第１３週
木構造と探索（２）
２分探索木における挿入と探索

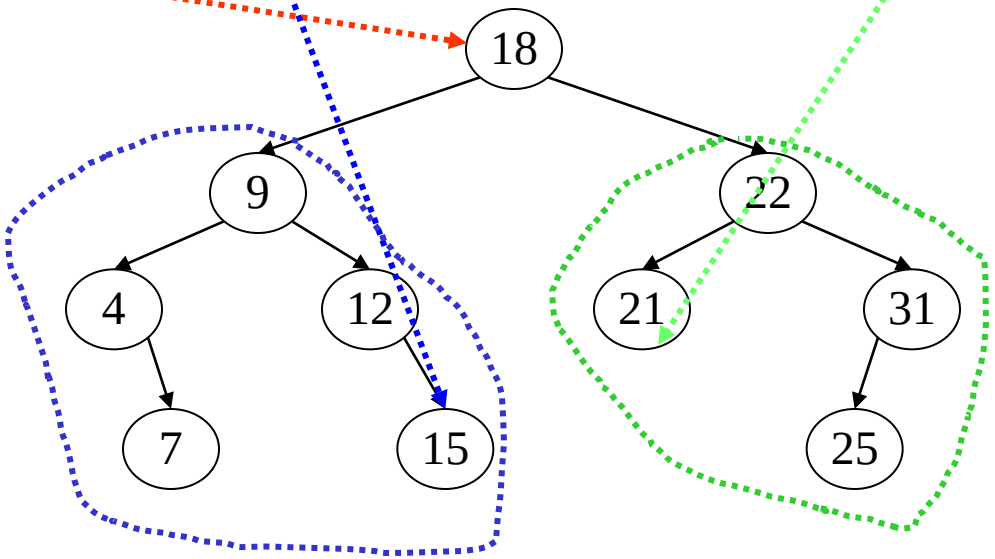
復習：2分探索木

●定義（広い意味での定義）

- 根付き2分木の各点に要素を１つずつ対称順に配置したもの
- 対称順とは
  - 例えば下記のような順序関係を指す

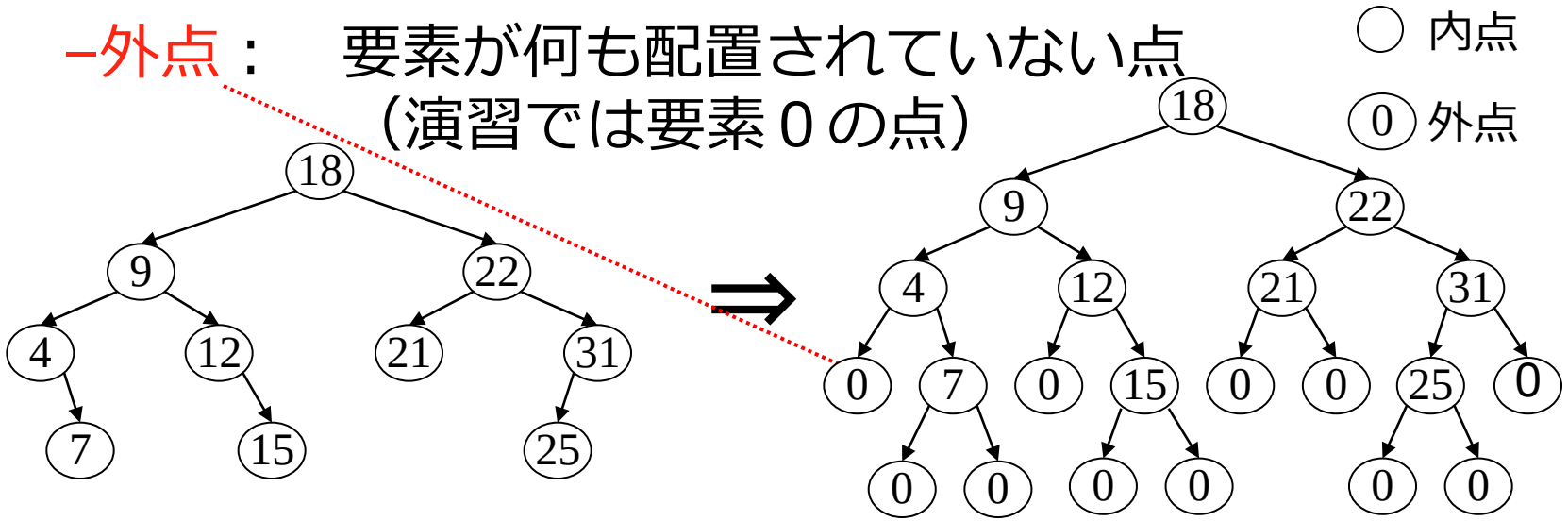
点vの左側の要素の最大値 ＜ 点vの要素 ＜ **点vの右側の要素の最小値**

例：18は、**左側最大（15）**より大きく、**右側最小（21）**より小さい



復習：正則2分木

- 定義
  - –各点が左子、右子の両方の子を持つか、または全く子を持たない根付き2分木
    - –（左子、右子の一方のみが存在する場合は正則2分木ではない）
- すべての2分探索木は正則2分木として考えられる？
  - –方法：　左子、右子の一方しか存在しない場合は外点を追加して正則2分木に変形
  - –外点：　要素が何も配置されていない点（演習では要素０の点）



必須課題13-１： new_node

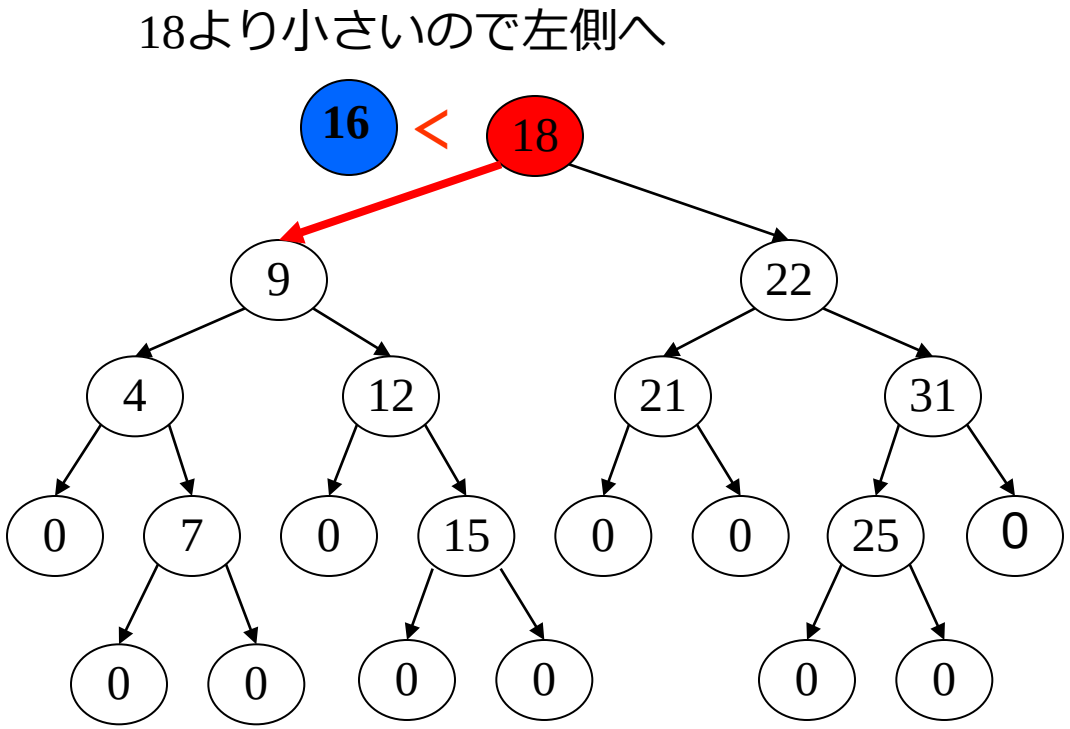
```
int insert(int key, struct node *root)
```

挿入する要素の値　　2 分探索木の根を示すポインタ

- 機能:
  2 分探索木を探索し，適切な位置に要素key を挿入する.
- 引数key:
  挿入する要素の値（キー）
- 引数root:
  2 分探索木の根を示すポインタ
- 戻り値:
  要素key を挿入できた場合は1,
  挿入できなかった場合（すでに同じ値の要素がある場合）は0 を返す

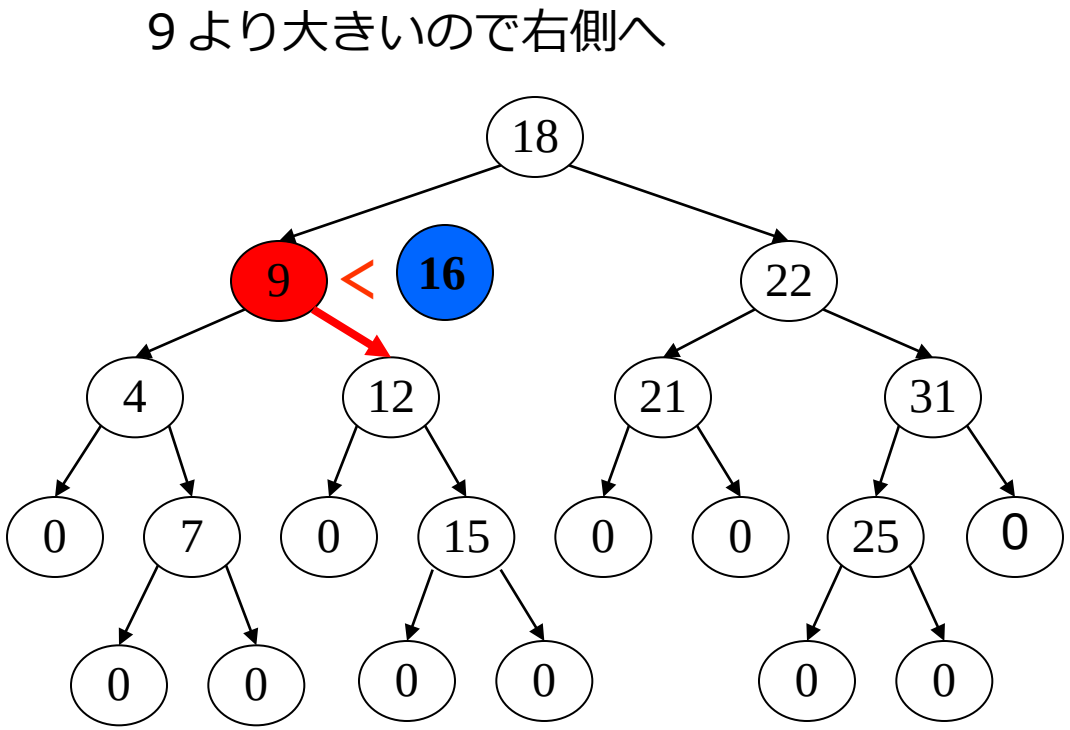
insert手順の例： 1 /5

例：16を挿入する場合



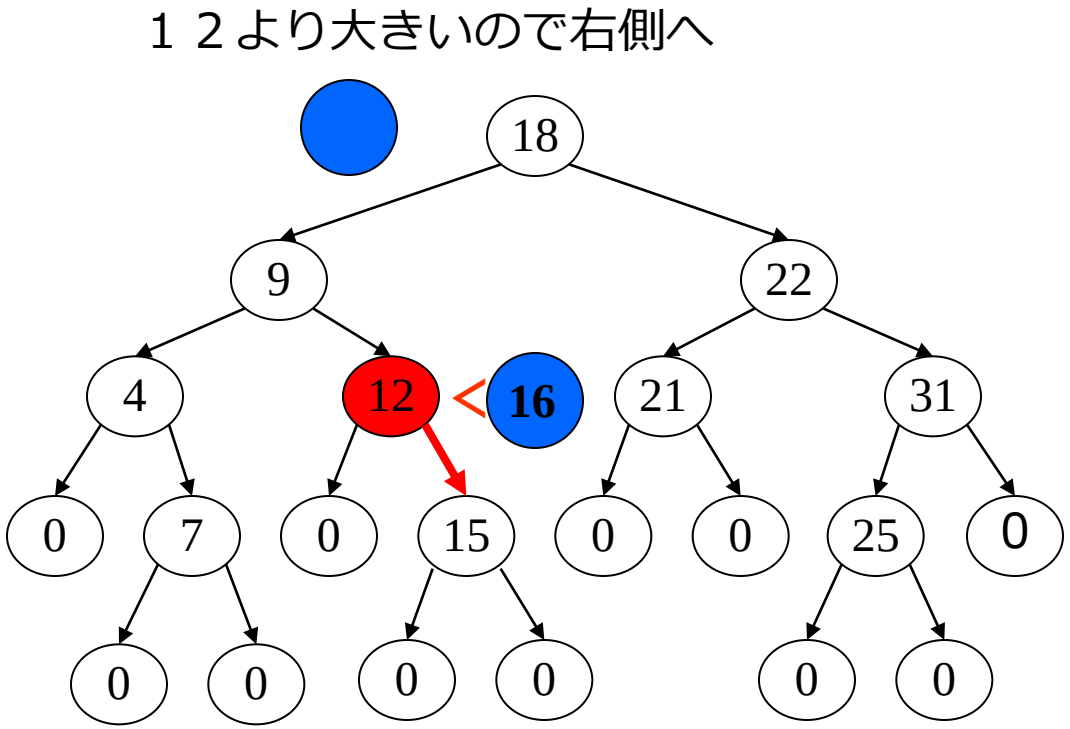
insert手順の例：2/5

例：16を挿入する場合



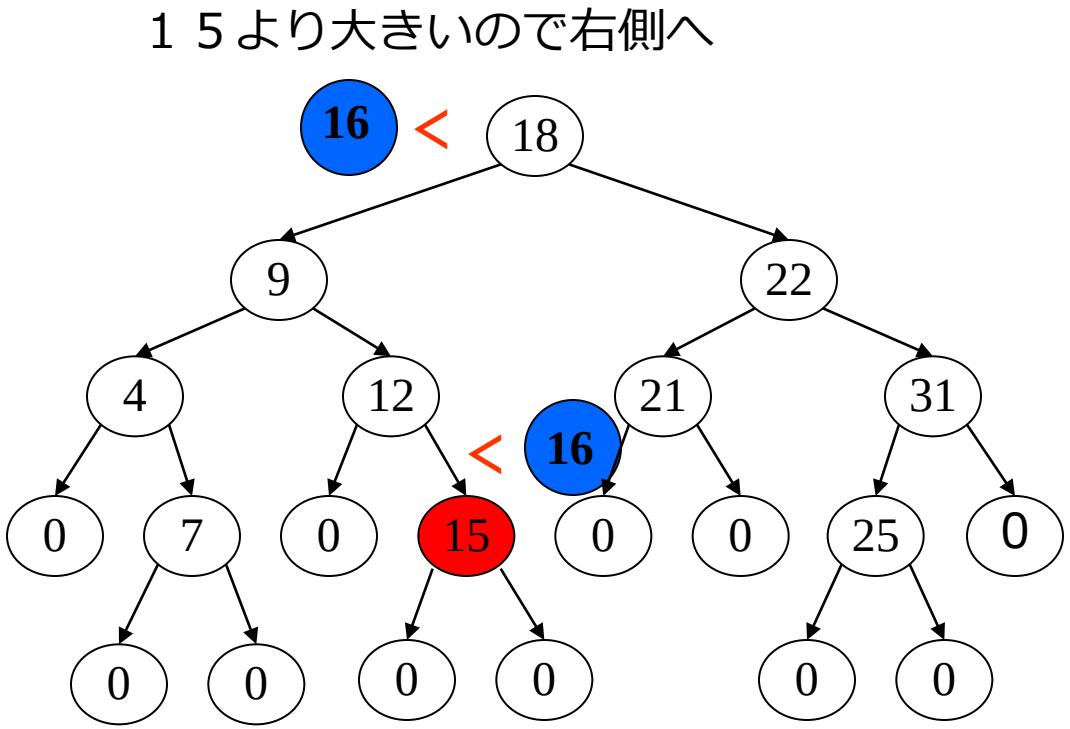
insert手順の例：3/5

例：16を挿入する場合



insert手順の例：4/5

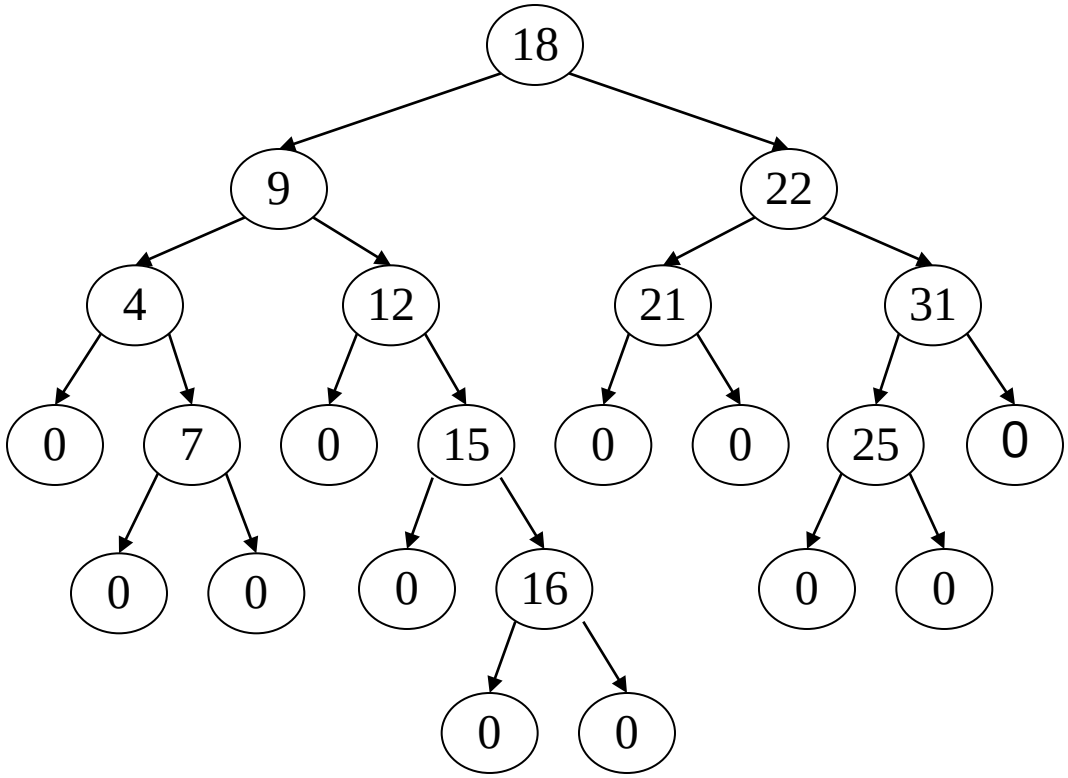
例：16を挿入する場合



insert手順の例： 1 /5

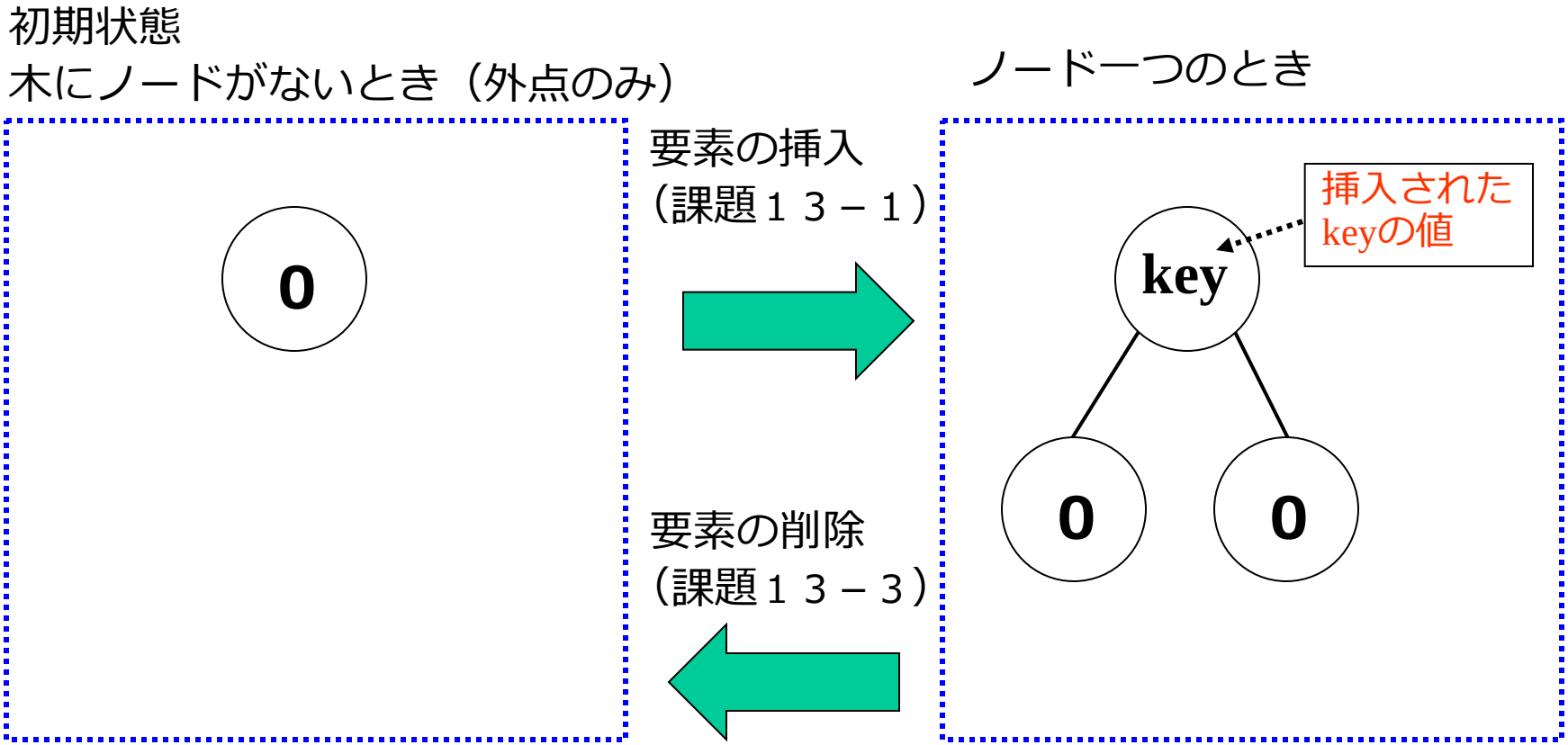
例：16を挿入する場合

外点に到達したので挿入し
新たに外点を追加する



必須課題13－1：2分探索木の初期状態

●2分探索木の初期状態（木が空）のときは、
外点が一つある



必須課題１３－１：参考プログラム

```
int main(void){
  int key;
  struct node *root = new_node(0);
  for(;;){
    printf("挿入するキーを入力して下さい：");
    scanf("%d",&key);

    if(key > 0){
      if(   insert(key,root) == 1){
        print_tree(root);
      }else{
        printf("その値はすでに存在します。¥n");
        print_tree(root);
      }
    }else{
      printf("正でない数値が入力されたので終了します。¥n");
      free_tree(root);
      return 0;
    }
  }
}
```

初期状態、外点のみ

12週目に提供された関数

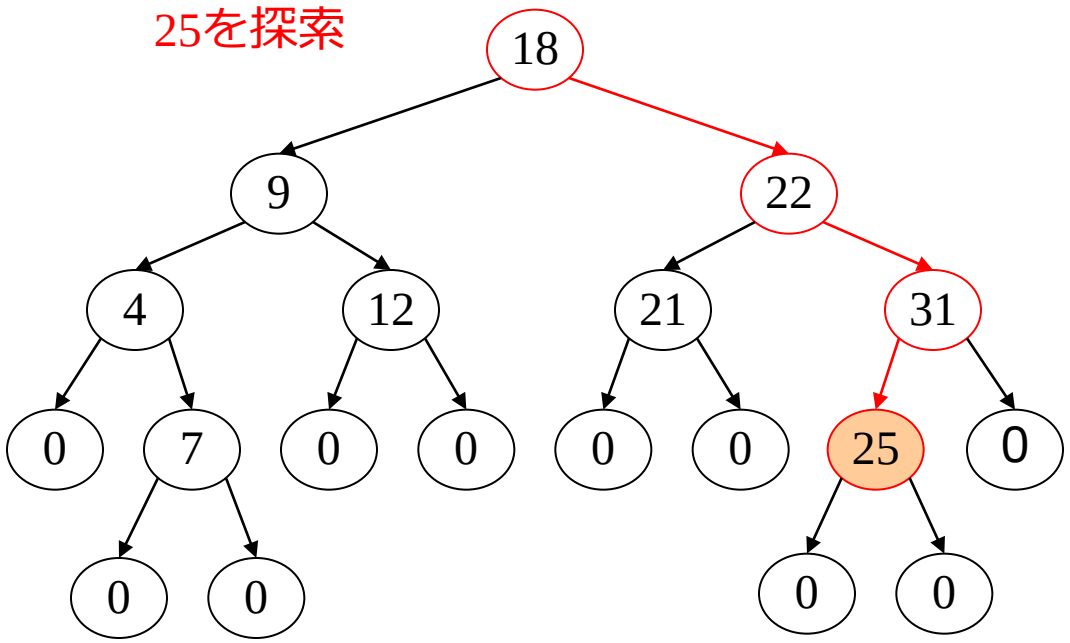
全てのノードのメモリを解放する
プログラムは次ページを参照

free_tree: すべてのnodeのメモリを解放する関数

```
void free_tree(struct node *node) {
        if(node != NULL){
                free_tree(node->left);
                free_tree(node->right);
                free(node);
        }
}
```

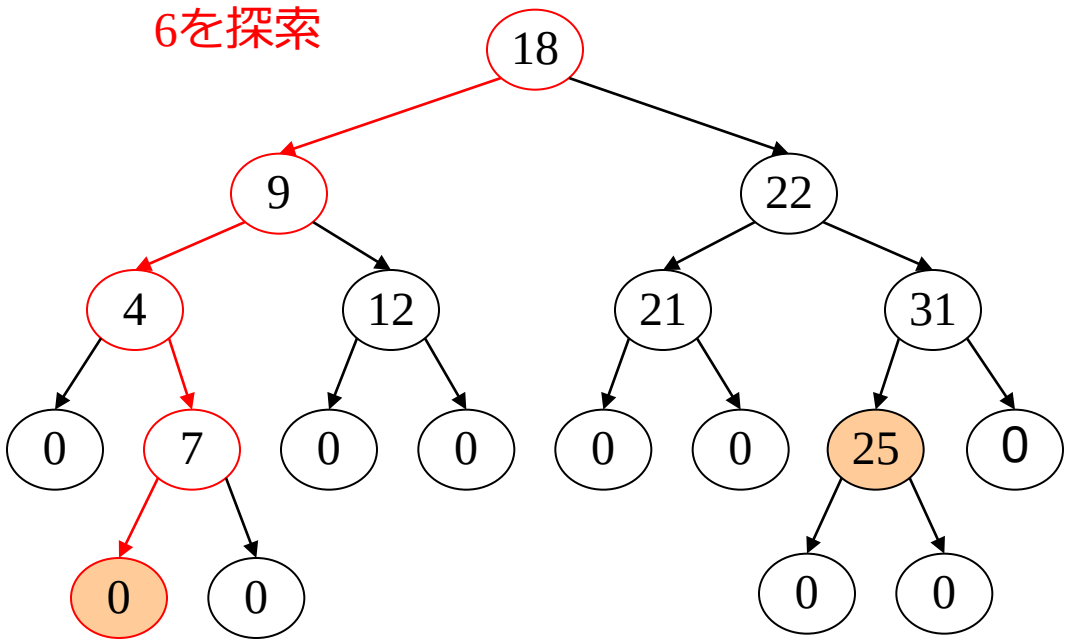
正則二分探索木の探索: member (1)

- 根からはじめ，ノードの値と探索対象の値を比較しながら木をたどる
  - 着目ノードの値 < 探索値ならば右子へ
  - 着目ノードの値 > 探索値ならば左子へ
  - 着目ノードの値 = 探索値ならば探索対象発見，終了
  - 着目ノードが外点ならば探索対象なし，終了



正則二分探索木の探索: member (2)

- 根からはじめ，ノードの値と探索対象の値を比較しながら木をたどる
  - 着目ノードの値 < 探索値ならば右子へ
  - 着目ノードの値 > 探索値ならば左子へ
  - 着目ノードの値 = 探索値ならば探索対象発見，終了
  - 着目ノードが外点ならば探索対象なし，終了



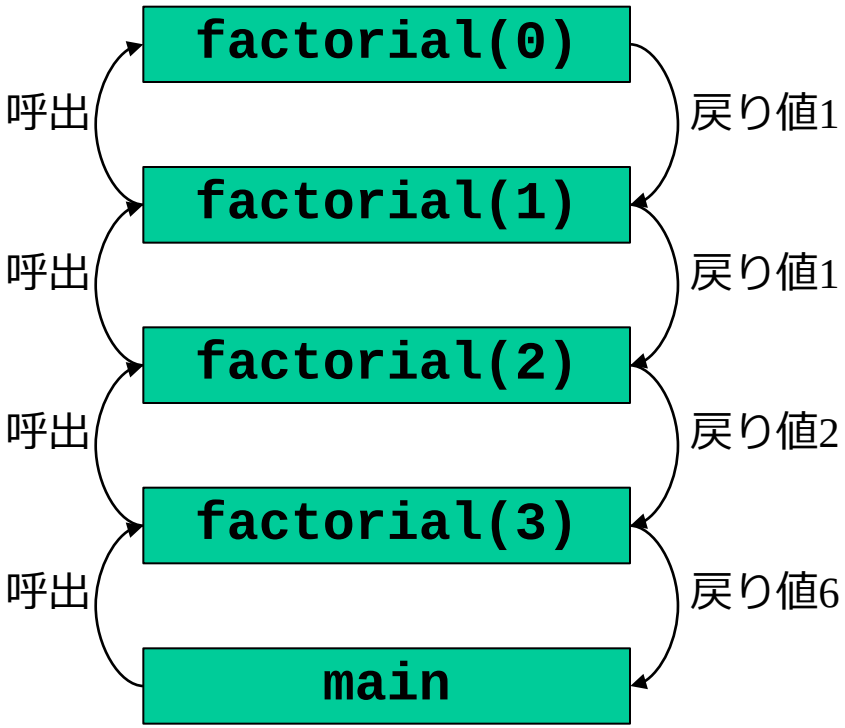
再帰

●関数内で自分自身を呼ぶこと

再帰呼び出しの例

```
int factorial(int x){
  if(x==0){
    return 1;
  }
  return x*factorial(x-1);
}

int main(){
  factorial(3);
  return 0;
}
```



factorial: 階乗

無限ループに注意！

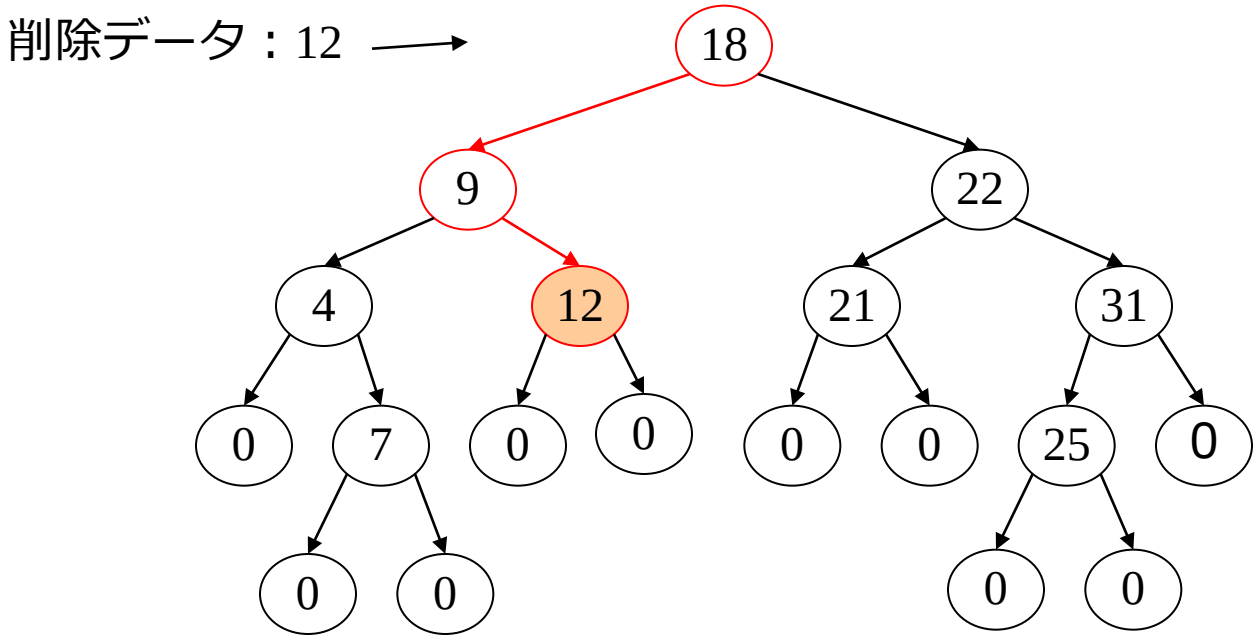
再帰 オプション課題13-2

- member_recursive関数の中で member_recursive関数を呼ぶ
  - 探索する値keyと現在のノードのキーを比較
  - 比較結果によって、再帰呼び出しの際、 member_recursiveに渡す引数を変える
  - 再帰呼び出しの終了は、 外点にぶつかるか、keyを発見したとき

recursive: 再帰的な

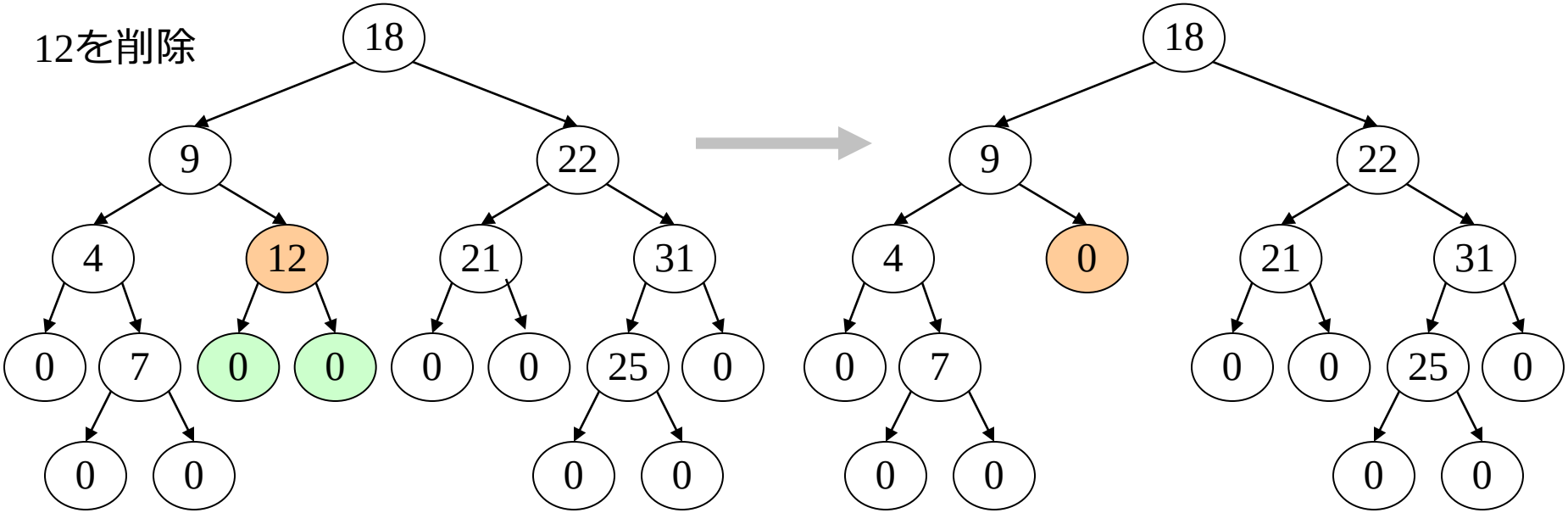
正則二分探索木からの削除(1)

- 根からはじめ，ノードの値と挿入する値を比較しながら木をたどり，削除対象となる内点を探索
  - 着目ノードの値 < 挿入する値ならば右子へ
  - 着目ノードの値 > 挿入する値ならば左子へ
  - 着目ノードの値 = 挿入する値ならば挿入せず終了



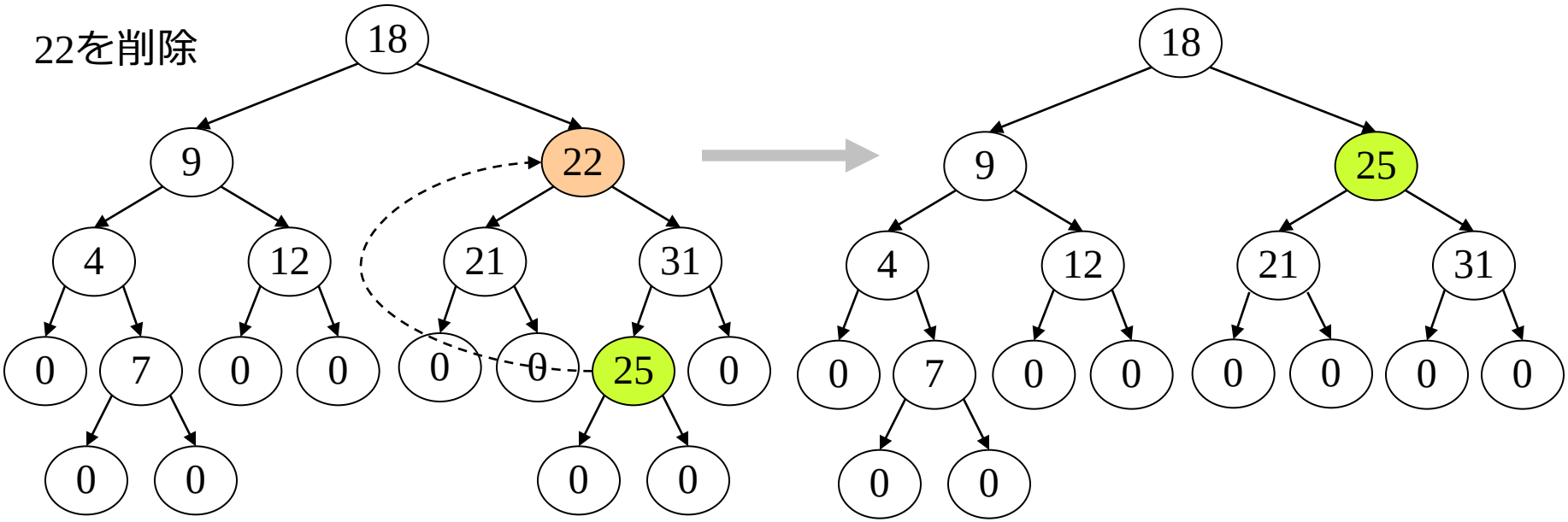
正則二分探索木からの削除(1)

- 削除対象ノードの右子・左子ともに外点のとき
  - 削除対象ノードを外点にする
    - データを外点値にする
    - 左子，右子をなくす



正則二分探索木からの削除(2)

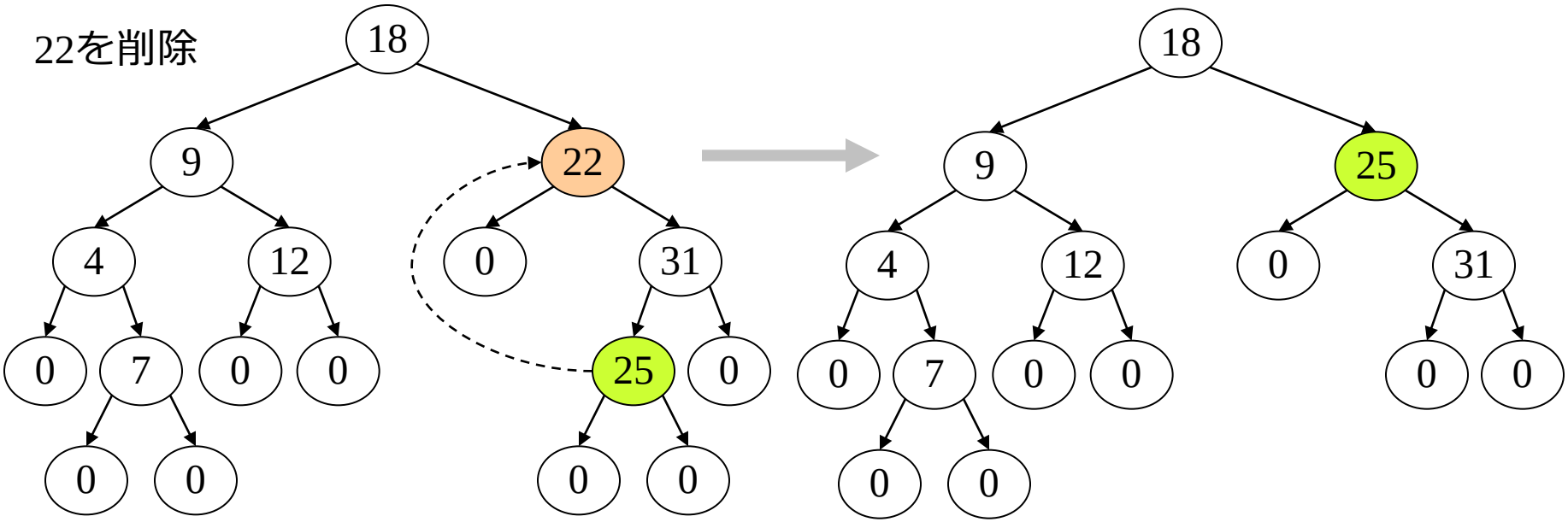
- 削除対象ノードの右子(左子)が内点のとき
  - 右子(左子)を根とする部分木の最小値(最大値)ノードを外点にする
  - 削除対象ノードの値を最小値(最大値)に変える



右部分木の最小値: 25

正則二分探索木からの削除(3)

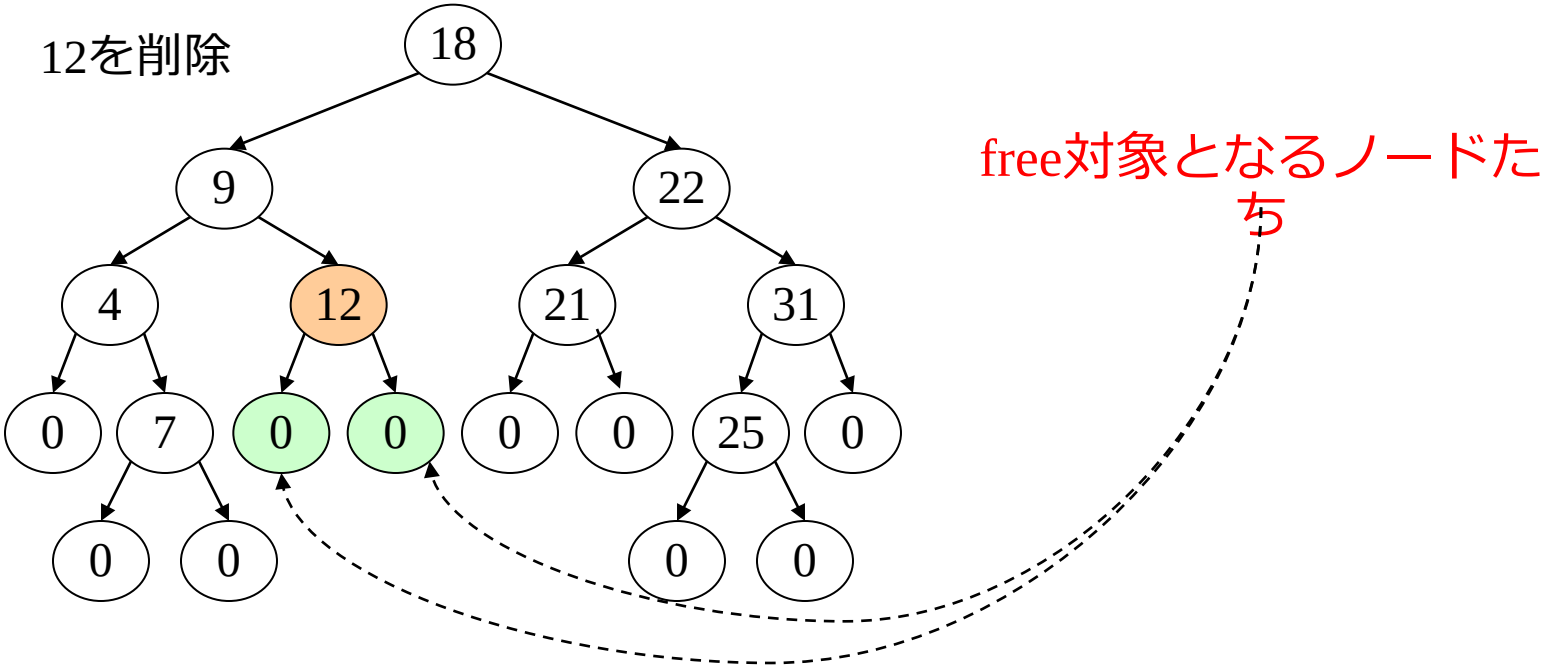
- 削除対象ノードの右子・左子ともに内点のとき
  - 右子のみが内点のときと同様にできる
  - 右子を根とする部分木の最小値ノードを外点にする
  - 削除対象ノードの値を最小値に変える



右部分木の最小値: 25

注意

- ノード作成にはmalloc関数を使うことになりますが…
  - ノードを木からなくすときには free関数によるメモリ解放 を忘れずに!!!
  - 例: 課題12-4 木からのノード削除



著者リスト

1. 安積 卓也（情報システム学科）
2. 大森 隆行（情報システム学科）
3. 原田 史子（情報システム学科）