情報構造 第十三回

辞書の実現

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

基本的な探索木
形がよくないと効率が悪い

形をよくする方法を導入

多分木に一般化

- 今日はAVL木を行います

2分探索木のバランス化（平衡化）

- 2分探索木の挿入操作の時間計算量
  - 平均でO(log N)
  - 最悪ではO(N) => 左右の部分木をバランス化すると O(log N)

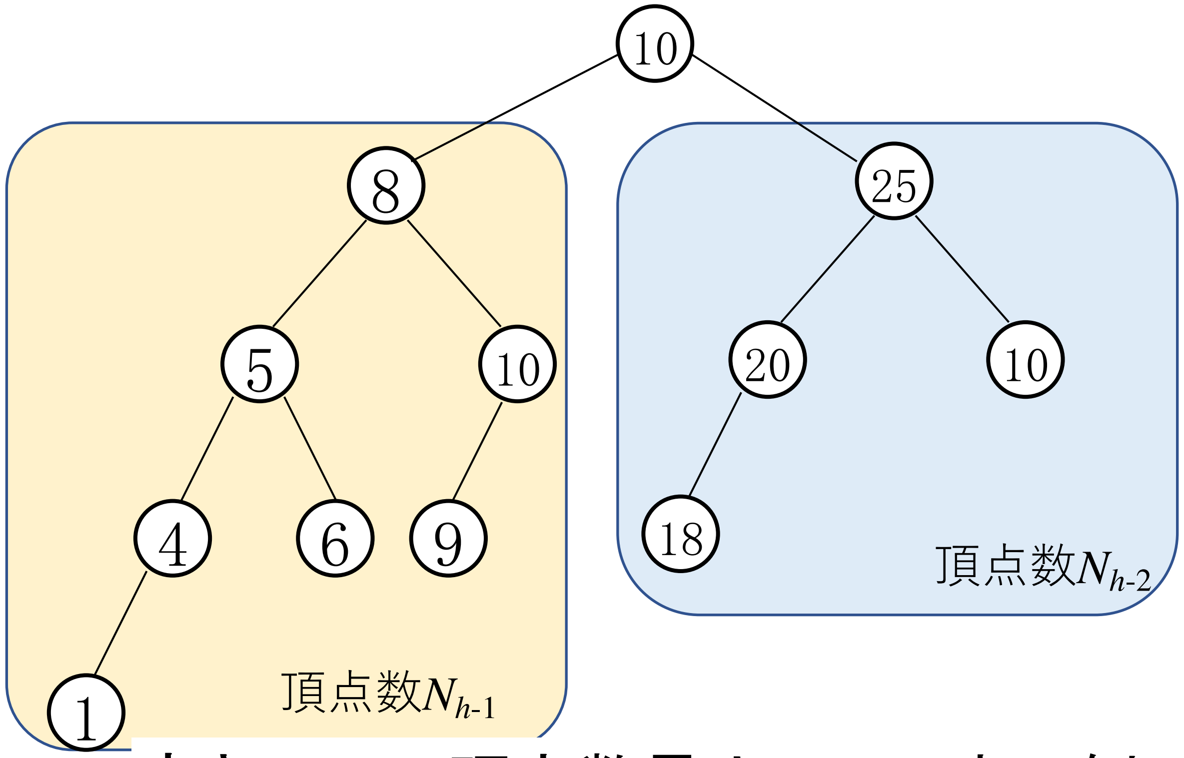
- 完全バランス木
  - 各節点で，左と右の部分木の節点数が高々ひとつしか違わない
  - 挿入・削除を行うたびに，木を完全にバランスさせる必要がある
    => バランス復元によって効率が落ちる
- AVL木
  - 各節点で「左右の部分木の高さが高々ひとつしか違わない」というバランスをもつ（回転とよばれる操作によってバランス復元を行う）
    => バランス復元が容易

完全バランスより
ゆるいバランス

AVL木

AVL木（Adelson-Velskii & Evgenii-Landie）

- どの頂点においても左部分木と右部分木の高さの差（バランス度）が +1, 0, -1 のいずれかである2分探索木



高さ h=4 の頂点数最小のAVL木の例

高さhのAVL木の最小頂点数

$$N_h = N_{h-1} + N_{h-2} + 1$$
$$N_0 = 1,\ N_1 = 2$$
$f_h = N_h + 1$とおくと
$$f_h = f_{h-1} + f_{h-2}$$
$$f_0 = 2,\ f_1 = 3$$

フィボナッチ数列

$$f_h = \frac{1}{\sqrt{5}}\left(\left(\frac{1+\sqrt{5}}{2}\right)^{h+3} - \left(\frac{1-\sqrt{5}}{2}\right)^{h+3}\right)$$

$$N_h = \frac{1}{\sqrt{5}}\left(\left(\frac{1+\sqrt{5}}{2}\right)^{h+3} - \left(\frac{1-\sqrt{5}}{2}\right)^{h+3}\right)\text{-}1$$

AVL木の最悪計算量

$$N_h = \frac{1}{\sqrt{5}}\left(\left(\frac{1+\sqrt{5}}{2}\right)^{h+3} - \left(\frac{1-\sqrt{5}}{2}\right)^{h+3}\right)\text{-}1$$

hが大きくなると

$$N_h \approx \frac{1}{\sqrt{5}}\left(\left(\frac{1+\sqrt{5}}{2}\right)^{h+3}\right)$$

Nに対してhは対数オーダーで増加

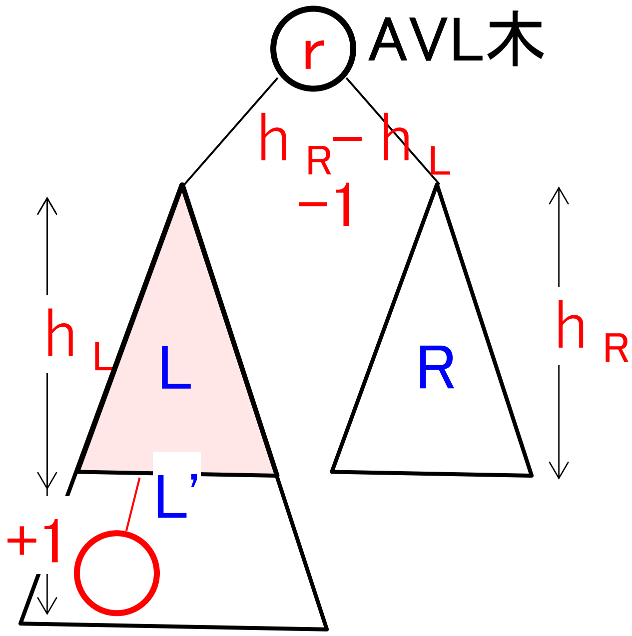
- 2分探索木に対する節点の挿入・削除・探索は，最悪で木の高さhに比例

⇒ AVL木に対する各操作における，最悪時間計算量はO(logN)

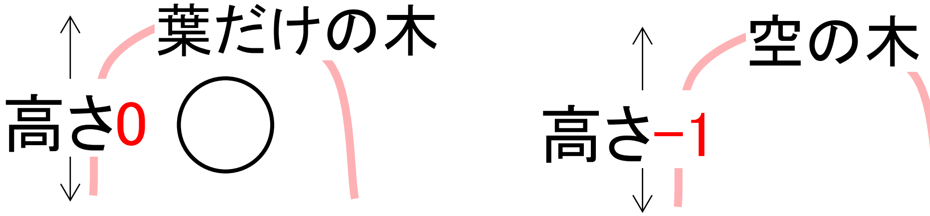
ちなみに，$\log_2 \frac{1+\sqrt{5}}{2} = \frac{1}{1.44}$ なので，
完全2分木と比較しても最悪1.44倍しか遅くならない！

AVL木への節点の挿入

- 挿入前の左部分木Lと右部分木Rの高さをh_Lとh_Rとしたとき，h_R-h_Lを根rのバランス度と呼ぶ
  - AVL木のバランスの基準は，バランス度=+1, 0, -1 のいずれか
- 節点が，根rの左部分木に挿入されて，
  - 左部分木は高さが1だけ増えた場合を考える．
  - 高さが不変なら何もしない

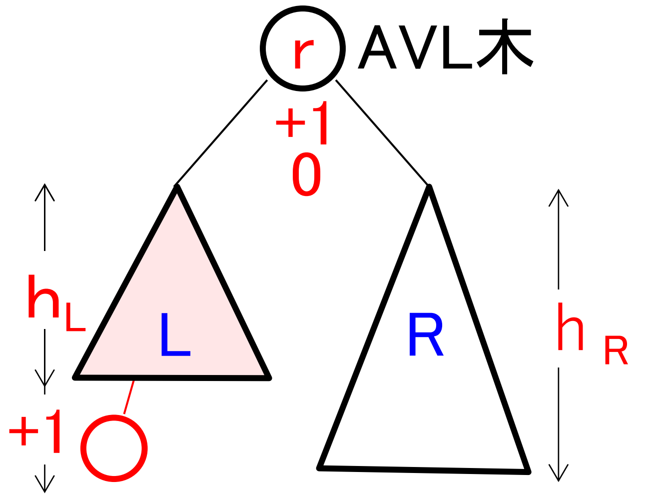


- 葉だけからなる木の高さは0
- 空の木の高さは -1



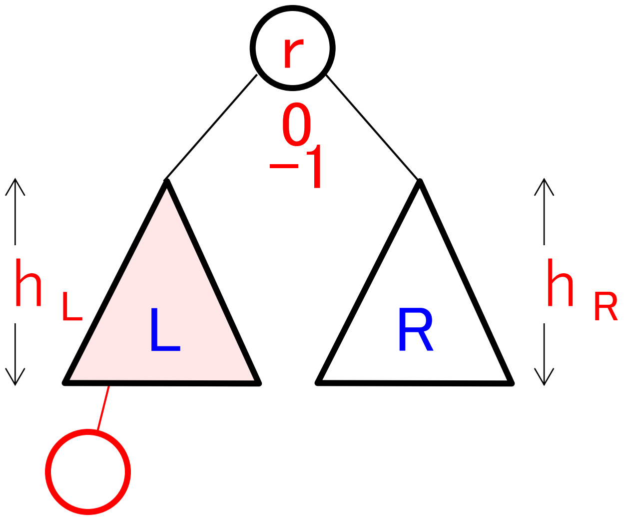
AVL木への節点の挿入：バランス度0に

- 場合B1: $h_L < h_R$
  - LとRは同じ高さになる
  - 節点rのバランス度は0になる
  - 挿入操作終了



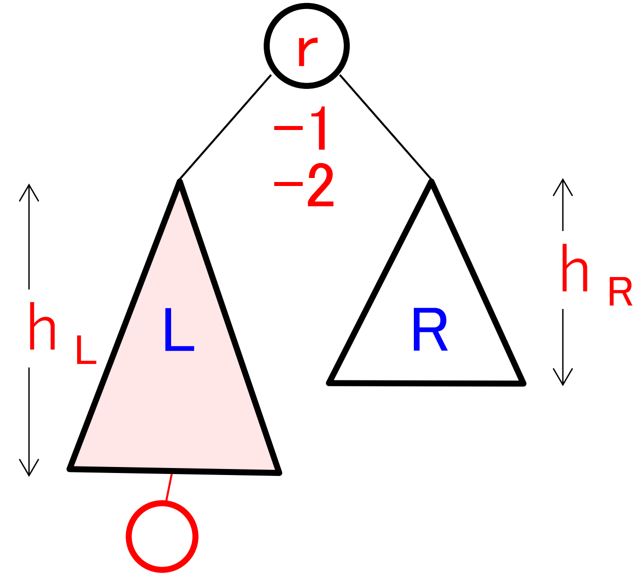
AVL木への節点の挿入：バランス度-1に

- 場合B2: $h_L = h_R$
  - Lの高さがRより1だけ高くなる
    - バランス基準には違反しない
  - 節点rのバランス度は-1になる
  - 根rの部分木の高さが増すので，節点rの親節点のバランス修正を実行する



AVL木への節点の挿入：バランス度-2に

- 場合B3: $h_L > h_R$
  - Lの高さがRより2つ高くなる
    - バランス基準に違反 => バランスの復元操作（木の作り替え）を行う
  - 根rの部分木の高さが挿入前と同じになり，挿入操作終了



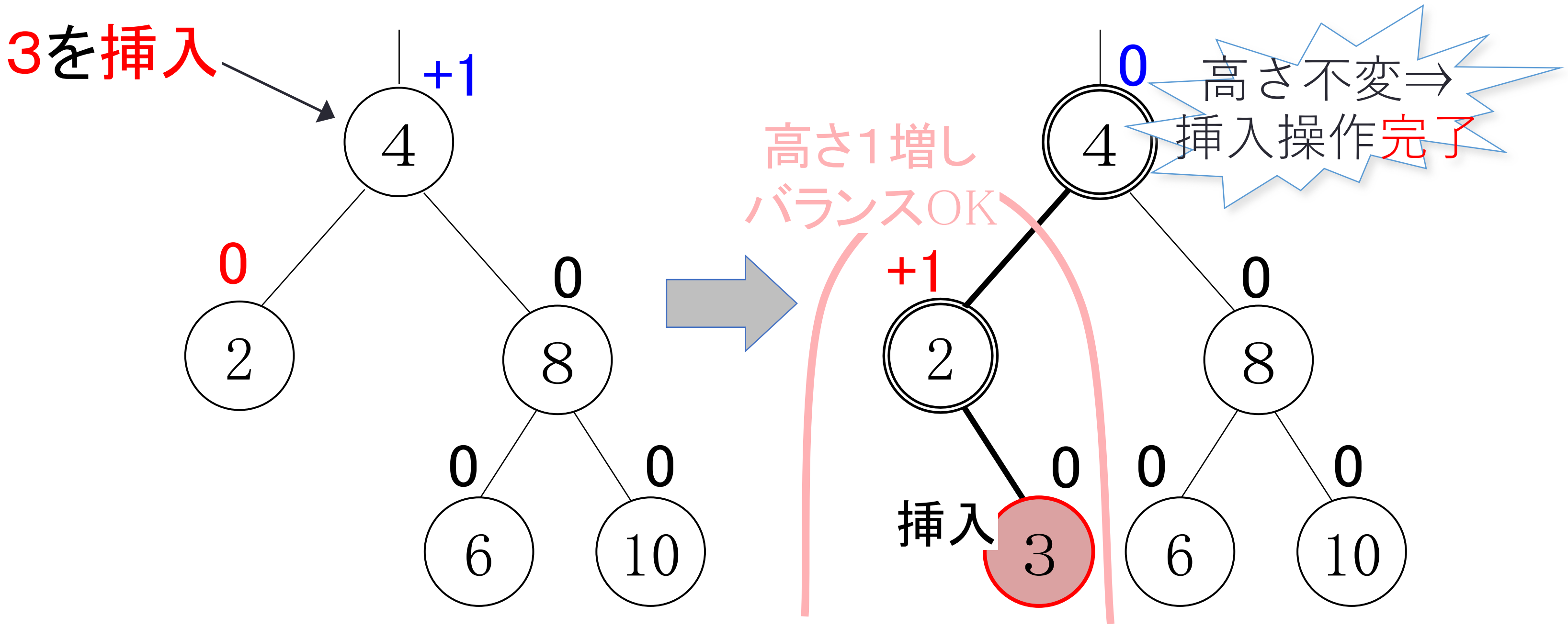
節点の左部分木への挿入：3つの場合わけ

- AVL木に新しい節点を挿入するとき
  - 挿入前のLとRの高さをh_Lとh_Rとする
  - 根rのバランス度: h_R-h_L

- 場合B1: バランス度+1のとき
  - バランス度0に
- 場合B2: バランス度0のとき
  - バランス度-1に（バランス基準内）
- 場合B3: バランス度-1のとき
  - バランス度-2に => バランス復元操作（木の作り替え）

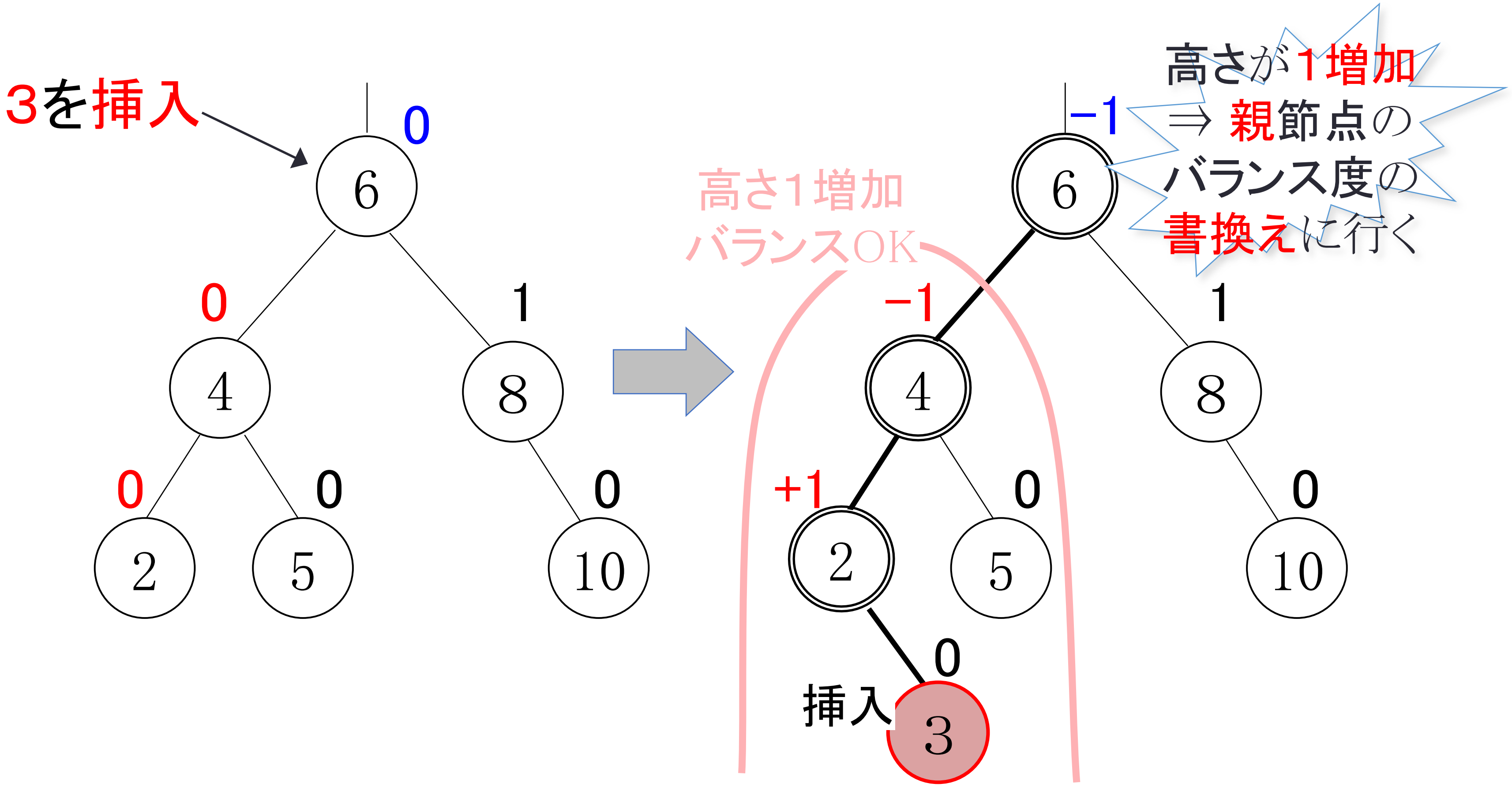
節点の挿入：場合B1

- バランス度: +1 → 0



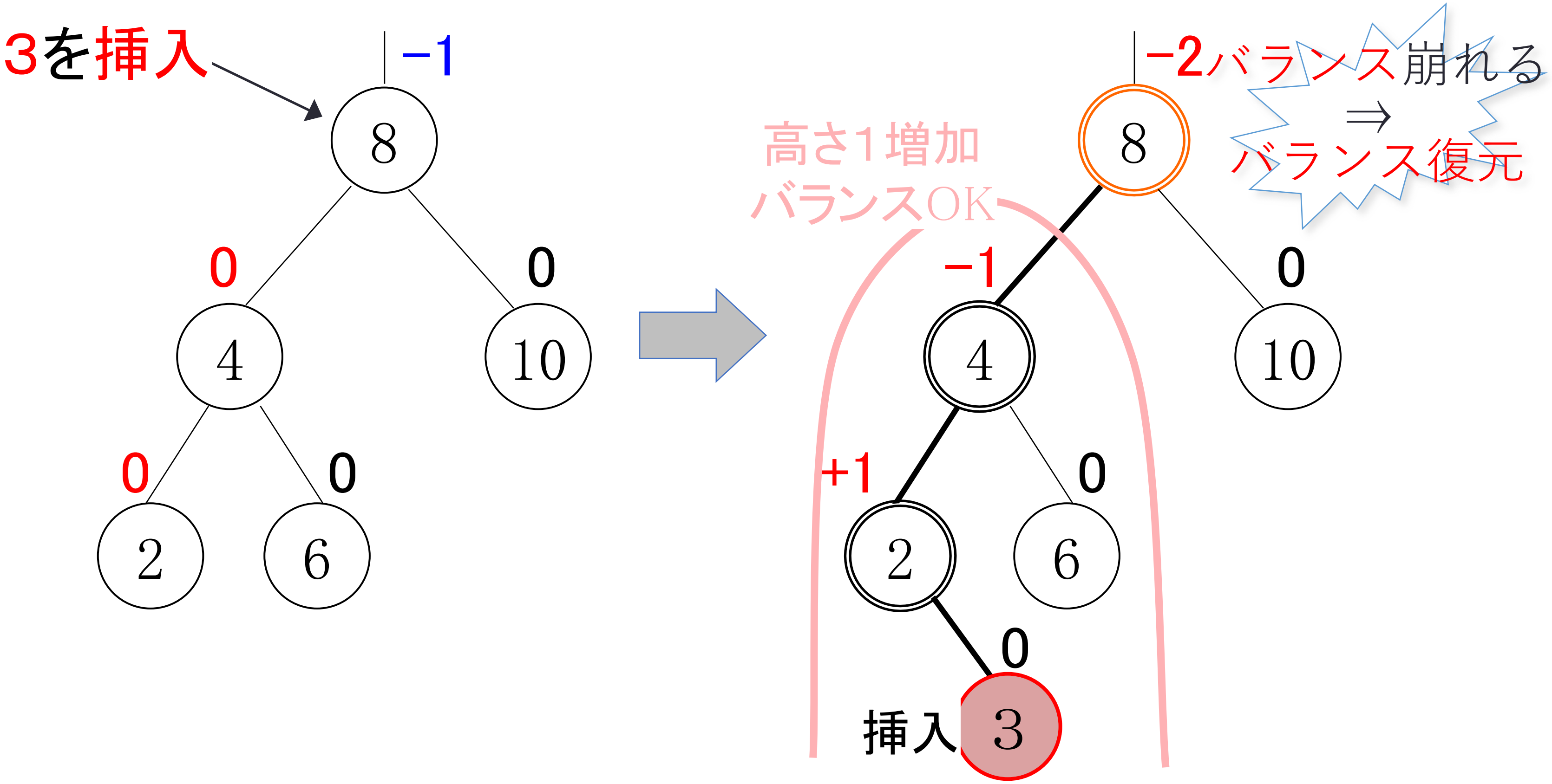
節点の挿入：場合B2

- バランス度: 0 → -1



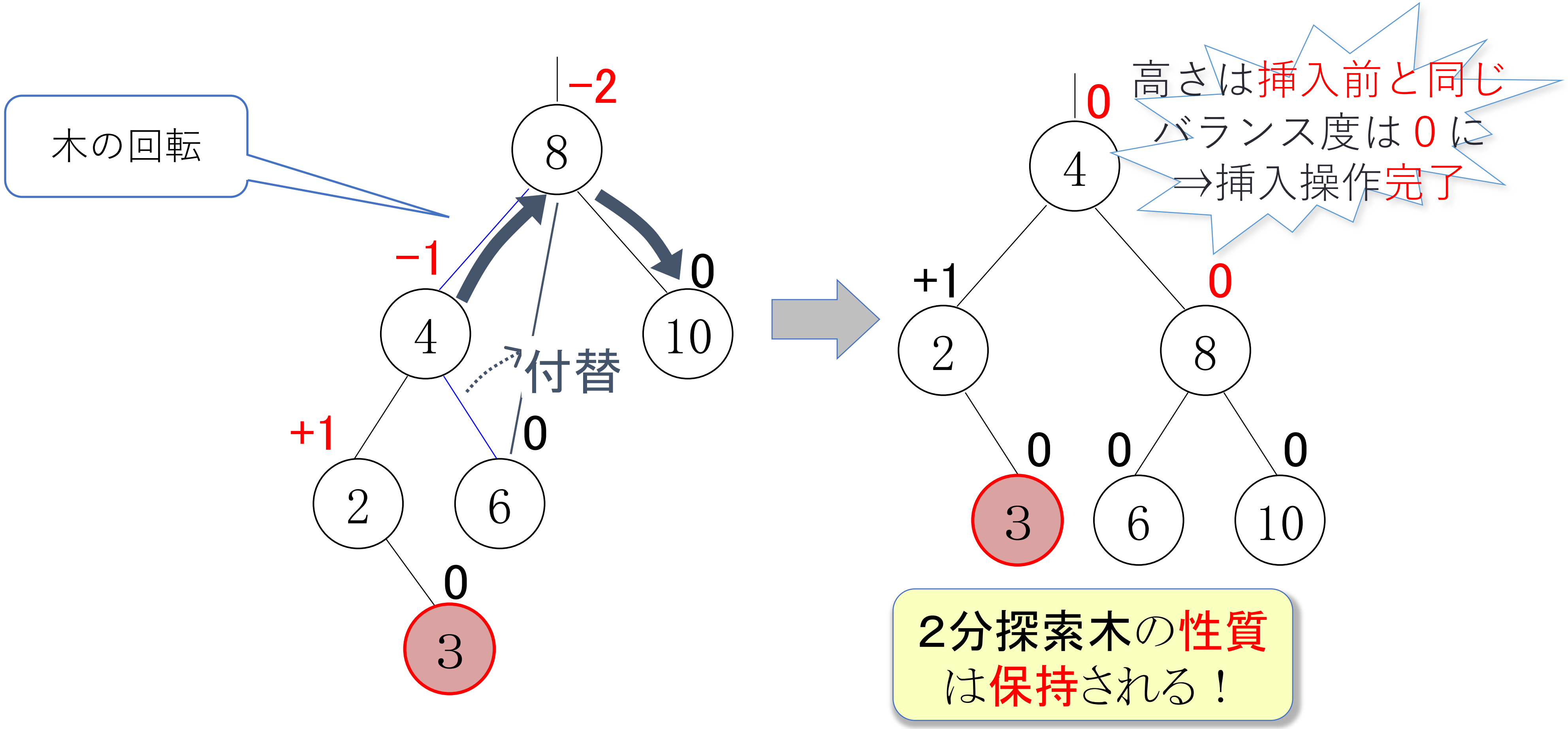
節点の挿入：場合B3

- バランス度: -1 → -2



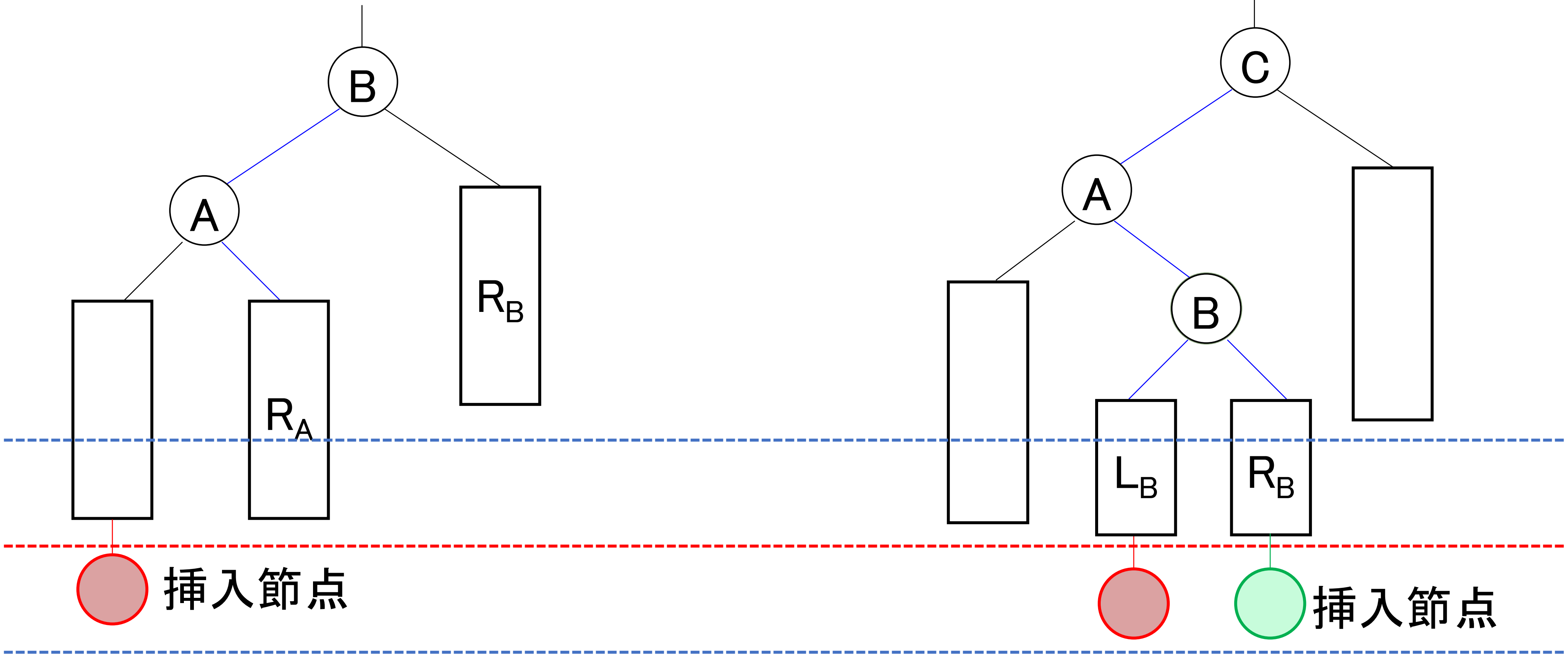
節点の挿入：場合B2（バランス復元）

- バランス度: -2 → 0 （バランス復元）

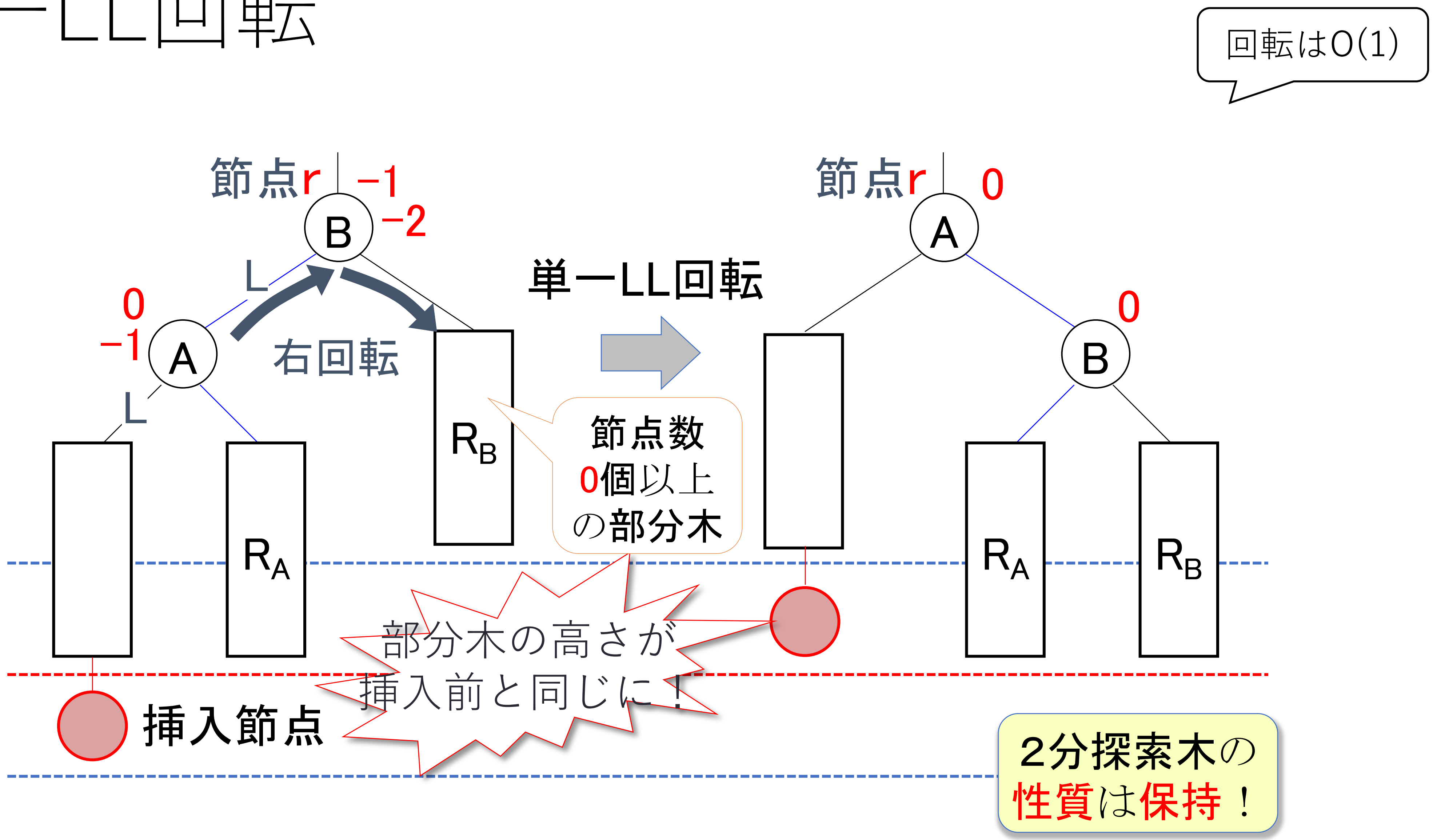


バランス復元操作

- 左の子の左部分木に節点が挿入されたとき
  - 単一LL回転でバランスを修正
- 左の子の右部分木に節点が挿入されたとき
  - 二重LR回転でバランスを修正

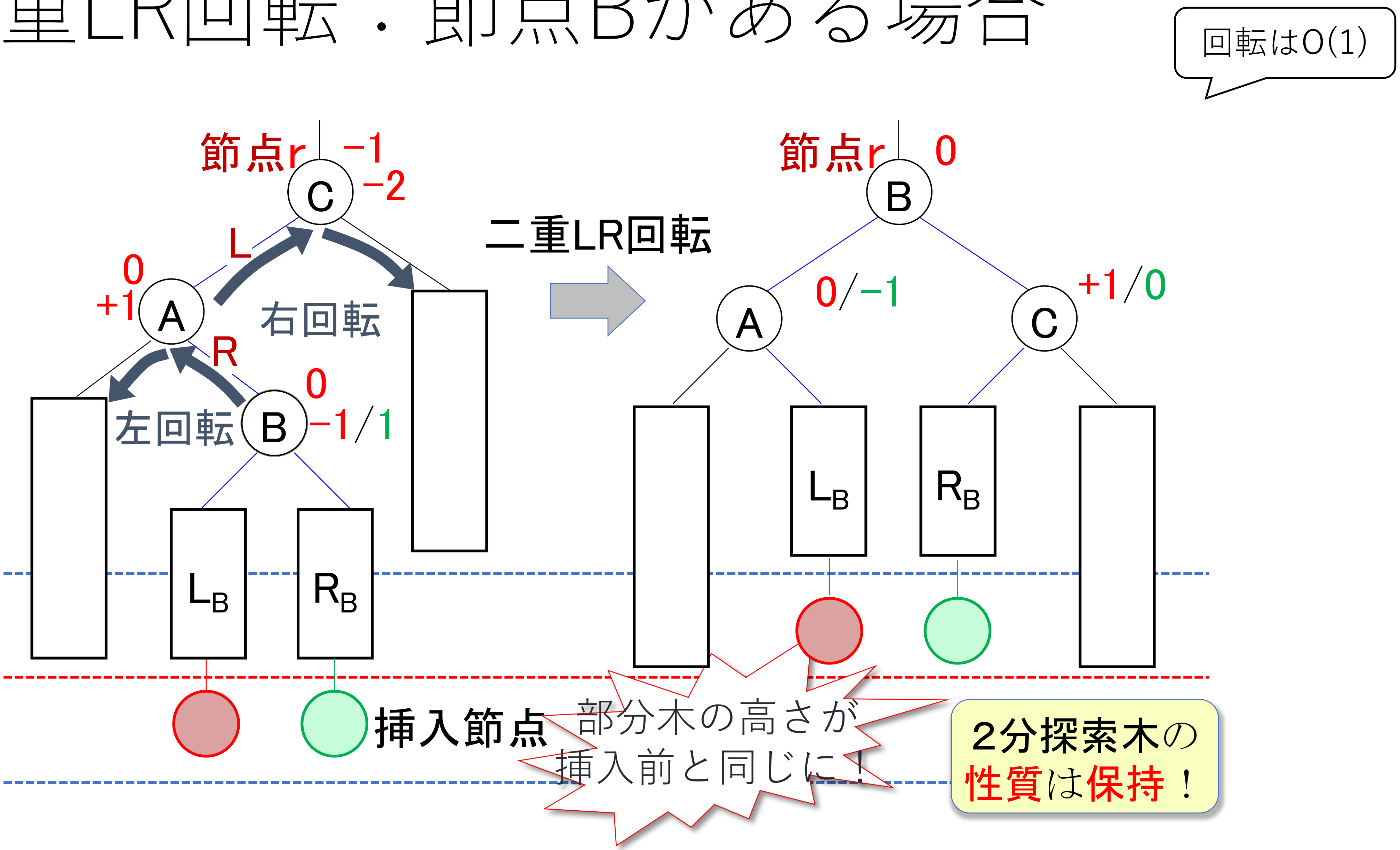


単一LL回転

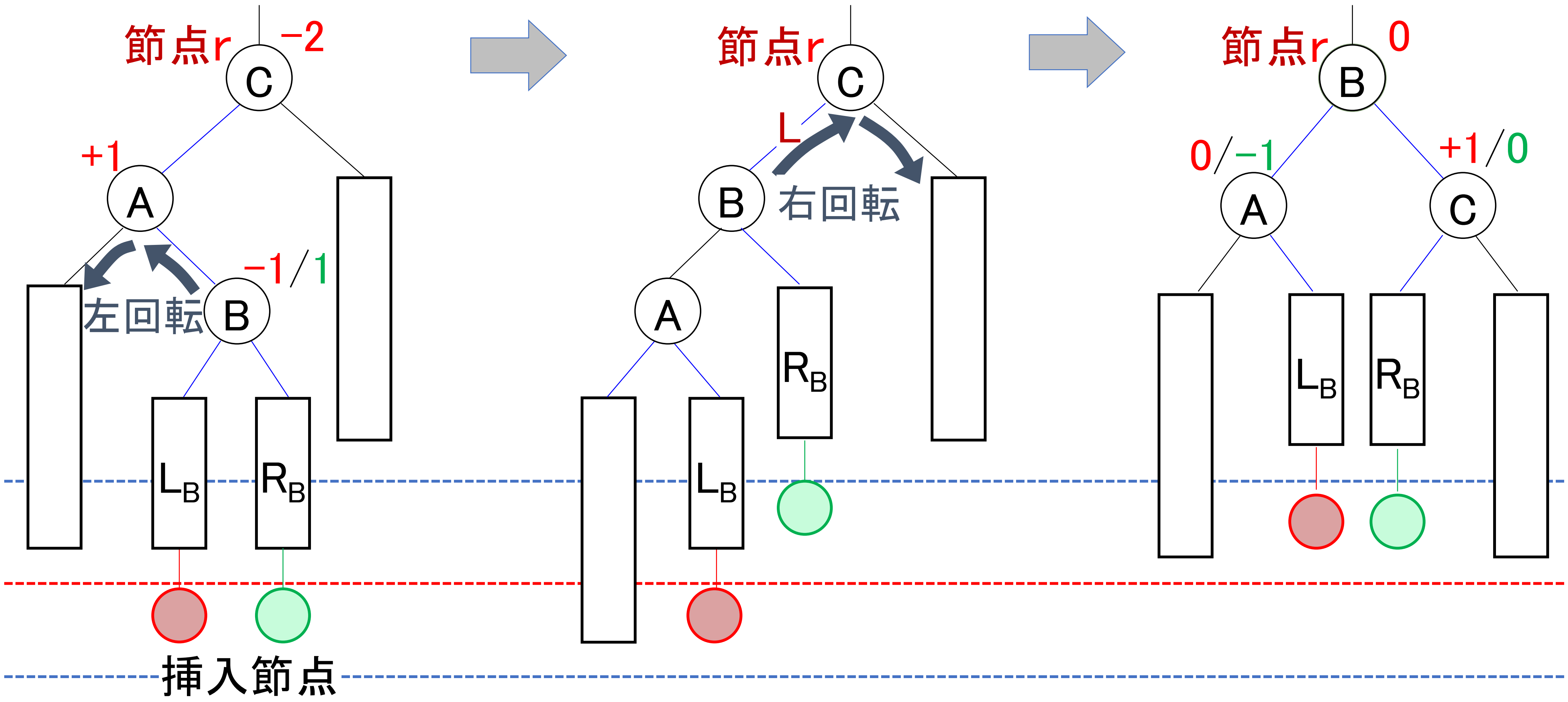


回転はO(1)
節点r
−1
−2
B
L
0
−1
A
L
右回転
RA
RB
節点数
0個以上
の部分木
単一LL回転
節点r
0
A
0
B
RA
RB
部分木の高さが
挿入前と同じに！
挿入節点
2分探索木の
性質は保持！

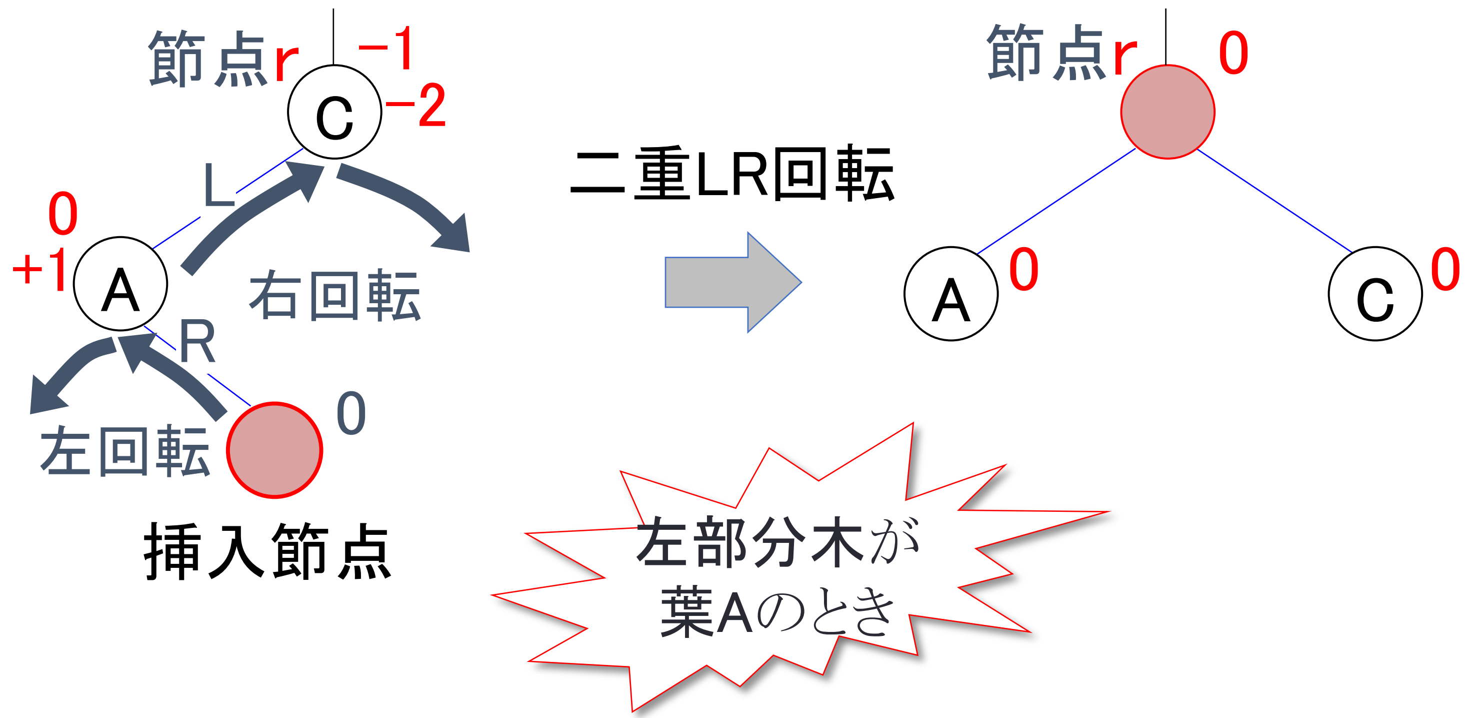
二重LR回転：節点Bがある場合



二重LR回転：二つの回転ステップ



二重LR回転：節点Bがない場合



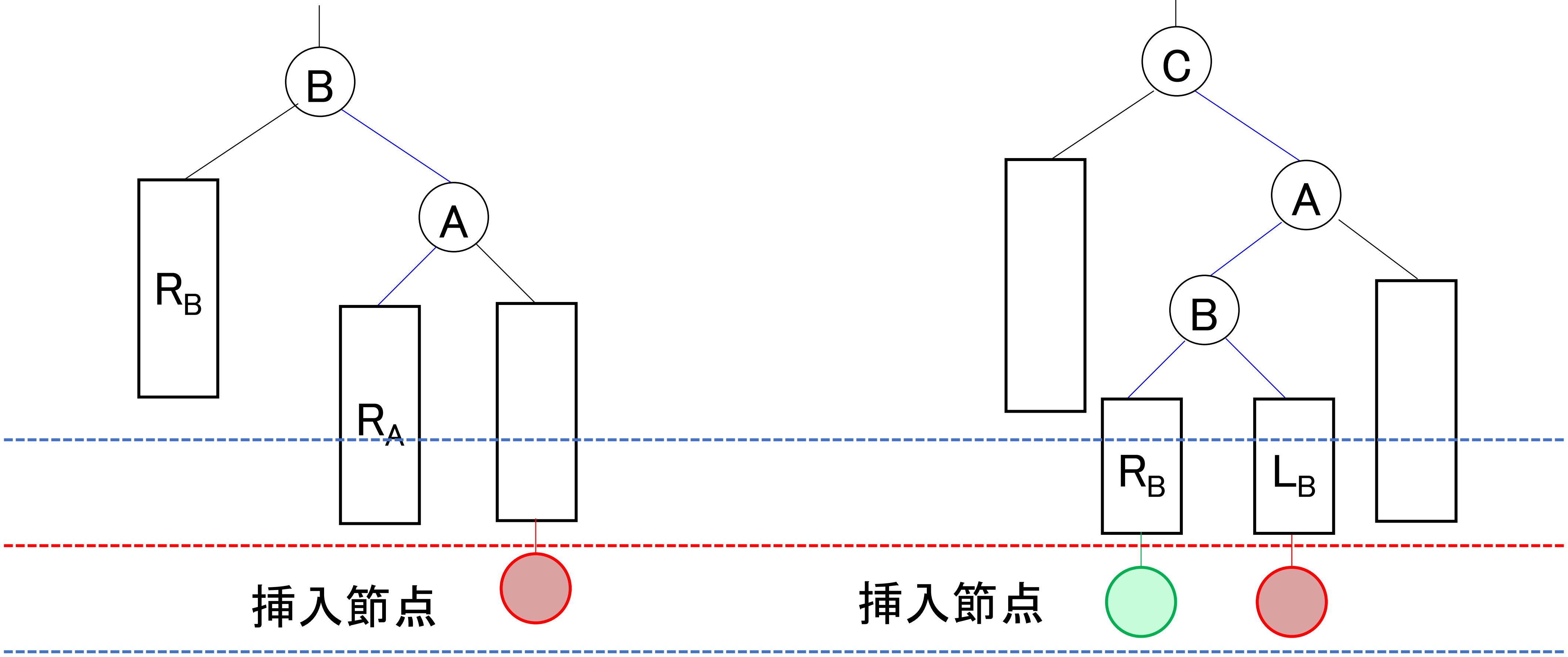
節点の右部分木への挿入： 3つの場合わけ

- AVL木に新しい節点を挿入するとき
  - 挿入前のLとRの高さをh_Lとh_Rとする
  - 根rのバランス度: h_R-h_L

- 場合B1’: バランス度-1のとき
  - バランス度0に
- 場合B2’: バランス度0のとき
  - バランス度+1に（バランス基準内）
- 場合B3’: バランス度+1のとき
  - バランス度+2に => バランス復元操作（木の作り替え）

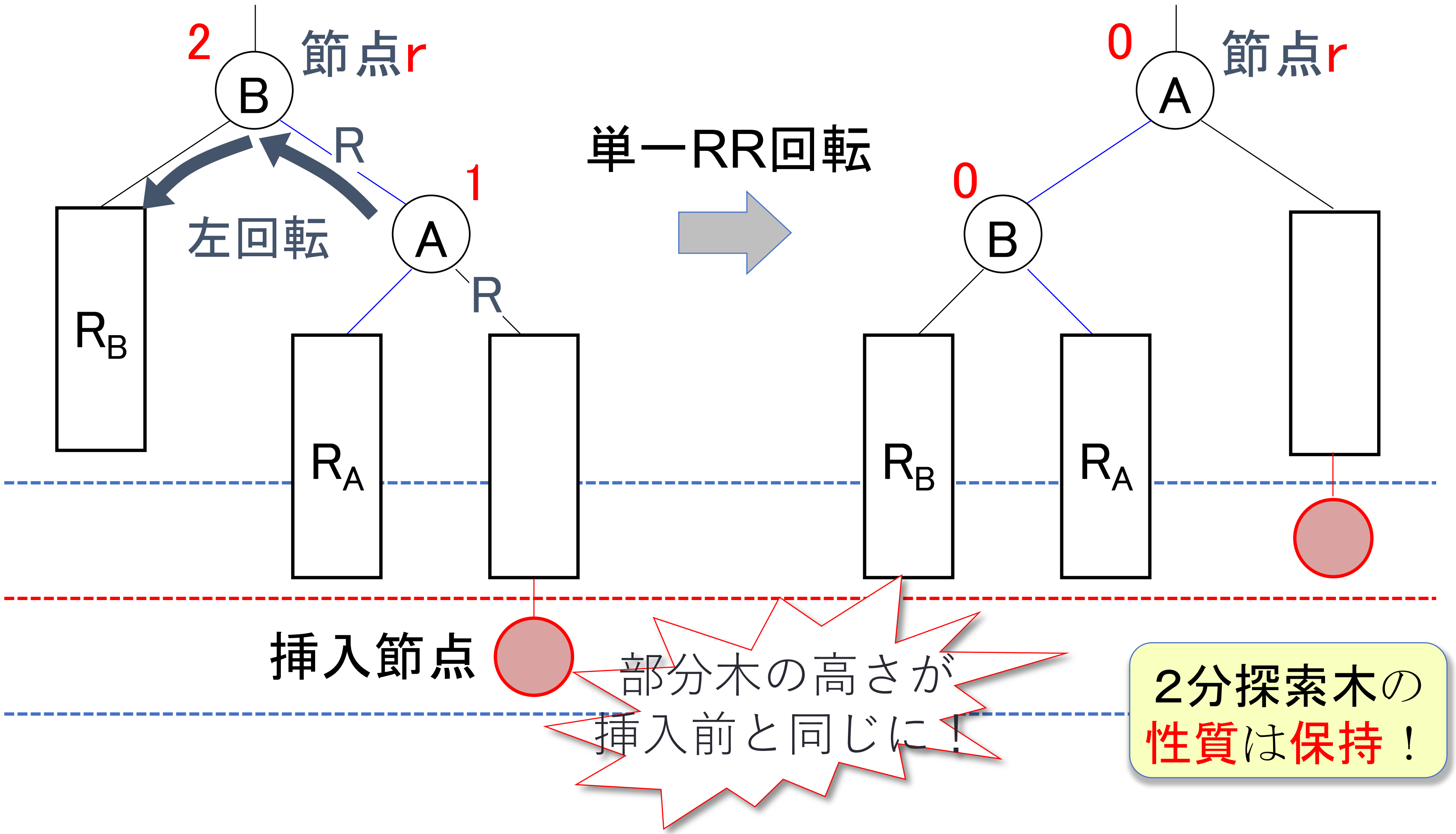
バランス復元操作

- 右の子の右部分木に節点が挿入されたとき
  - 単一RR回転でバランスを修正
- 右の子の左部分木に節点が挿入されたとき
  - 二重RL回転でバランスを修正



単一RR回転

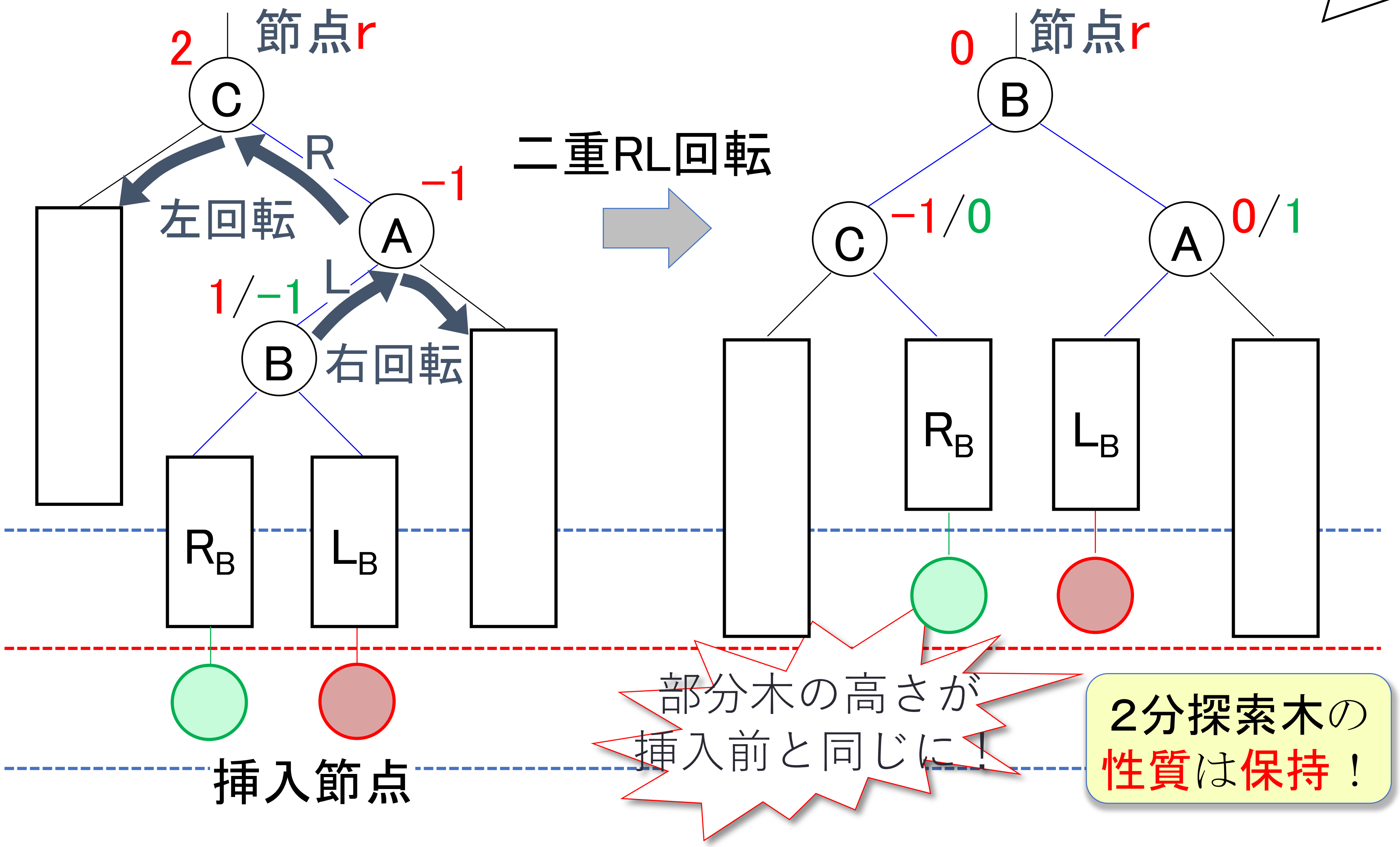
回転はO(1)



2分探索木の
性質は保持！

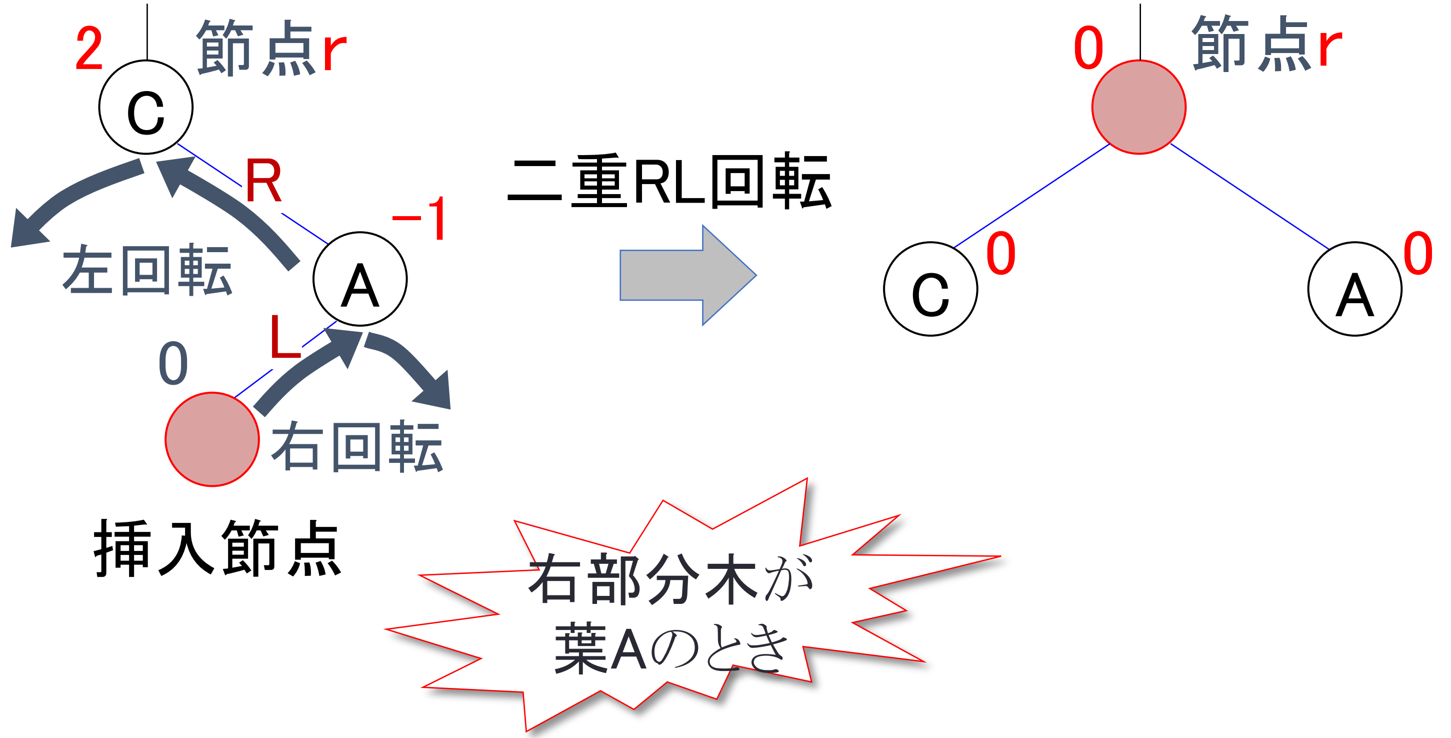
二重RL回転：節点Bがある場合

回転はO(1)

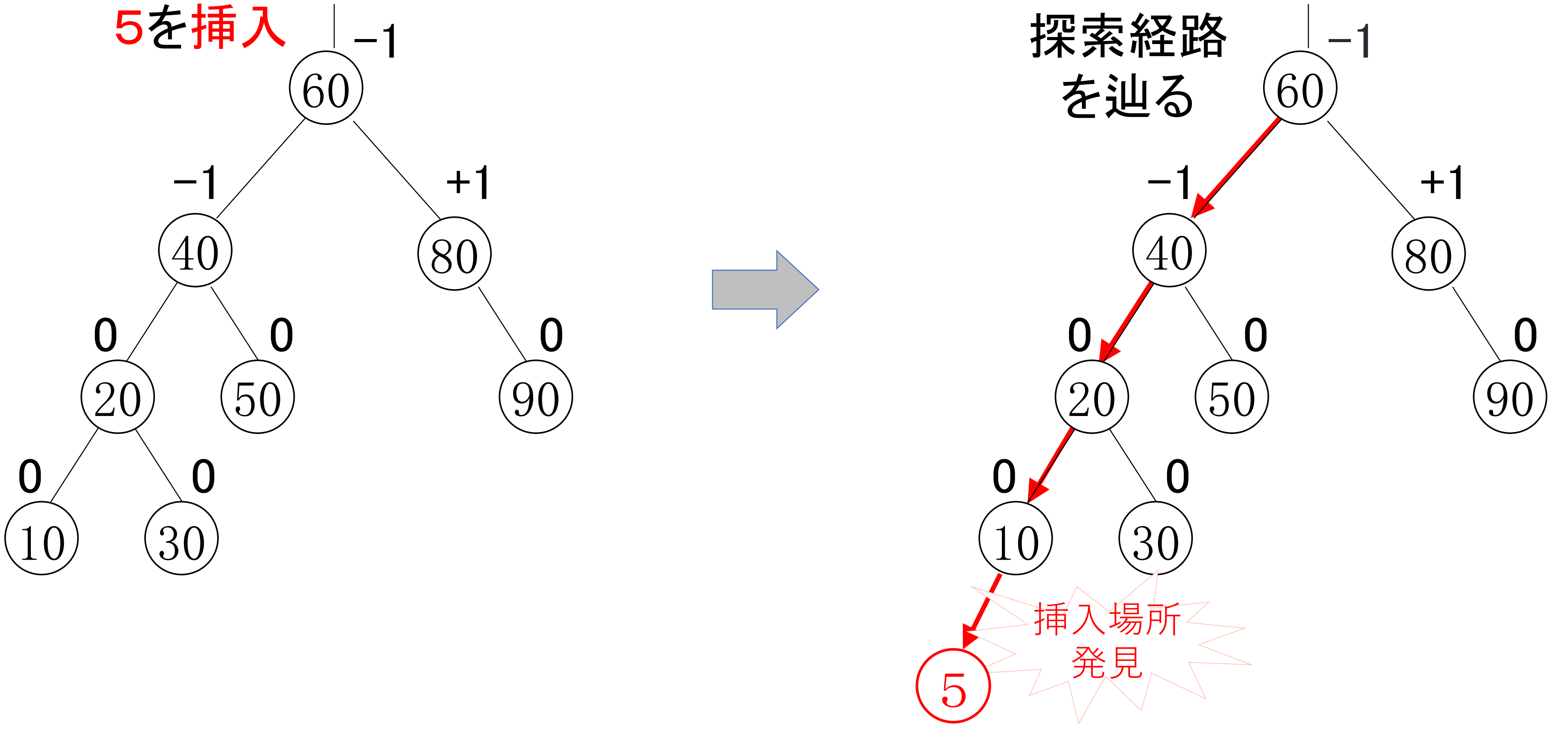


2分探索木の
性質は保持！

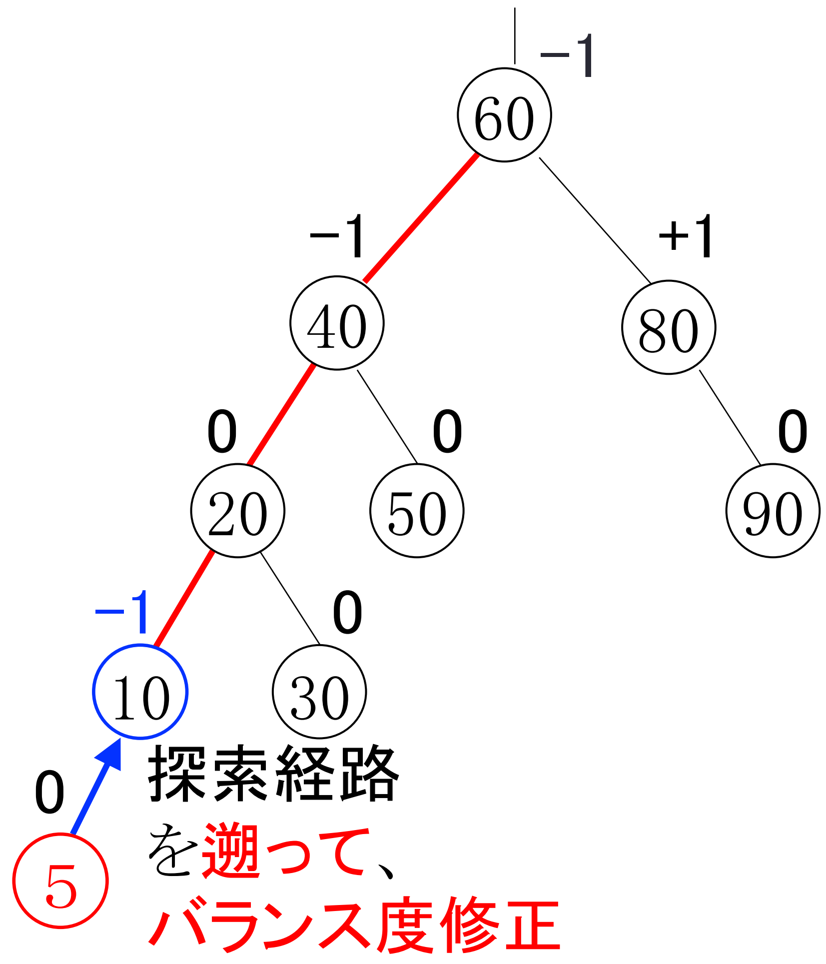
二重RL回転：節点Bがない場合

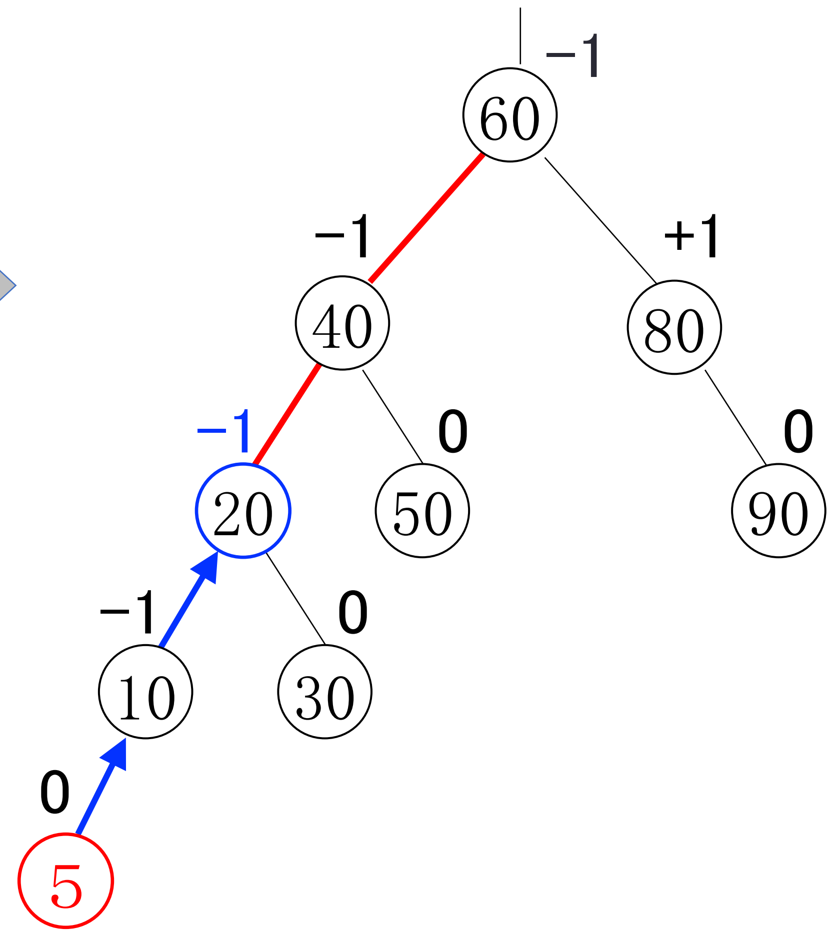


【例】節点挿入手順：探索・挿入

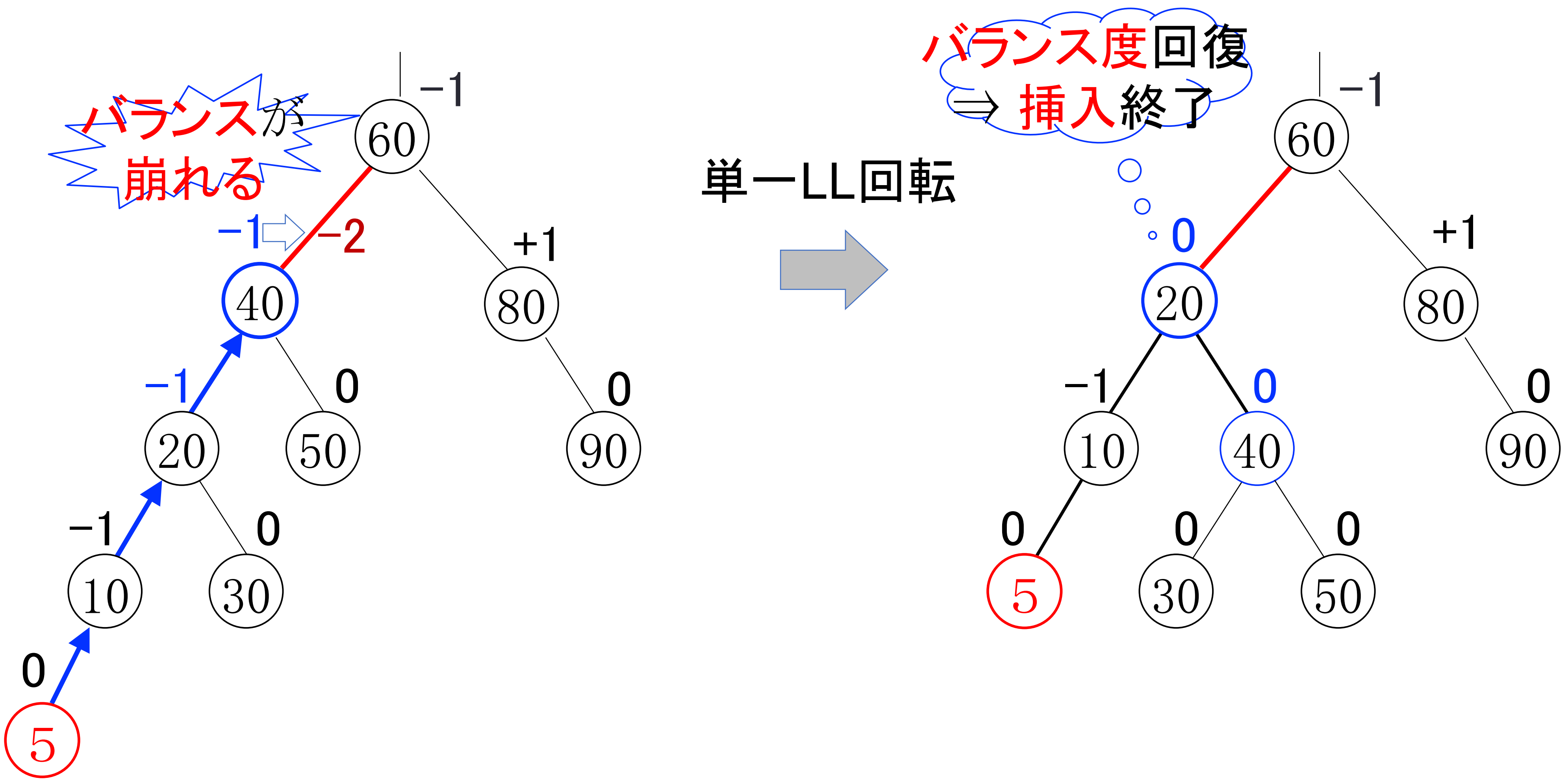


【例】節点挿入手順：逆戻り・バランス度修正





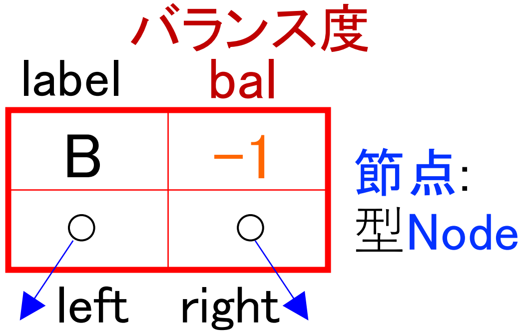
【例】節点挿入手順：バランス度崩壊⇒回復



AVL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;
    NodePointer right;
    int bal    /* バランス度 */
} Node;
```

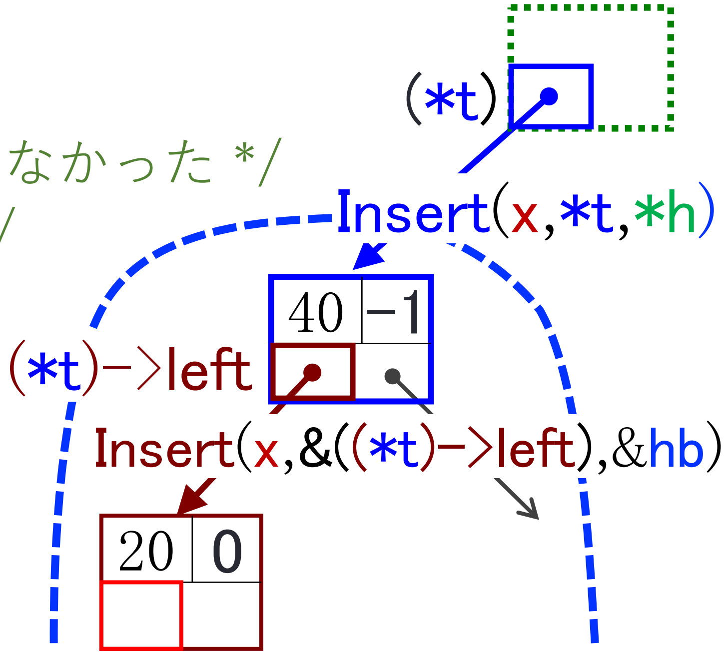


- AVL木変数
  - 根の節点（構造体）を指すポインタを格納する変数

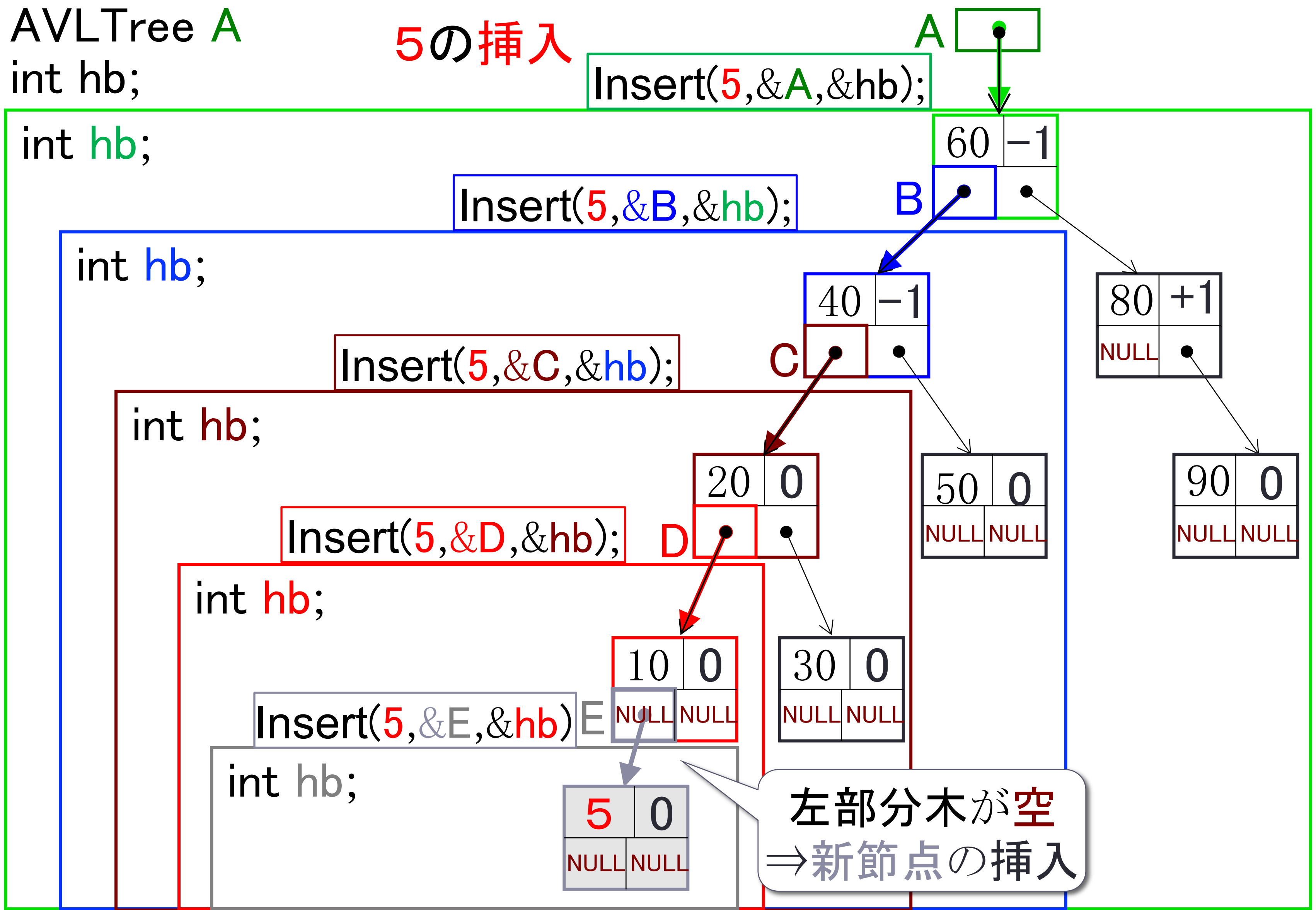
```
typedef NodePointer AVLTree;
```

AVL木の実現：Insert

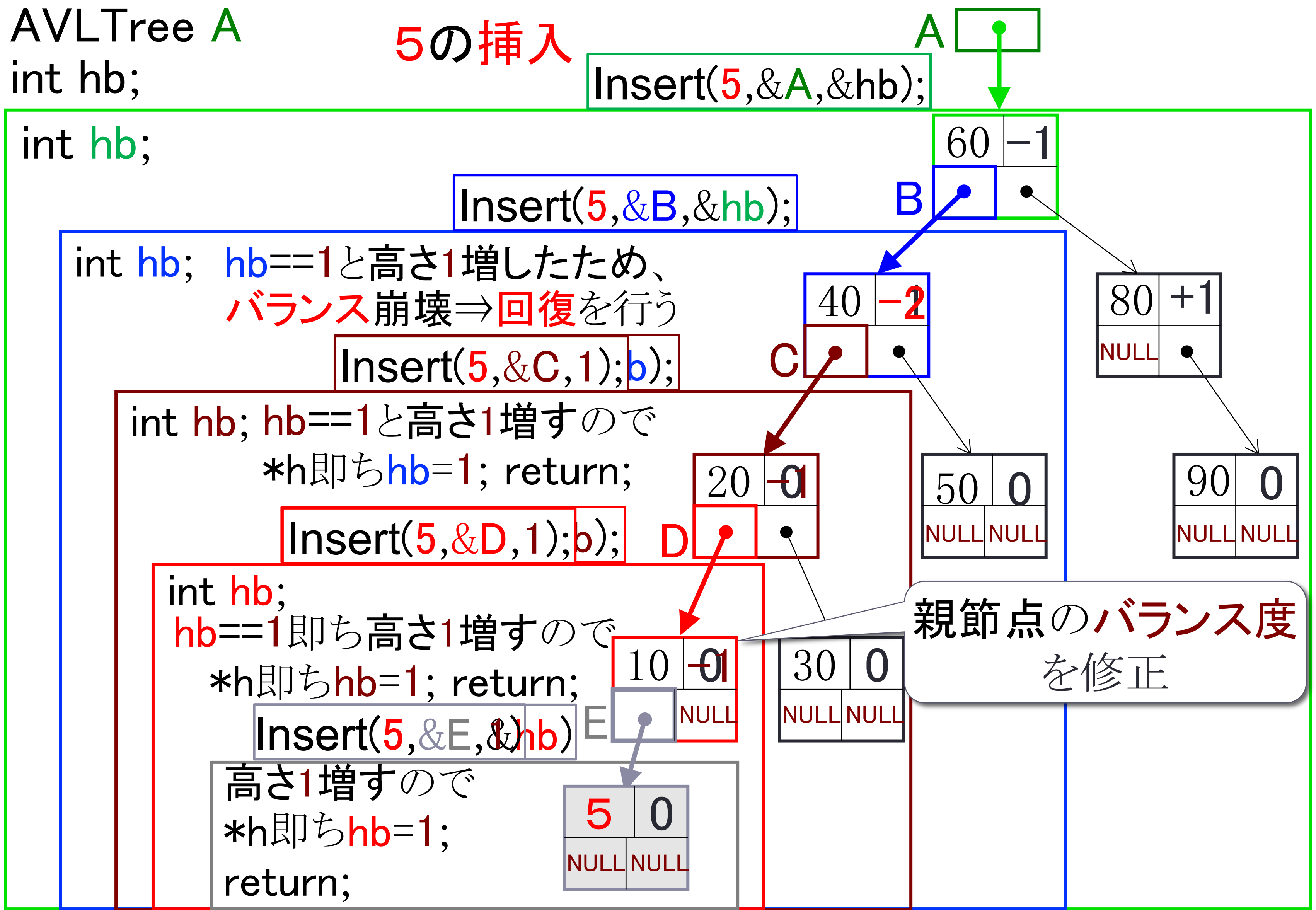
```
void Insert(Element x, AVLTree *t, int *h){                /* *hは木の高さ変化 */
    int hb;                          /* 挿入される部分木の高さが高くなったらhb = 1 */
    if((*t) == NULL){ *t = malloc(…); …; *h = 1; return;}    /* 部分木が空: 節点x挿入 */
    else if(x < (*t)->label){            /* 左部分木に挿入 */
        Insert(x, &((*)->left), &hb);     /* 再帰 */
        if(!hb){ *h = 0; return;}         /* 左部分木高さが増えなかった */
        else switch((*t)->bal){           /* 左部分木高さ増加 */
            case 1: (*t)->bal = 0; *h = 0; return;    /*場合B1*/
            case 0: (*t)->bal = -1; *h = 1; return;   /*場合B2*/
            case -1: {…}                  /* 場合B3 バランス復元 */
    }else if(x > (*t)->right){        /* 右部分木に挿入 */
        Insert(x, (*t)->right), &hb);      /* 再帰 */
        {…}             /* 右部分木挿入後のB1'B2'B3'の処理 */
    }else{*h = 0; return;}             /* xが存在 挿入せず */
}
```



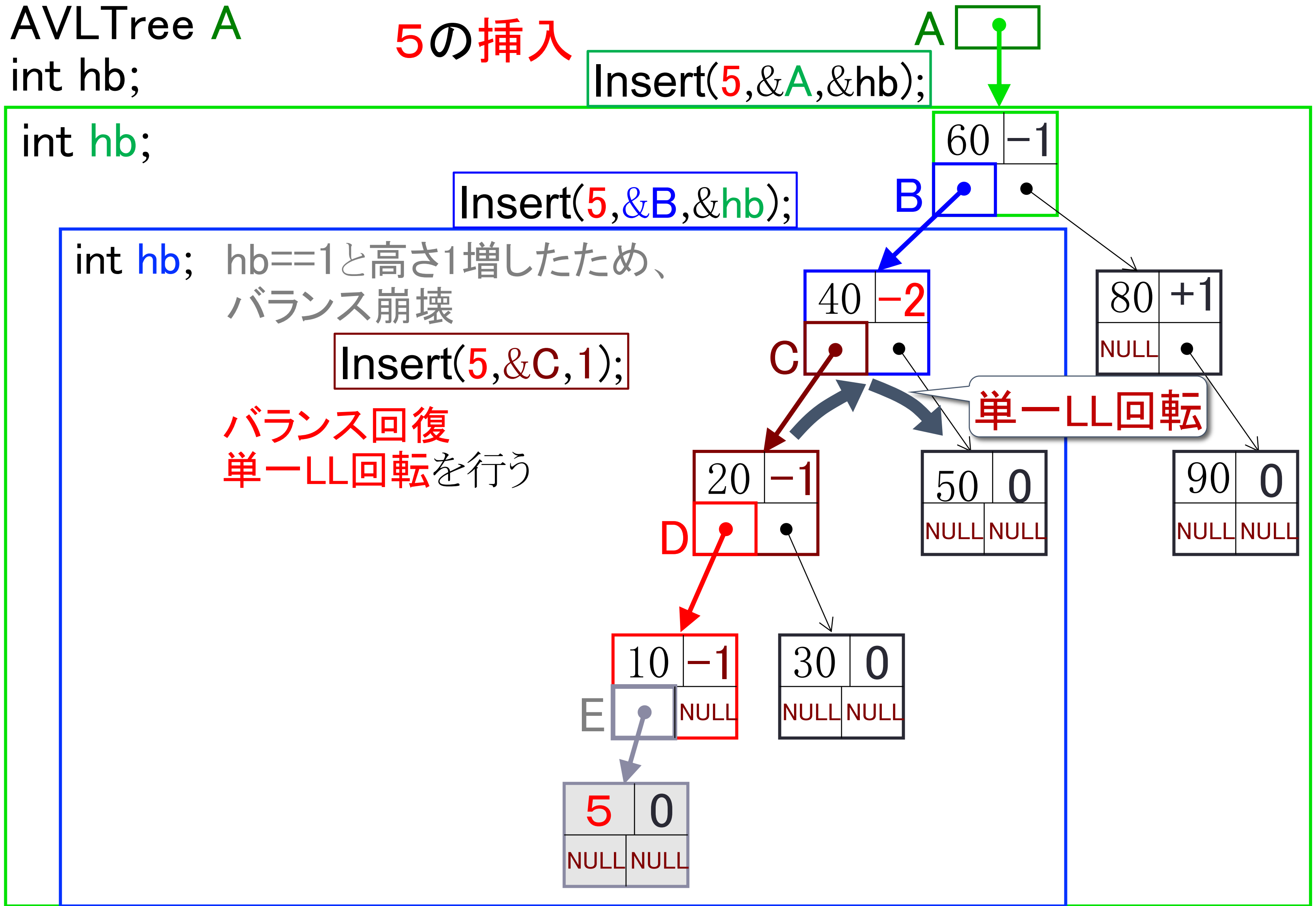
節点の再帰による挿入：探索・操作



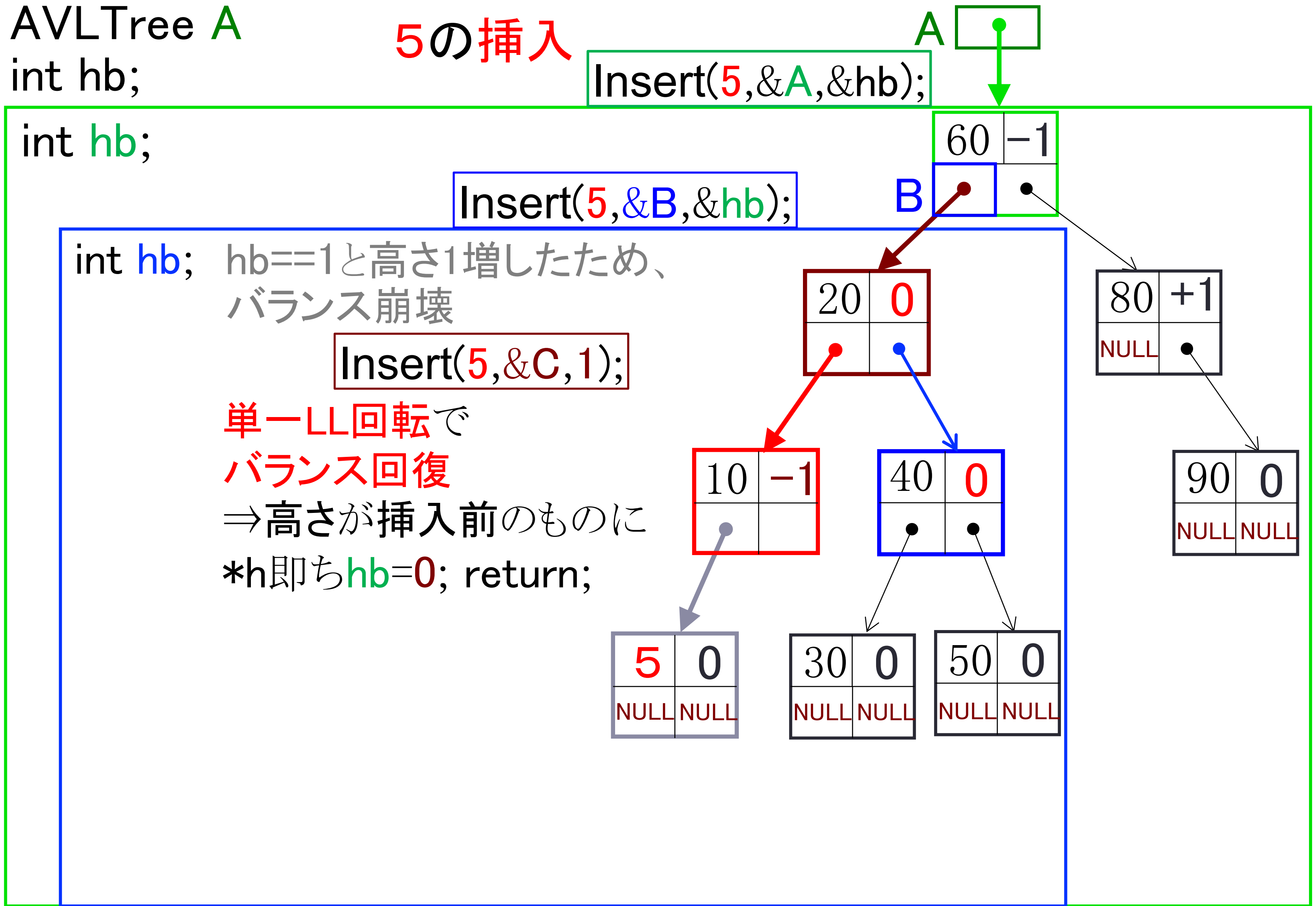
節点の再帰による挿入：逆戻り



節点の再帰による挿入：バランス回復



節点の再帰による挿入：単一LL回転直後



節点の再帰による挿入：単一LL回転直後

AVLTree A
int hb;

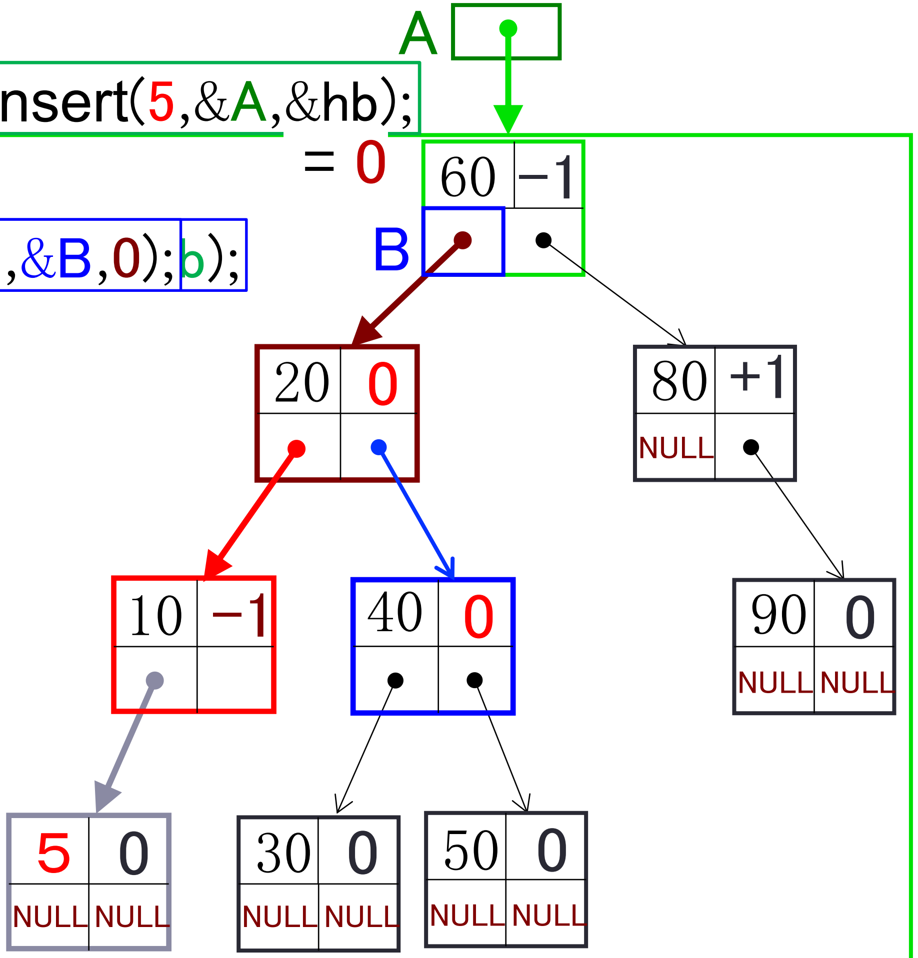
5の挿入

Insert(5,&A,&hb);
= 0

int hb;

Insert(5,&B,0);b);

hb==0 なので
木Bの高さが挿入前のものに
⇒ 挿入終了のため
*h即ちhb=0; return;



AVL木の削除処理

- 削除処理は
  - 二分探索木と同様にデータを削除し
  - AVL木の挿入と同様に，必要であればバランス回復する

まとめ

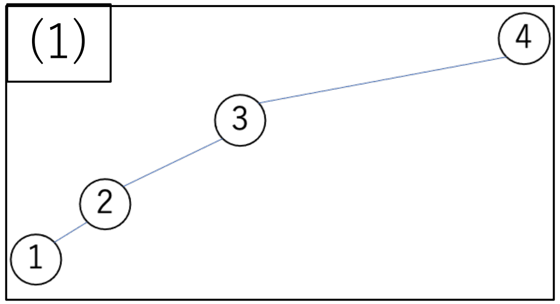
- AVL木による，辞書の実現を行った
  - 二分探索木は木のバランスが崩れると計算量が増える
  - AVL木はバランス度-1, 0, +1を許容する
  - これを超えた場合，回転によって木の作り替えバランスを保つ
  - バランスによって計算量を抑える

演習問題

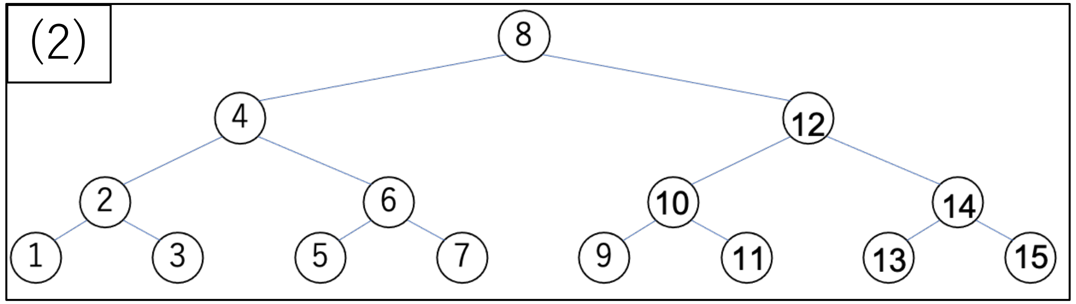
演習 AVL木 1

- 次の(1)から(5)のなかでAVL木でないものはどれか（理由ものべよ）

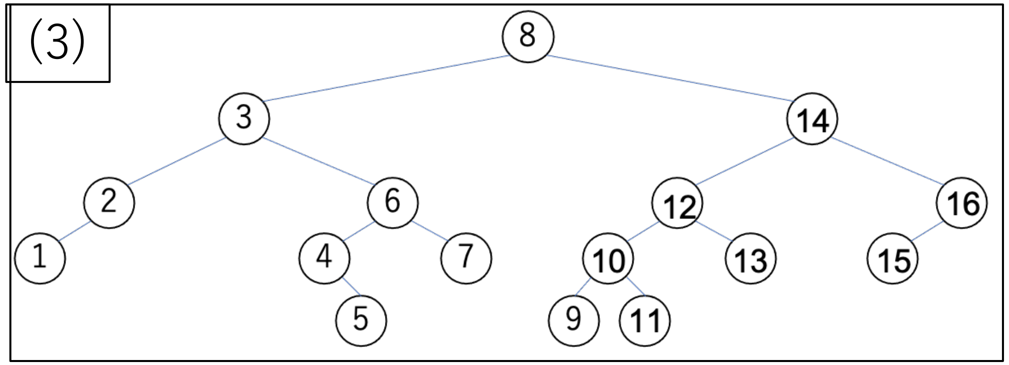
(1)



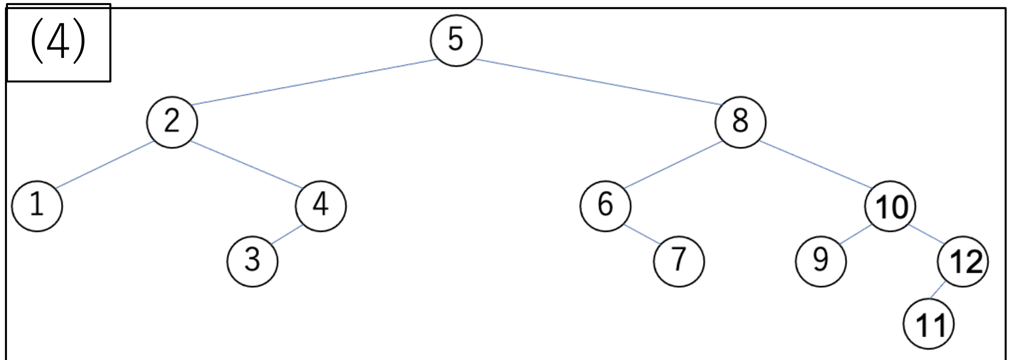
(2)



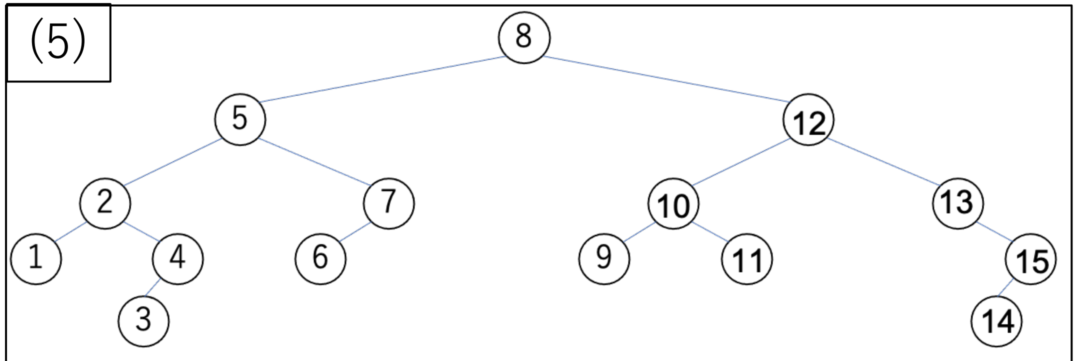
(3)



(4)

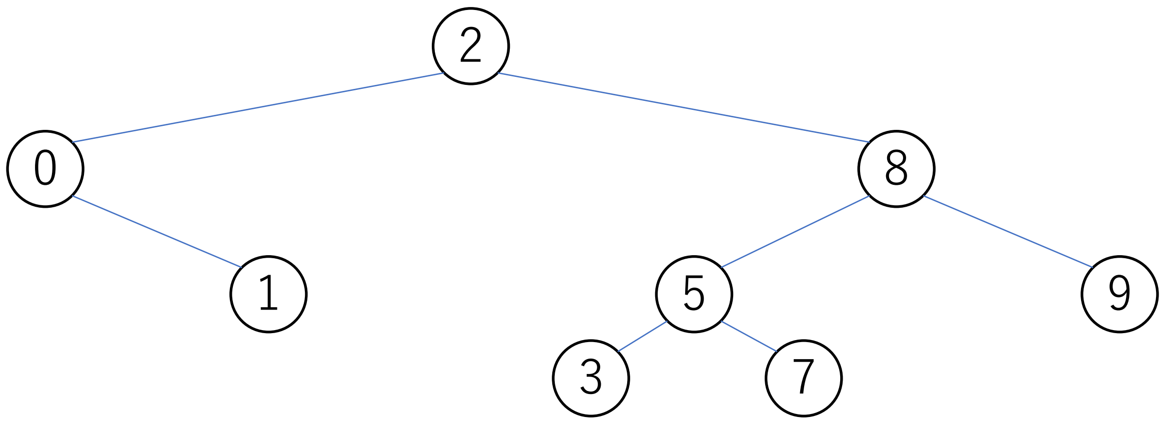


(5)



演習 AVL木 2

- 下記のAVL木に対して（A）から（F）の操作を順に行うと，各操作後に得られるAVL木はどうなるか？



（A）6の挿入　（B）1の削除　（C）4の挿入

（D）3の削除　（E）8の削除　（F）1の挿入

演習 C言語で実現（余裕があったら）

- AVL木をC言語で実装
  - Insertの実装が省略部分があったり，Deleteのアルゴリズムも省略してあるので，結構大変かもしれません.
  - 余裕があったらチャレンジしてください

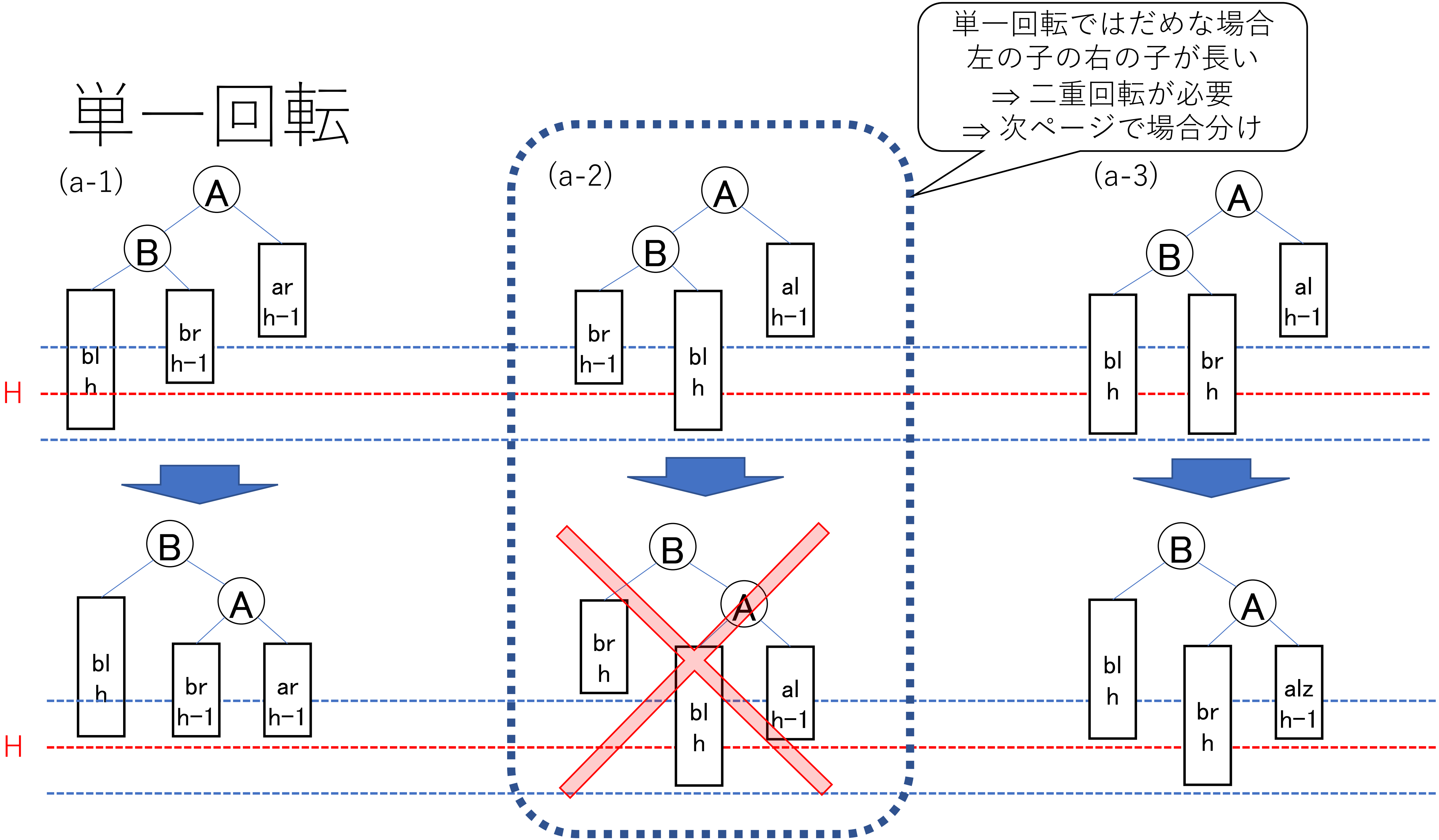
提出方法

- 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
- C言語の実現した場合は，テキストファイルのソースコードを提出してください

- 提出方法：LETUS
- 締め切り：2023/7/17 10:30まで

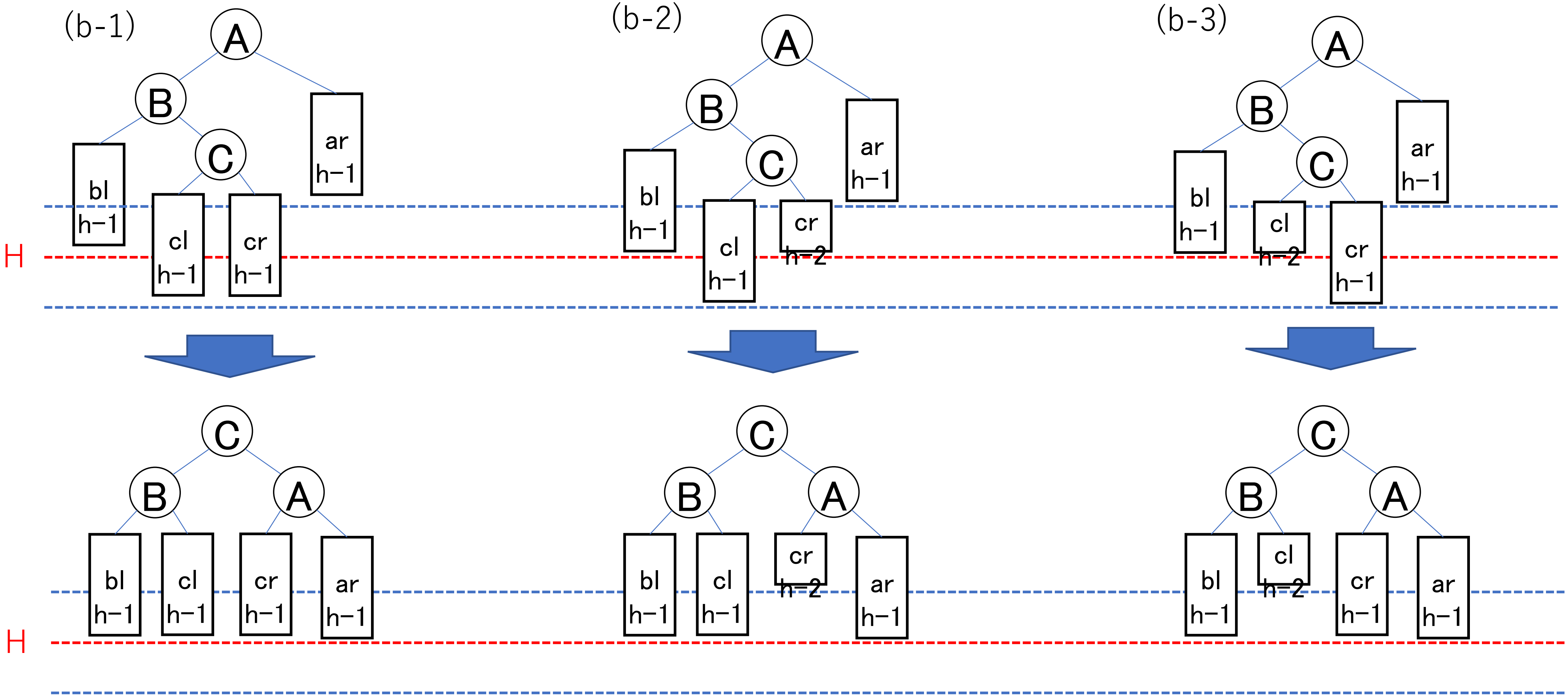
参考：回転と高さの変化

単一回転



単一回転ではだめな場合
左の子の右の子が長い
⇒ 二重回転が必要
⇒ 次ページで場合分け
(a-1)
A
B
bl
h
br
h−1
ar
h−1
H
B
A
bl
h
br
h−1
ar
h−1
H
(a-2)
A
B
br
h−1
bl
h
al
h−1
B
A
br
h
bl
h
al
h−1
(a-3)
A
B
bl
h
br
h
al
h−1
B
A
bl
h
br
h
alz
h−1

単一回転ではだめな場合：二重回転



高さの変化

	追加場所	追加前	追加後	追加時回転後	削除場所	削除前	削除後	削除時回転後
A-1	bl	H	H+1	H	ar	H+1	H+1	H
A-2	br				ar			
A-3	なし				ar	H+1	H+1	H+1
B-1	なし				ar	H+1	H+1	H
B-2	cl	H	H+1	H	ar	H+1	H+1	H
B-3	cr	H	H+1	H	ar	H+1	H+1	H