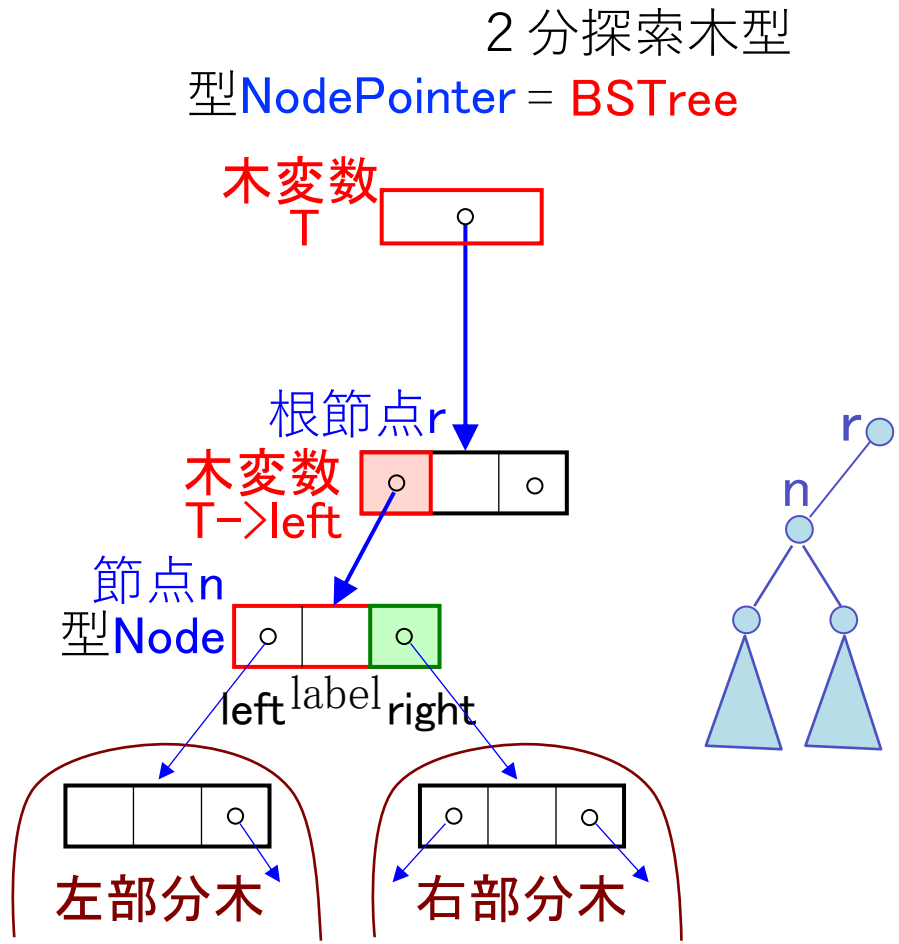
2分探索木の実現：表現

- 木の節点：構造体型Node：

```
typedef struct node_tag *NodePointer;
typedef struct node_tag{
    Element label;
    NodePointer left, right;
} Node;
```

- 木変数：
  - 根の節点を指すポインタを格納する変数

```
typedef NodePointer BSTree;
```



- Tは，根節点rを表す構造体を指すポインタを格納する木変数
- T->leftは，節点nを表す構造体を指すポインタを格納する木変数

2分探索木の実現：引数の指定

- （部分）木を指定する引数：部分木を指すポインタを保持する変数（木変数，節点のleft/rightフィールド）を指す
- 必要性：操作で部分木が変わったとき，2分探索木に変更を反映する必要がある

【例】節点の挿入

定義 Insert(Element x, BSTree *t)

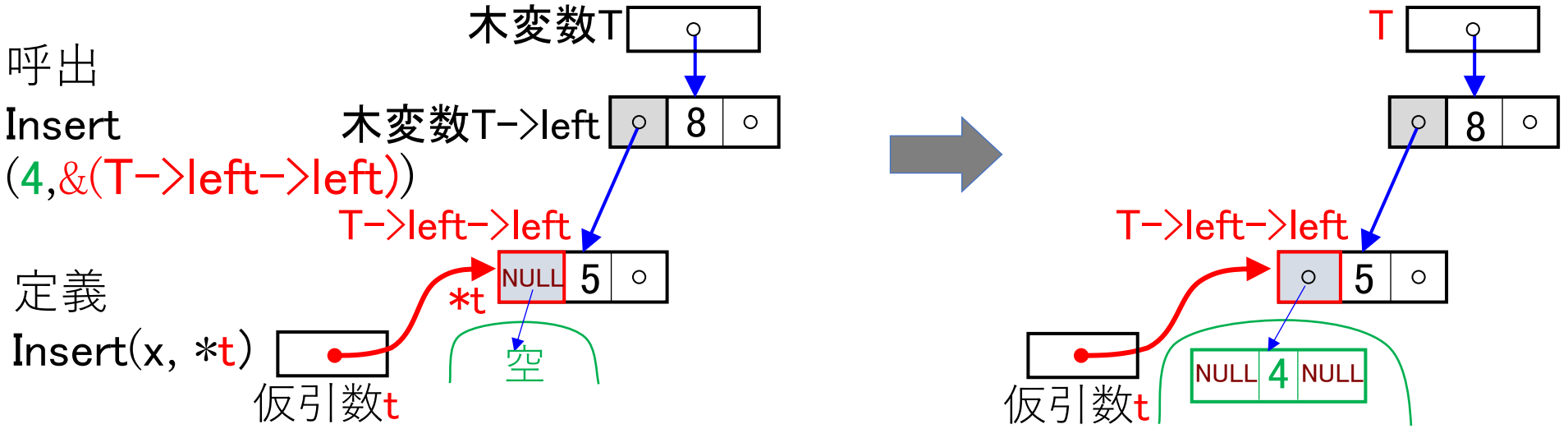
【例】節点の削除

定義 Delete(Element x, BSTree *t)

*tが部分木の根を指す変数

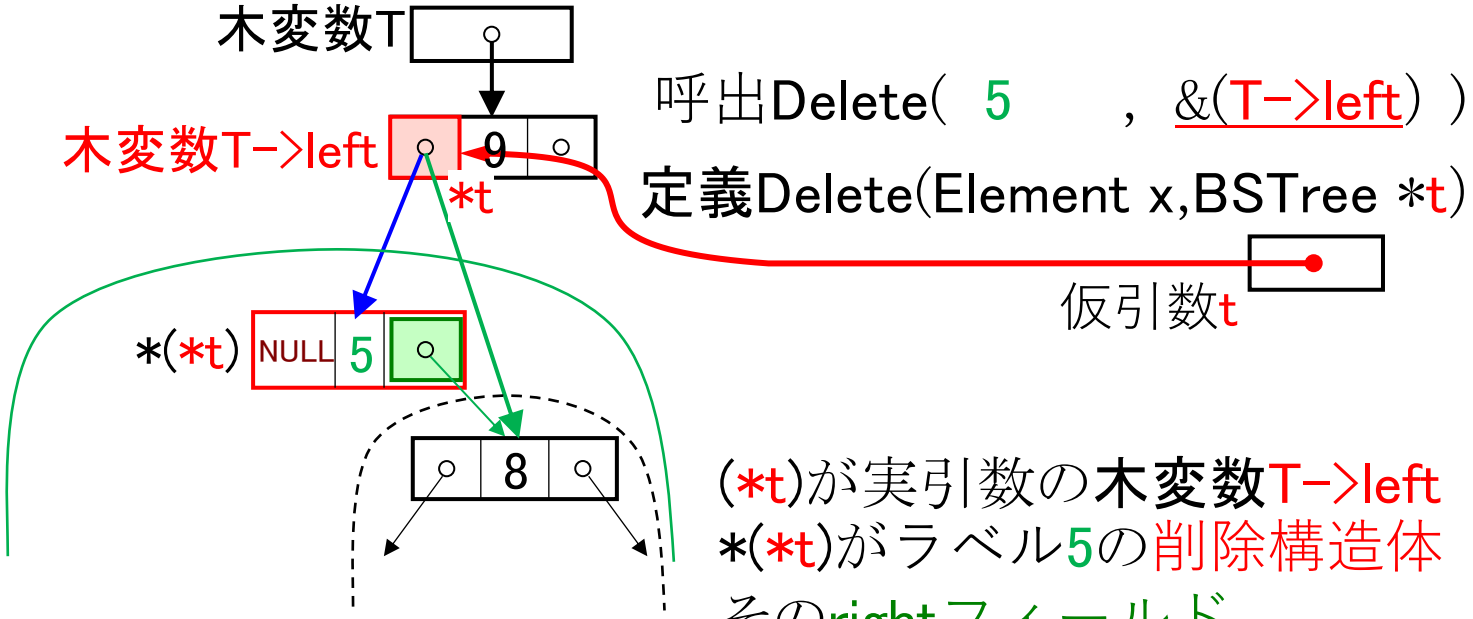
【例】節点の挿入

- 定義 Insert(Element x, BSTree *t)
- 呼出 Insert(4, &(T->left->left))



【例】節点の削除

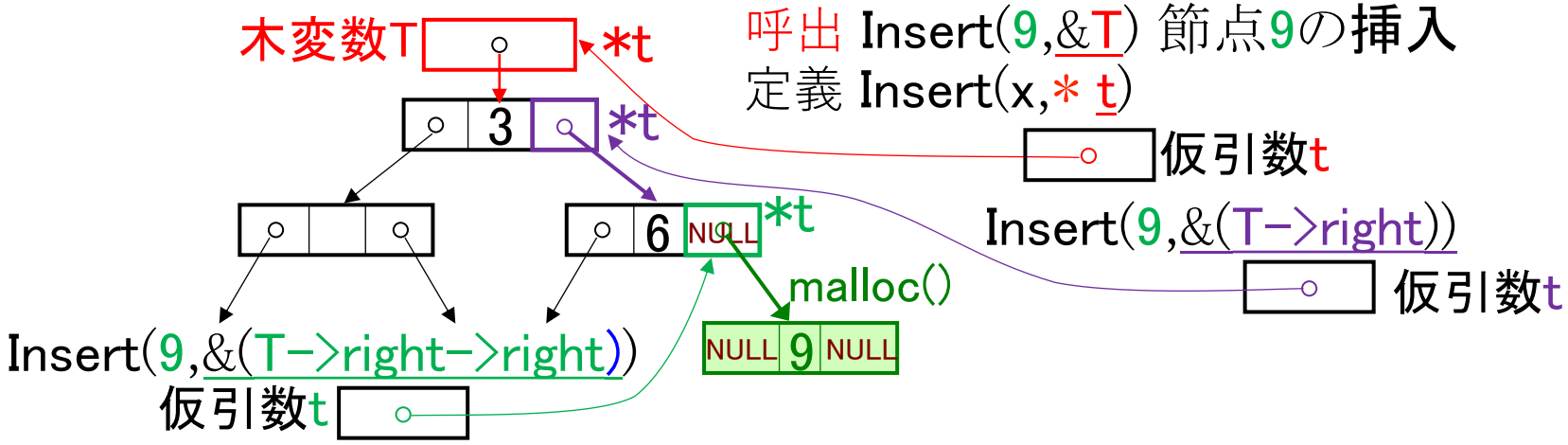
- 定義 Delete(Element x, BSTree *t)
- 呼出 Delete(5, &T->left)



(*t)が実引数の**木変数**T->left
*(*t)がラベル5の削除構造体
そのrightフィールド
(*t)->right (= *(*t).right)が
節点8を根とする部分木の親
⇒**親**を*tに付け替える

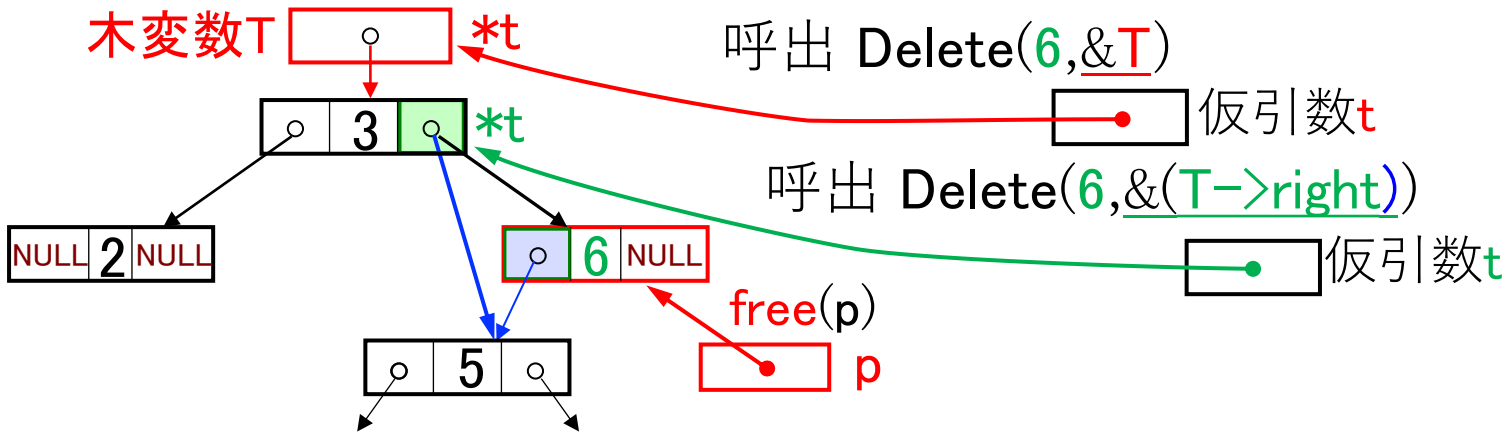
2分探索木 実現：Insert

```
void Insert(Element x, BSTree *t){          /* 辞書型 BSTree, 要素型 Element */
   if((*t) == NULL){                        /* 空の木なら新節点を挿入 */
      *t = malloc(sizeof(Node));
      (*t)->left = (*t)->right = NULL;  (*t)->label = x;
   }
   else if(x < (*t)->label)  Insert(x, &((*t)->left));
   else if(x > (*t)->label)  Insert(x, &((*t)->right));
   else return;                             /* 要素xがあったとき，何もしない */
   return;
}
```



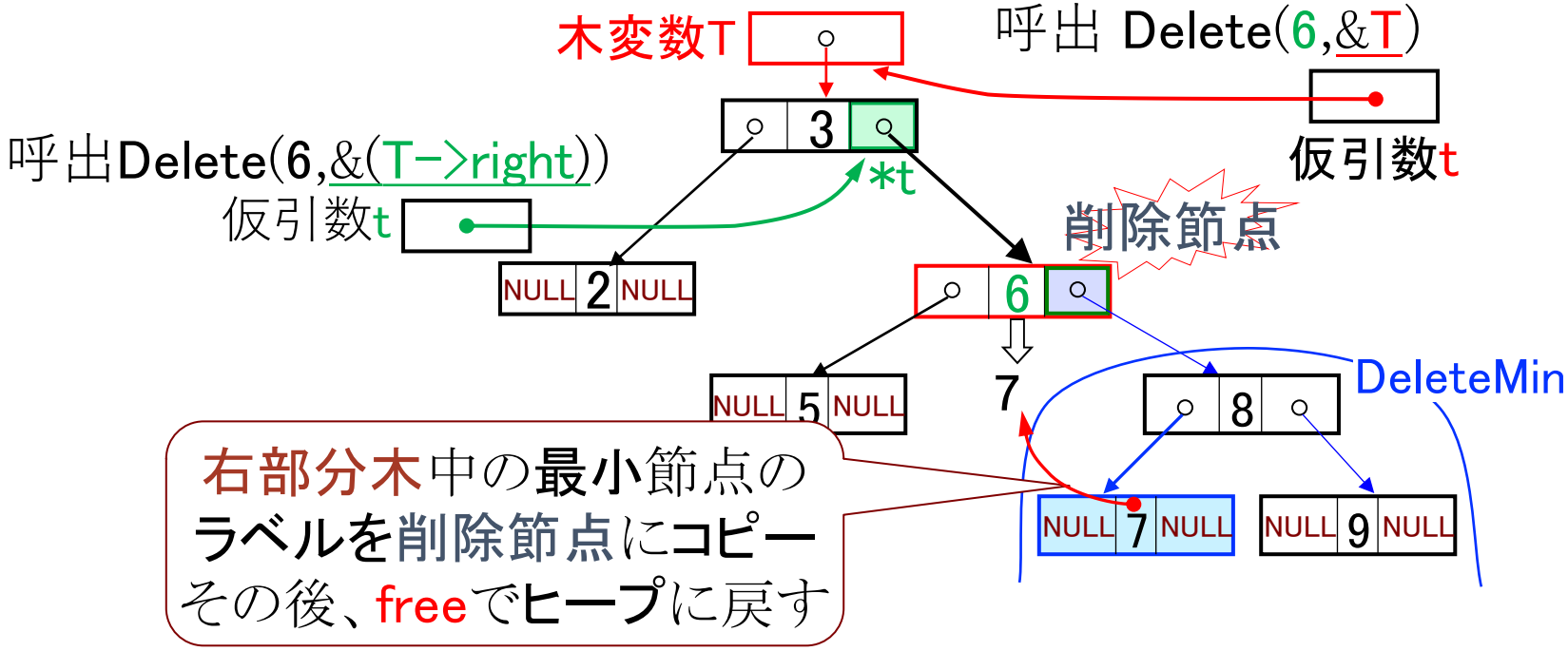
2分探索木 実現：Delete

```
void Delete(Element x, BSTree *t){
    NodePointer p;
    if(*t == NULL) return;                                  /* 要素xがない */
    if(x < (*t)->label)  Delete(x, &((*t)->left));          /* 左部分木を探索 */
    else if(x > (*t)->label)  Delete(x, &((*t)->right));    /* 右部分木を探索 */
    else if((*t)->left == NULL){                            /* 要素xを発見 */
        p = (NodePointer)(*t); (*t) = (*t)->right; free(p); return; }      /* 左部分木が空 */
    else if((*t)->right == NULL){
        p = (NodePointer)(*t); (*t) = (*t)->left; free(p); return;}        /* 右部分木が空 */
    else{(*t)->label = DeleteMin(&((*t)->right));}          /* 左右部分木が存在 */
    return;
}
```



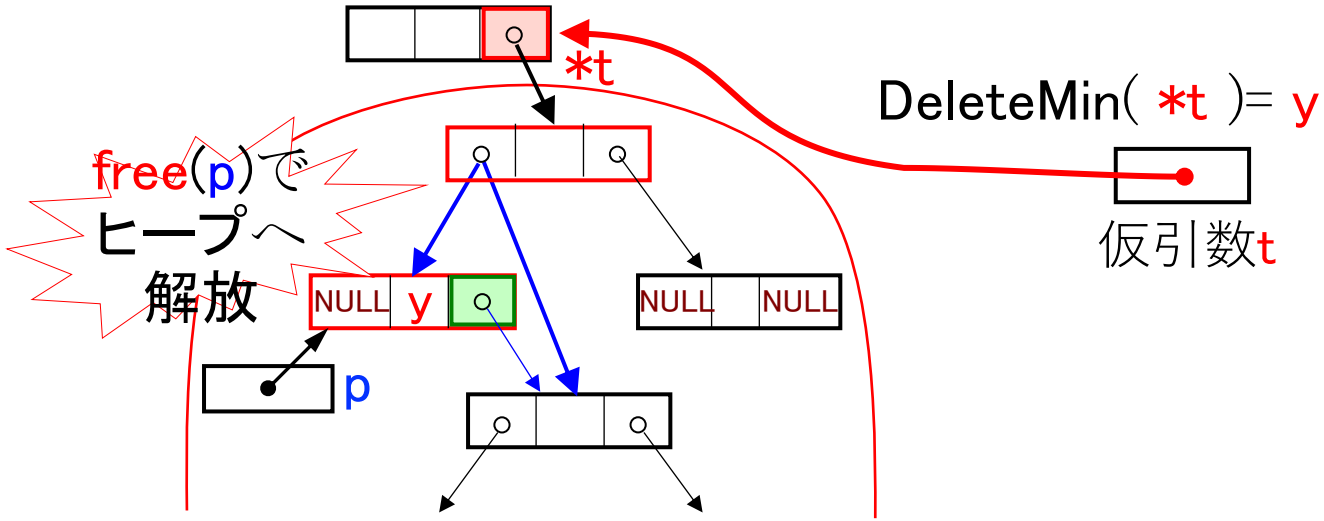
2分探索木 実現：Delete つづき

else{(*t)->label = DeleteMin(&((*t)->right));}



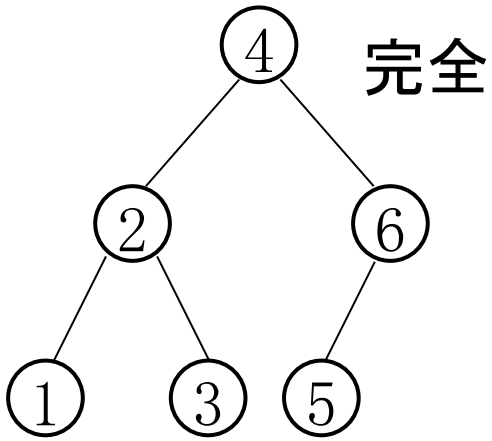
補助関数：DeleteMin

```
Element DeleteMin(BSTree *t){
    Element y; NodePointer p;
    if((*t) == NULL) ERROR("Fatal Error");
    else if((*t)->left == NULL){
        y = (*t)->label; p = (NodePointer)(*t); free(p);     /* 最小節点の削除 */
        (*t) = (*t)->right; return y; }
    else{ y= DeleteMin(&((*t)->left)); return y;}          /* 左部分木をたどる */
}
```

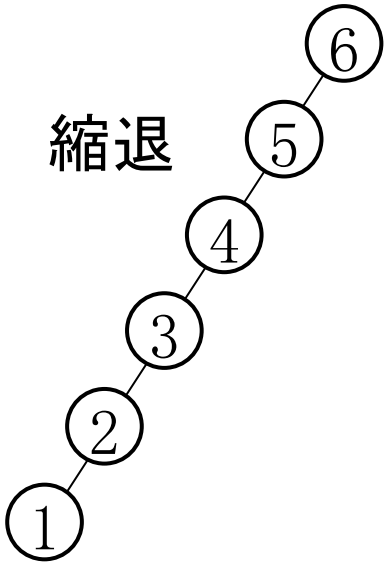


時間計算量

- 操作 Member, Insert, Delete, DeleteMin の時間計算量は，接点数をNとしたとき
  - 2分探索木が完全：O(logN)
    - 左右の部分木の接点数が高々ひとつしか違わない木（完全バランス木）
  - 縮退した2分木：O(N)
    - 最悪の場合，連結リストの形



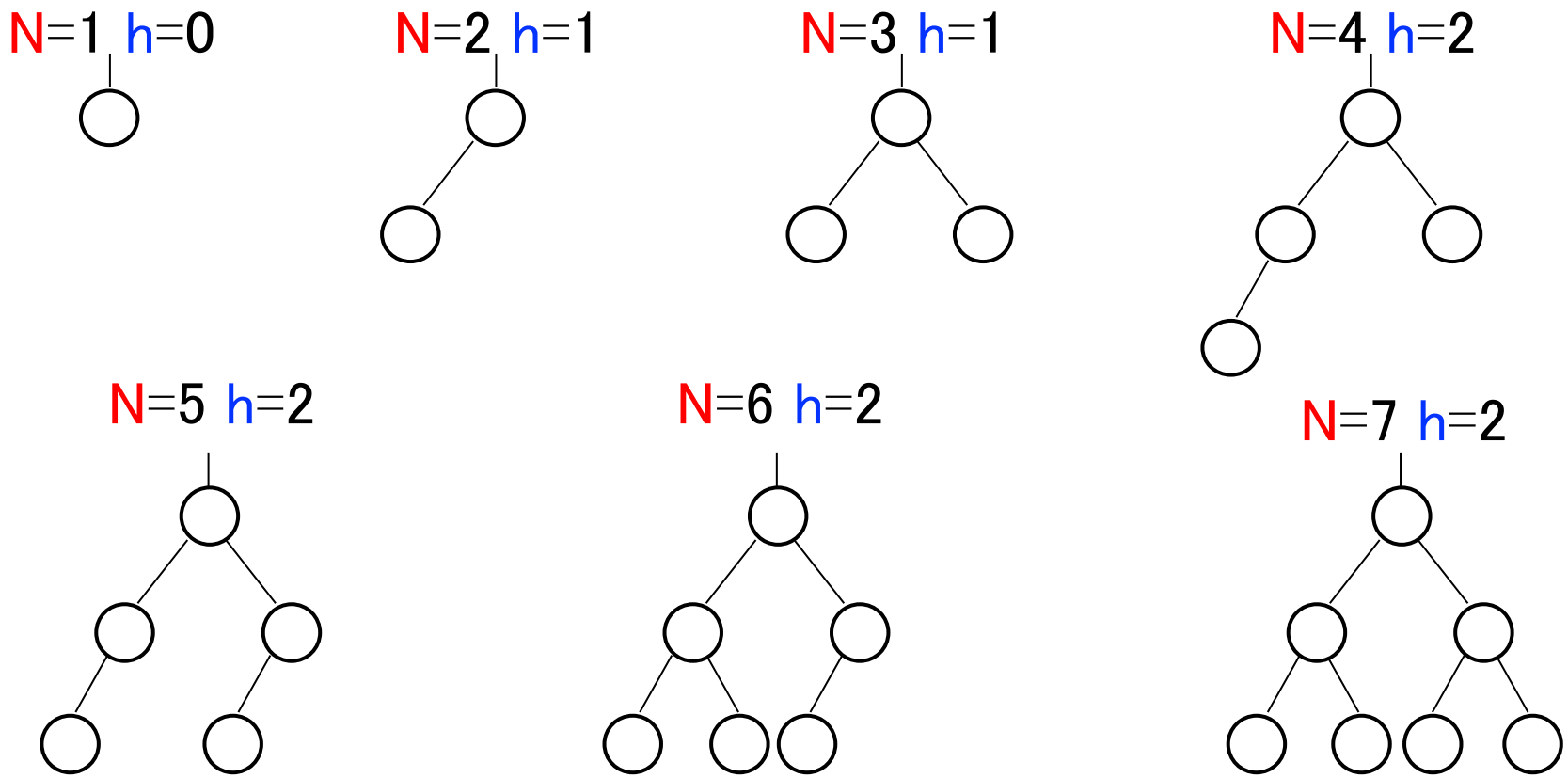
完全



縮退

完全バランス木とその高さ

- 左と右の部分木の節点数が高々ひとつ
  - Nを接点数としたときの完全バランス木の高さ $h=\log_2 N$



2分探索木における探索の平均時間計算量

- 2分探索木をN個の節点の挿入だけでつくる
- データ挿入順序のN!通りが等確率で生じると仮定

- すべての2分探索木の探索の平均の長さ
- 約1.386 $\log_2 N + 2\gamma - 4$（$\gamma \fallingdotseq 0.577$: オイラー定数）

- 完全バランス木の探索における最悪の経路長：$\log_2 N$ なので
- 2分探索木を完全バランス化で，約39%改善できる

まとめ

- 木による辞書の実現
  - 2分探索木

- つづけて，AVL木をやります

演習：2分探索木

- 整数値をラベルに持つ2分探索木について以下を答えよ

1. 2分探索木とは，どのような性質を持つ木か
2. 空の2分探索木に次の整数をラベルとする節点を挿入していったときにできあがる2分探索木を描け
   - 7, 2, 10, 1, 4, 8, 11, 3, 6, 9, 12, 5
3. 2. で描いた2分探索木における根の高さ，深さ，レベルは何か
4. 2. で描いた2分探索木において，通りがけ順にたどったときの節点のラベル列をのべよ
5. 2. で描いた2分探索木において，根を削除したときの結果の2分探索木を描け

提出方法

- ドローイングソフトを使ってもかまいませんが，手書きを写真でとったものでOKです
- pdfや画像フォーマットで提出してください

- 提出方法：LETUS
- 締め切り：2023/7/17 10:30まで