アルゴリズム論

2017年6月12日

樋口文人

目次

- 木構造
  - 高さ
  - 節／葉
  - 路長
    - 内部路長
    - 外部路長
- 部分木
- 二分探索木
  - 平衡木
    - AVL Adel’son-Vel’skii-Landis
  - 平衡の回復
    - １重の回転
    - ２重の回転

木構造

木に関する用語

葉の高さ：根から葉までの枝の数

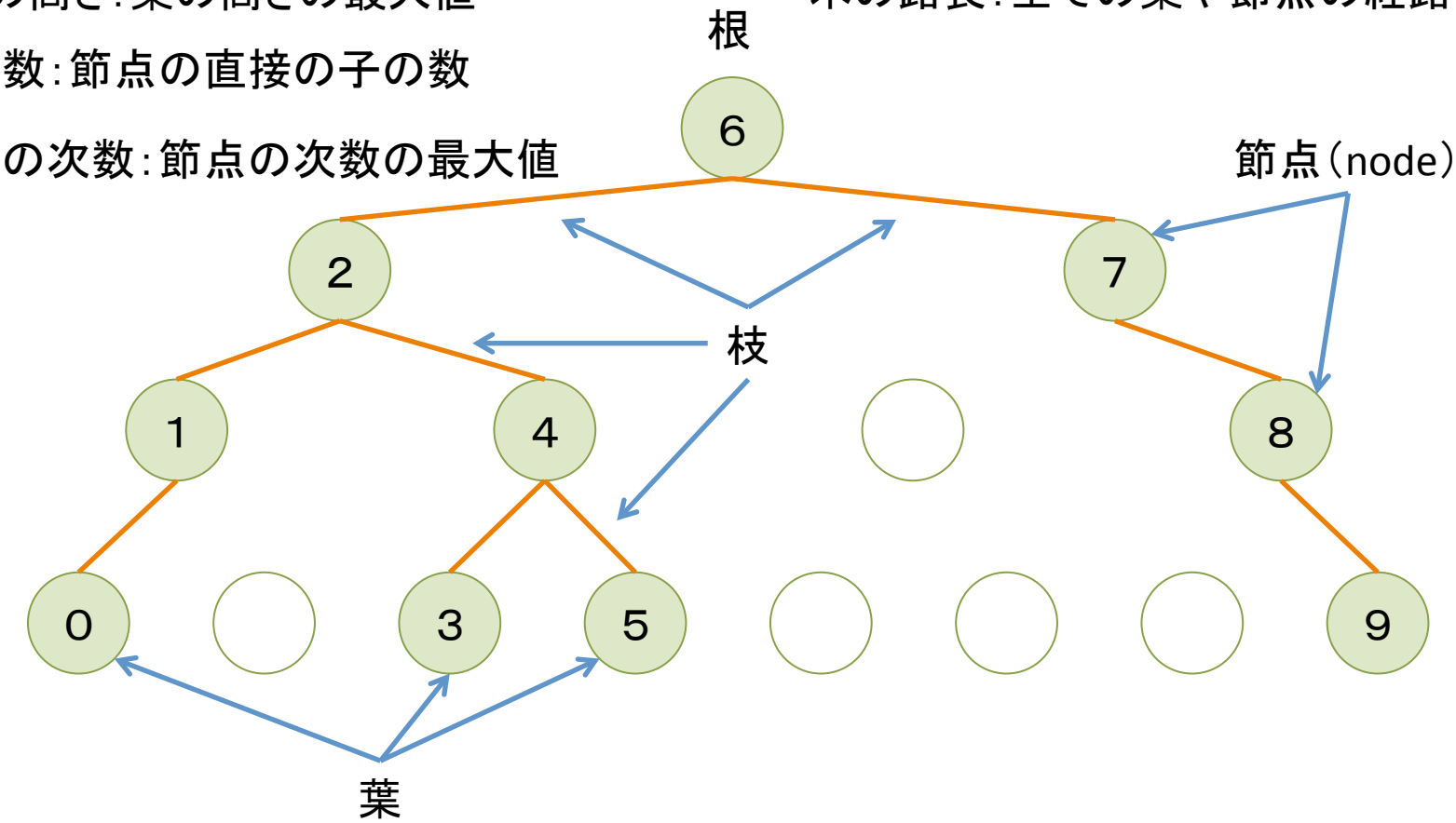
木の高さ：葉の高さの最大値

次数：節点の直接の子の数

木の次数：節点の次数の最大値

路長：根から葉や節点までの枝の数

木の路長：全ての葉や節点の経路の総和



木に関する用語つづき

拡張木：全ての節の次数を同じにする
（欠けている子を子を持たない節で補足）

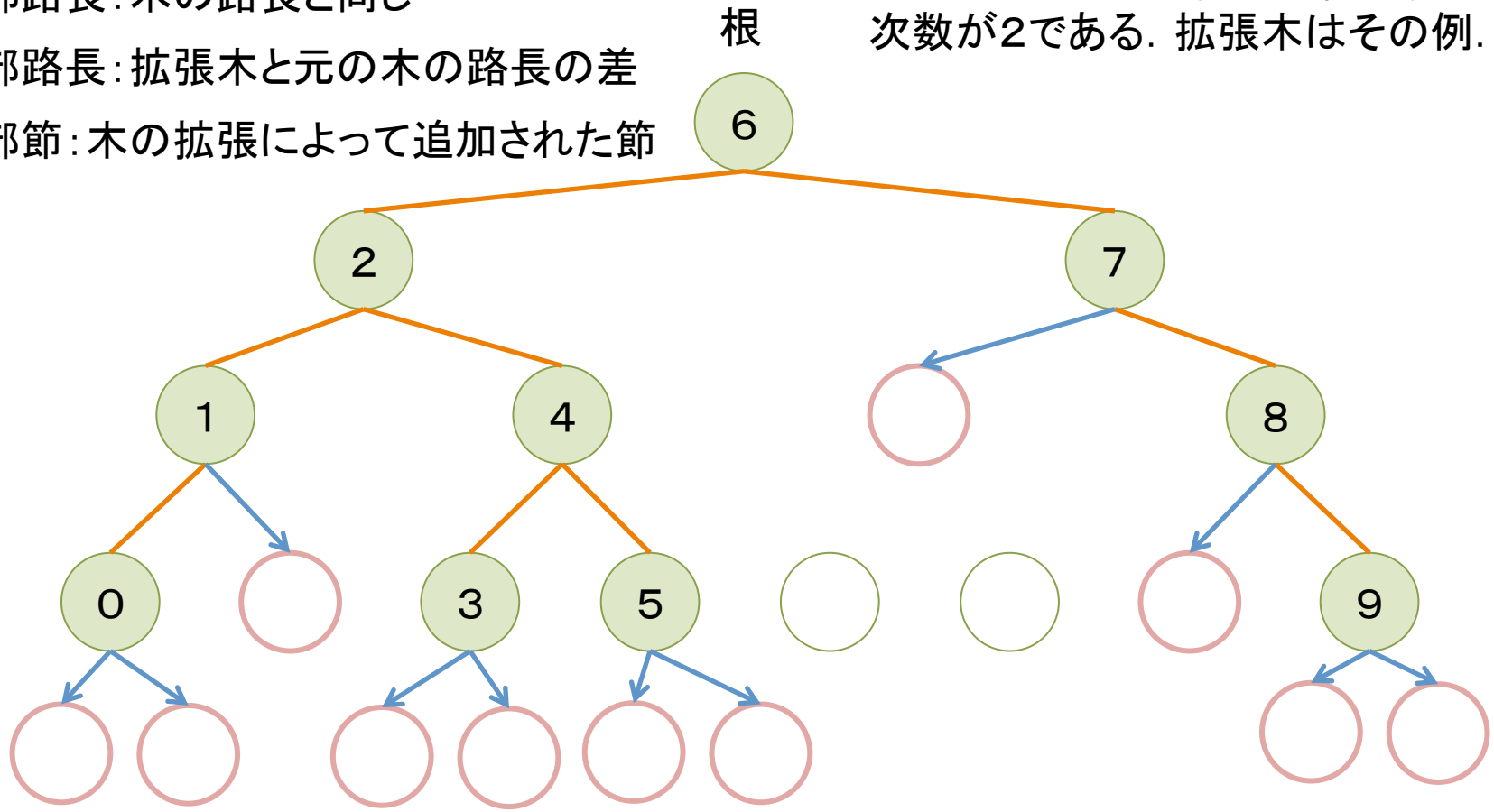
内部路長：木の路長と同じ

外部路長：拡張木と元の木の路長の差

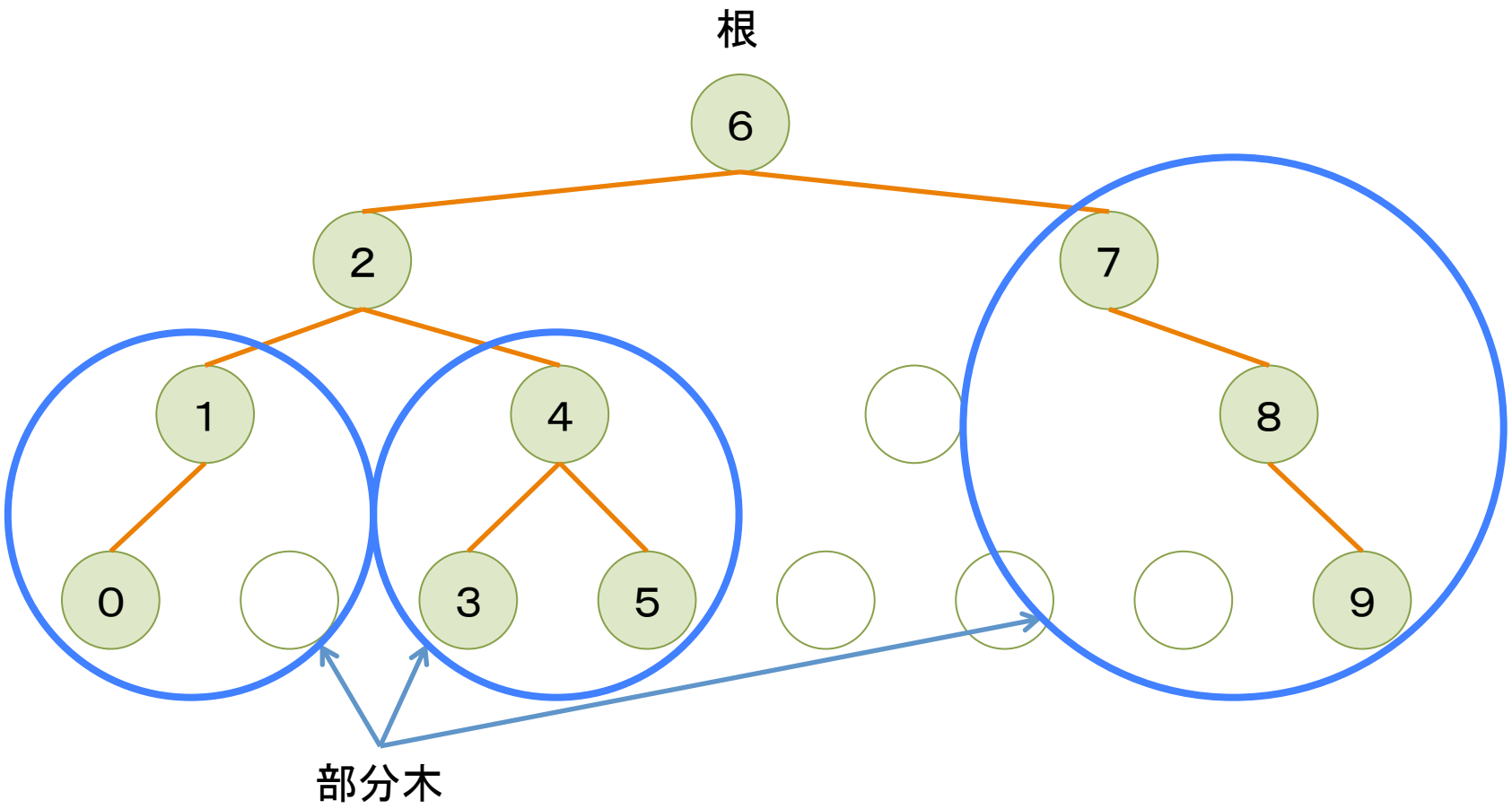
外部節：木の拡張によって追加された節

完全2分木：全ての葉の高さが同じである木

全2分木：全ての節点は葉であるか
次数が2である．拡張木はその例．

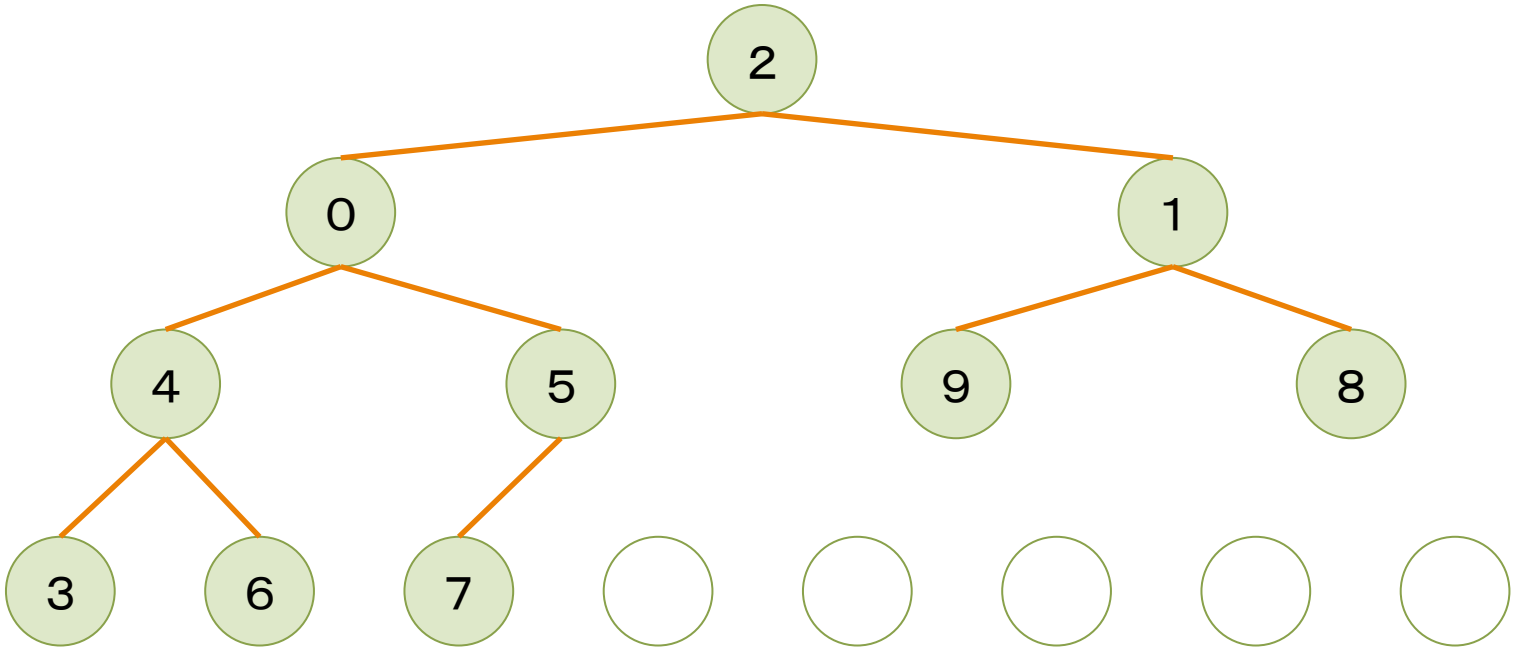


部分木



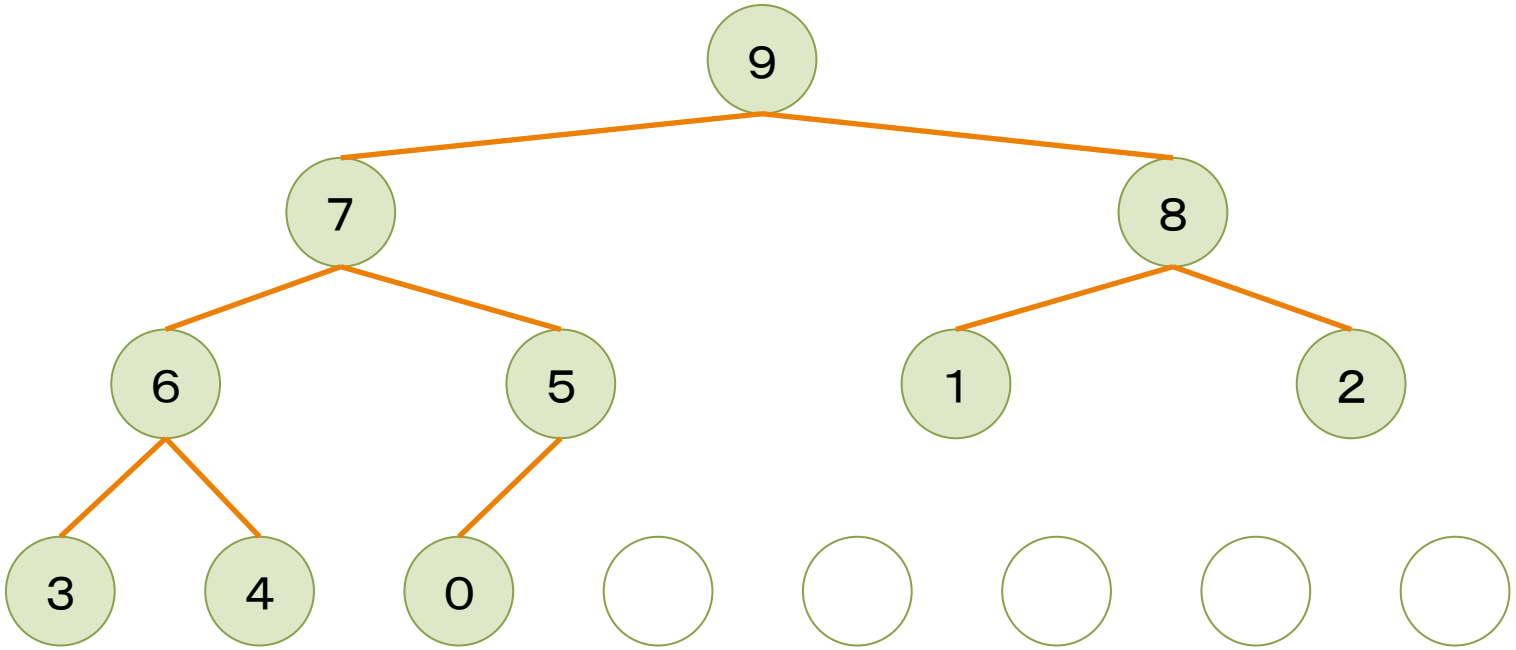
ヒープソート

2 0 1 4 5 9 8 3 6 7



ヒープ

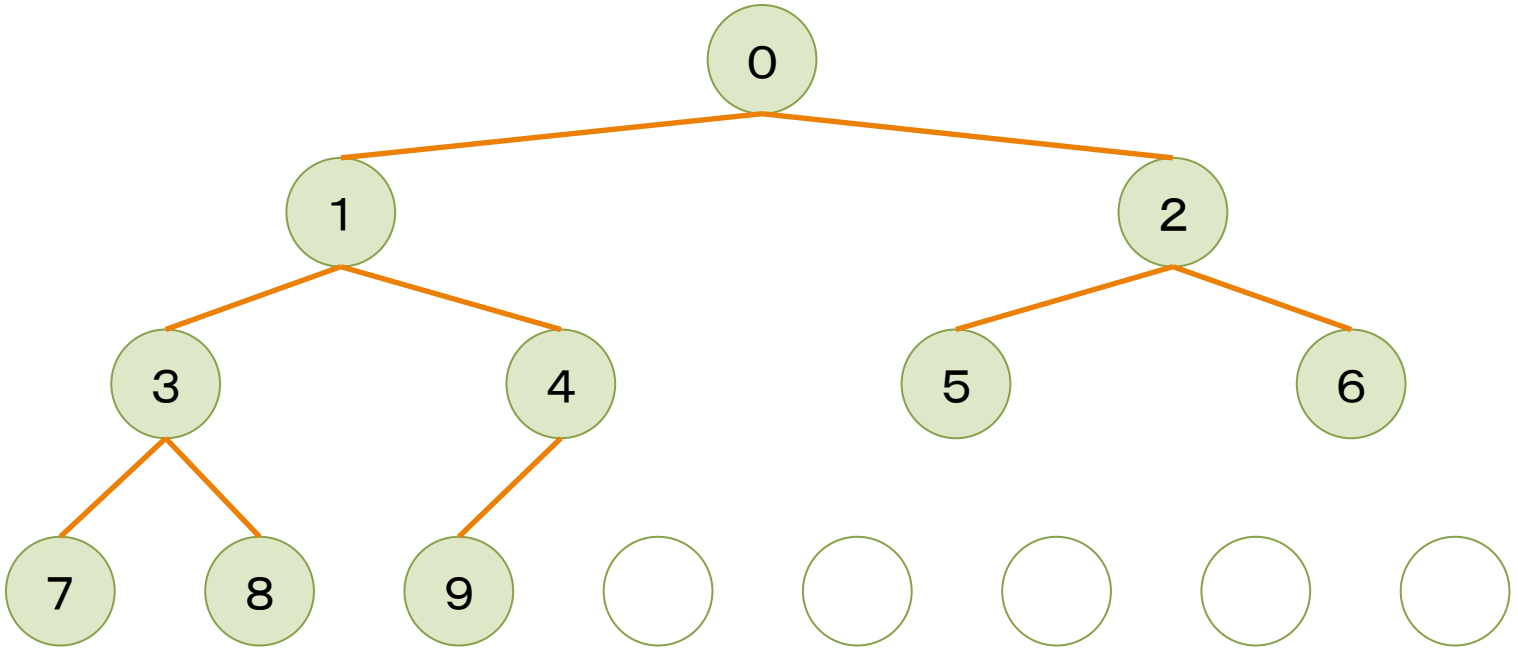
９７８６５１２３４０



親は子より大きい．子同士の大小は問わない．

ソートの完成

０１２３４５６７８９



木構造で探索

- 探索の開始点は根（ルート）から
- 子供同士も含めた親子の大小関係に基づく構造化
  - ２分探索木
- 配列では2分探索
- ヒープは親子間の大小関係
  - 子供同士の大小関係は問わ無い

２分探索木

左の子孫　＜　親　＜　右の子孫

左部分木の最大値　＜　親　＜　右部分木の最小値

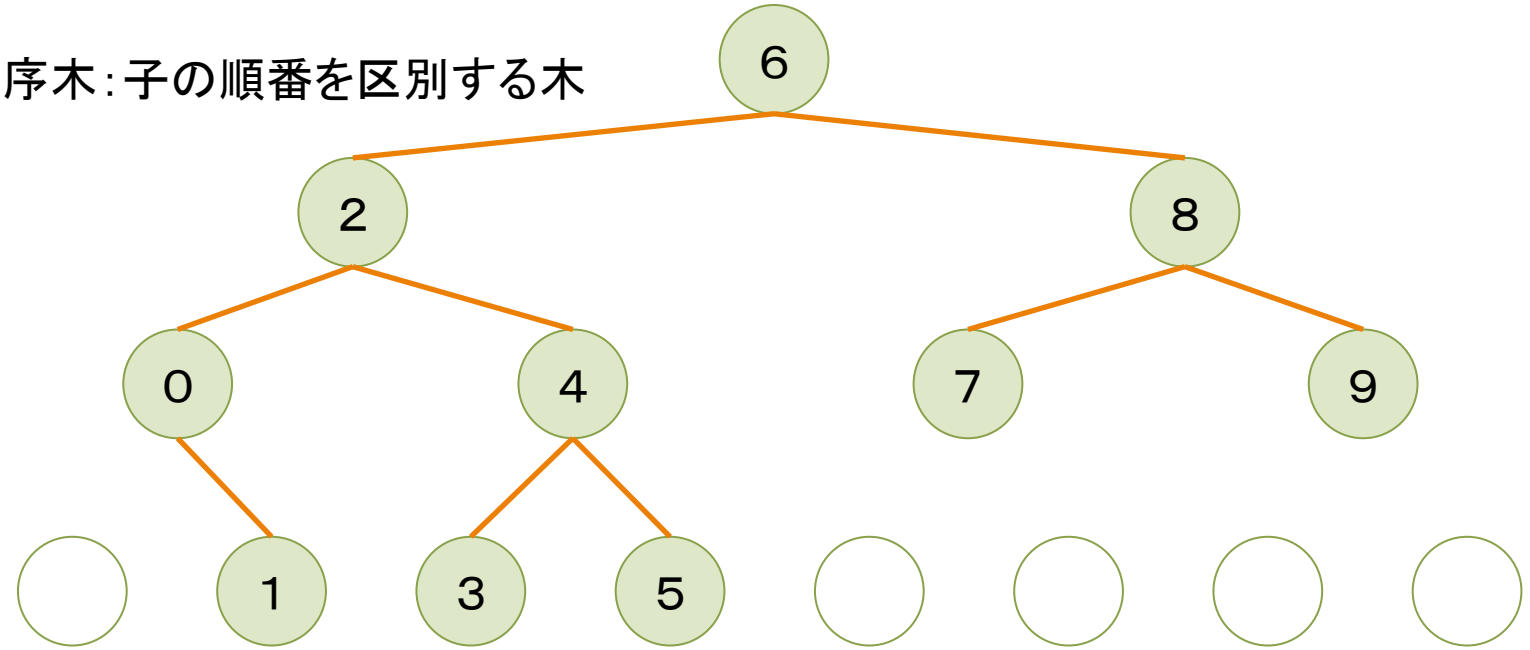
簡単のため値の重複は考えない

2分探索木の例

ヒープのように内部構造を工夫する

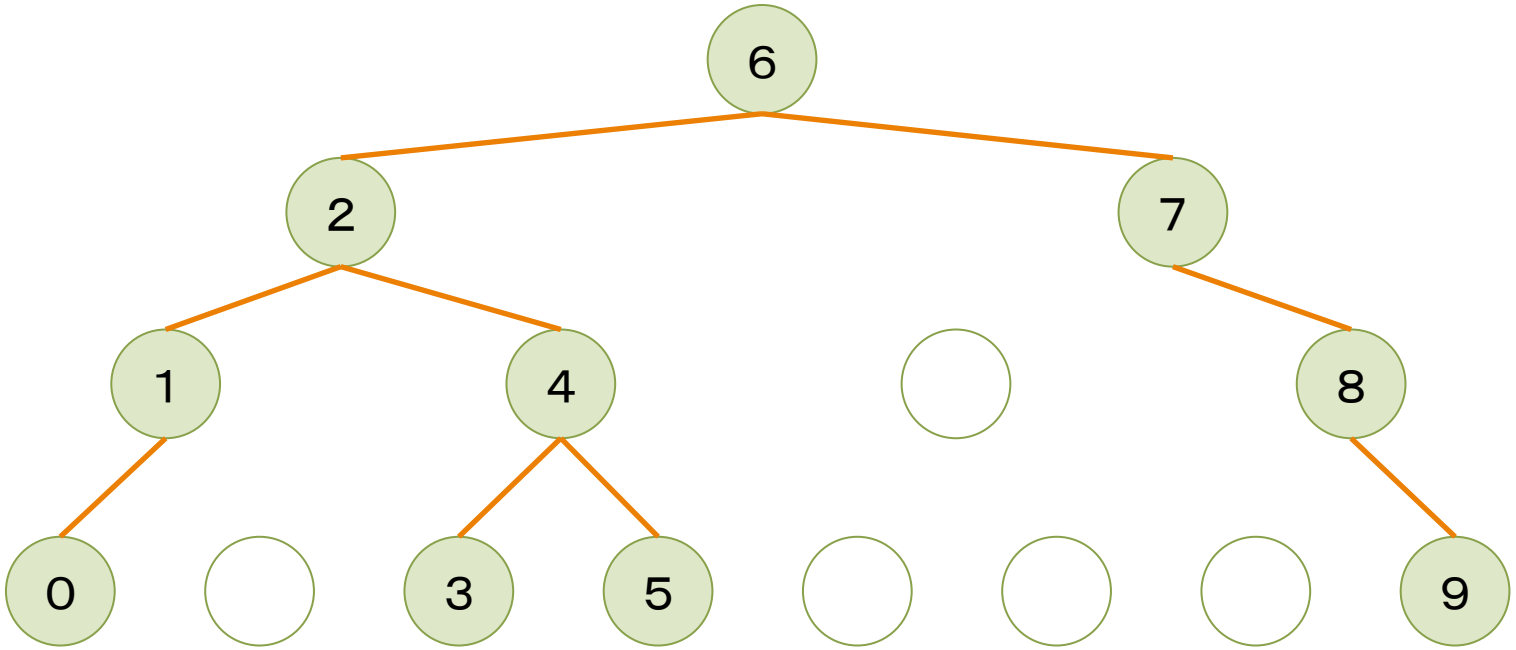
左の子孫　＜　親　＜　右の子孫

順序木：子の順番を区別する木



2分探索木つづき

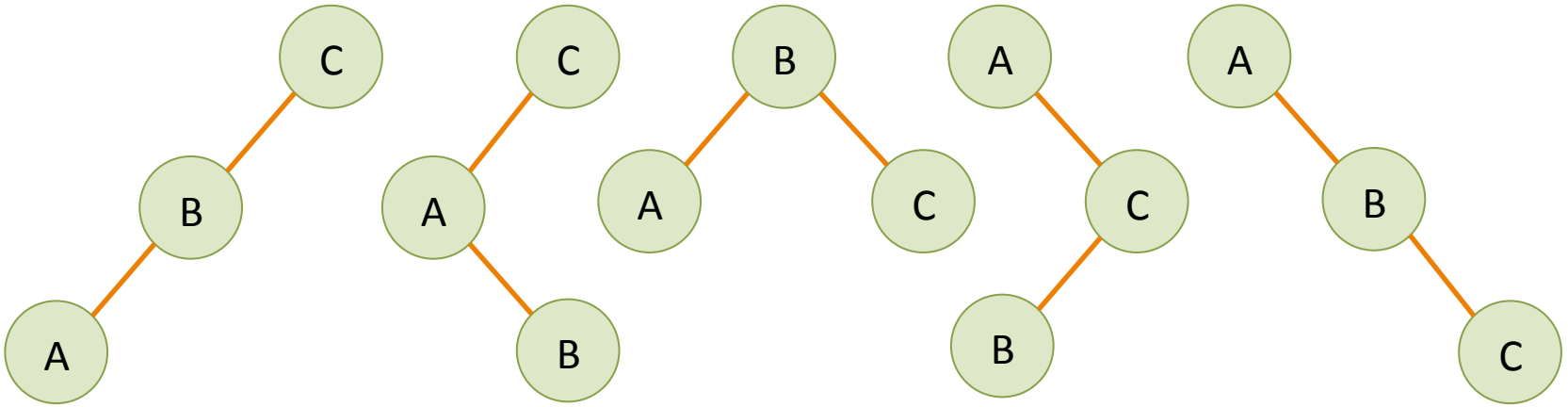
左の子孫　＜　親　≦　右の子孫



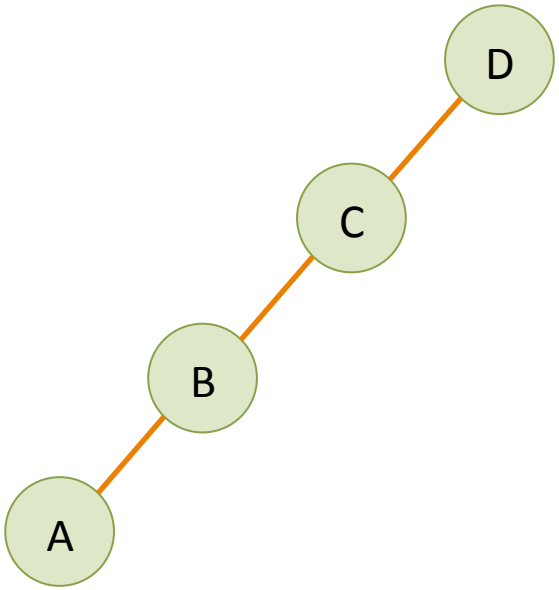
注意：　条件を満たすパターンは1つではない！

２分探索木：ノード数２

2分探索木:ノード数3



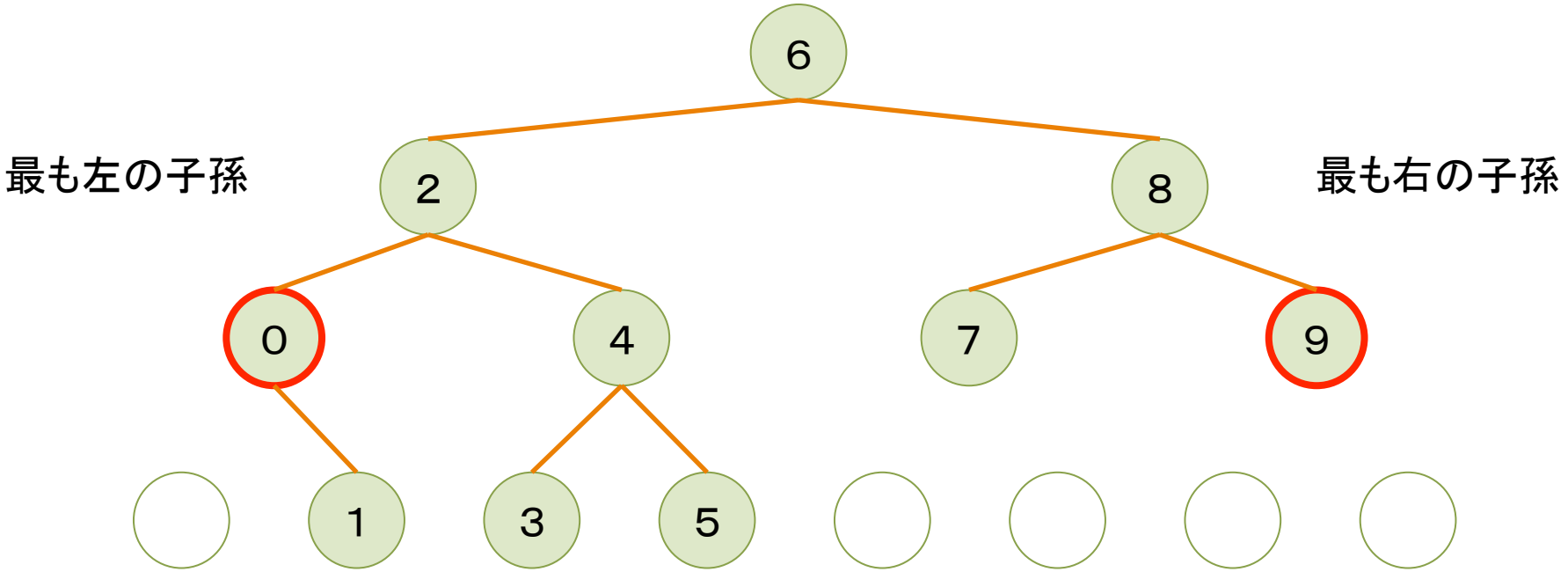
2分探索木：ノード数4



… ? …

最大値最小値

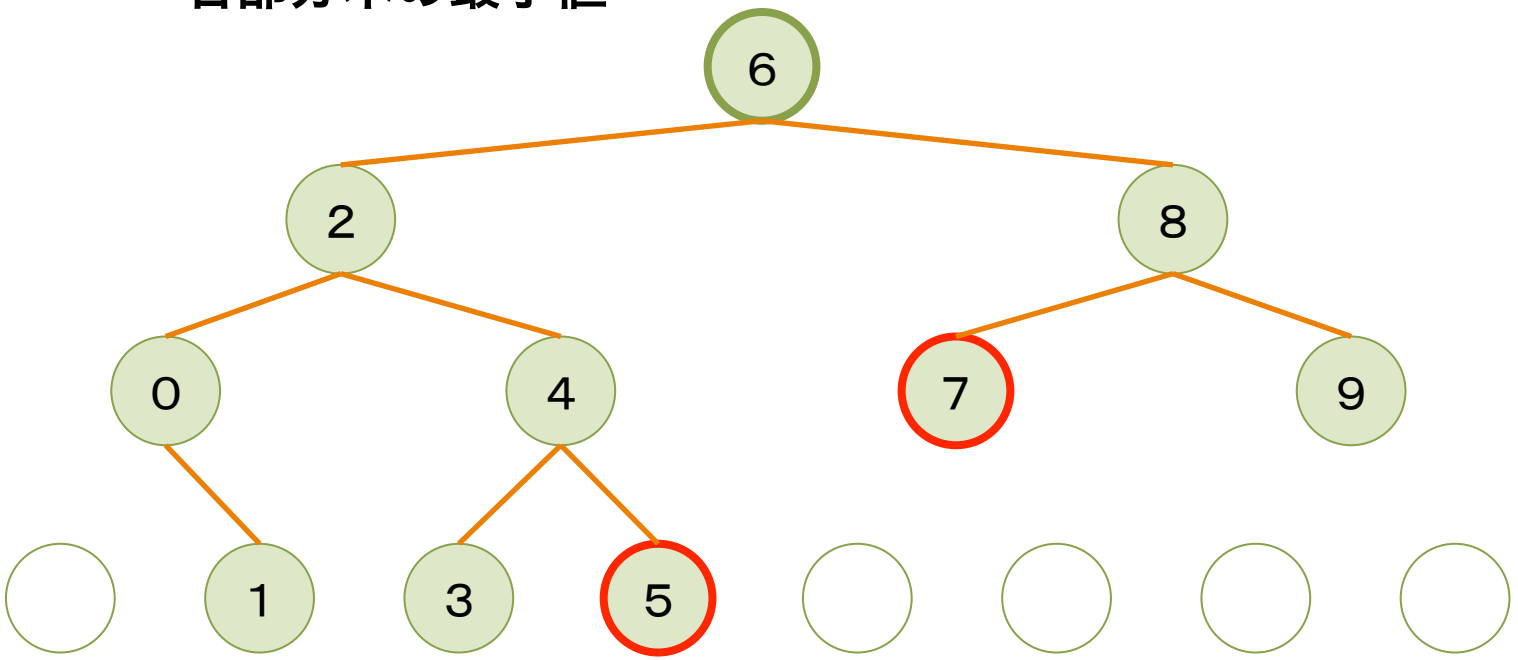
左の子孫　＜　親　≦　右の子孫



左部分木の最大値 ＜ 親 ≦ 右部分木の最小値

親の値に隣接する値：

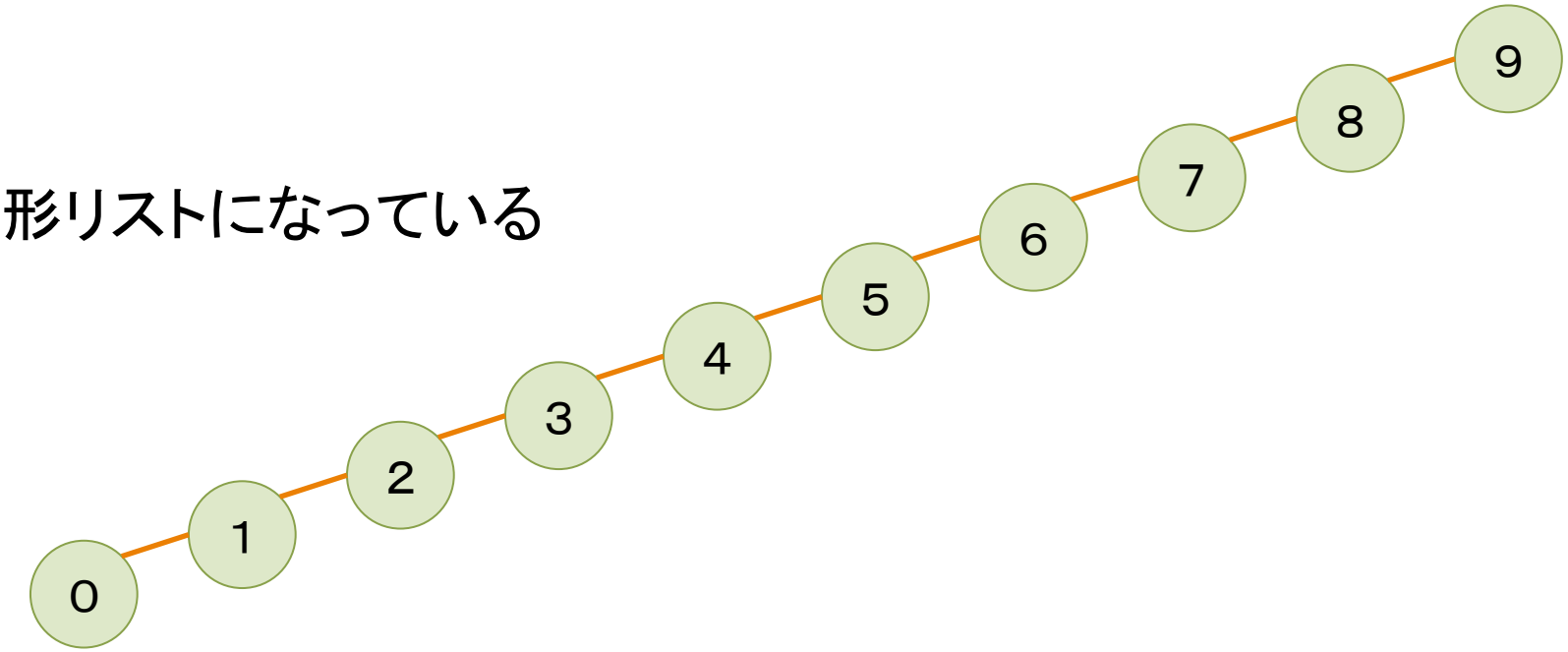
- **左部分木の最大値**
- **右部分木の最小値**



最悪の2分探索木

左の子孫　＜　親　≦　右の子孫

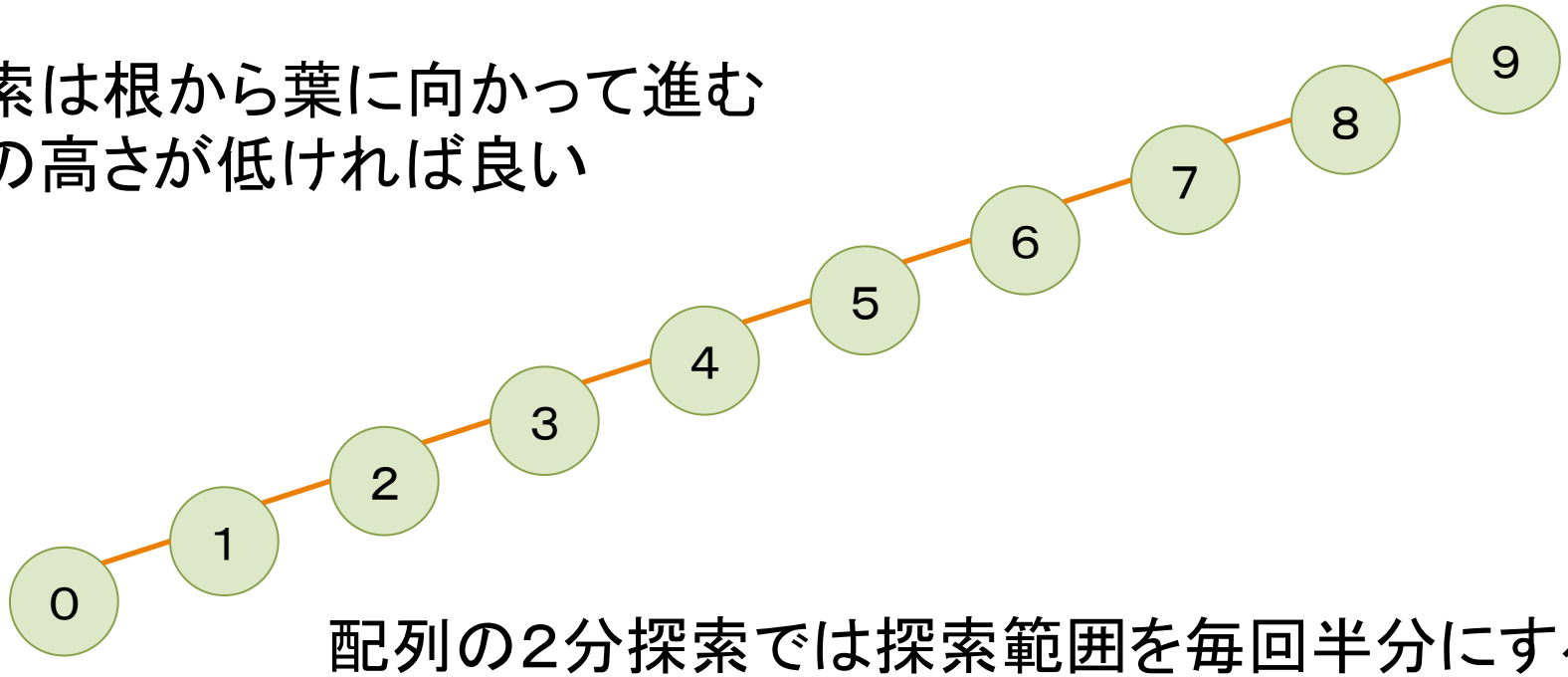
線形リストになっている



良い2分探索木の条件

探索の効率を維持する

探索は根から葉に向かって進む
葉の高さが低ければ良い



配列の2分探索では探索範囲を毎回半分にする
各節で左右の部分木が同じ大きさなら都合が良い

平衡木

- 左右の部分木の節の個数が高々１個しか異ならない（完全平衡）
- 全ての節で左右の部分木の高さの差が１以内（平衡）
  - AVL木
  - ２色木（赤黒木）

Adelson-Velskii-Landis Tree

AVL木

AVL木

- 全ての節で左右の部分木の高さの差が1以内となる2分木
- （次数が1の節の子は葉である）

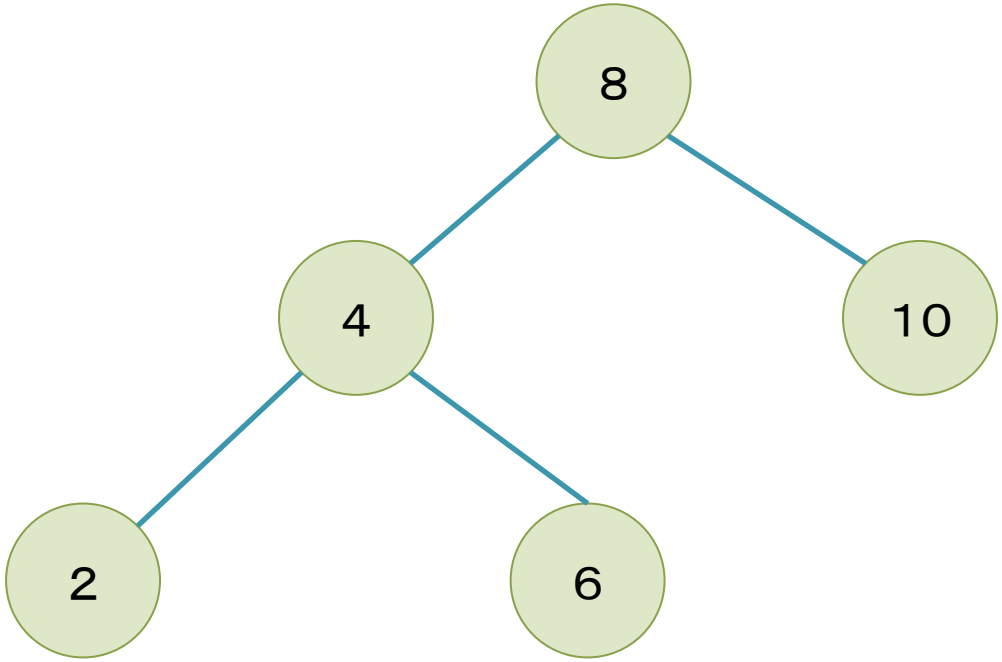
平衡の維持

- 木に対する節の挿入･削除により平衡を失うとき平衡を回復するように操作

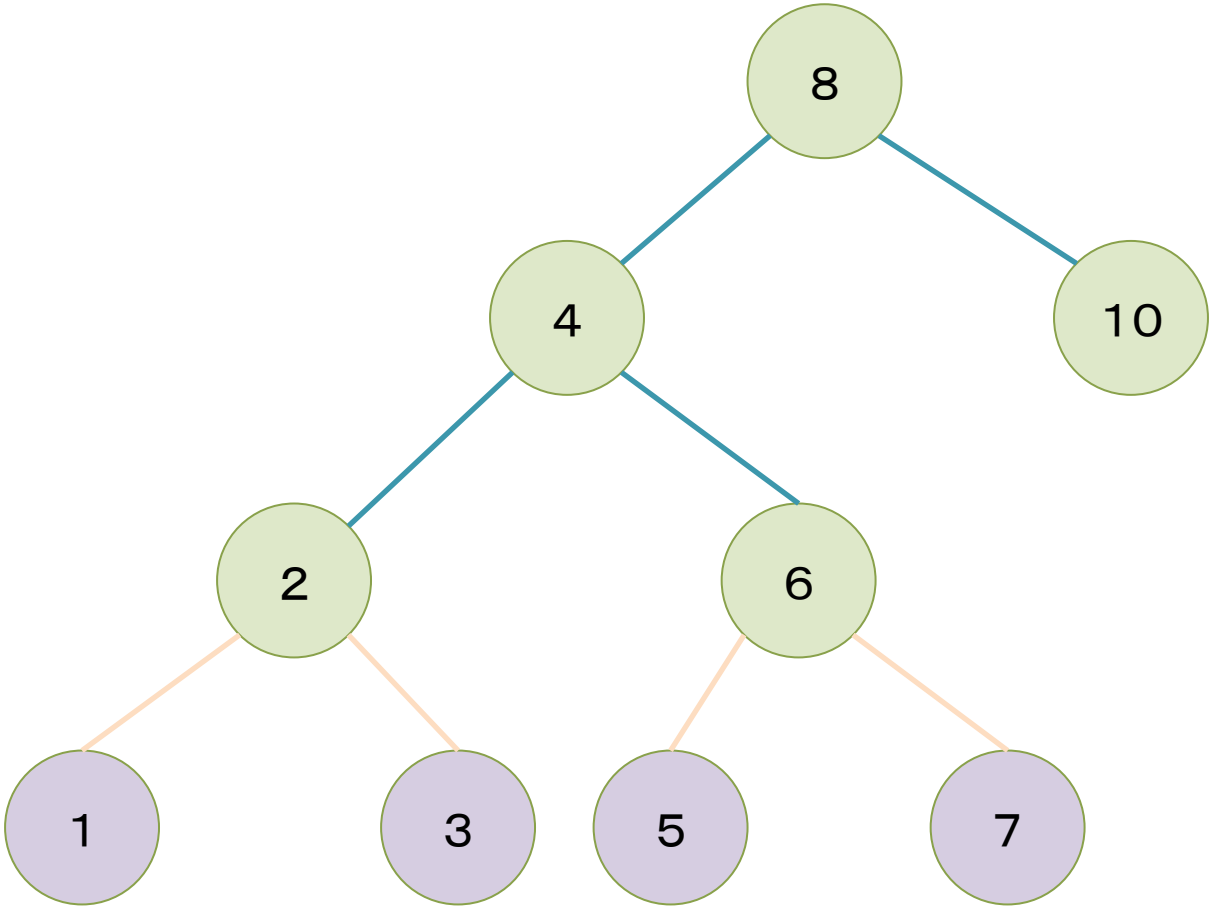
挿入

- 根（ルート）からスタート
- 挿入する値を節（ノード）の値と比較
  - 大きいなら
    - 子孫があれば右部分木に挿入
    - 子がなければ右の子とする
  - 小さいなら
    - 子孫があれば左部分木に挿入
    - 子がなければ左の子とする

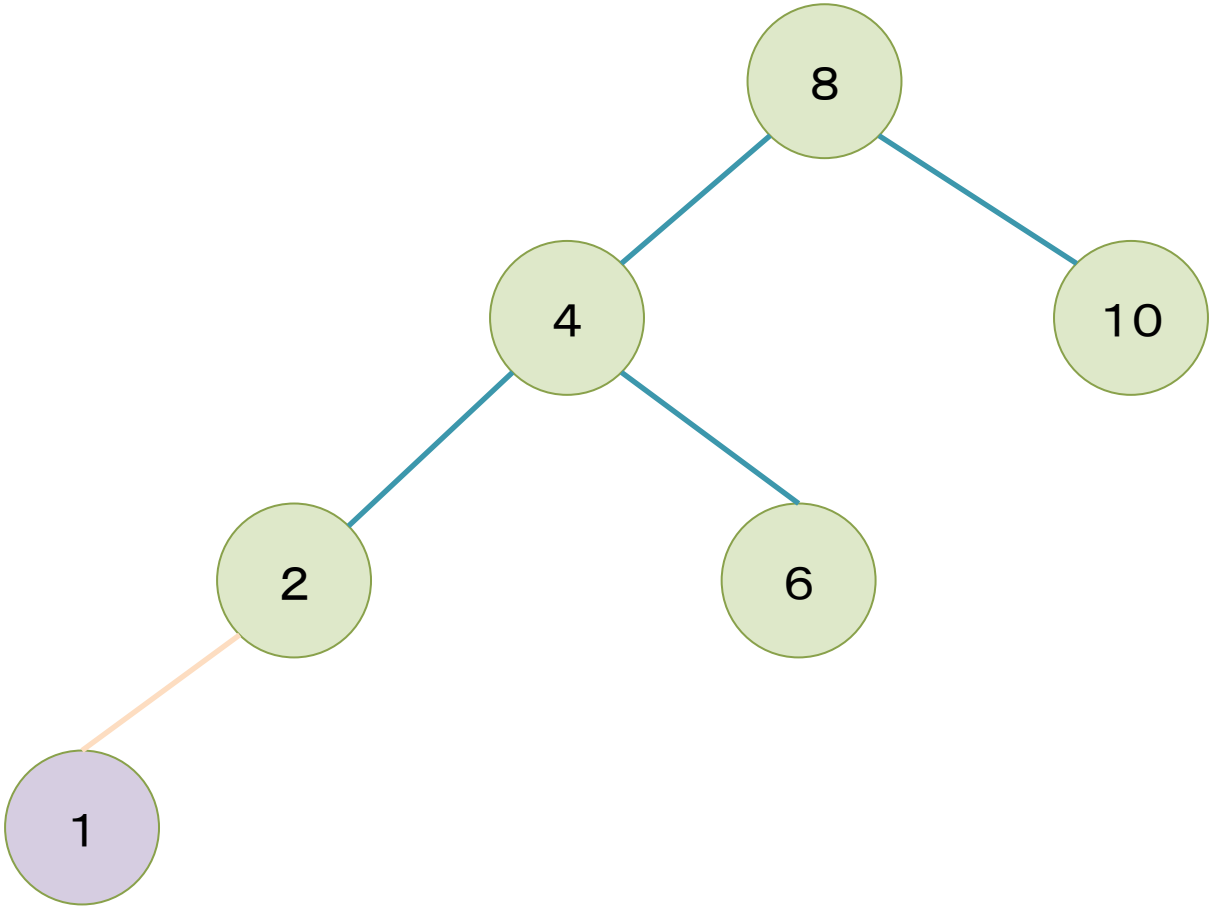
平衡な状態の木（挿入前）



平衡を崩す挿入

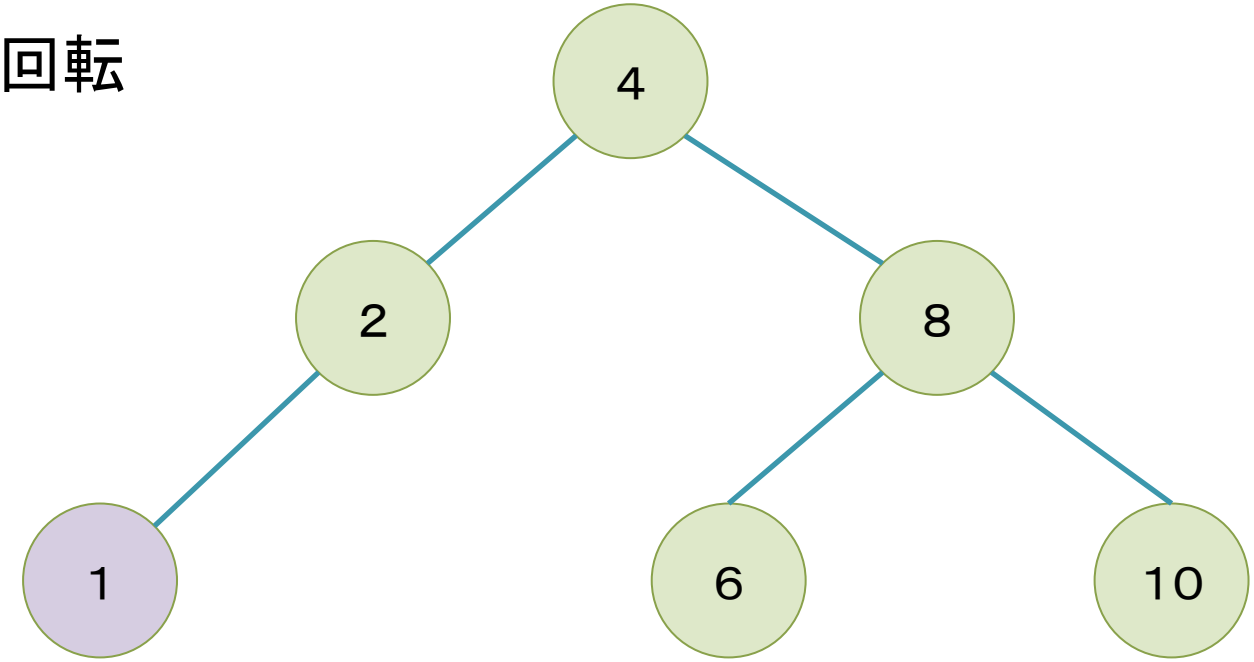


平衡を崩す挿入パターン1

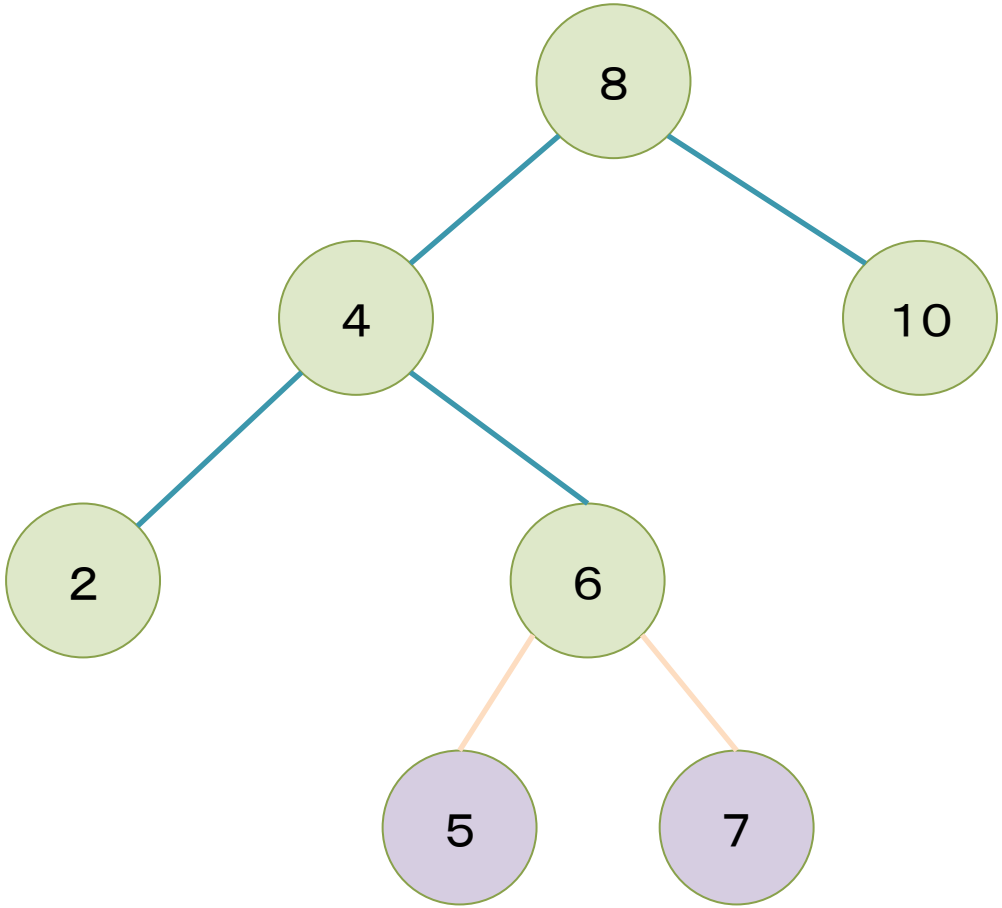


パターン1の挿入後

一重の回転

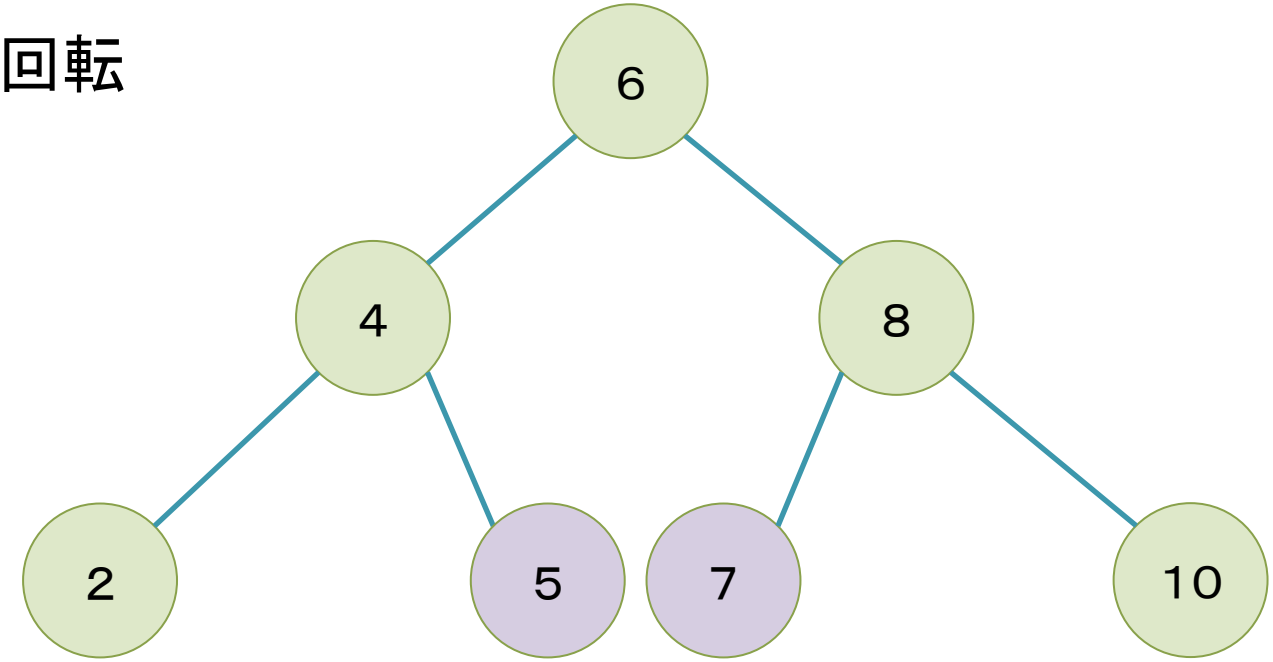


平衡を崩す挿入パターン2



パターン2の挿入後

二重の回転



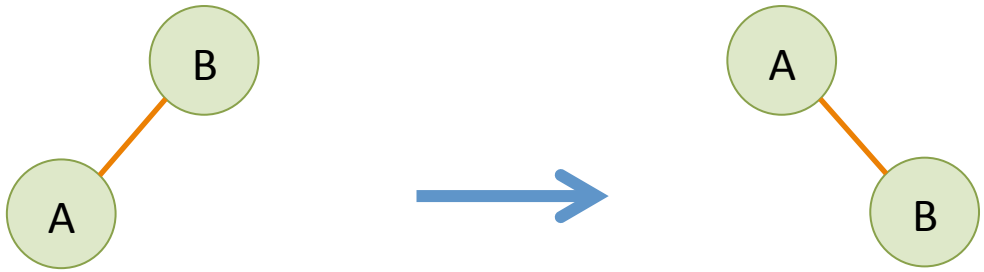
削除

- 根から対象となる節または葉を探索
- 対象の節を削除後に平衡を回復操作
  - 対象となる節から根へ戻りながら各節の部分木の高さを比較
  - 差が2以上なら回復操作
    1. 一重回転
    2. 二重回転

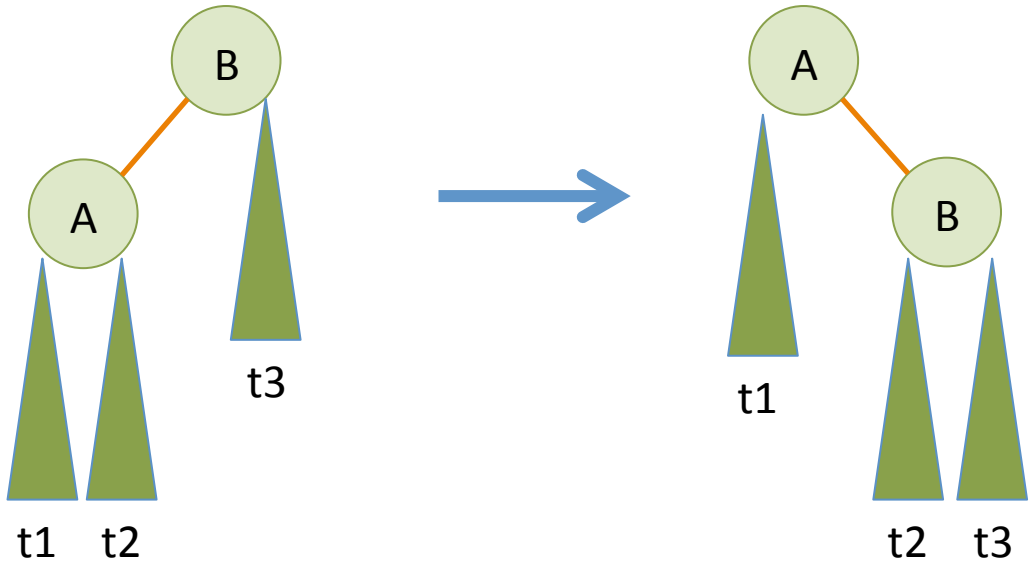
削除

- 削除対象が:
  - 葉ならそのまま削除
  - 子を持つ(その先の子孫が無い)場合
    - 右または左の子で削除対象の節を置き換え
  - 子孫を持つ場合
    - 右部分木の最小値，または左部分木の最大値で置き換え
      - 最小値が右部分木を，最大値が左部分木を持つ場合
      - 最小値，最大値を各部分木の根で置き換え

１重の回転

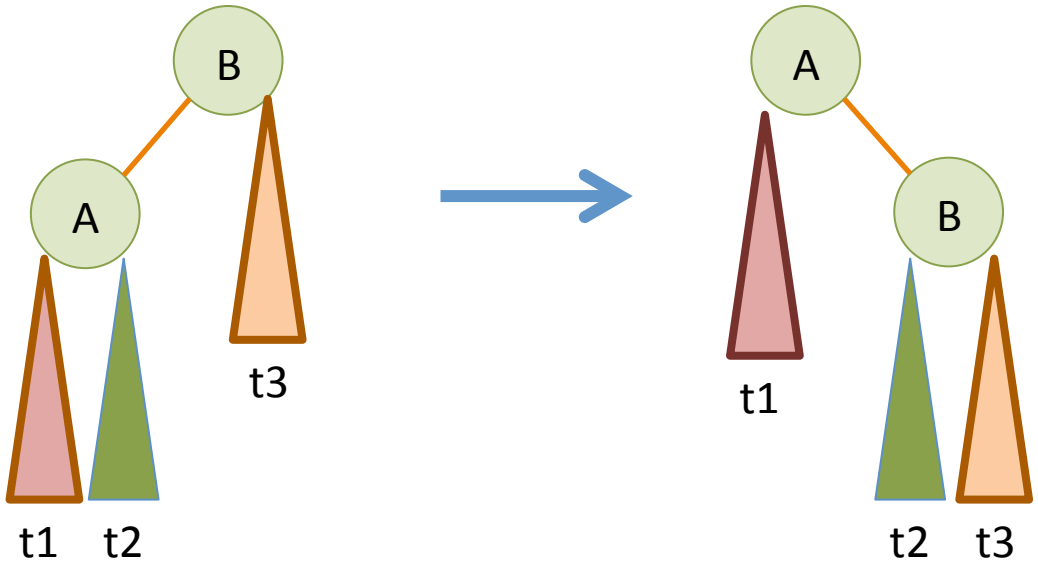


１重の回転

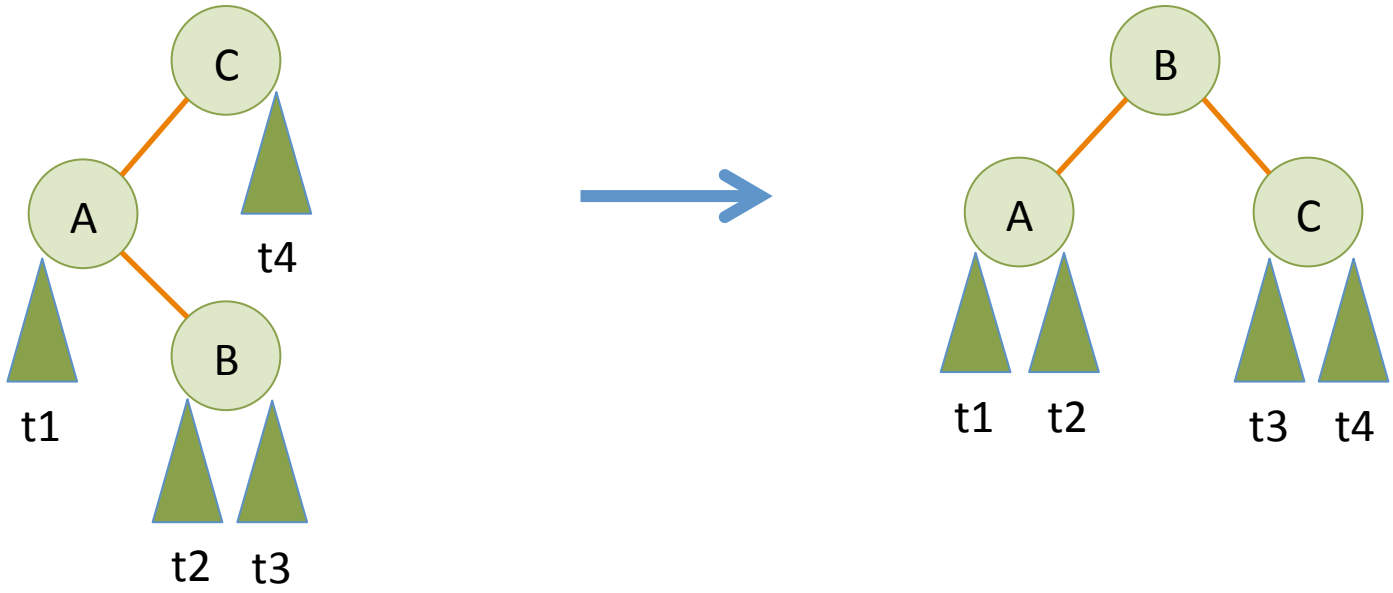


１重の回転の特徴

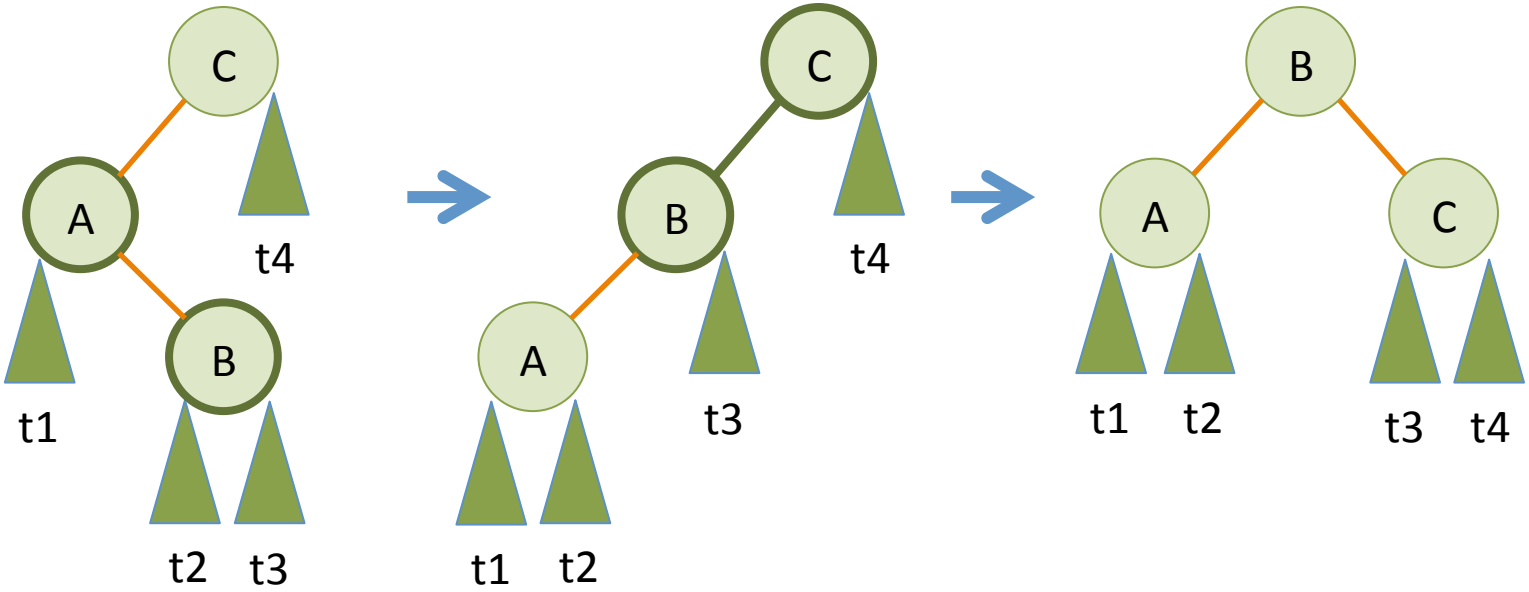
t1の高さが１減り，t3の高さが１増える
t2は左部分木から右部分木に移動する。高さは変化しない



２重の回転

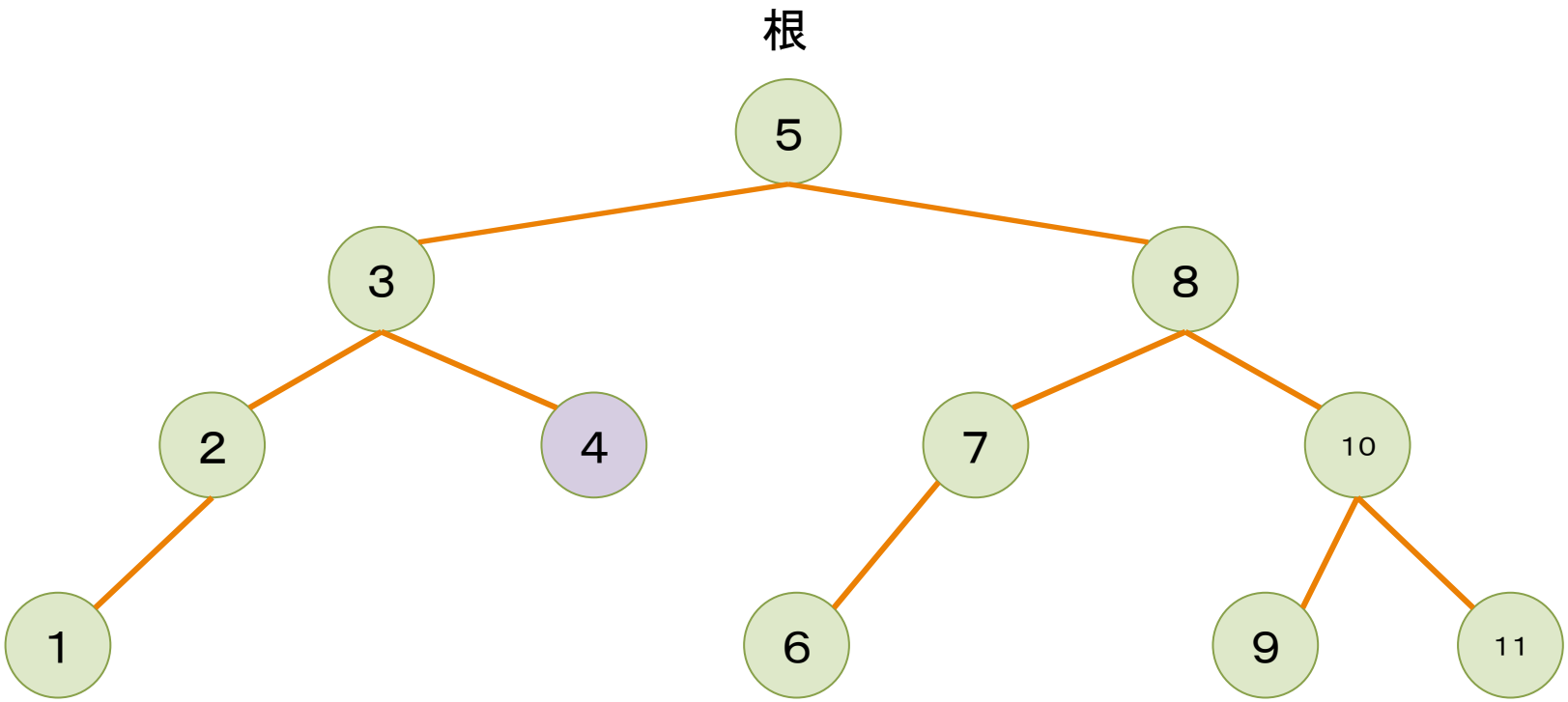


2重の回転



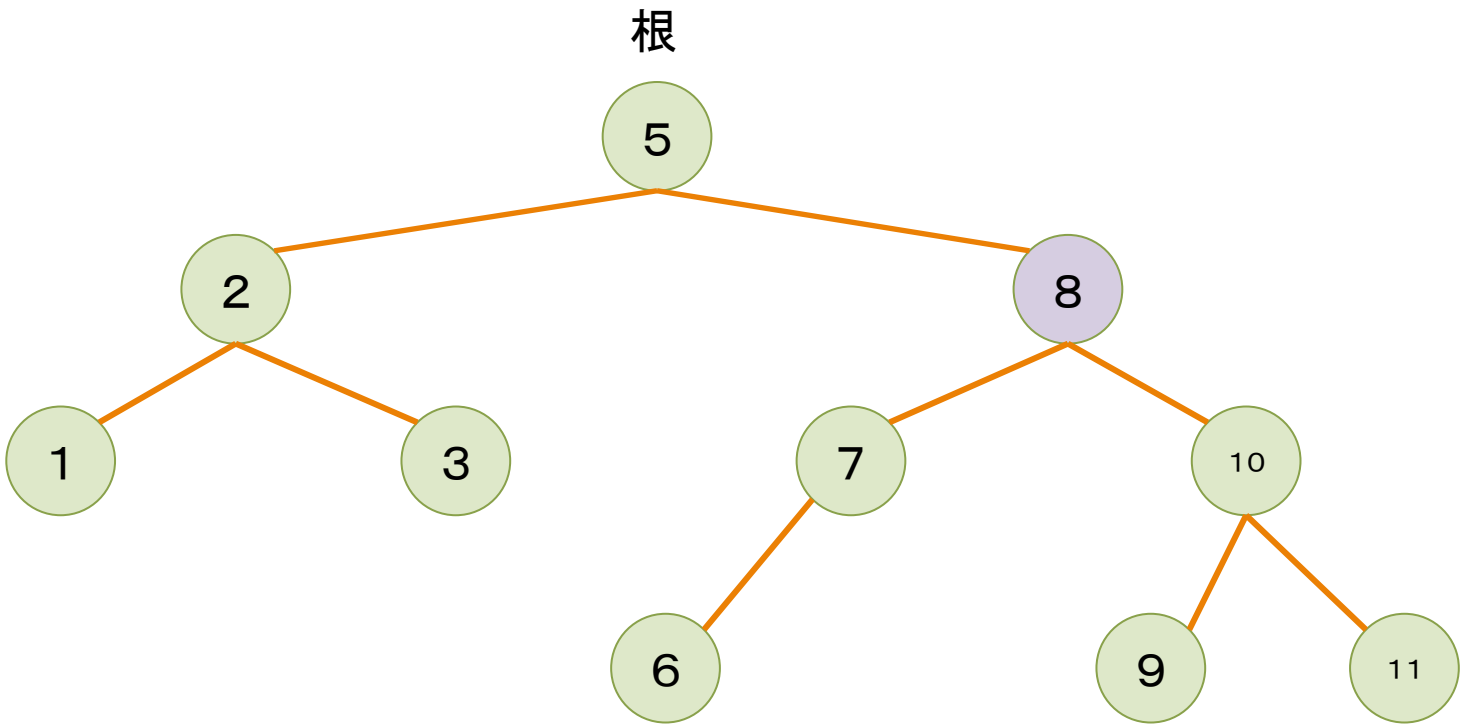
削除の例

ヴィルト著「アルゴリズムとデータ構造」より



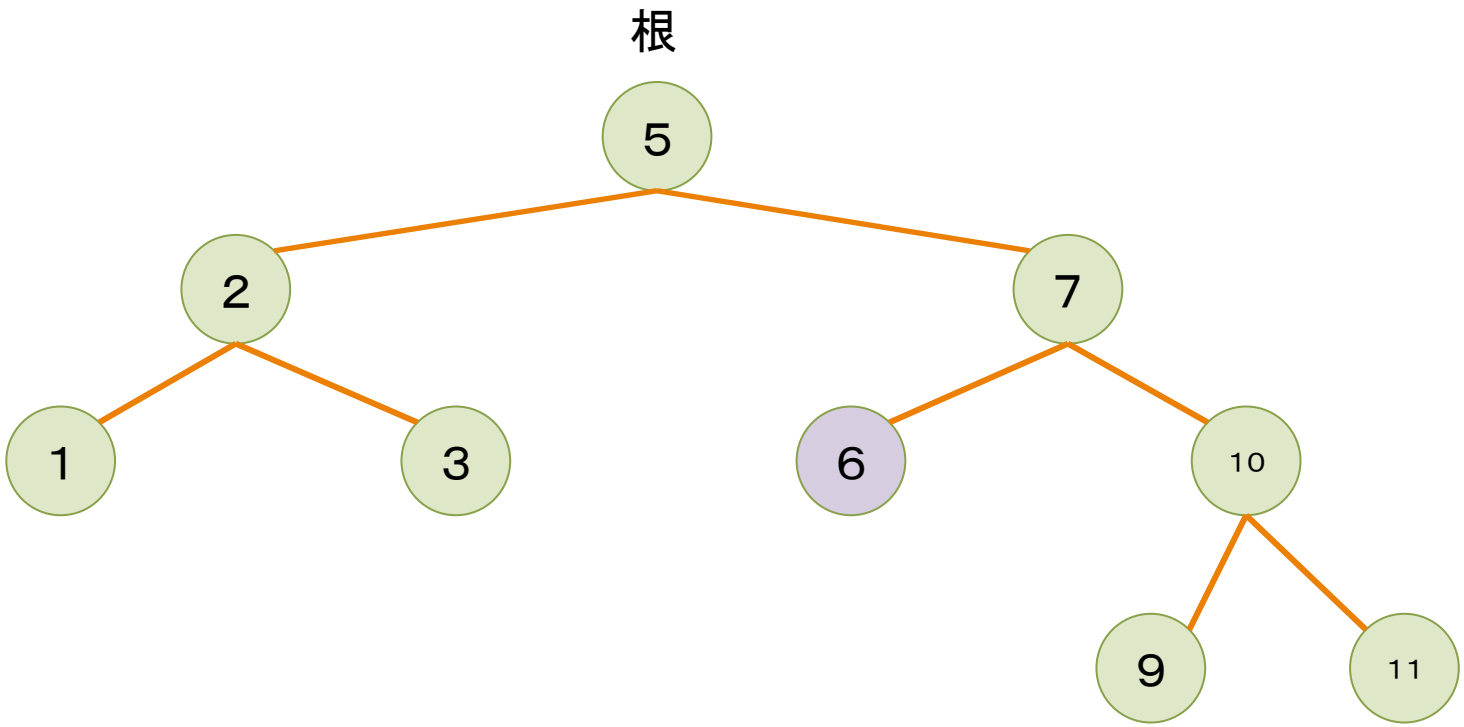
削除の例

ヴィルト著「アルゴリズムとデータ構造」より



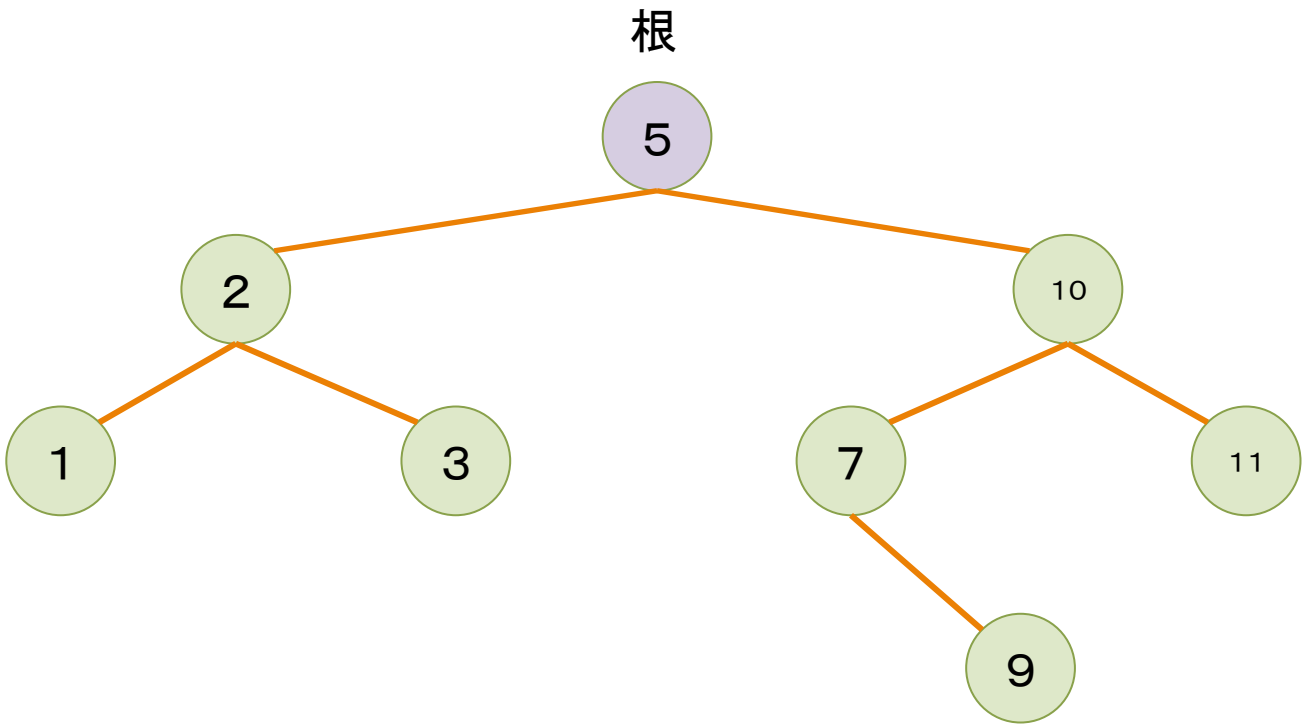
削除の例

ヴィルト著「アルゴリズムとデータ構造」より



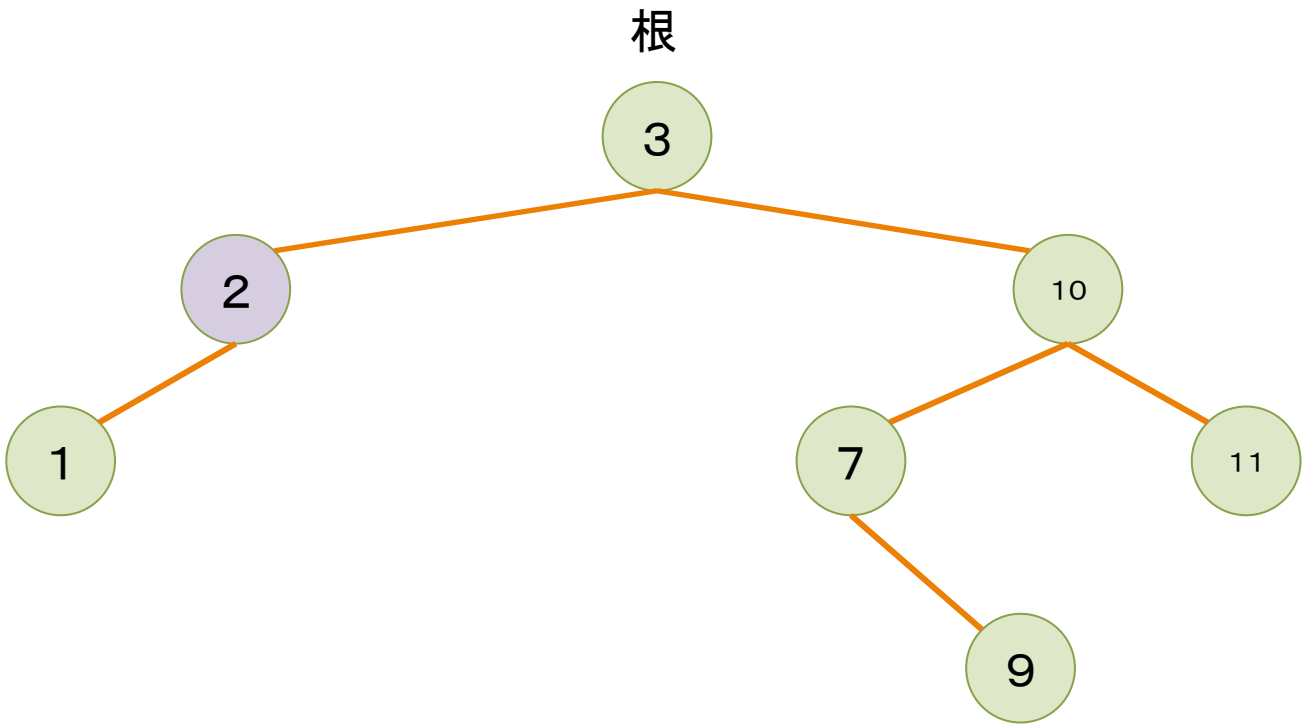
削除の例

ヴィルト著「アルゴリズムとデータ構造」より



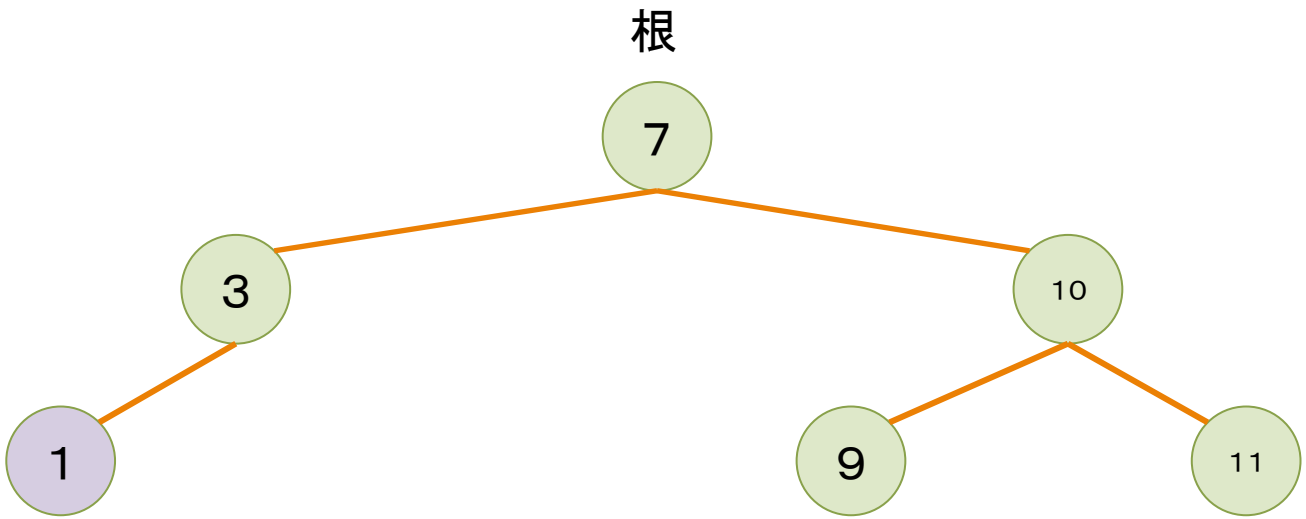
削除の例

ヴィルト著「アルゴリズムとデータ構造」より



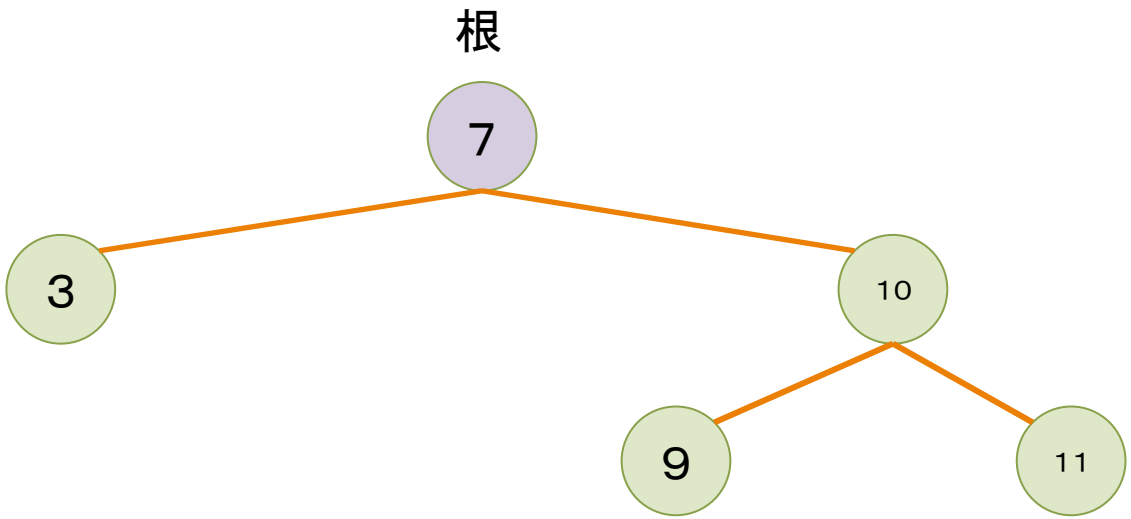
削除の例

ヴィルト著「アルゴリズムとデータ構造」より



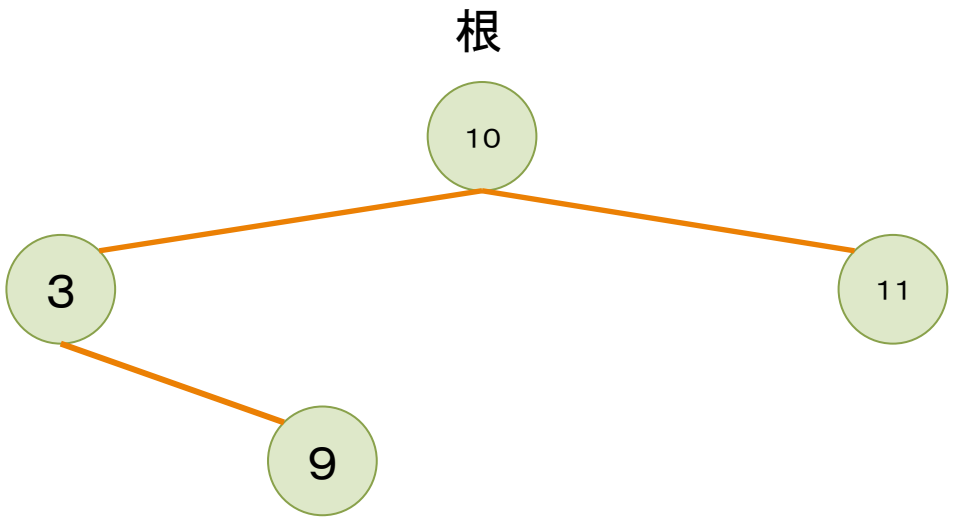
削除の例

ヴィルト著「アルゴリズムとデータ構造」より



削除の例

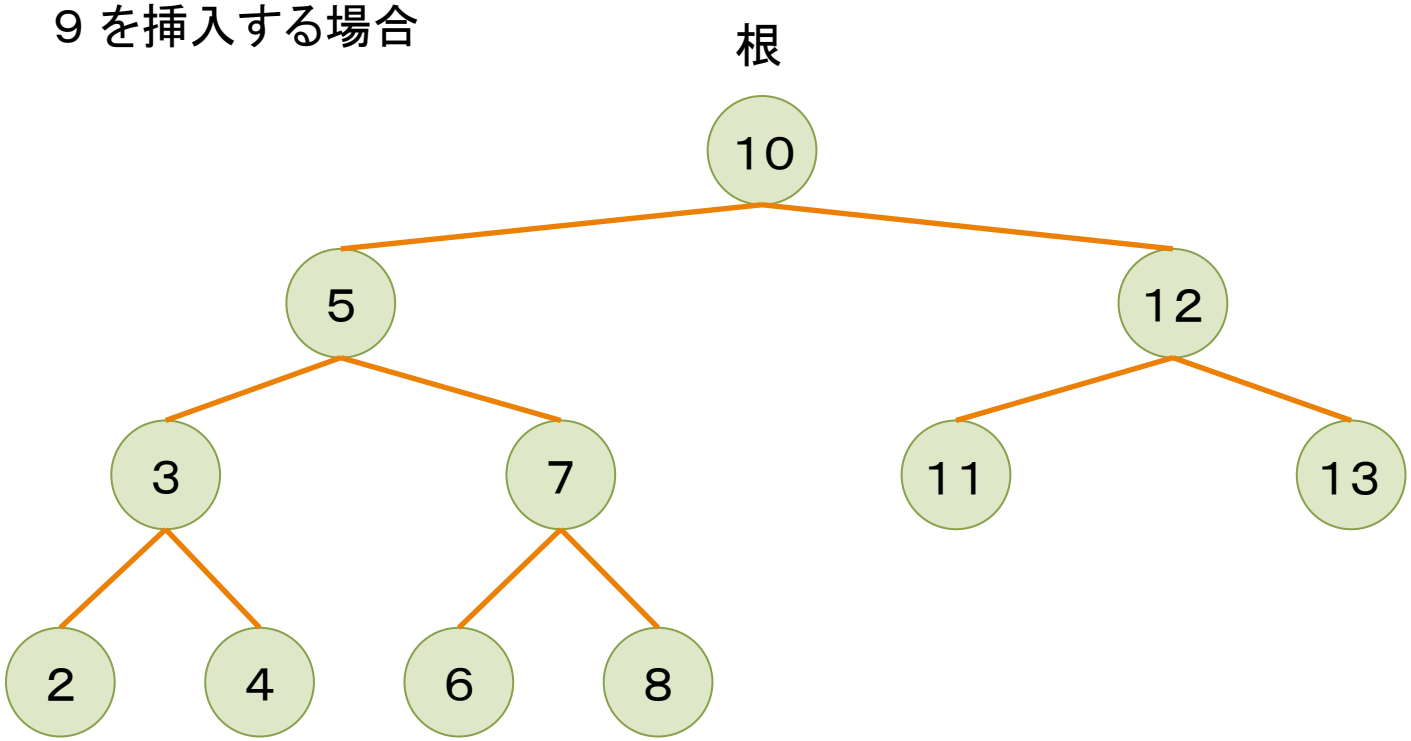
ヴィルト著「アルゴリズムとデータ構造」より



練習問題

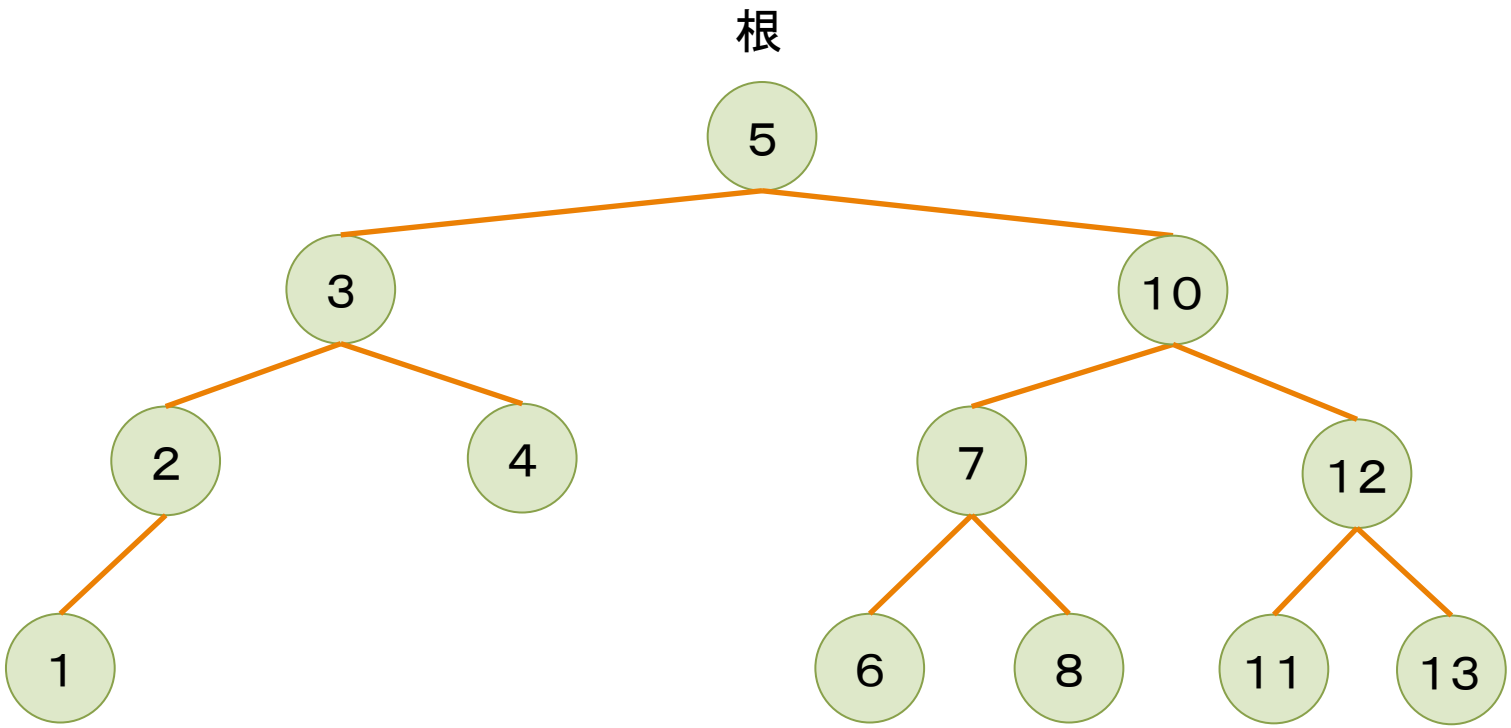
このAVL木に次の操作を行うとそれぞれ結果は？

- １を挿入する場合
- ９を挿入する場合



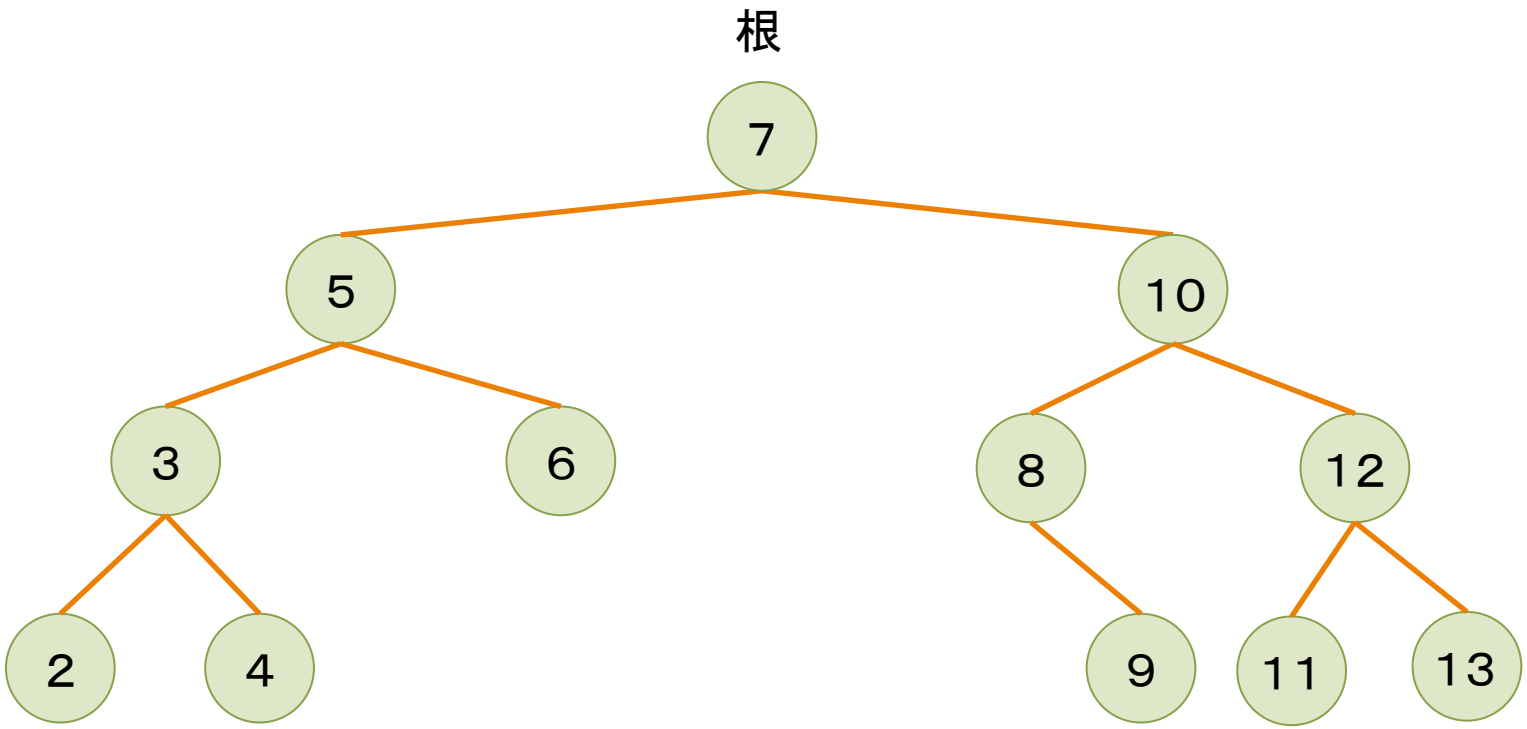
解答

- 1 を挿入



解答2

- ９を挿入

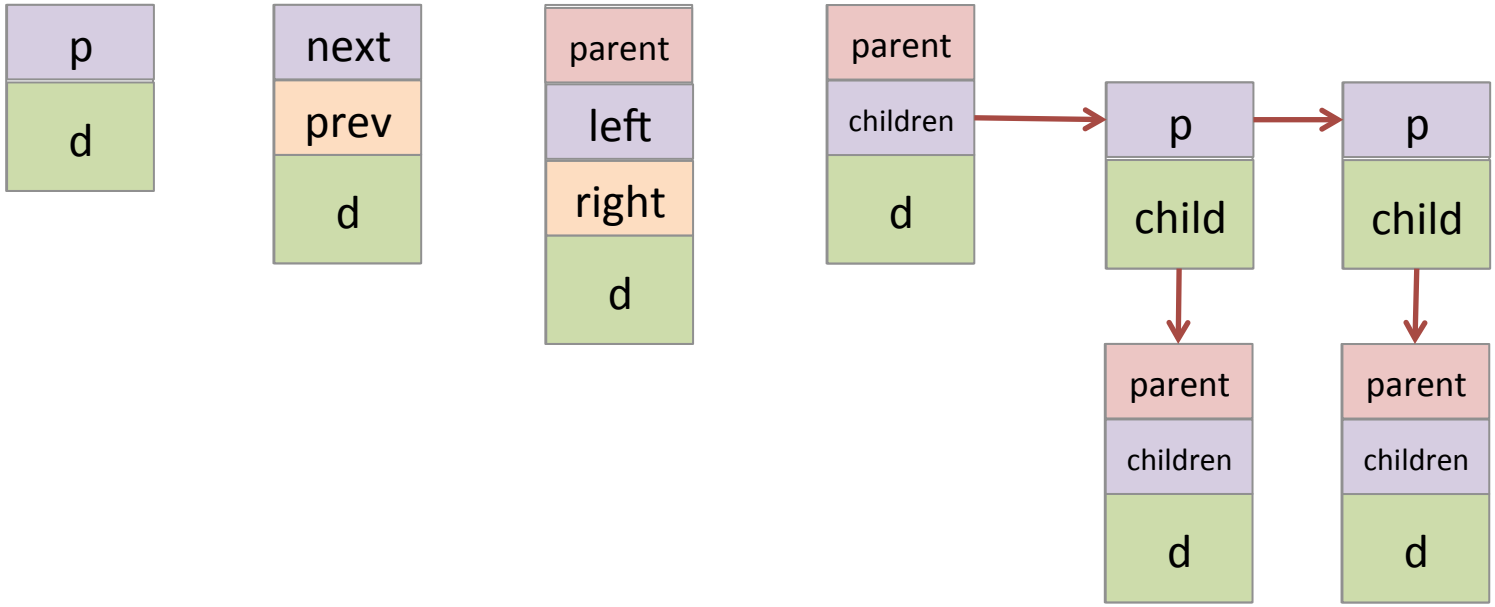


巡回

- Traverse
  - すべての節と葉に処理をする
    - 出力
    - 初期値設定
    - リセット
  - ２分木では三種類
    1. 行きがけ順：　節，左部分木，右部分木
    2. 通りがけ順：　左部分木，節，右部分木
    3. 帰りがけ順：　左部分木，右部分木，節

リストの応用と発展

- スタックとキュー
- セルの拡張



ポインタ変数

```
// int型の変数
int x = 3;

// int型へのポインタ変数
int *y;

// アドレスを代入
y = &x;

printf(“%d\n”, *y);
```

変数名	型	アドレス	内容
x	int	3689	3
y	pointer to int	6549	3689

表記	意味
&x	変数 x のアドレス
*y	ポインタ y の指すアドレスの内容

構造体へのポインタ変数

```
struct node {
    int data;
    struct node *next;
}
struct node p, *q;
p.data = 3;
p.next = NULL;
q = &p;
printf(“%d\n”, q -> data);
```

NULL

ポインタが何も指していないいいない状態

構造体のメンバへのアクセス

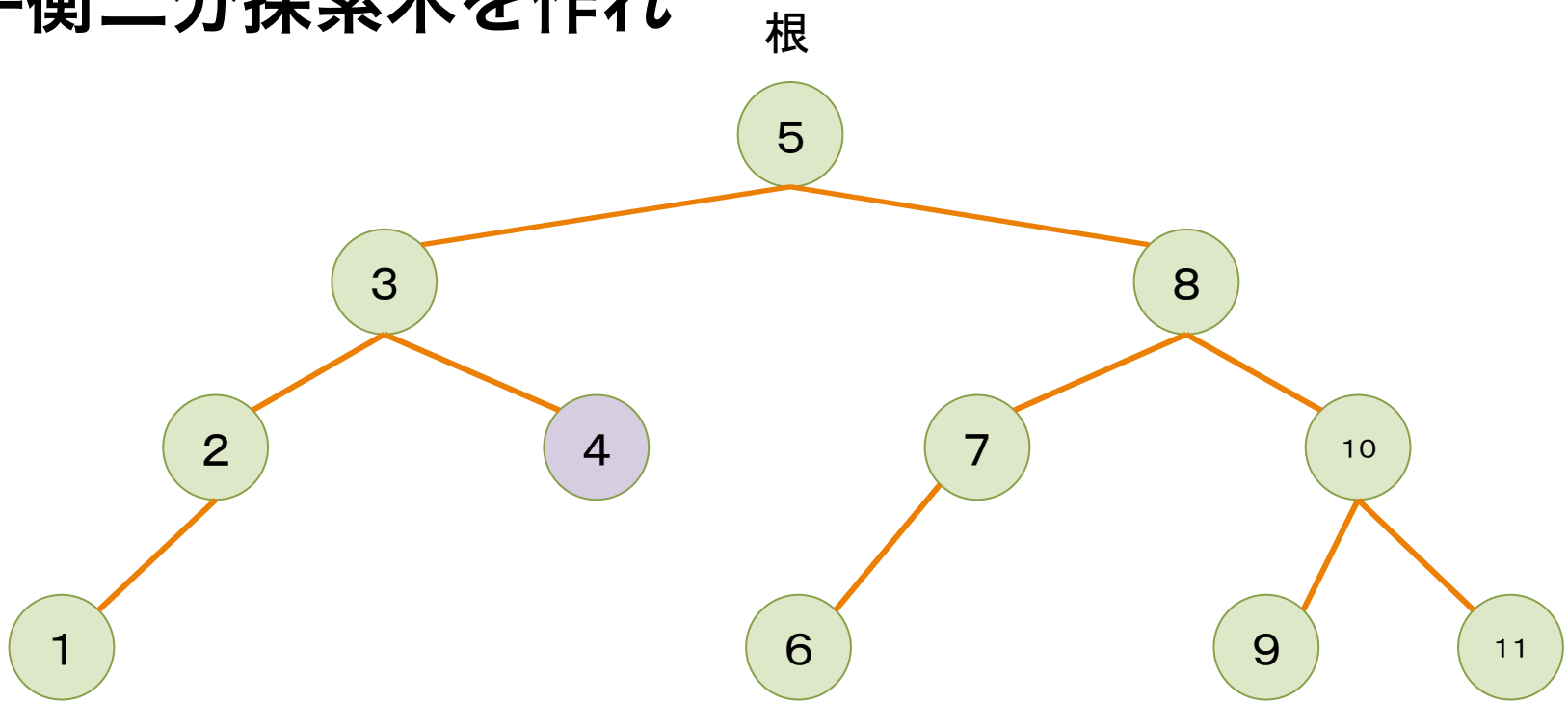
ドット表記（.）

ポインタが指す構造体のメンバへのアクセス

->

やってみよう：作業

2014598367 の順に与えられるデータに対して平衡二分探索木を作れ



4865217 の順に平衡を保ちながらノードを削除せよ

宿題：ex08

- 線形リストに要素を追加する以下の関数を作成してください。
  - リストの先頭に要素を追加する関数
  - リストの末尾に要素を追加する関数
  - リストの先頭からi番目に要素を追加する関数
- これらの関数を利用して線形リストを作成し、その内容を出力するプログラムを作成してください
- ねらい
  - 引き続き、ポインタ変数への理解と習熟を図る

提出についての注意

- プログラム名
  - デフォルトでは sketch_yymmdda などだが…
  - higuchi_fumito_ex08 のように氏名と宿題番号に変えること
  - higuchi_fumito_c5_ex08 (同姓同名はクラスを付加）
- プログラムの冒頭に氏名、学年、クラス、番号等をコメントとして記入
  - 参考にした資料の他、簡単な感想も付け加えてください
- Oh-o!Meijiから提出（次回の授業開始までに）

連絡先

樋口文人

wenren@meiji.ac.jp