７．木構造

- ７－１．データ構造としての木
  - グラフ理論での木の定義
  - 根付き木
- ７－２．２分探索木
- ７－３．高度な木（平衡木）
  - AVL木
  - B木

７－1.データ構造としての木

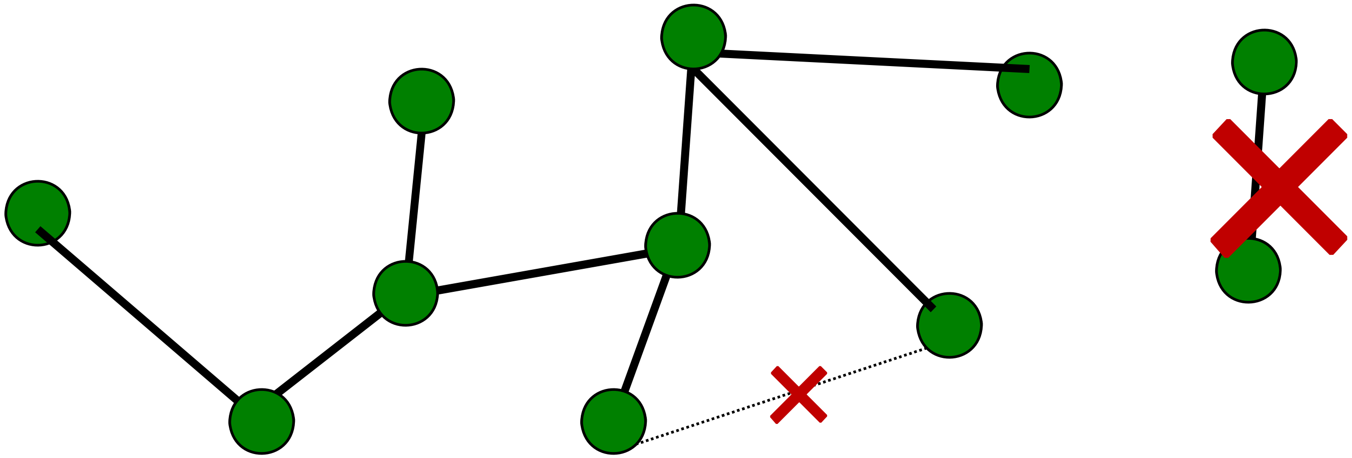
木構造

- 木構造を表すデータ構造の一つとしてヒープがある。しかし、ヒープでは、配列を用いるため、要素数で木の形状が一通りに決定してしまった。

- ここでは、再帰的なデータ構造を用いることにより、より柔軟な木構造が構築可能なことを見ていく。

グラフ理論における木

- グラフ理論では、木は以下のように定義される。



定義：（グラフ理論での）木

閉路のない連結なグラフ。

木の性質

- Ｎ点からなる木の辺数はＮ－１である。
- 木に１辺を加えると、閉路ができる。（閉路の無い連結グラフで辺数が最大である。）
- 木から１辺を削除すると、非連結になる。（木は、連結グラフで辺数が最小である。
- 任意の２点からなる道は唯一に定まる。

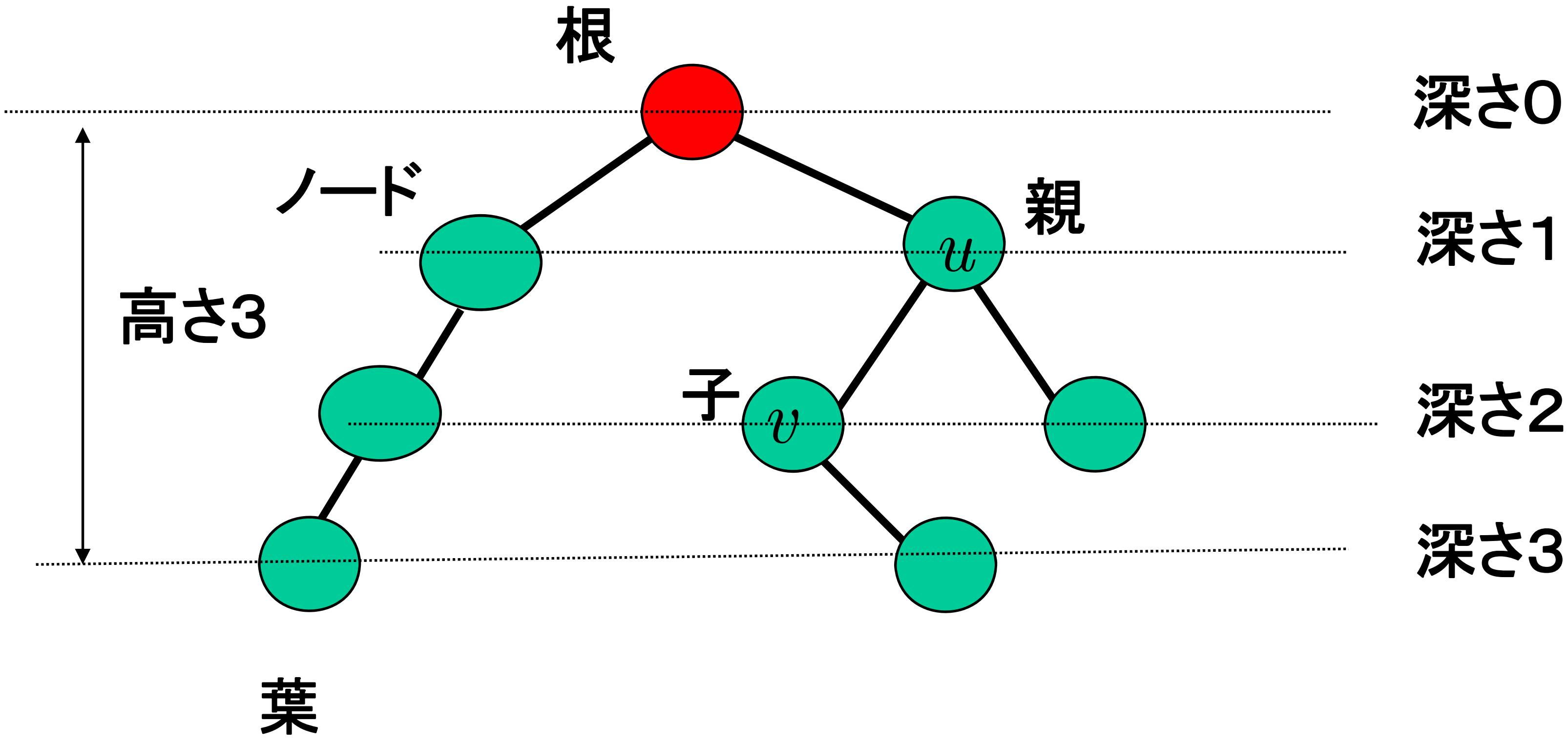
特に、最後の性質は、ファイルシステムに利用されている。

木の用語定義

- 木の各頂点をノードという。
- 木の特別な1つの頂点を根といい、根の指定された木を根付き木という。
- （根以外の）次数1の点を葉という。
- 根からの道の長さを深さという。
- 最大の道の長さを高さという。
- ある頂点vに対して、根に向かう道で、一番近い頂点をvの親という。
- 頂点vを親とする頂点wを、頂点vの子という。
- ある頂点vに対して、vの子孫からなる部分グラフを頂点vにおける部分木という。

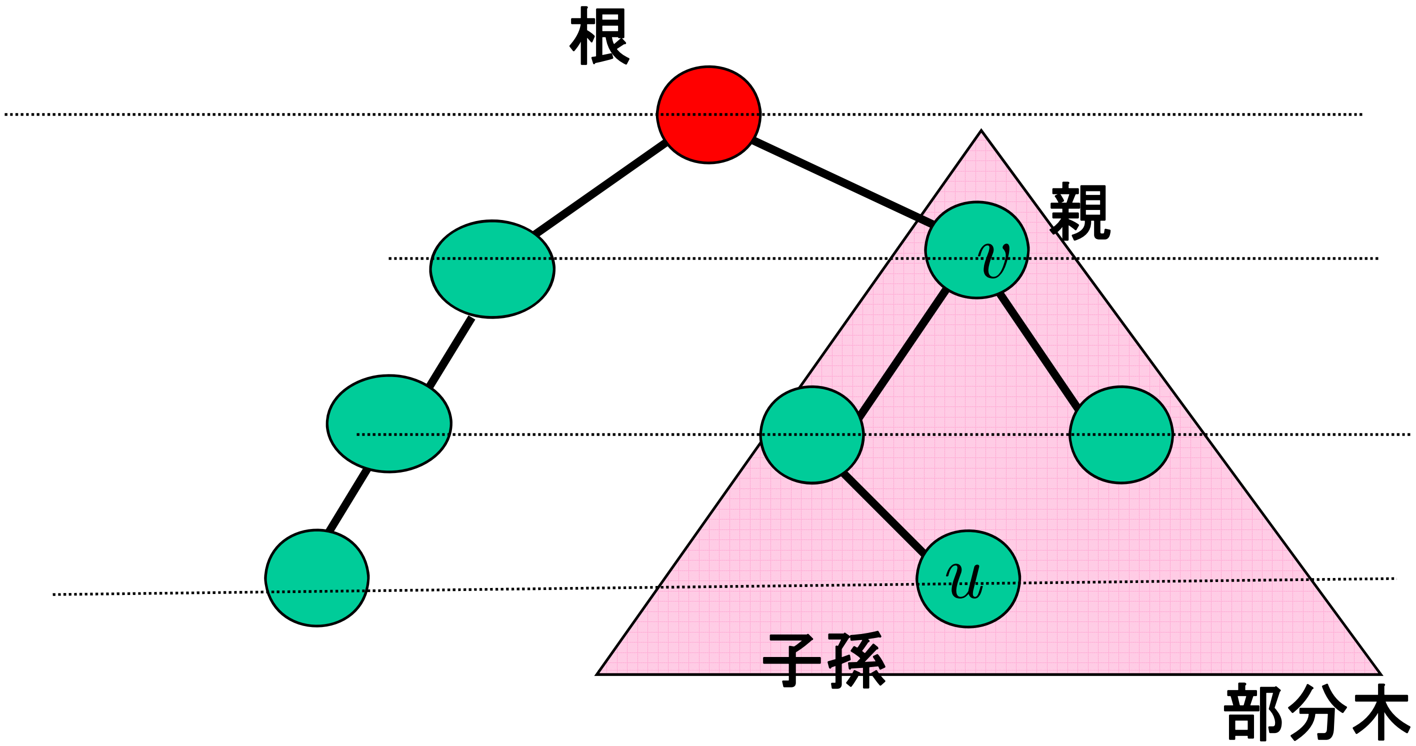
木に関する用語1

- 深さ: 根までの道の長さ
- 高さ: 木中の最大の深さ



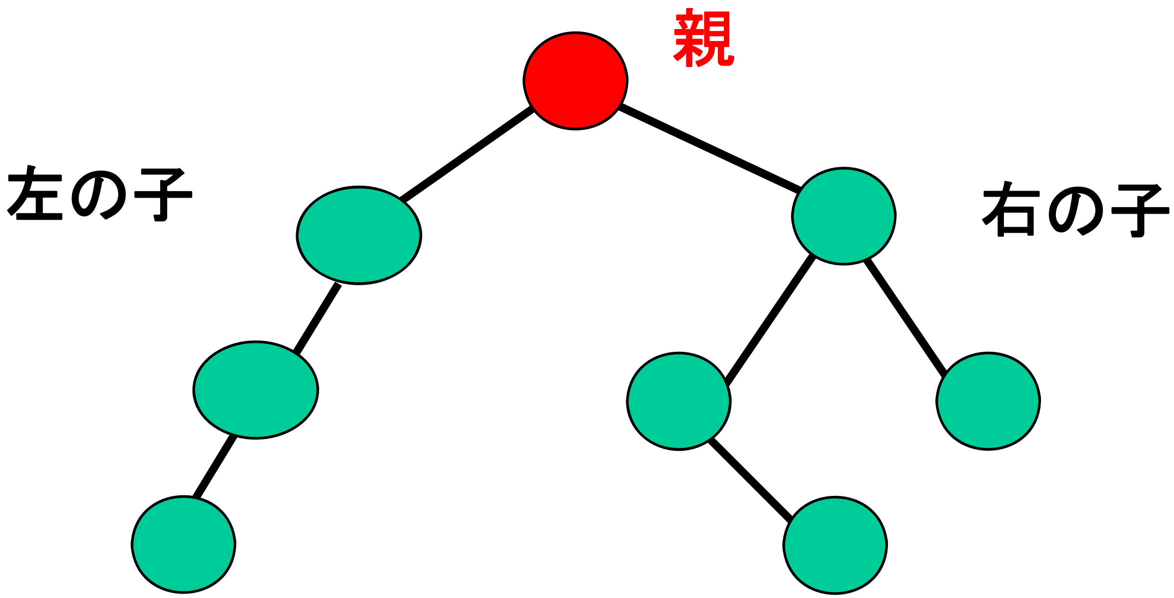
木に関する用語2

部分木:ある頂点の子孫からなる部分グラフ



２分木

- 高々２つの子しかない木。
- 左と右の子を区別する。



データ構造としての木

- ２つの子供を直接ポインタで指すようにする。
- ノードを再帰的なデータ構造として定義する。
- 葉では、子供を指すポインタ2つに対して、双方ともNULLにする。

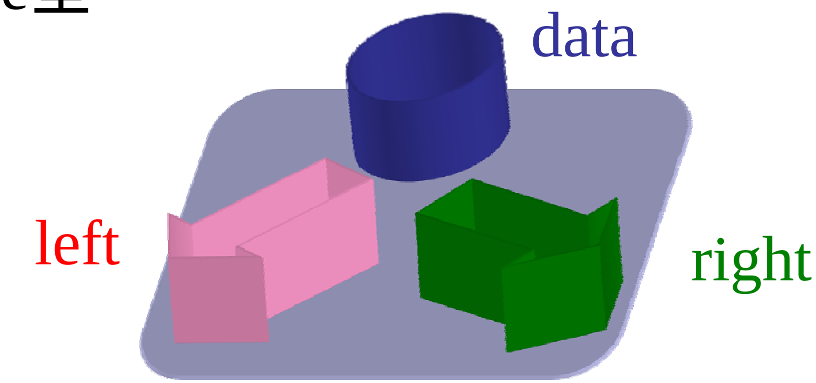
データ構造の基本単位(ノード)

- 自己参照構造体を用いる。

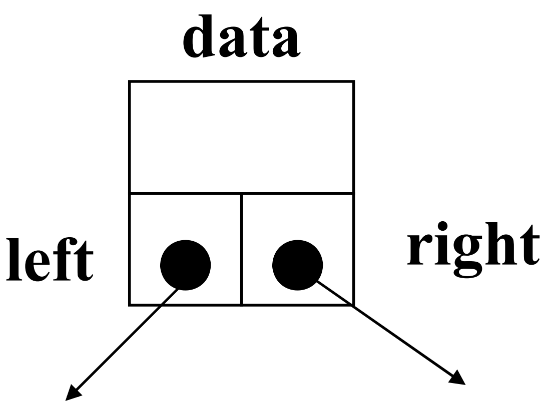
```
 struct  node
{
          double  data;
          struct node * left;/*左の子供を指す。*/
          struct node * right;/*右の子供を指す*/
} ;
```

イメージ

strcut node型

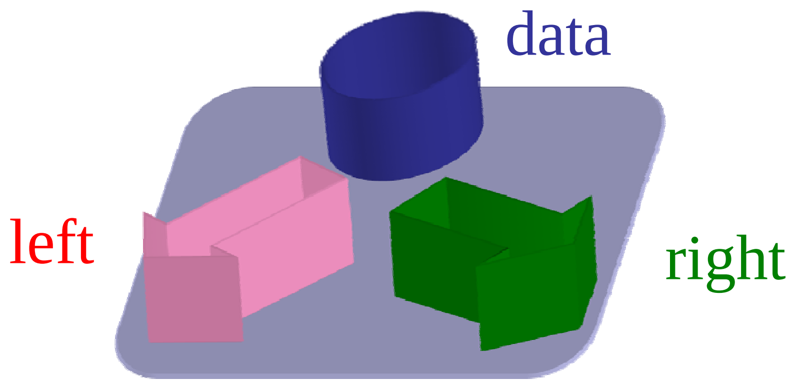


strcut node型



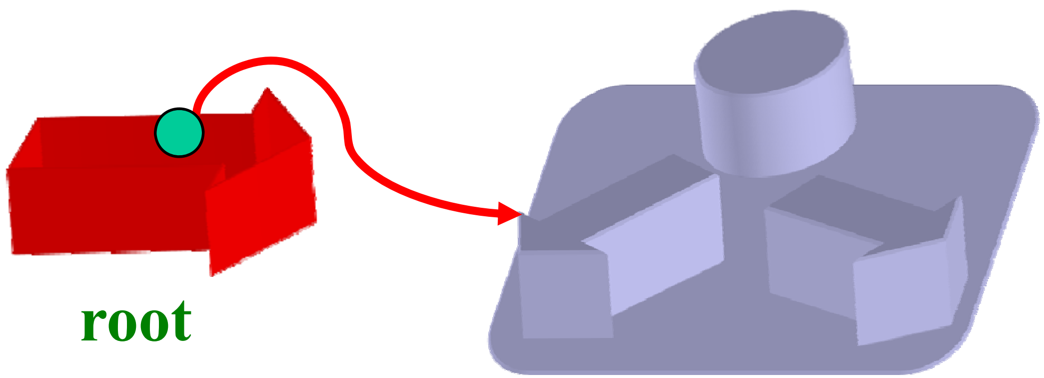
ノード型の定義

```
typedef  strcuct node   Node;
```

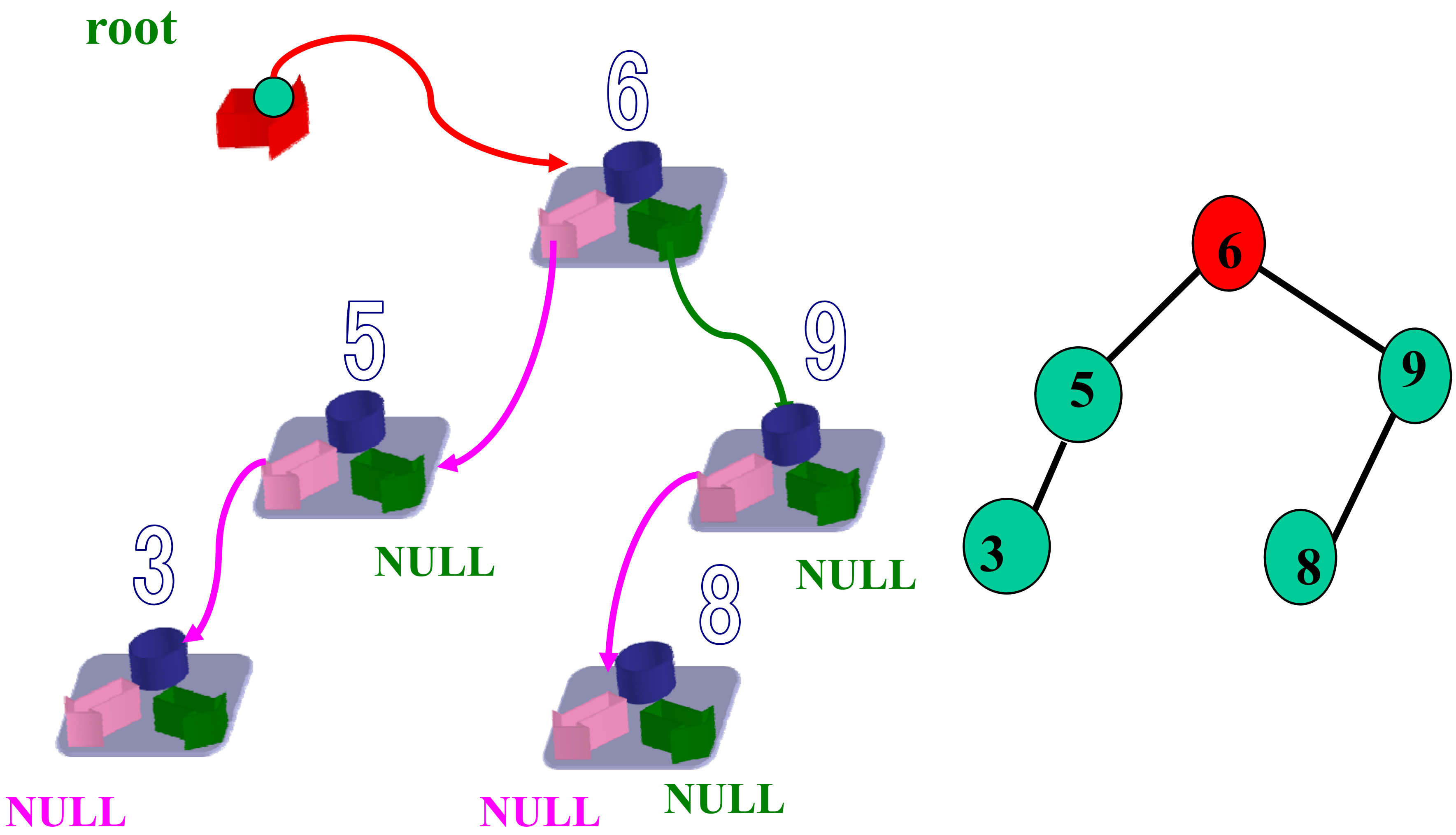


Node型

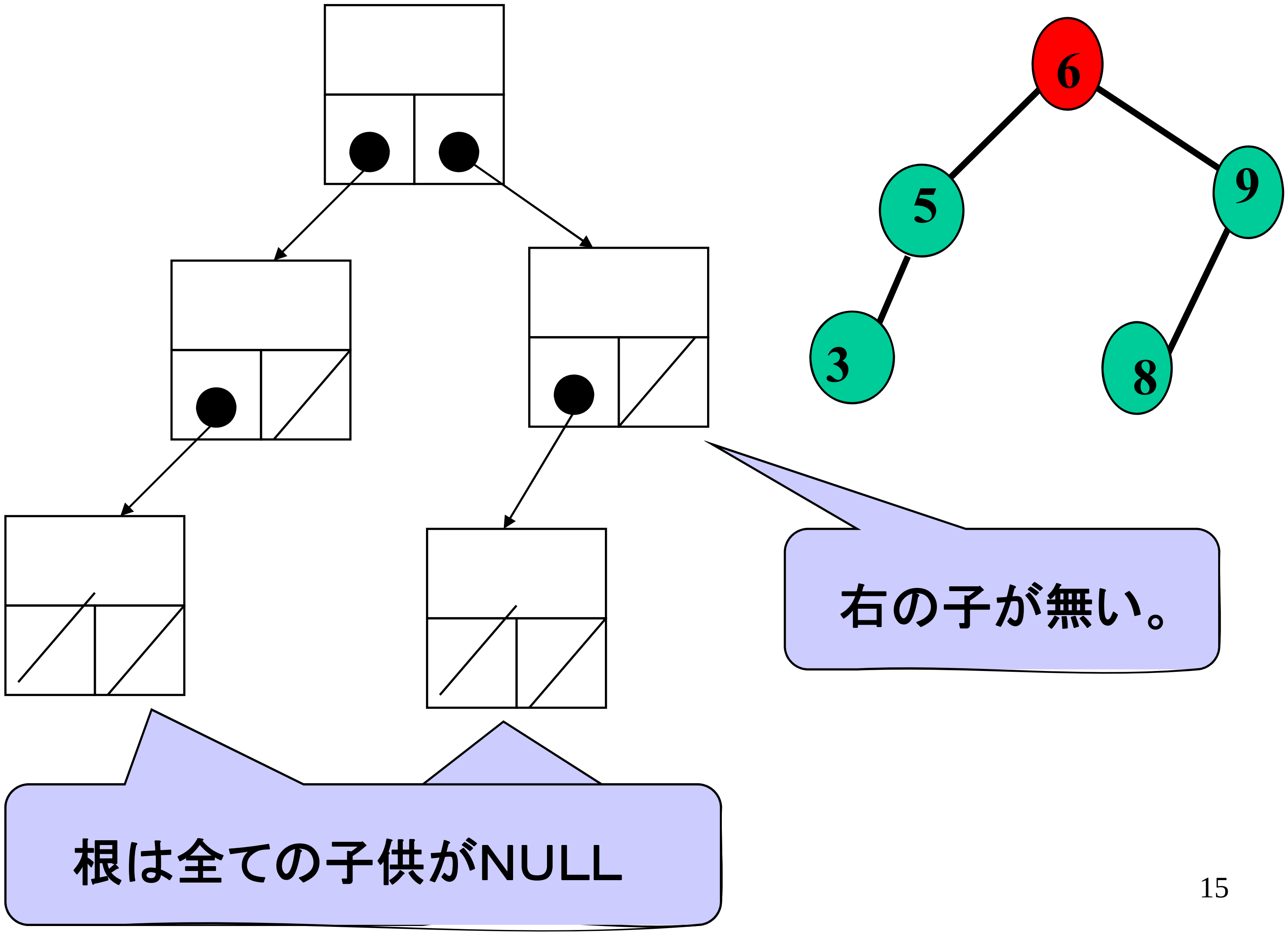
Node ＊ 型



データ構造としての2分木



データ構造としての２分木2

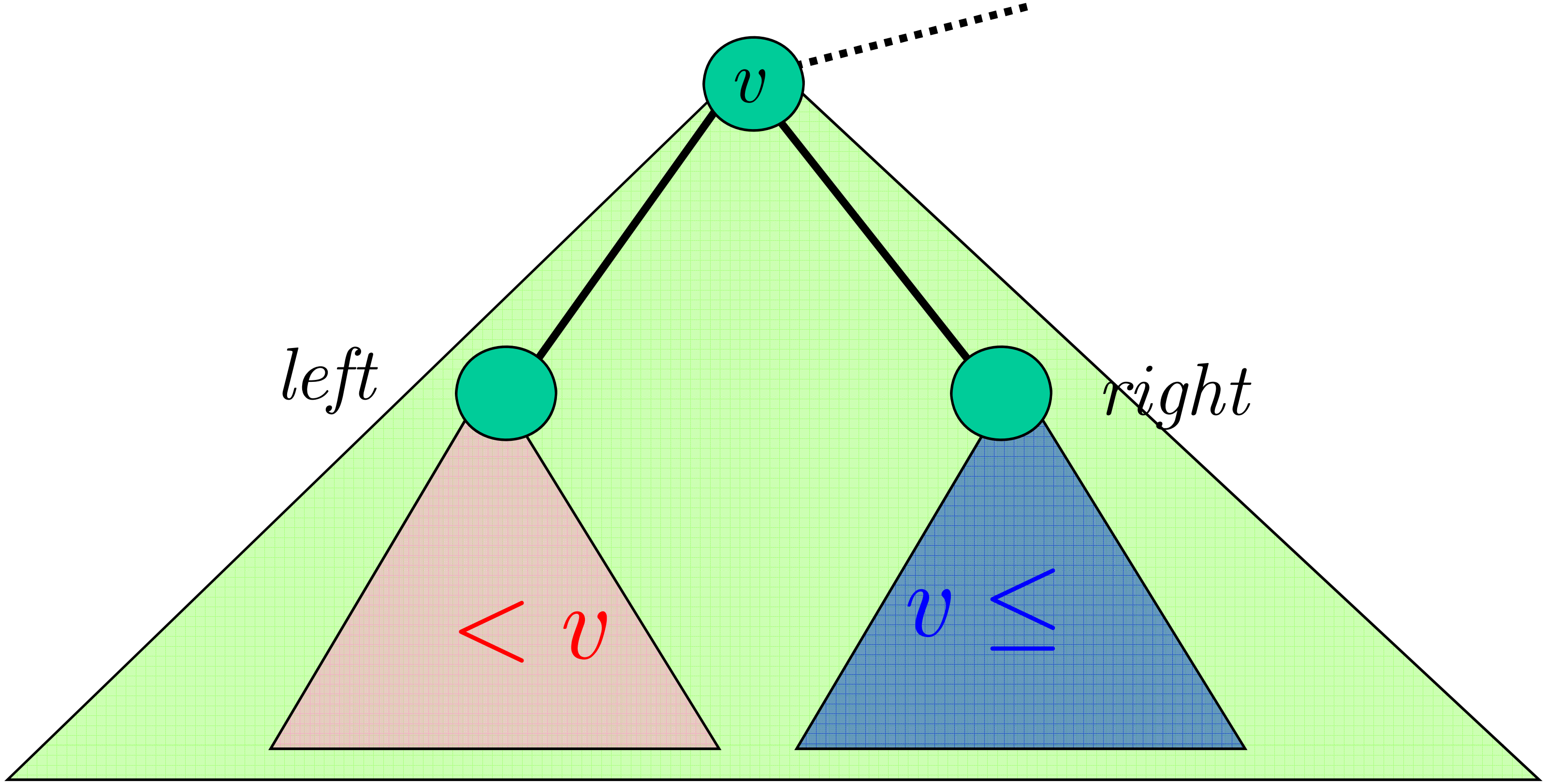


７－２．２分探索木

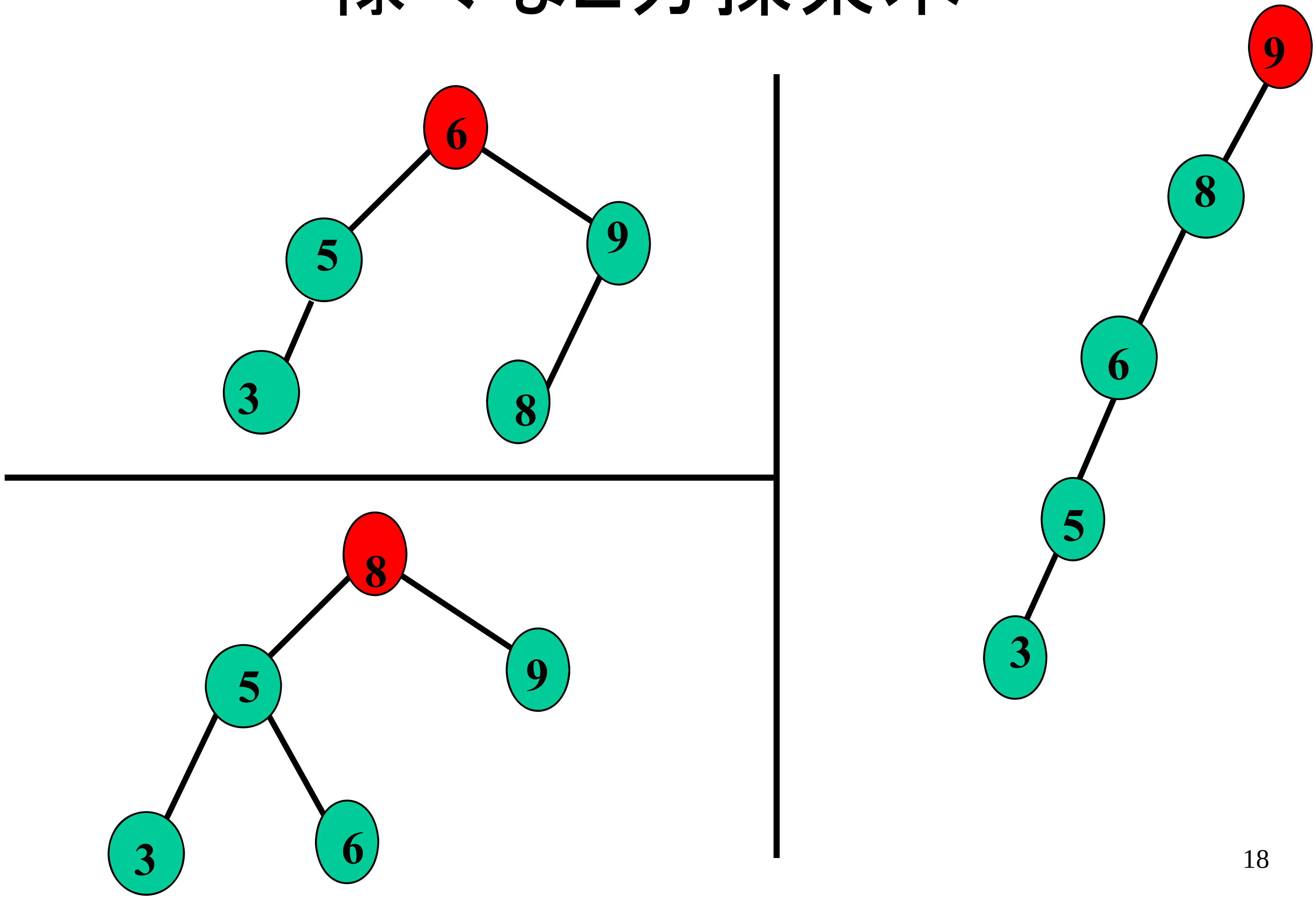
- ２分木
- 各頂点vに対して、
  - 左の子を根とする部分木（左の子孫）のデータは頂点vのデータ未満
  - 右の子を根する部分木（右の子孫）のデータは頂点vのデータ以上

イメージ（2分探索木）

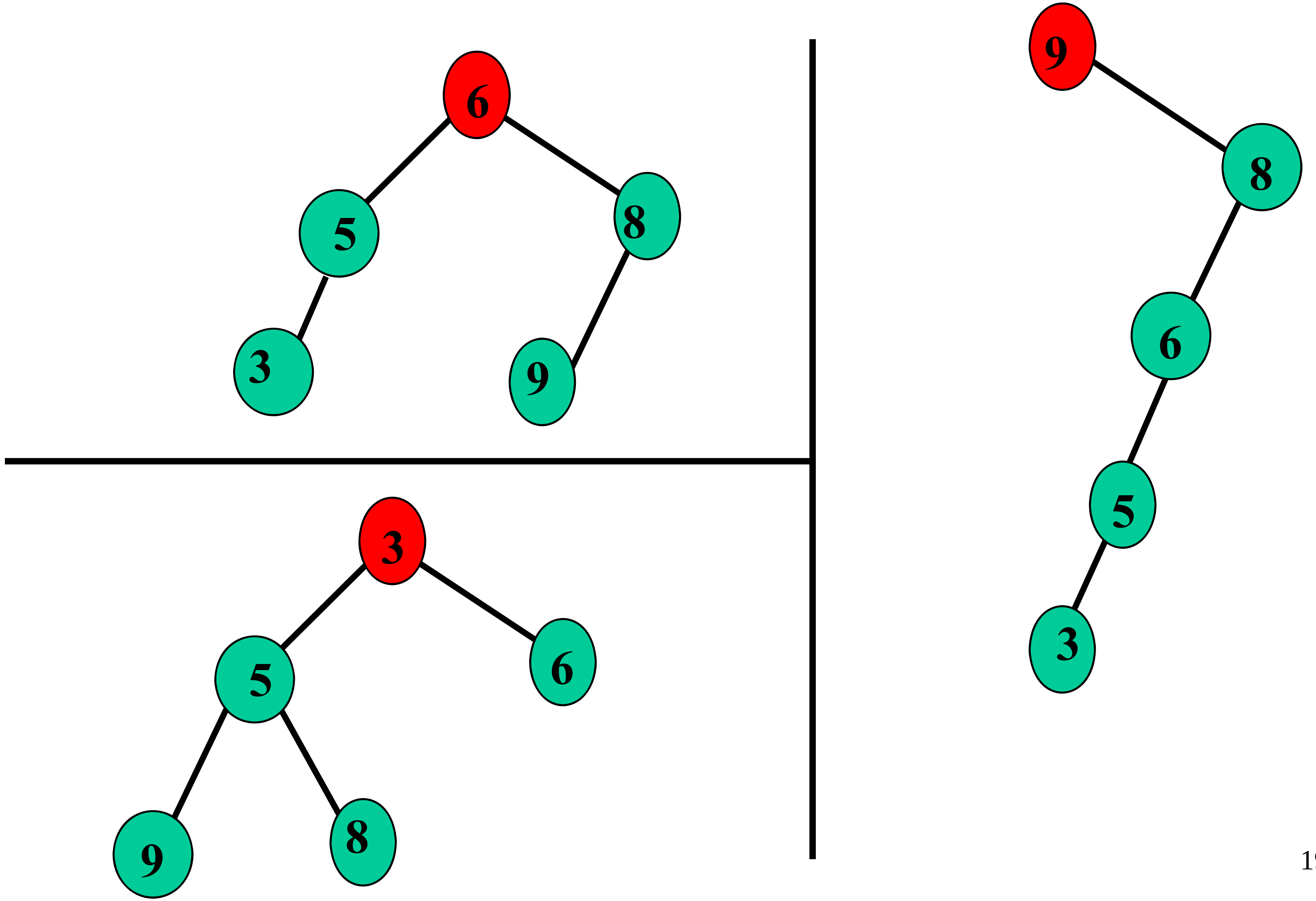
- 条件が再帰的になっていることに注意する。



様々な2分探索木

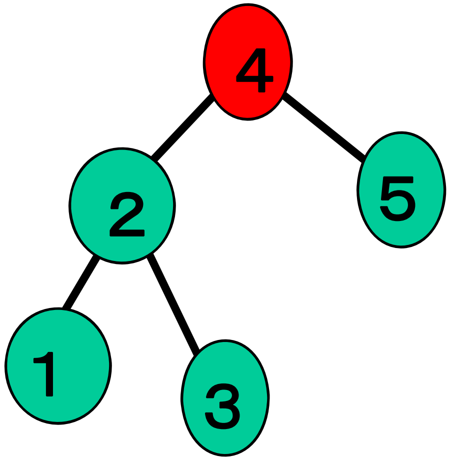


２分探索木ではない木

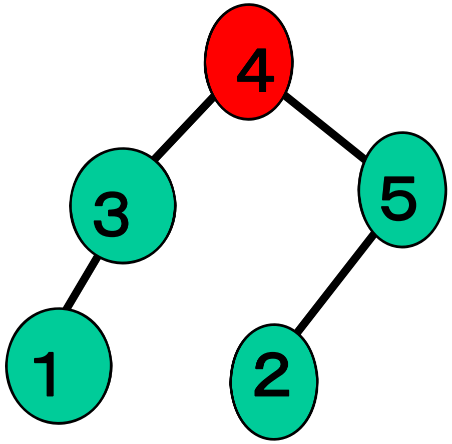


練習次の木が2分探索木であるか答えよ。

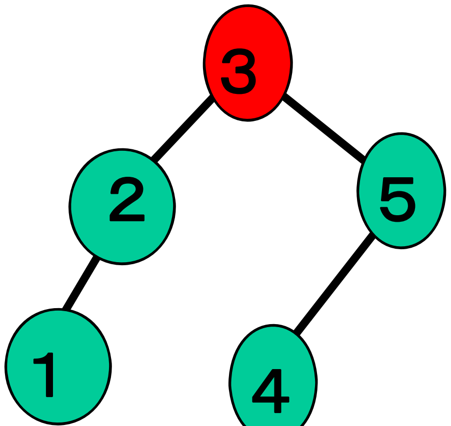
（1）



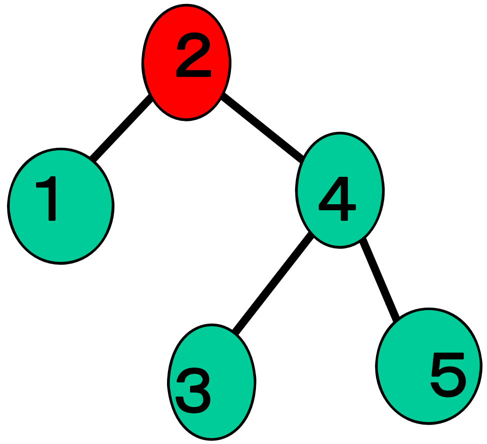
（2）



（3）



（4）



練習

- $\{1,2,3\}$ の３つのデータを2分探索木に保存するとき、２分探索木の形状を全て示せ。

２分探索木における探索

- ２分探索木の性質を利用する。
- ある頂点vのデータと、キーの値の大小関係を調べる。
- キーが小さければ、左の子孫を調べる。

（左の子に再帰的に探索を繰り返す。）

キーが大きければ、右の子孫を調べる。

- 根から探索を開始する。

（探索の概略は、配列における2分探索との類似点がある。）

２分探索木を用いた探索の実現

```
/*  ２分探索木による探索*/
Node*  search(Node* node,double  key){
       if(node==NULL)return NULL;/*基礎*/
       else{  /*      帰納   */
              if(node->data==key)return node;/*発見*/
              else if(key<node->data){/*小さい方*/
                     return search(node->left,key);
              }else if(node->data<key){/*大きい方*/
                     return search(node->right,key);
              }
       }
}
```

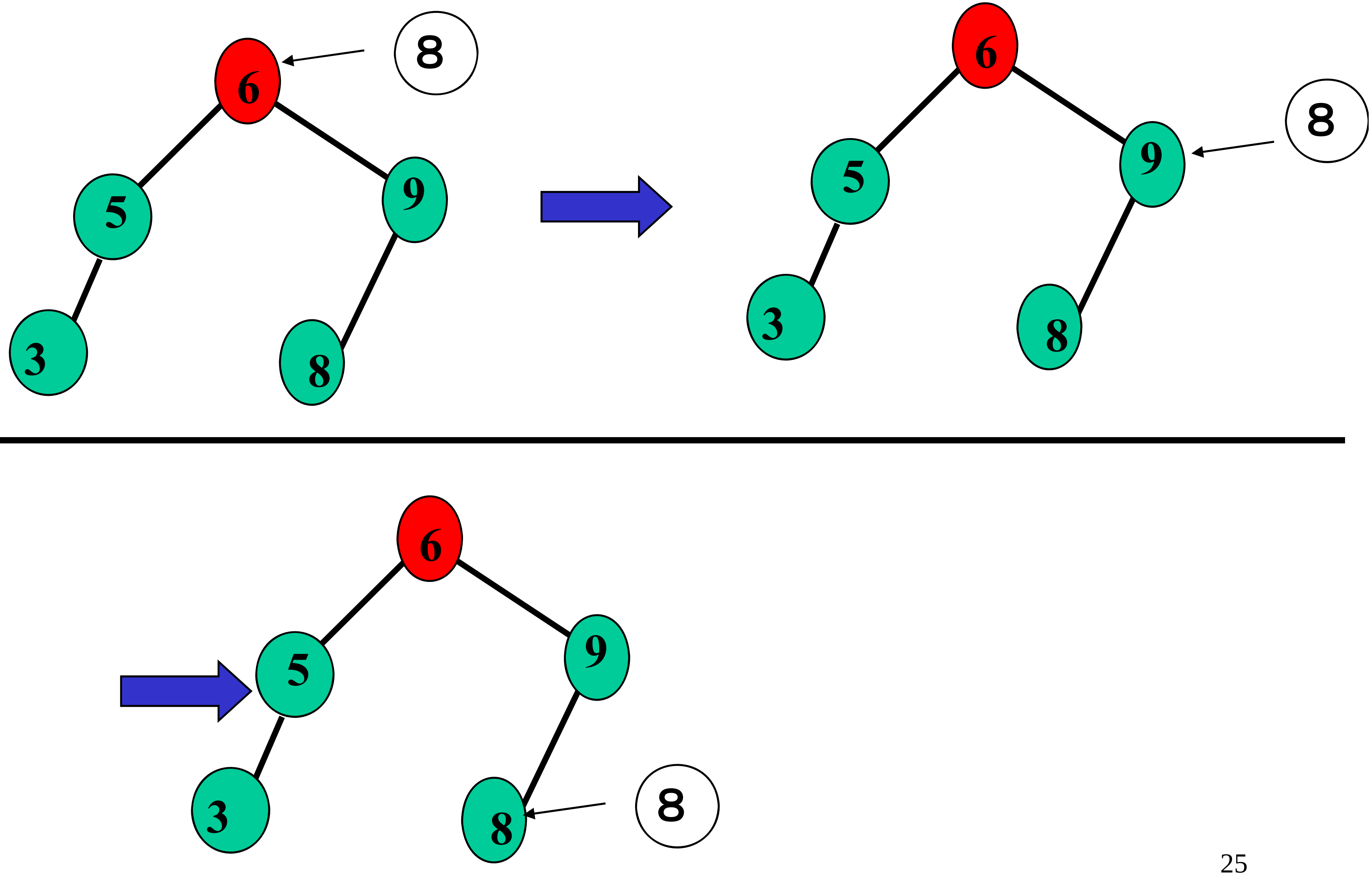
呼び出し方

```
Node *pos;
pos=search(root,key);
```

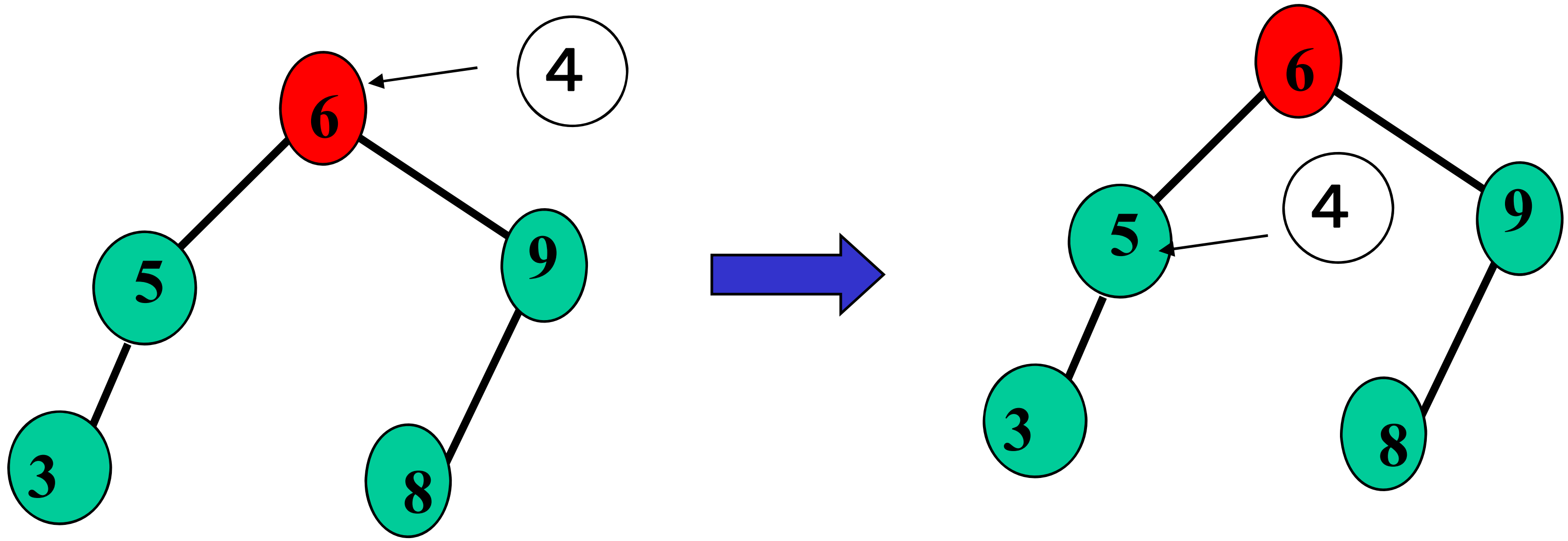
参考2分探索の実現(再帰版)

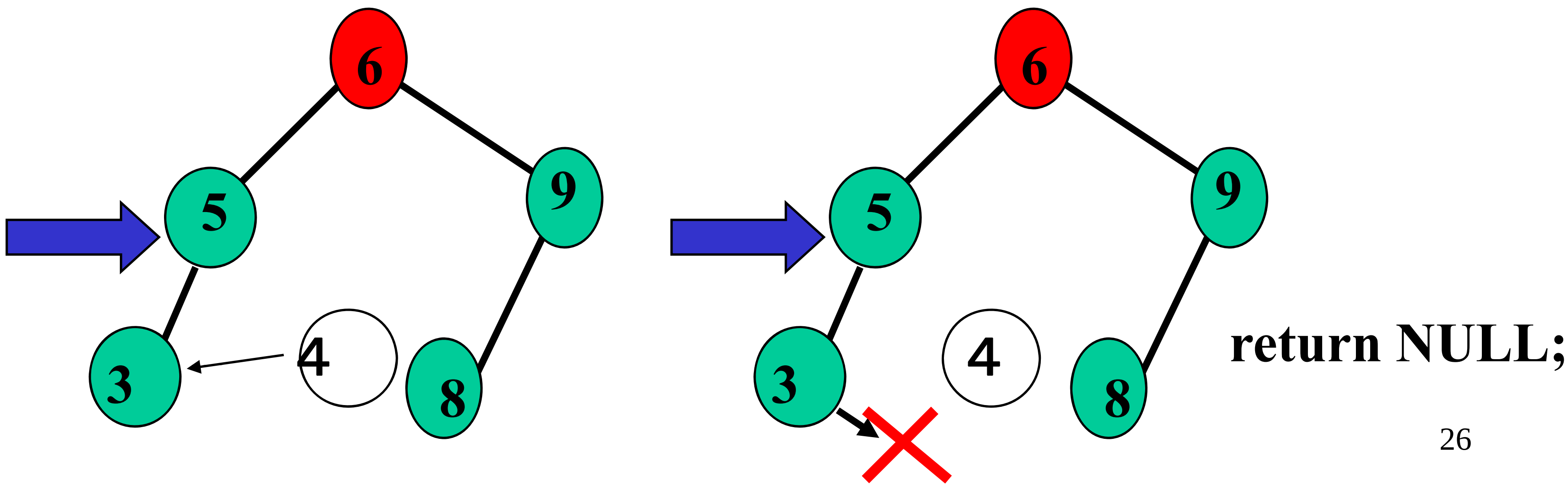
```
/*  再帰版2分探索*/
int search(double k,int left,int right){
    int mid;
    if(left>right)return -1;/*基礎*/
    else{ /*     帰納   */
        mid=(left+right)/2;
        if(A[mid]==k)return mid;/*発見*/
        else if(k<A[mid]){/*小さい方*/
            return search(k,left,mid-1);
        }else if(A[mid]<k){/*大きい方*/
            return search(k,mid+1,right);
        }
    }
}
```

探索の動き1



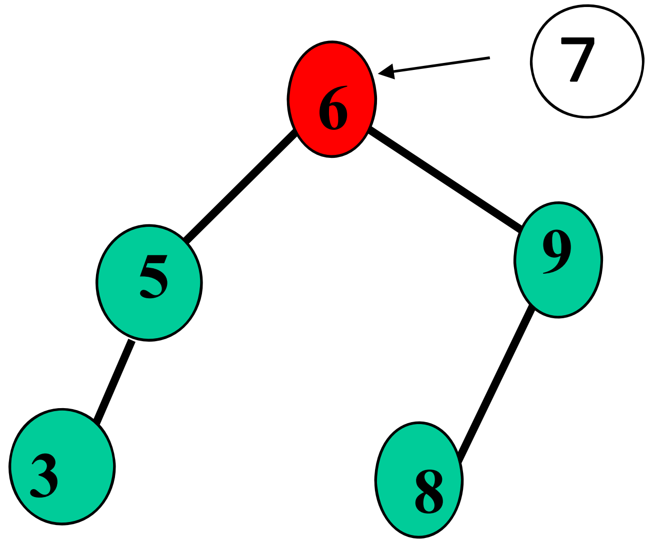
探索の動き2



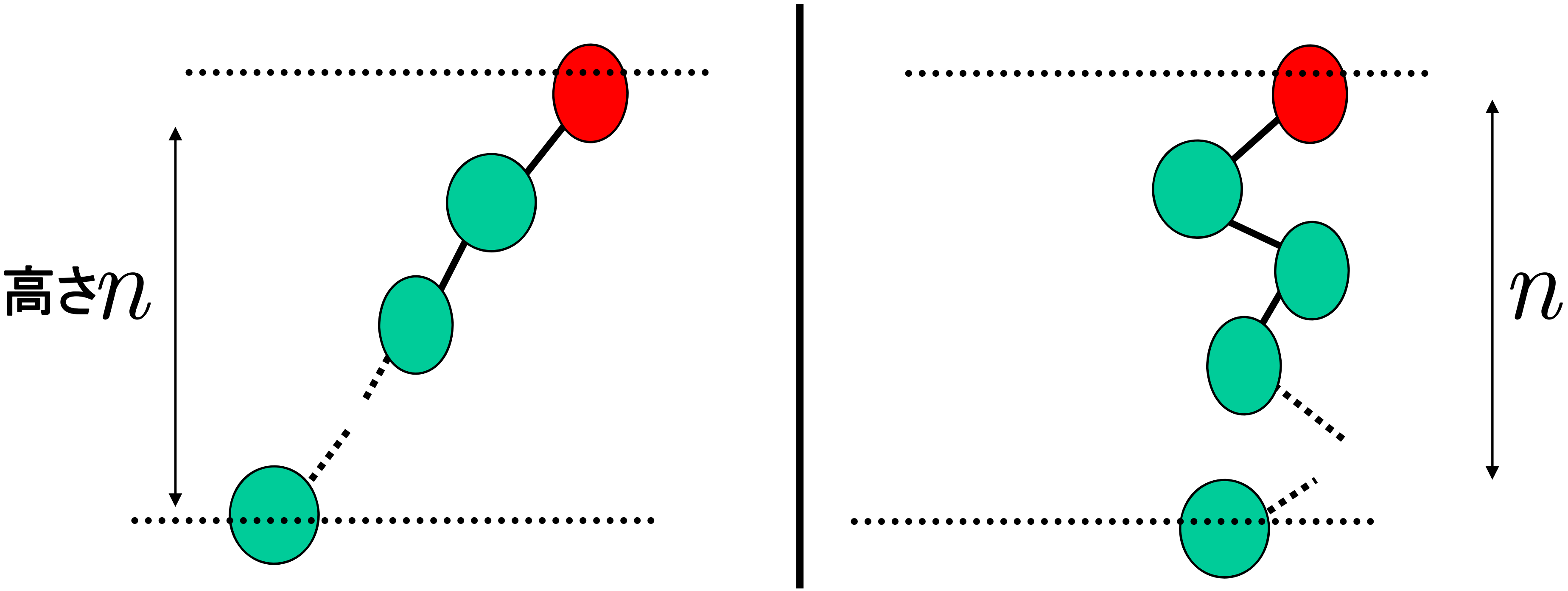


練習

下記のデータ構造に対して、7を探索するときの動作および10を探索するときの動作を示せ。

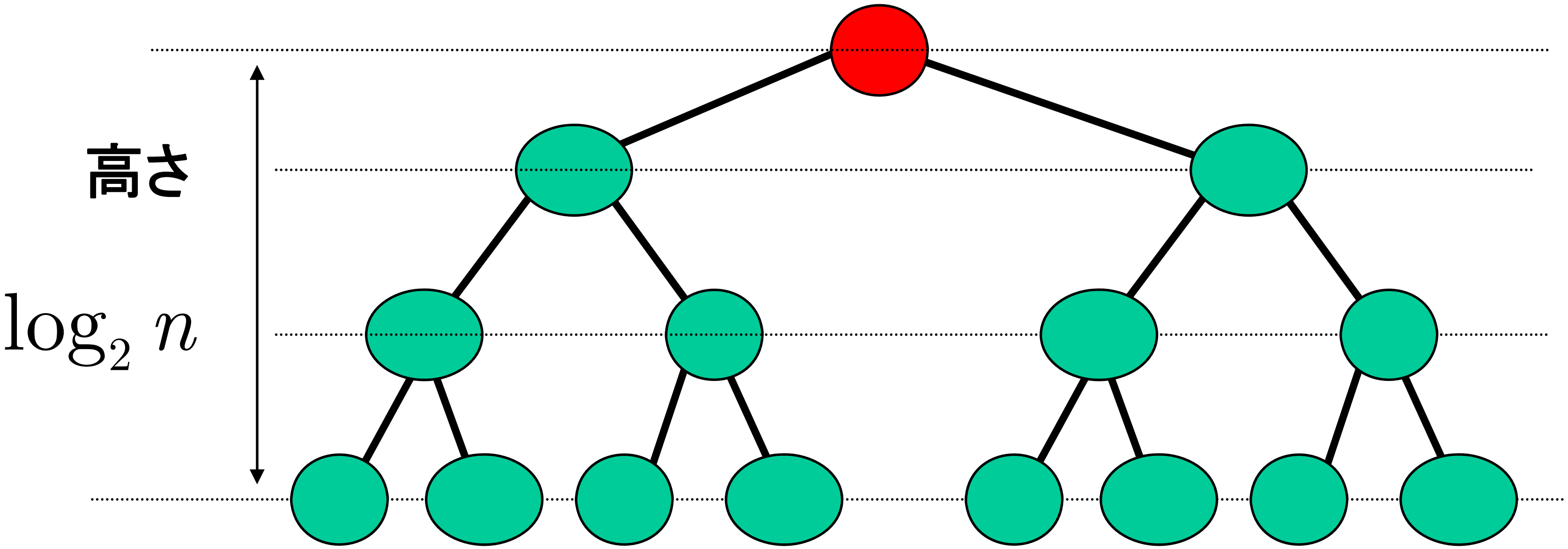


高さの高い2分探索木



２分探索木の高さは、 n になるこもある。

高さの低い2分探索木



完全2分木状になれば、2分探索木の高さは
$\log_2 n$ である。

２分探索木における探索計算量

２分探索木における探索では、高さに比例した時間計算量が必要である。最悪の場合を考慮すると、高さが n の場合が存在する。

したがって、２分探索木における探索の最悪時間計算量は、

$$O(n) \text{ 時間}$$

である。この場合は線形探索と同じように探索される。

２分探索木への挿入

- 探索と同様に、挿入データvの２分探索木での位置を求める。
- 子供がない位置に、新しくvを子供として追加する。

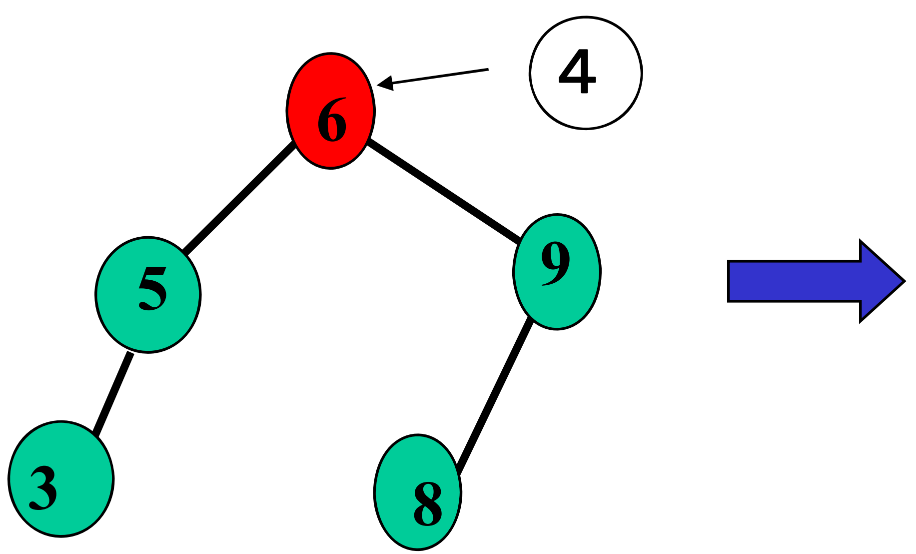
２分探索木への挿入の実現1

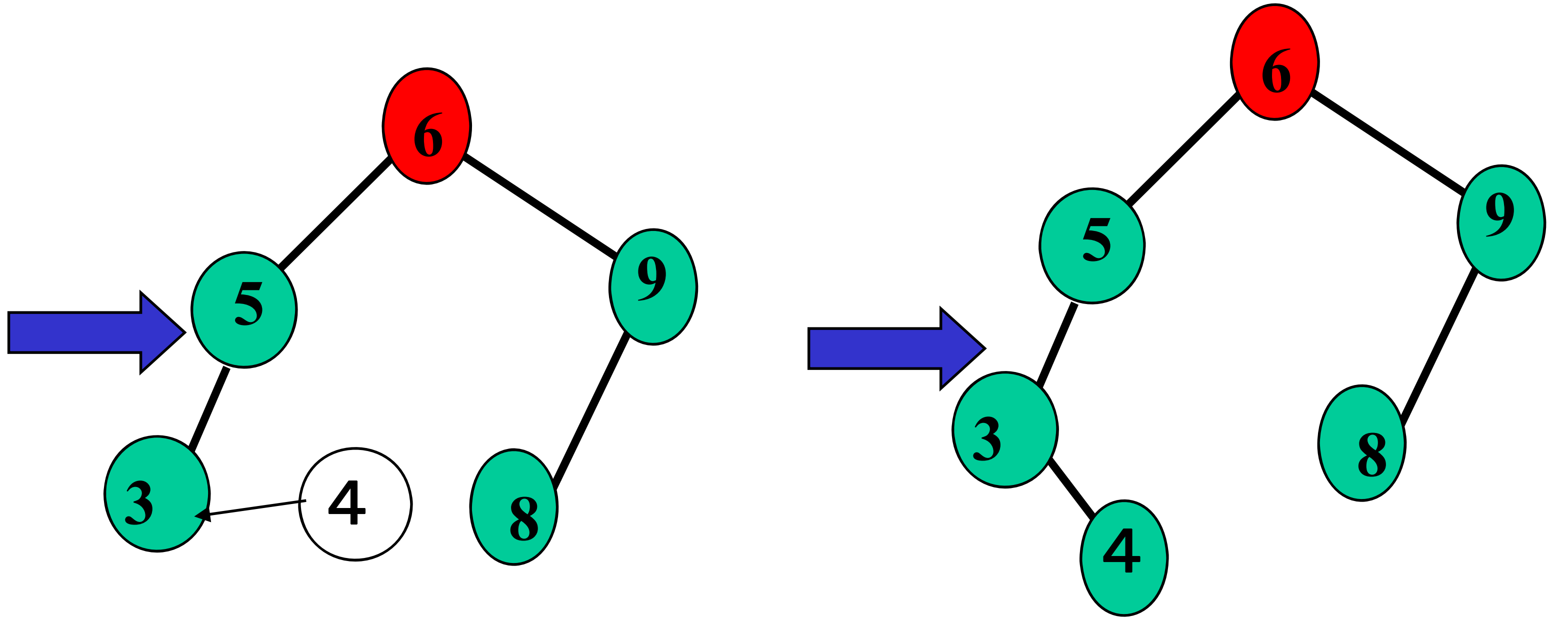
```
/*  ２分探索木への挿入位置を求める。親を返す（概略） */
Node* find_pos(Node* node,double  value){
    if(value< node->data){/*左部分木への挿入*/
        if(node->left==NULL){/*左子が挿入場所*/
            return node;
        }
        else  return find_pos(node->left,value);
    }
    else{/*左部分木への挿入*/
        if(node->right==NULL){/*右子が挿入場所*/
            return node;
        }
        else retrun find_pos(node->right,value);
    }
}
```

2分探索木への挿入の実現2

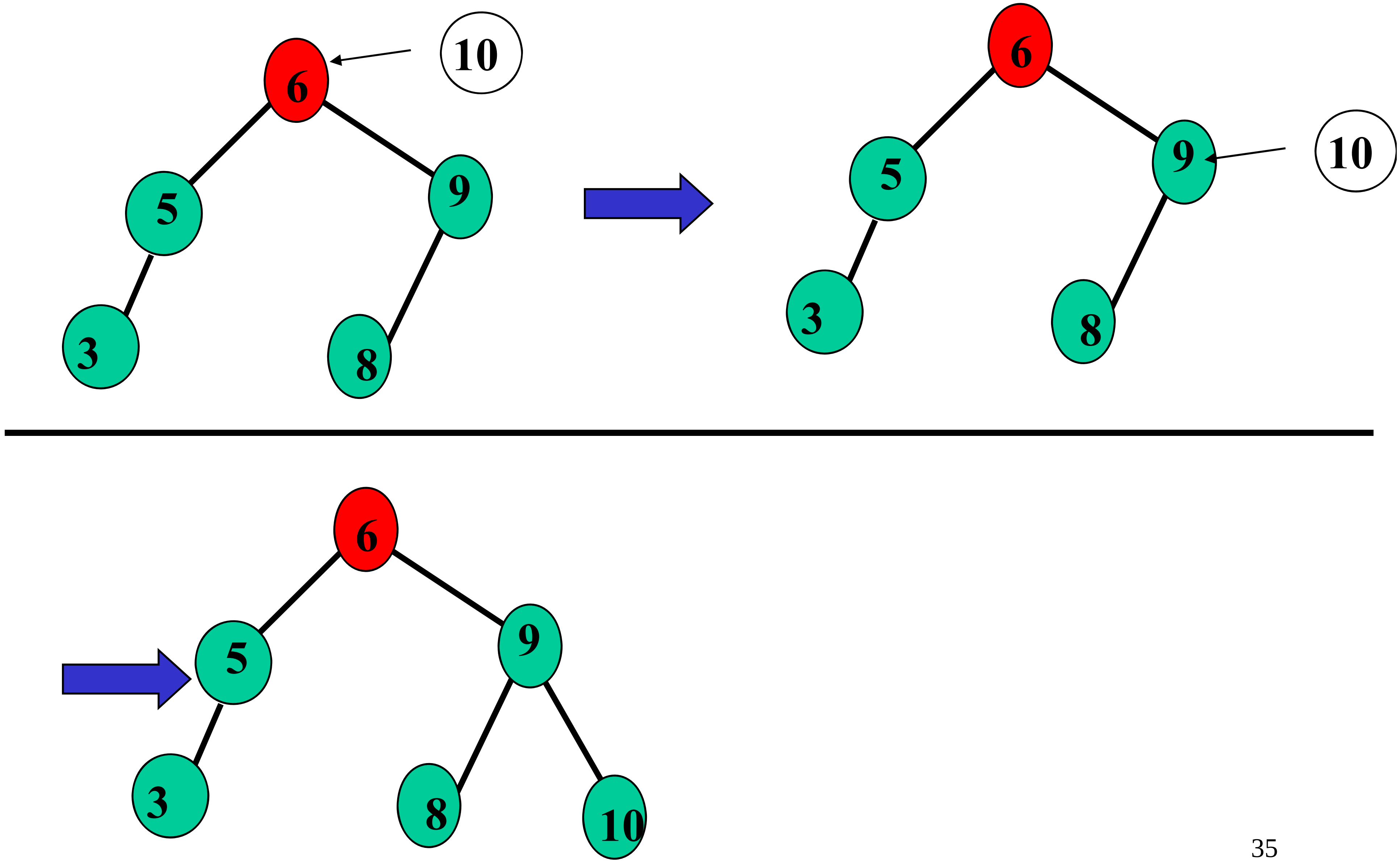
```
/*  2分探索木への挿入する。      */
void insert(Node* root,double  value){
     Node* pos;/*挿入位置*/
     Node* new;/*挿入点*/
     new=(Node*)malloc(sizeof(Node));
     new->data=value;
     new->left=NULL;
     new->right=NULL;
     pos=find_pos(root,value);
     if((value< pos->data)&&(pos->left==NULL))
          pos->left=new;
     }
     else if((pos->data<value)&&(pos->right==NULL))
          pos->right=new;
     }
     return;
}
```

挿入の動き1





挿入の動き2



挿入の最悪時間計算量

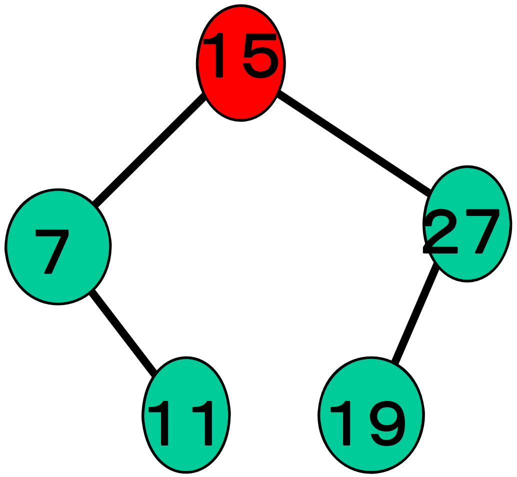
挿入には、最悪、2分探索木の高さ分の時間計算量が必要である。したがって、

$$O(n) \text{ 時間}$$

である。

練習

次の2分探索木に以下で示す要素を順に挿入せよ。

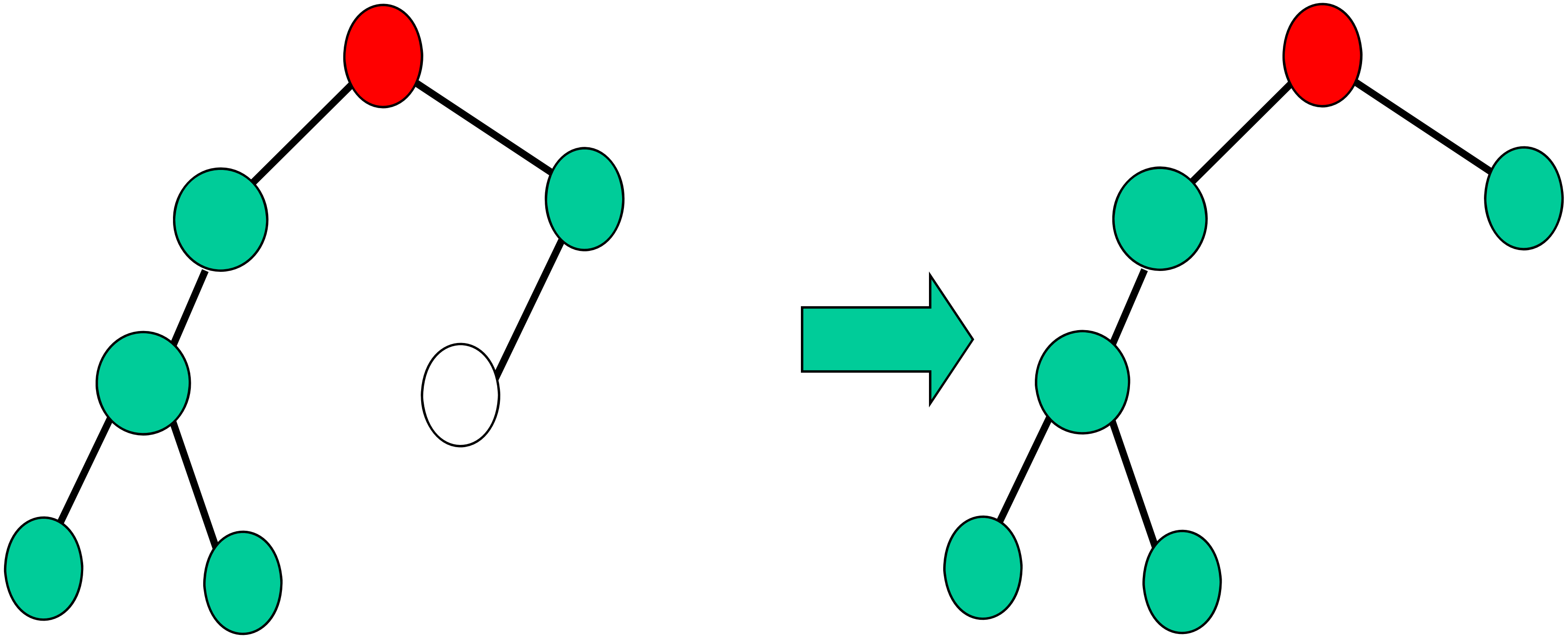


５→１２→２０→２３→１０

２分探索木からの削除

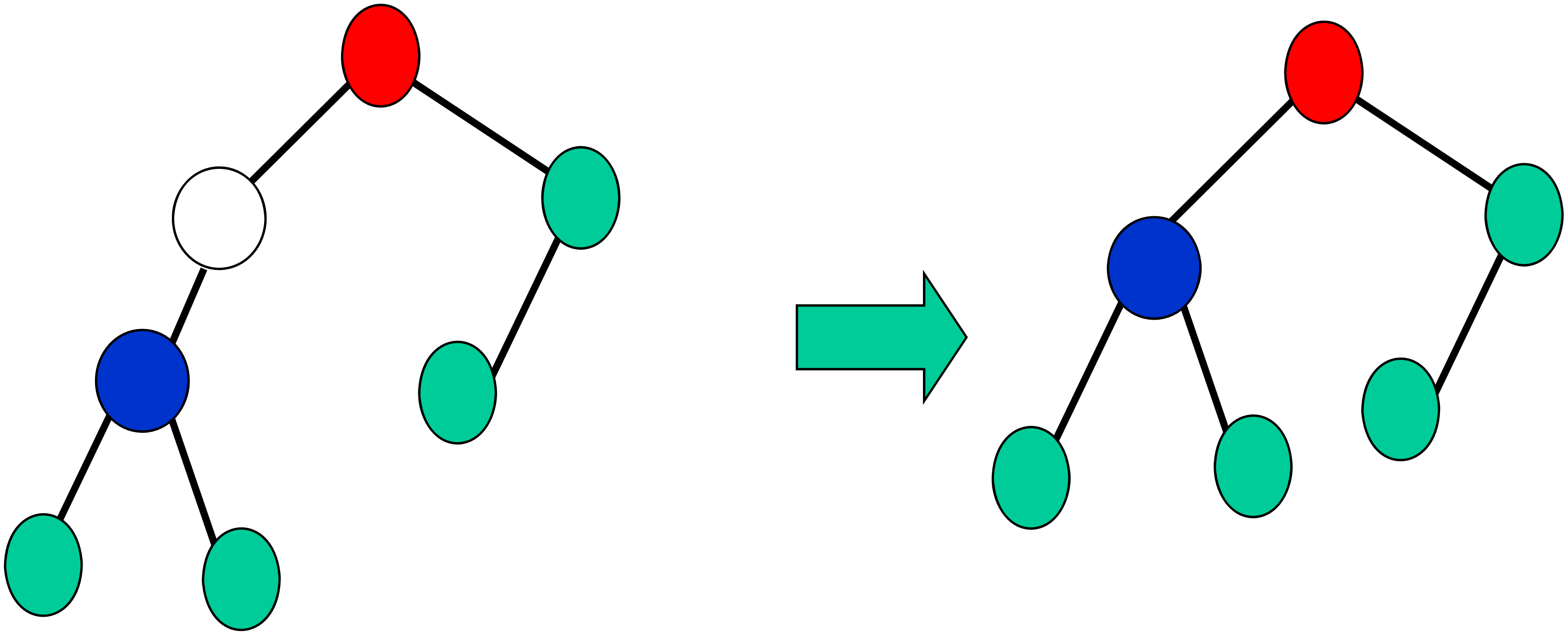
- 削除する点を根とする部分木中の、最大値あるいは最小値で置き換える。
- 削除は、少し煩雑なので、コードは示さずに、動作だけを示す。

削除動作１（葉の削除）



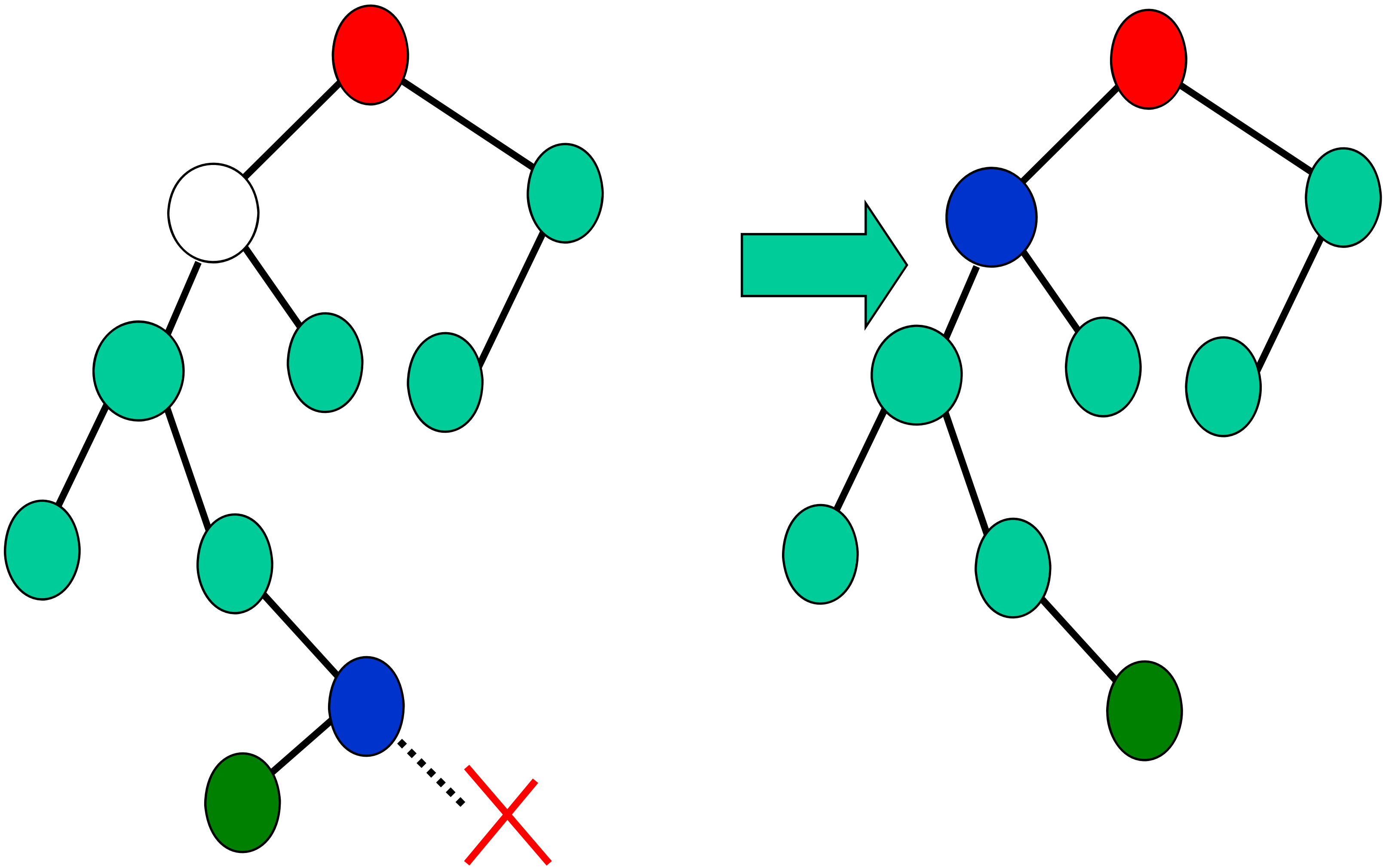
単に削除すればよい。

削除動作２（子供が一つの場合の削除）



唯一の子供を、親にリンクする。

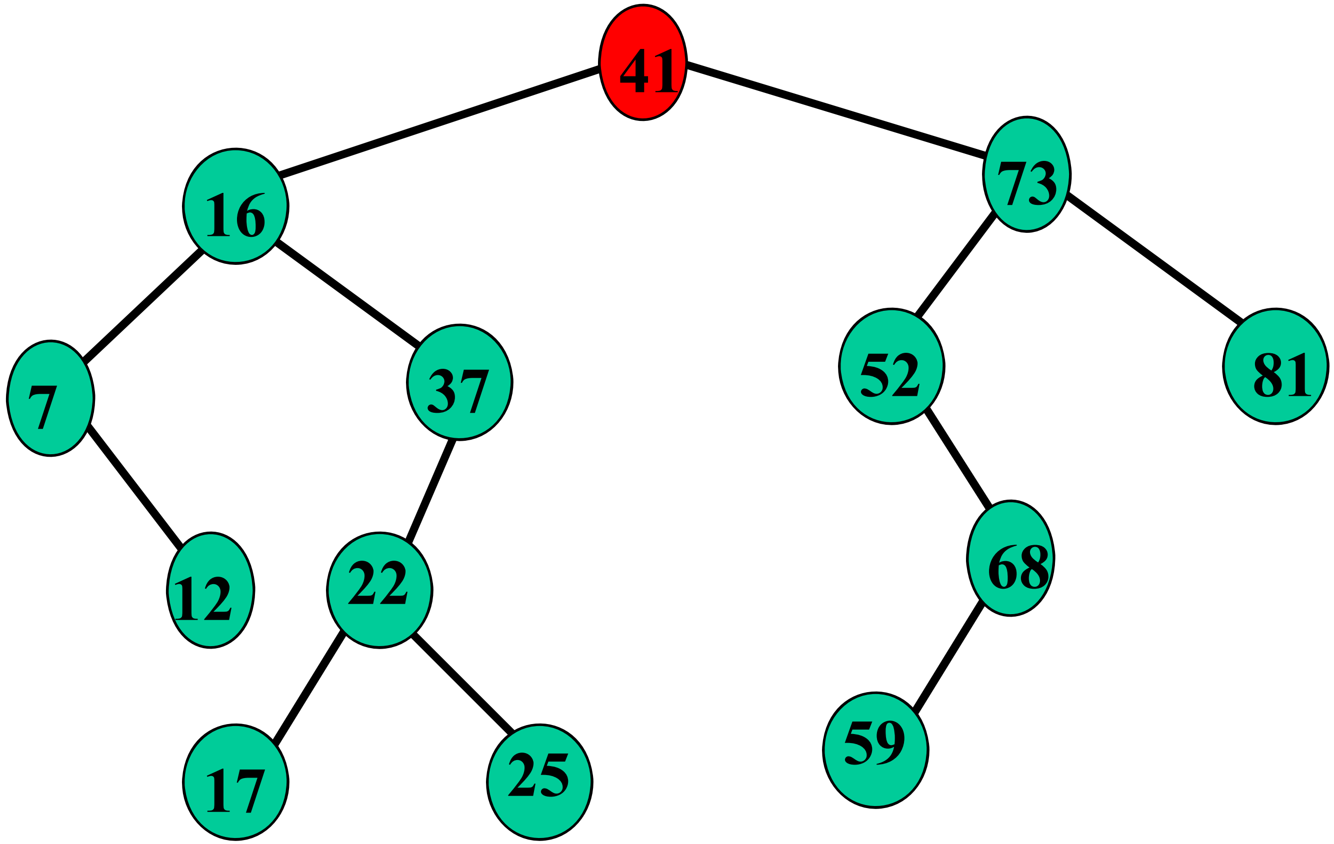
削除動作3（子供が2つの場合の削除）



左部分木の最大値（あるいは右部分木最小値）を求め更新する。

練習

次のから2分探索木から、以下で示す順序に要素を削除せよ。
ただし、2つの子がある点が削除される場合には、
右部分木の最小値を用いて更新すること。



12→37→73→68→22

削除の最悪時間計算量

挿入には、最悪、2分探索木の高さ分の時間計算量が必要である。したがって、

$$O(n) \text{ 時間}$$

である。

2分探索木における各操作の平均時間量解析

- 各操作は、2分探索木の高さに比例する時間量で行える。
- ここでは、空木（データの無い木）からはじめて、n 個のデータをランダムに挿入して作成される2分探索木の高さ（平均の深さ）を評価する。

ここでのランダムとは、$n!$ 個の順列が均等におきると仮定して、その順列に従って挿入することである。

次のように記号を定義する。

$D(n) =$ (n要素の２分探索木の平均の深さ)

この $D(n)$ を求めるために、ランダムに挿入する際の比較回数 $C(n)$ を考察する。ここで、

$C(n) =$ (n要素の２分探索木を構成するときの平均比較回数)

である。

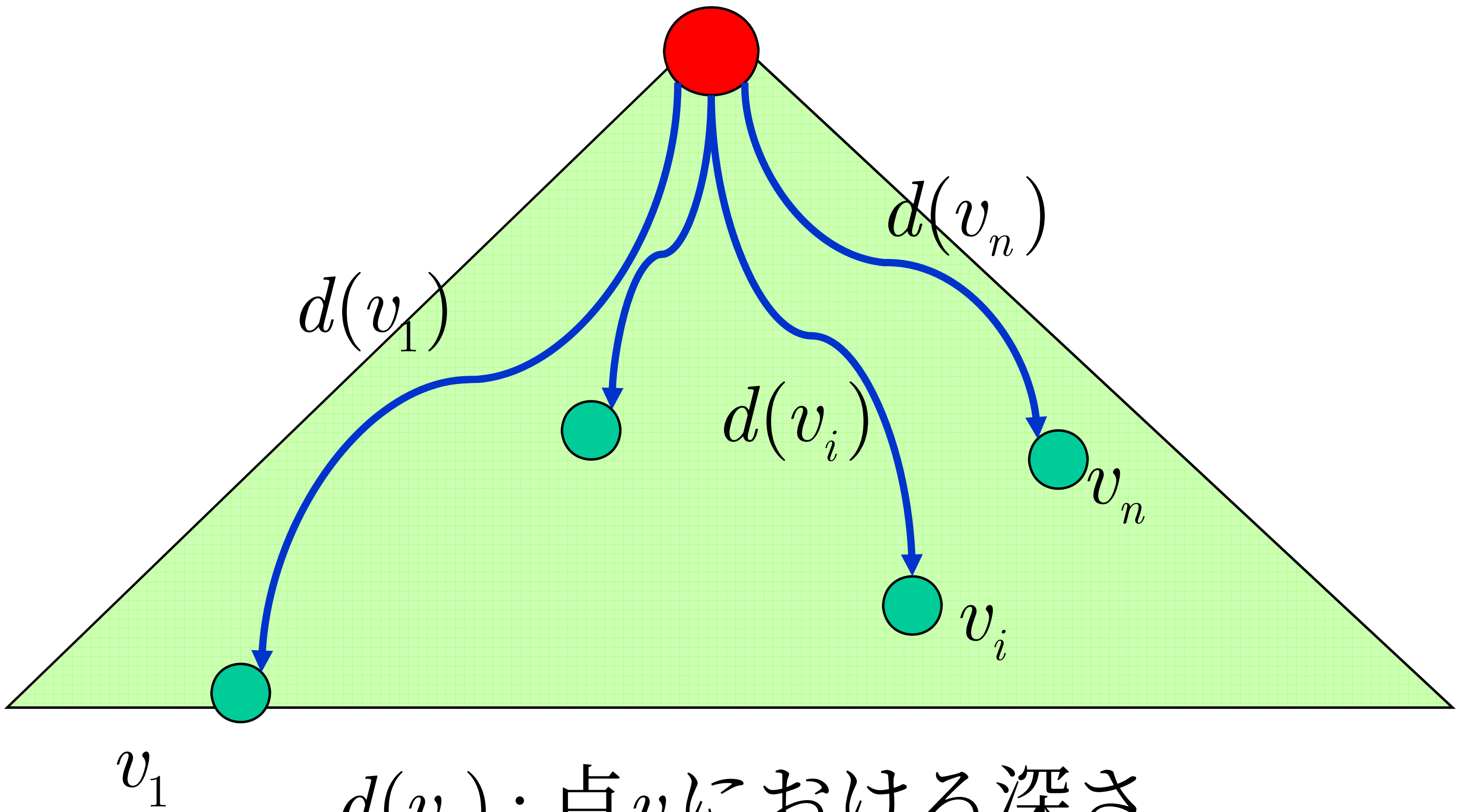
このとき、各頂点vに対して、作成時に深さ－1回の比較を行っていることに注意すると、次の関係が成り立つ。

$$D(n) - 1 \simeq \frac{1}{n} C(n)$$

平均の深さ

作成時の平均比較総数

イメージ



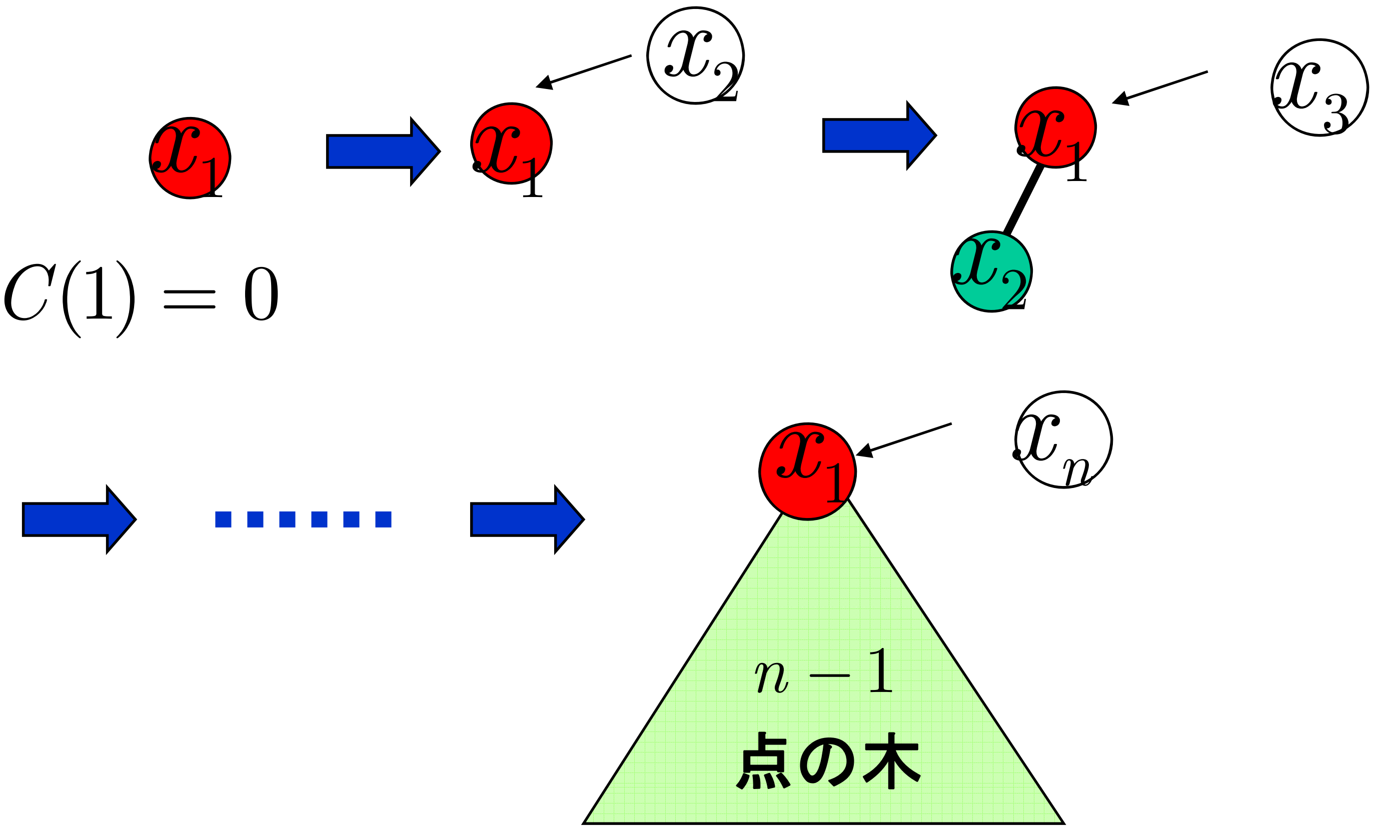
$d(v_i)$: 点v_iにおける深さ

$$D(n) = \frac{1}{n}\sum_{i=1}^{n} d(v_i)$$

$$D(0) = 0, D(1) = 0$$

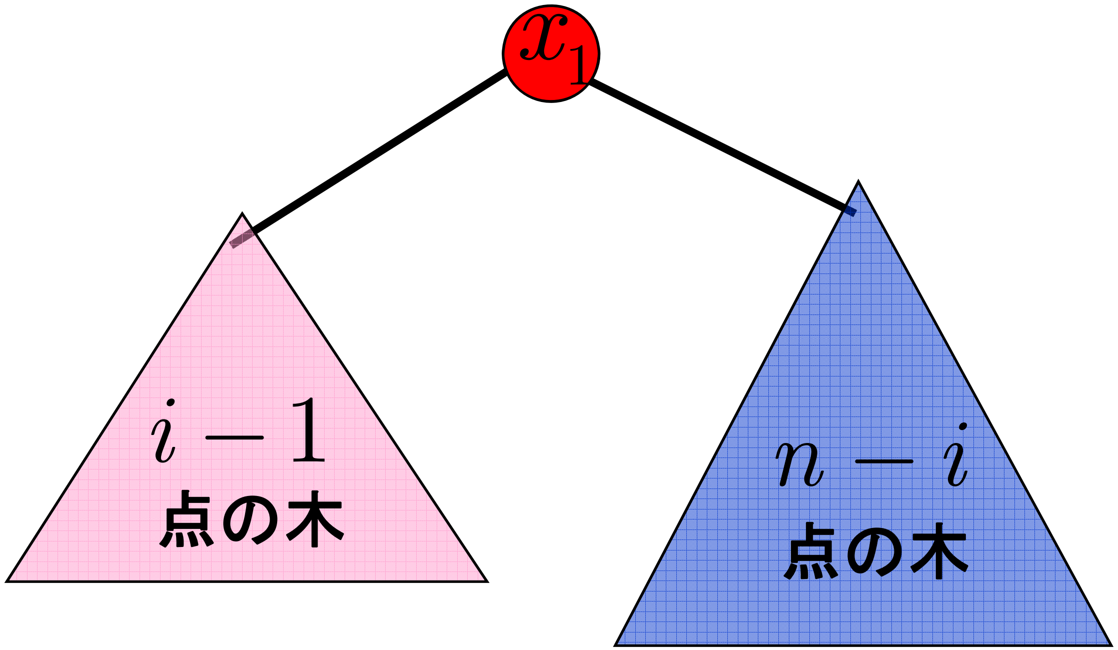
次にデータの挿入される順に、$x_1, x_2, \cdots, x_n$
と定める。

このとき、x_1 は根におかれ、2分探索木完成までには、
$n-1$ 回の比較が行われる。



$C(1) = 0$

一方、 x_1 の大きさが i 番目であるとする。



ランダムなので、順位 i は1からnの全て均等におきることに注意する。

これらのことを考慮すると、2分探索木の構成時における平均の総比較回数は、次の漸化式を満たす。

$$C(n) = \frac{1}{n}\sum_{i=1}^{n}(n-1+C(i-1)+C(n-i))$$

根との比較数

左部分木の
平均比較総数

右部分木の
平均比較総数

ランダムなので、全ての順位が均等に起こる。全ての場合の総和を求めて、nで割れば、平均比較総数となる。

クイックソートの平均時間計算量が満たすべき漸化式とまったく同じである。

忘れた人のために、もう一度解く。

$$C(n) = \frac{1}{n}\sum_{i=1}^{n}(n-1+C(i-1)+C(n-i))$$

$$\therefore\ nC(n) = n(n-1) + 2\sum_{i=0}^{n-1} C(i) \quad \cdots\cdot ①$$

①に、$n-1$　を代入して、

$$(n-1)C(n-1) = (n-1)(n-2) + 2\sum_{i=0}^{n-2} C(i)$$

$\cdots\cdot$②

①－②

$$nC(n)-(n-1)C(n-1)$$

$$= n(n-1)-(n-1)(n-2)+2C(n-1)$$

$$\therefore\ nC(n)-(n+1)C(n-1) = n(n-1)-(n-1)(n-2)$$

$\cdots\cdot$③

③のすべての項を $n(n+1)$ で割ってまとめる。

$$\therefore \frac{C(n)}{n+1} - \frac{C(n-1)}{n} = \frac{n(n-1)}{n(n+1)} - \frac{(n-1)(n-2)}{n(n+1)}$$

$$\therefore \frac{C(n)}{n+1} - \frac{C(n-1)}{n} \leq \frac{2}{n}$$

辺々加えてまとめる。

$$\therefore \frac{C(n)}{n+1} \leq 2(H_{n-1} - 1)(\because C(1) = 0)$$

$$\therefore C(n) \leq 2n \log_e n$$

以上、より n 点をランダムにして2分探索木を構築するための総比較回数(平均時間計算量)は、 $O(n \log n)$ である。

ここで、n 点の2分探索木における各頂点の平均深さと、
n 点の2分探索木構築する平均比較総数の関係を思い出す。

$$D(n) - 1 \simeq \frac{1}{n} C(n)$$

この関係式より、

$$D(n) = O(\log n)$$

である。

２分探索木における各操作に必要な平均時間計算量は、
平均深さ $D(n)$ に比例すると考えられる。したがって、
n 点からなる2分探索木における「探索」「挿入」「削除」の
各操作を行うための平均時間計算量は、

$$O(\log n)$$

である。

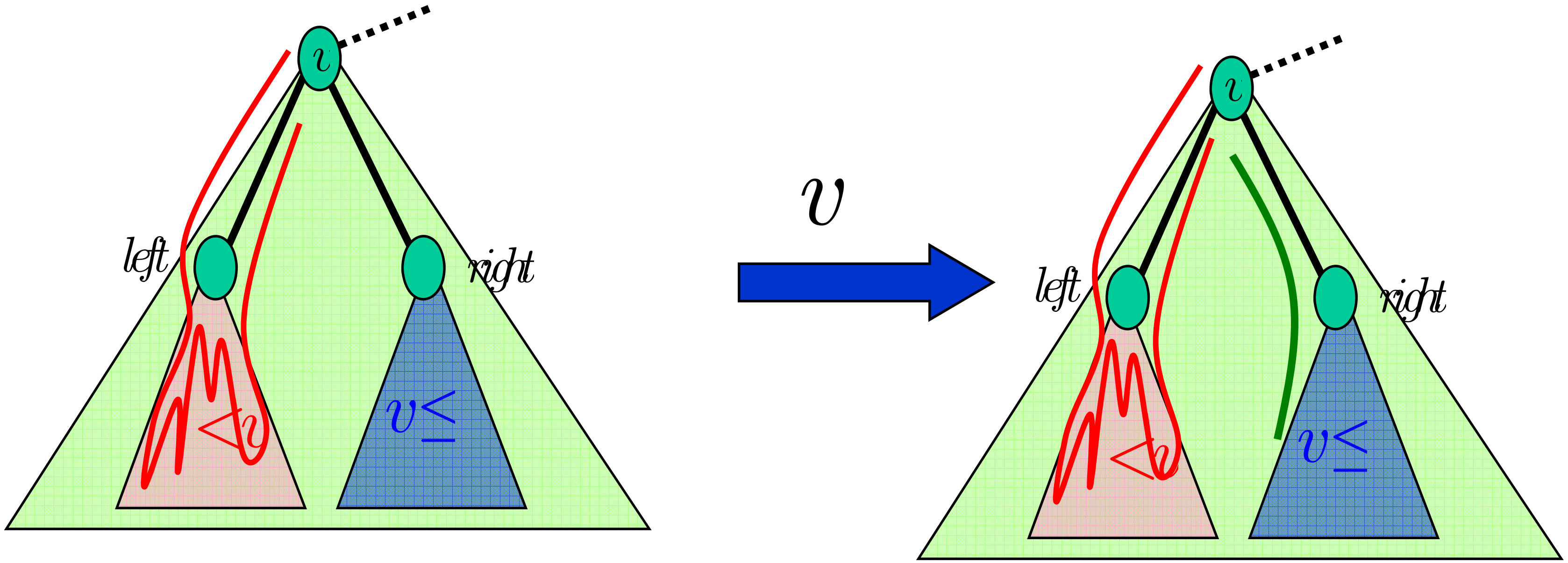
２分探索木のまとめ

	最悪 時間計算量	平均 時間計算量
探索	$O(n)$	$O(\log n)$
挿入	$O(n)$	$O(\log n)$
削除	$O(n)$	$O(\log n)$
構築	$O(n^2)$	$O(n\log n)$

n :データ数

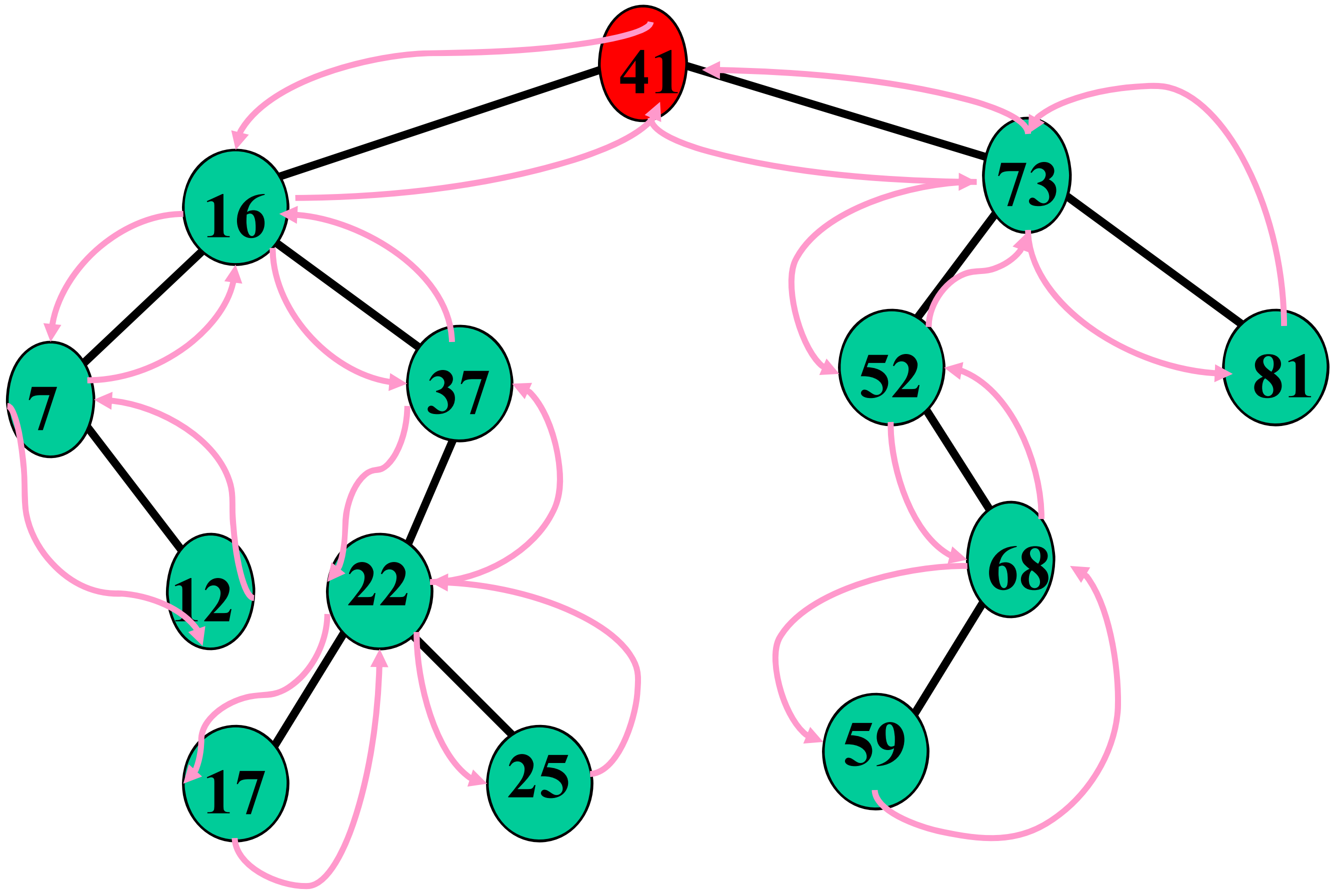
２分探索木と整列

２分探索木を用いても、ソートを行うことができる。

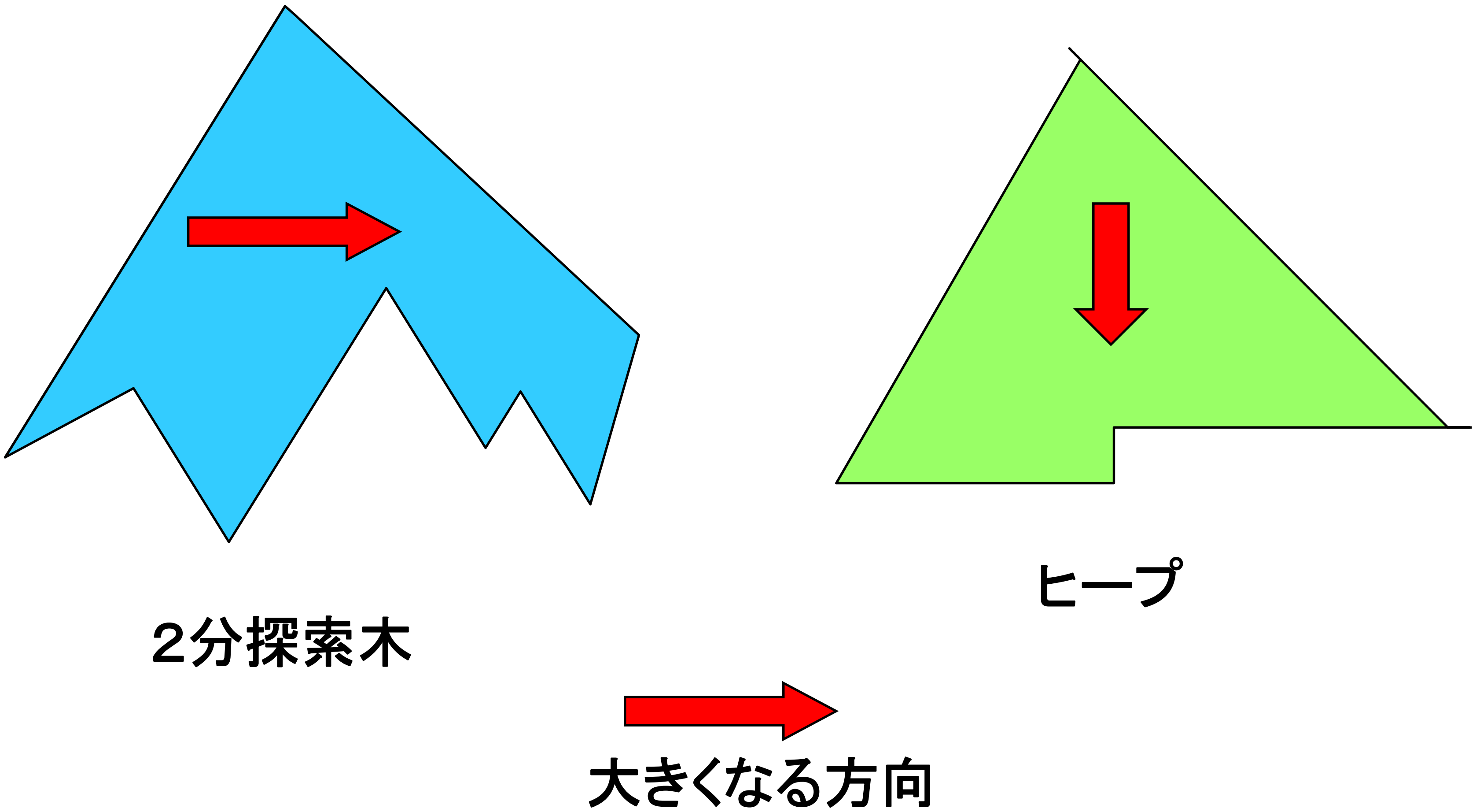


左優先で木をなぞったとき、
点 v において v の左部分木のすべてをなぞったら、
vを出力し、右の部分木をなぞるようにすればよい。

２分探索木と整列



２分探索木とヒープ
（イメージ）



７－３．高度な木
（平衡木）

- ＡＶＬ木

  平衡２分木。回転操作に基づくバランス回復機構により平衡を保つ。

- Ｂ木

  平衡多分木。各ノードの分割、併合操作により平衡を保つ。

２分探索木の問題点

- 高さが $O(n)$ になることがある。
- 各操作の最悪計算量は、 $O(n)$ 時間になってしまう。

（平均計算量は、$O(\log n)$ 時間である。）

⇩

最悪計算時間でも$O(\log n)$ 時間にしたい。

n：保存データ数

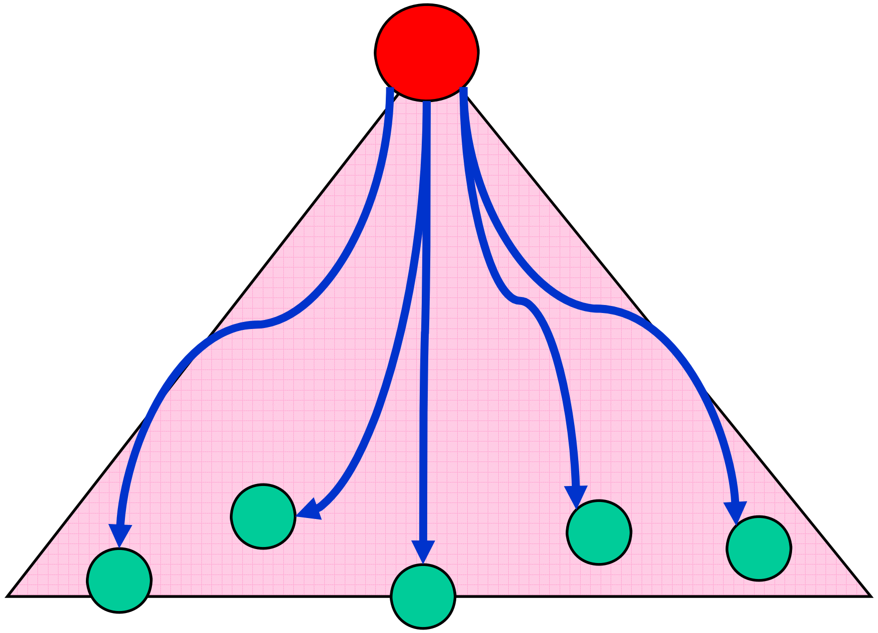
平衡木とは

- 根から、葉までの道の長さが、どの葉に対してもある程度の範囲にある。

  （厳密な定義は、各々の平衡木毎に定義される。概して、平衡木の高さは、$O(\log n)$ である。）

- 平衡木に対する各操作は、最悪計算時間で $O(\log n)$ 時間にできることが多い。

平衡木のイメージ



ほぼ完全(2分)木に近い形状をしている。

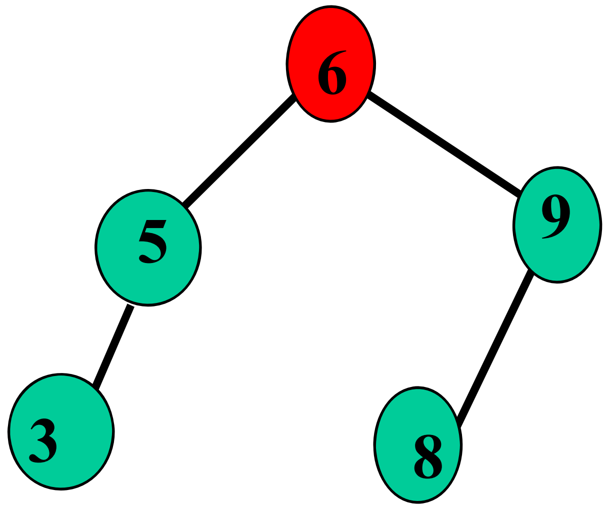
葉までの経路長がほぼ等しい。

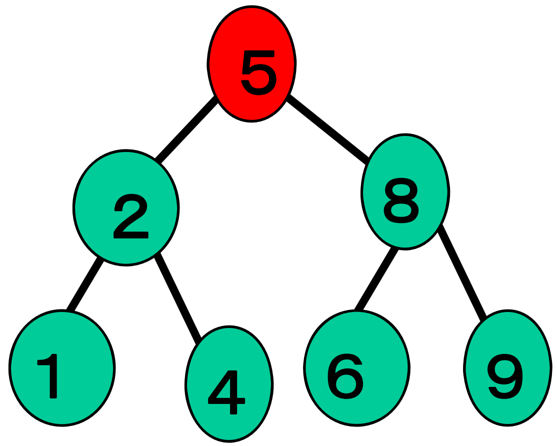
AVL木

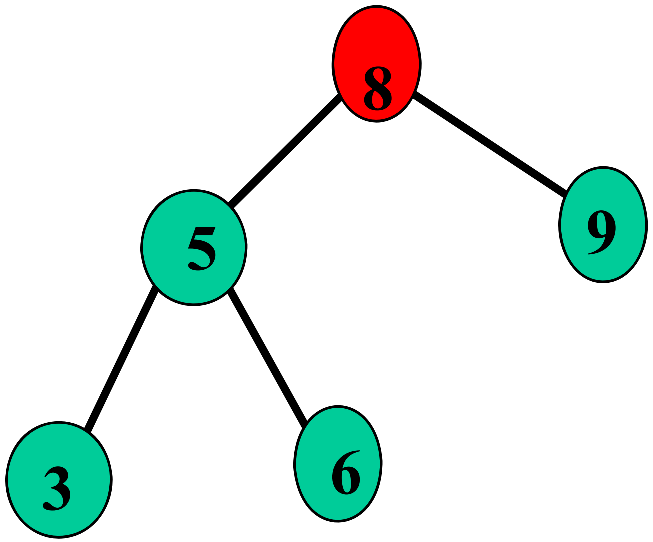
- Adel'son-Vel'skiiとLandisが考案したデータ構造
- 探索、挿入、削除の操作が最悪でも、
  $O(\log n)$時間で行える2分探索木の一種。
- 全てのノードにおいて、左部分木と右部分木の高さの差が1以内に保つ。

最後の、性質を保つために、バランス回復
操作を行う。
また、この性質より、高性能となる。

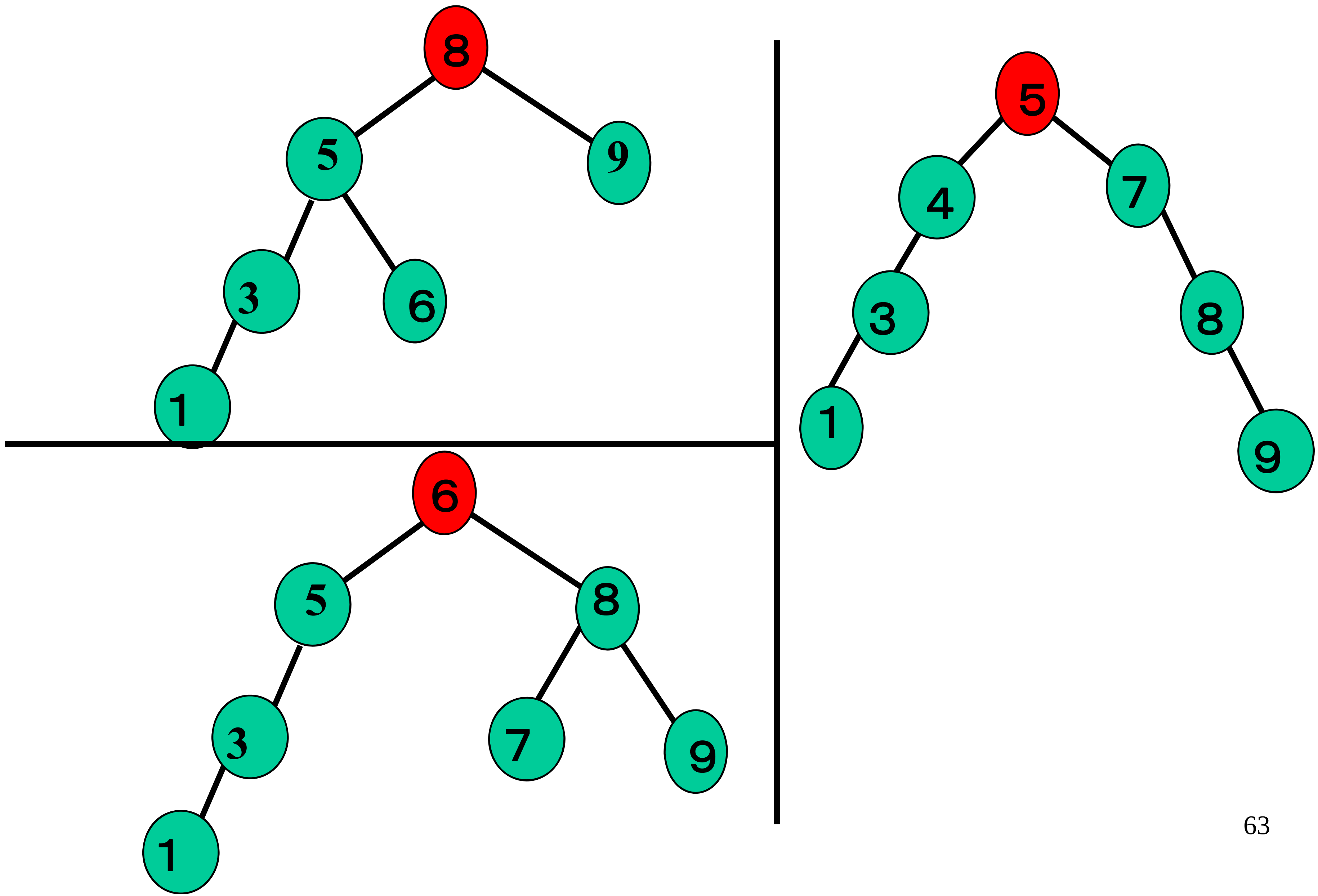
様々なAVL木







AVL木でない例



AVL木の高さの導出

- 「各ノードにおいて、右部分木の高さと左部分木の高さの差が高々1」

という条件からAVL木の高さが、

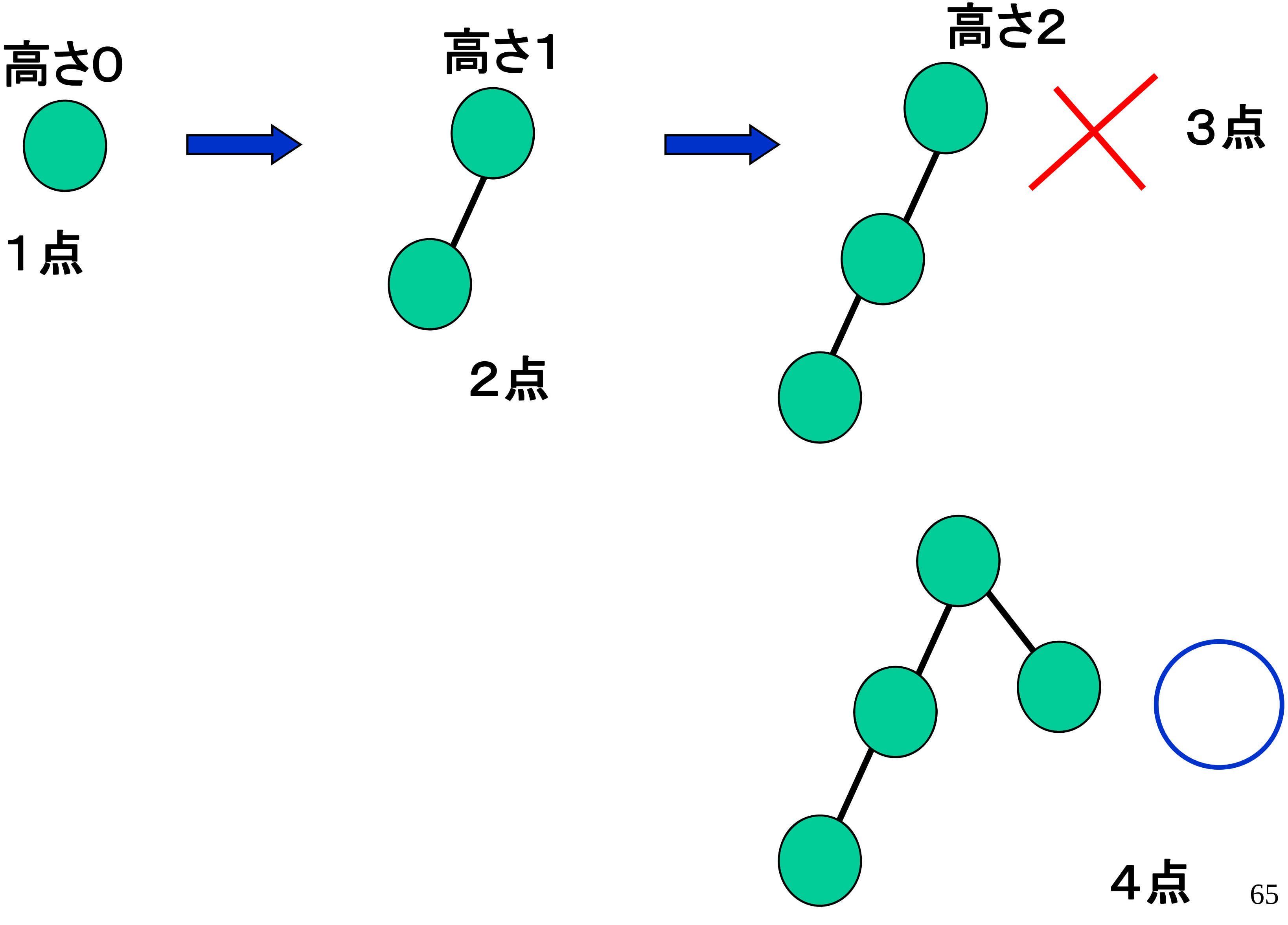
$$O(\log n)$$

になることが導かれる。

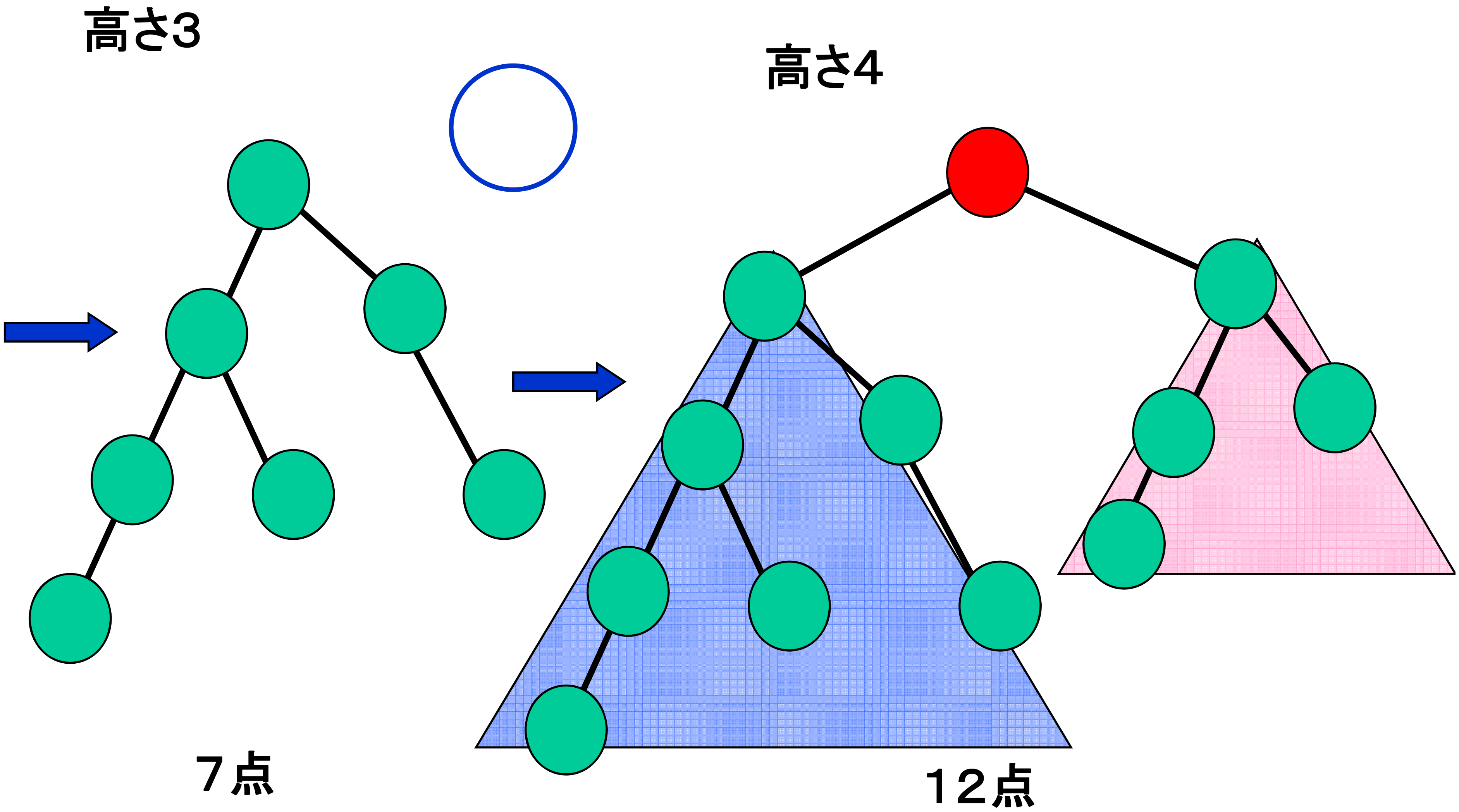
AVL木の
バランス条件

ここでは、できるだけ少ないノードで、
高さを増加させることを考える。

少ないノードのAVL木１



少ないノードのAVL木２



高さ h のAVL木を実現する最小のノード数を $N(h)$ と表す。

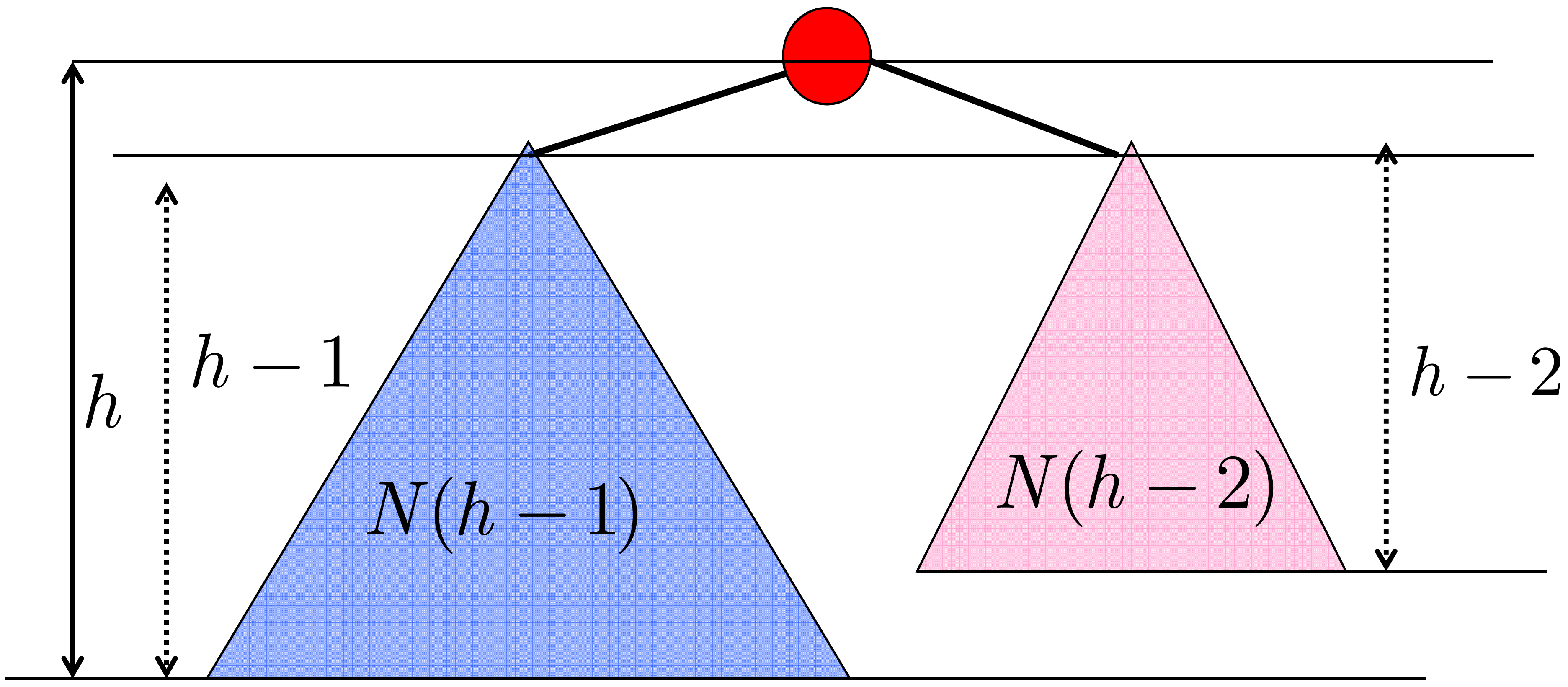
例より、

$$N(0) = 1, N(1) = 2, N(3) = 4, N(4) = 7, \cdots$$

という数列になるはずである。

ここで、この数列 $N(h)$ が満たすべき漸化式を導く。

高さ h を実現する最小ノード数のAVL木



$$\therefore N(h) = N(h-1) + N(h-2) + 1$$

左部分木の点数
(右部分木の点数)

右部分木の点数
(左部分木の点数)

根

以上の考察より、次の漸化式が成り立つ。

$$\begin{cases} N(0) = 1 & h = 0 \\ N(1) = 2 & h = 1 \\ N(h) = N(h-1) + N(h-2) + 1 & h \geq 2 \end{cases}$$

この漸化式を解けば、高さ h を実現する最小のノード数 $N(h)$ が求められる。

特殊解を N とする。

再帰式より、

$$N = N + N + 1$$

$$\therefore N = -1$$

この同次解を求める。
すなわち、以下の漸化式を満たす解を求める。

$$\widetilde{N}(h) - \widetilde{N}(h-1) - \widetilde{N}(h-2) = 0$$

特性方程式を解く。

$$x^2 - x - 1 = 0$$

$$\therefore x = \frac{1 \pm \sqrt{5}}{2}$$

よって、

$$\alpha \equiv \frac{1+\sqrt{5}}{2}, \beta \equiv \frac{1-\sqrt{5}}{2}$$

と置くと、任意定数 c_1, c_2 を持ちいて、次のようにあらわせる。

$$N(h) = c_1\alpha^h + c_2\beta^h + N = c_1\alpha^h + c_2\beta^h - 1$$

$$N(0) = c_1 + c_2 - 1 = 1$$

$$N(1) = c_1 \frac{1+\sqrt{5}}{2} + c_2 \frac{1-\sqrt{5}}{2} - 1 = 2$$

これを解いて、

$$c_1 = \frac{2+\sqrt{5}}{\sqrt{5}} = \frac{1}{\sqrt{5}}\alpha^3, c_2 = -\frac{2-\sqrt{5}}{\sqrt{5}} = \frac{-1}{\sqrt{5}}\beta^3$$

$$\therefore N(h) = c_1\alpha^h + c_2\beta^h + N = \frac{1}{\sqrt{5}}\left(\alpha^{h+3} - \beta^{h+3}\right) - 1$$

これより、n 点のAVL木の高さは、次式を満たす。

$$\therefore N(h) \leq n$$

これより、

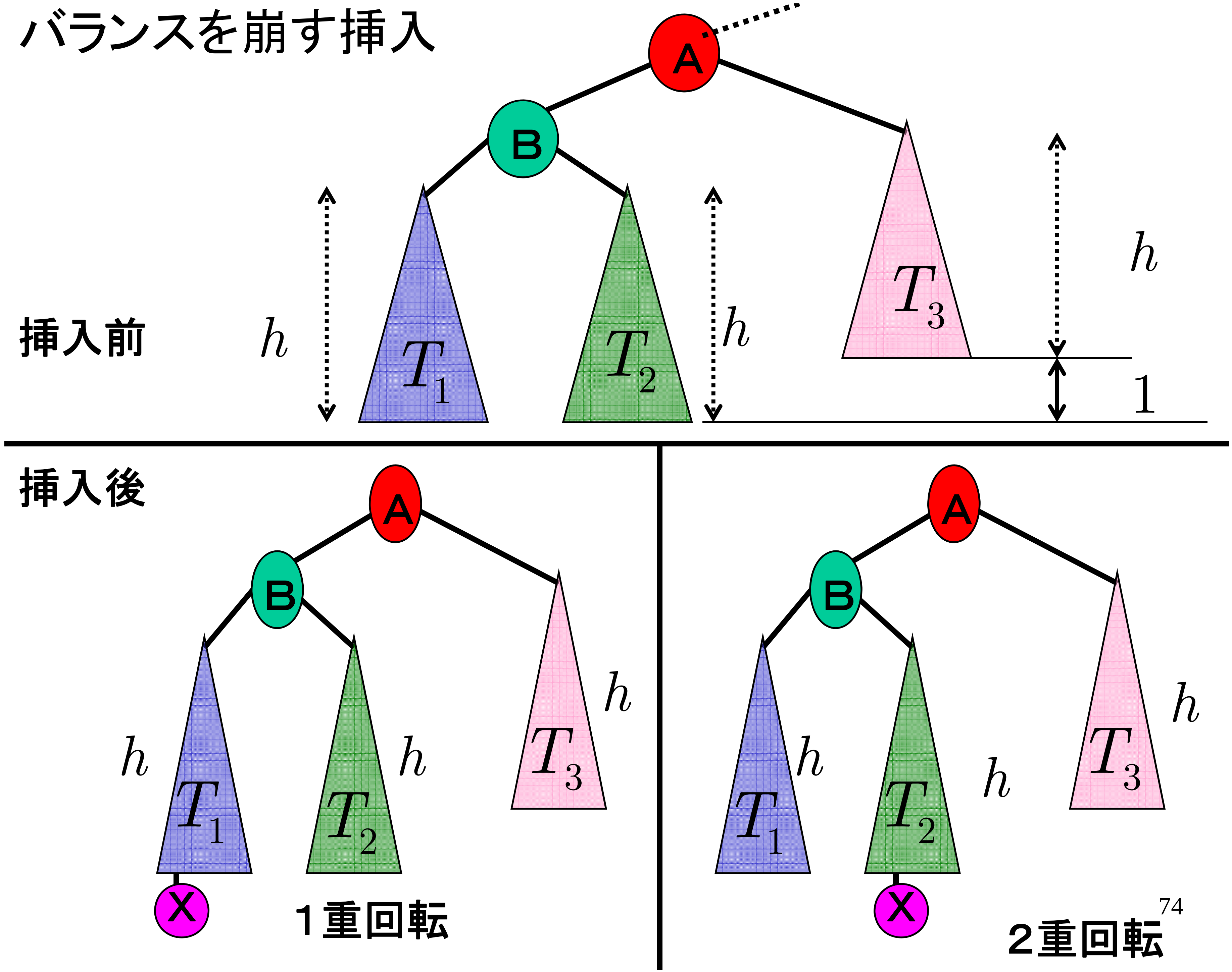
$$h = O(\log n)$$

と高さを導くことができる。
（この評価は、最悪時も考慮されていることに注意する。）

ＡＶＬへの挿入

- 挿入によっても、ＡＶＬのバランス条件を満足していれば、通常の２分探索木の挿入をおこなう。
- 挿入によりバランス条件を破ってしまったとき、挿入状況により、バランス回復操作をおこなう。
  - １重回転操作
  - ２重回転操作

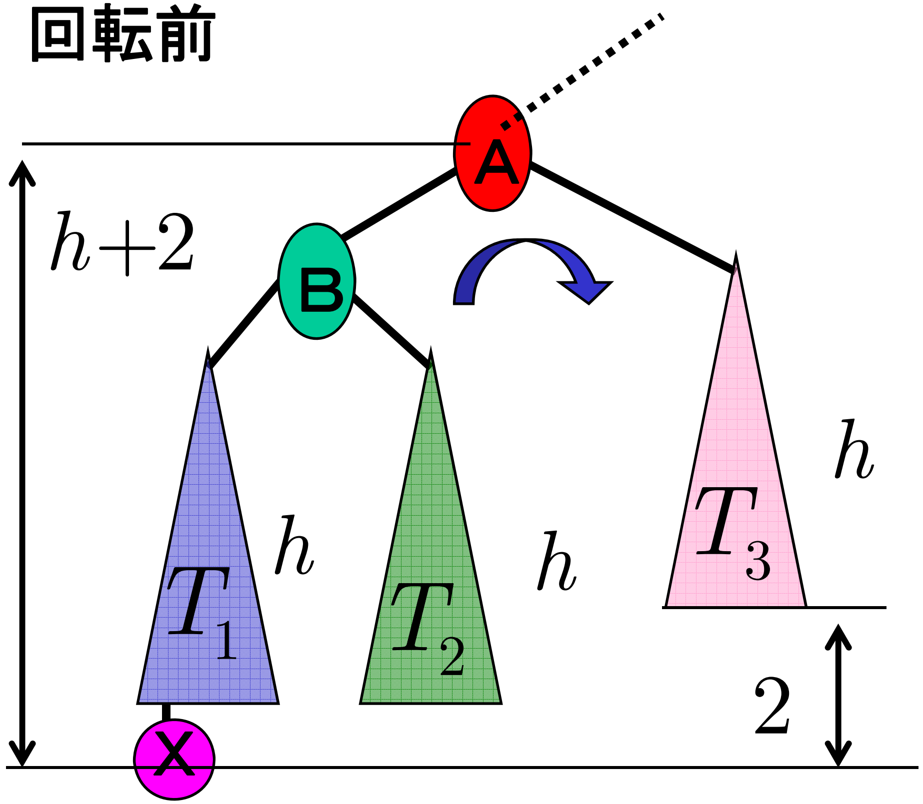
バランスを崩す挿入



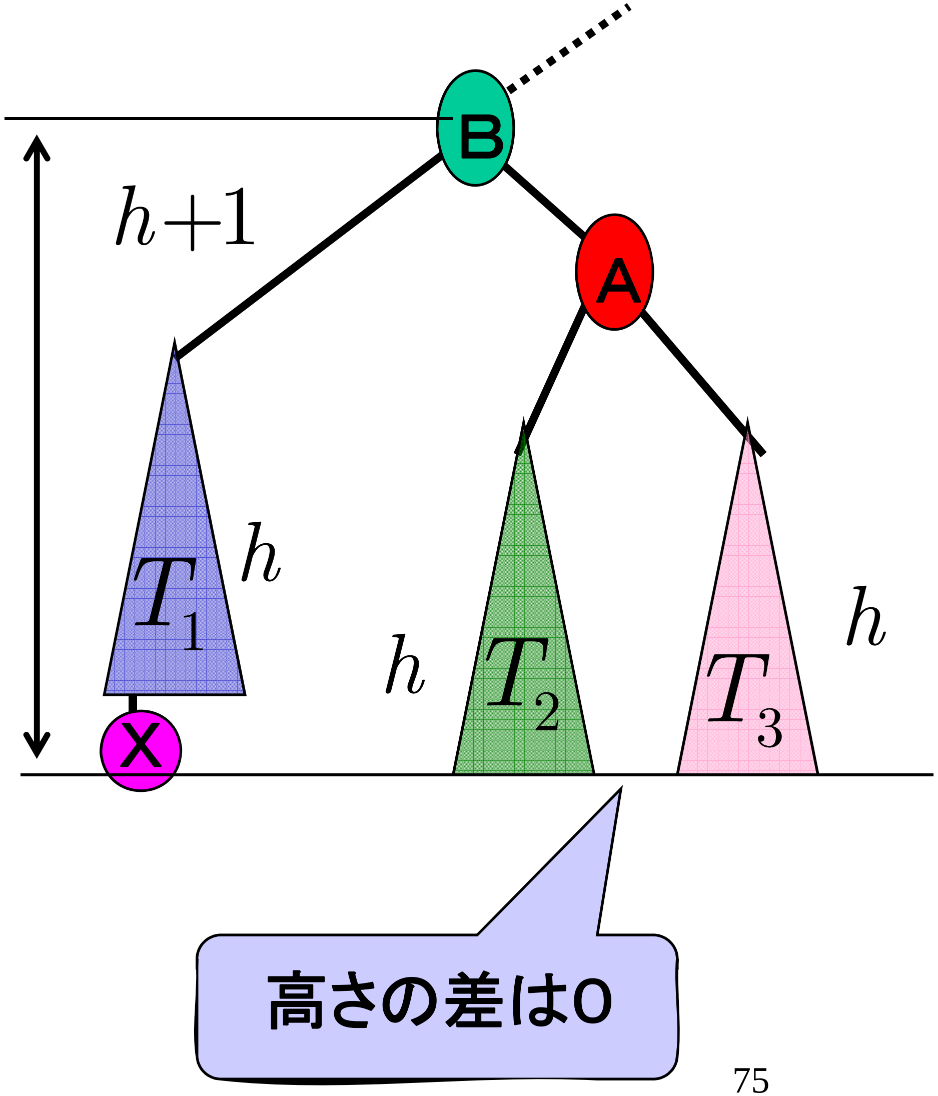
A
B
挿入前
h
T_1
T_2
h
T_3
h
1
挿入後
A
B
h
T_1
T_2
h
T_3
h
X
１重回転
A
B
T_1
h
T_2
h
T_3
h
X
２重回転

１重回転

回転前

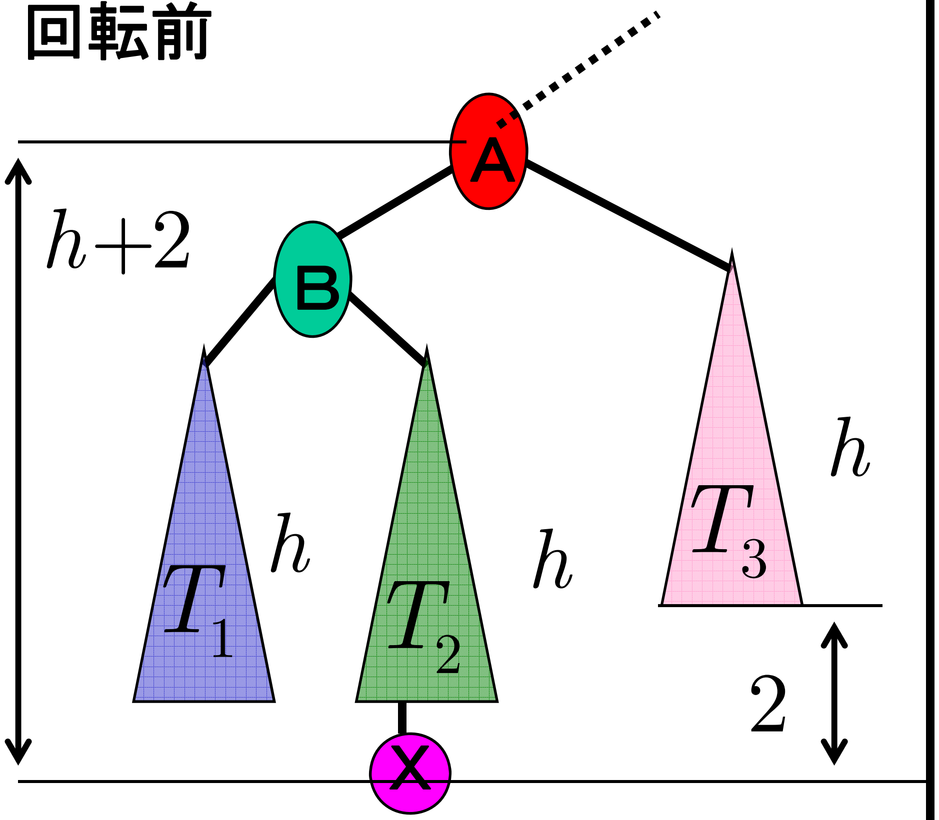


回転後

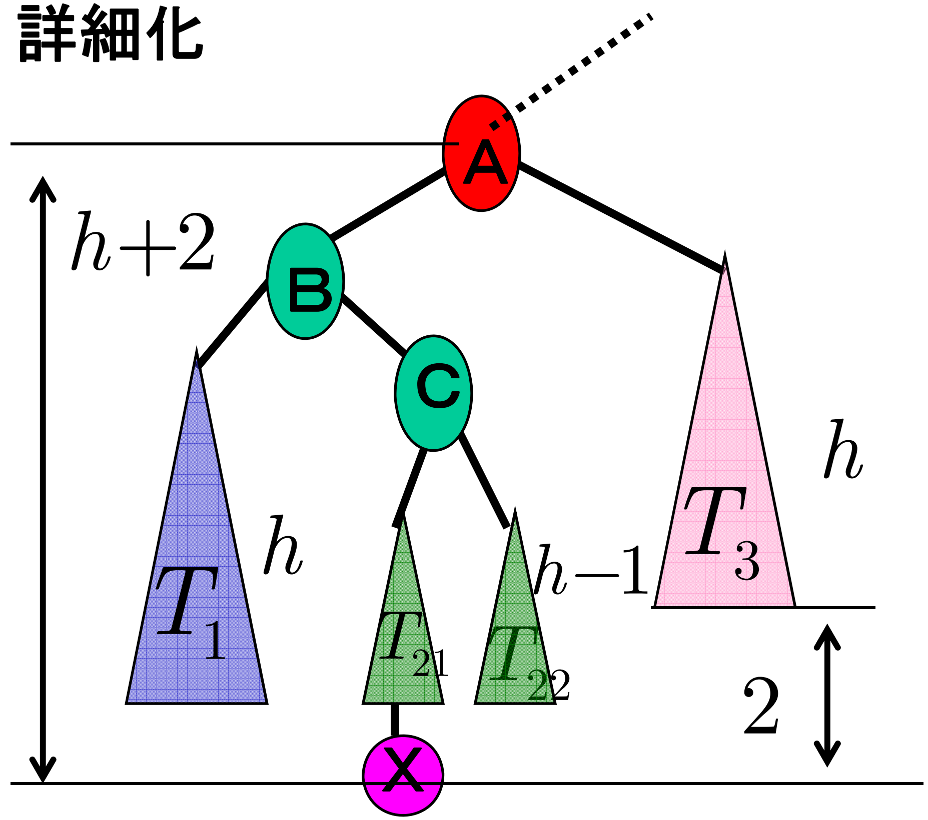


２重回転１

回転前

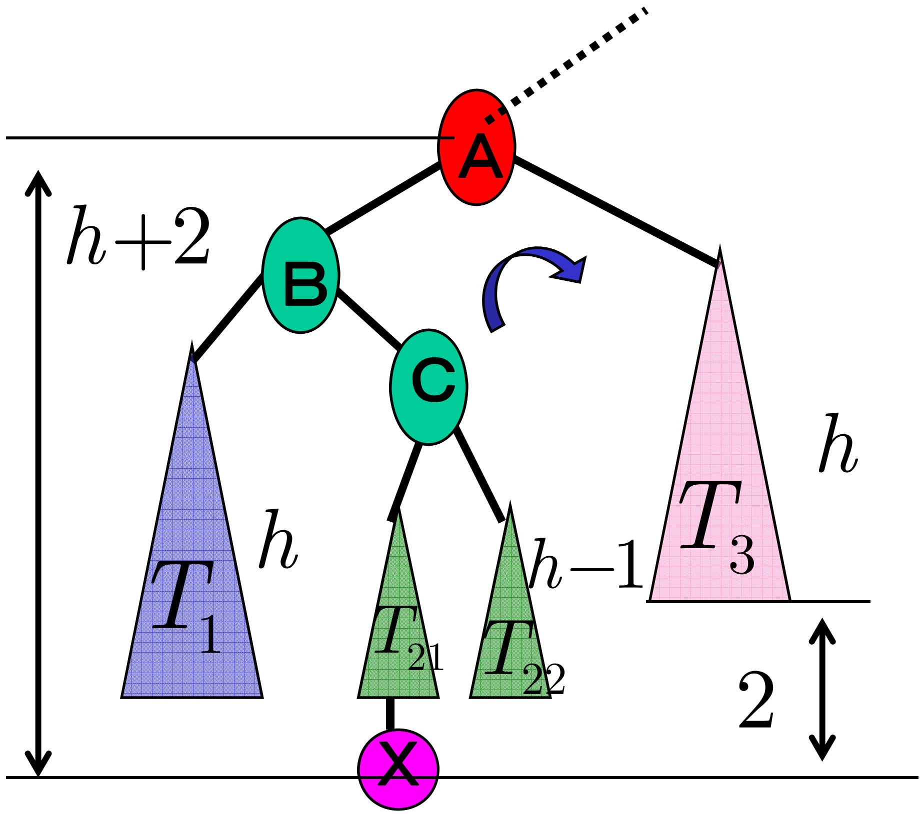


詳細化

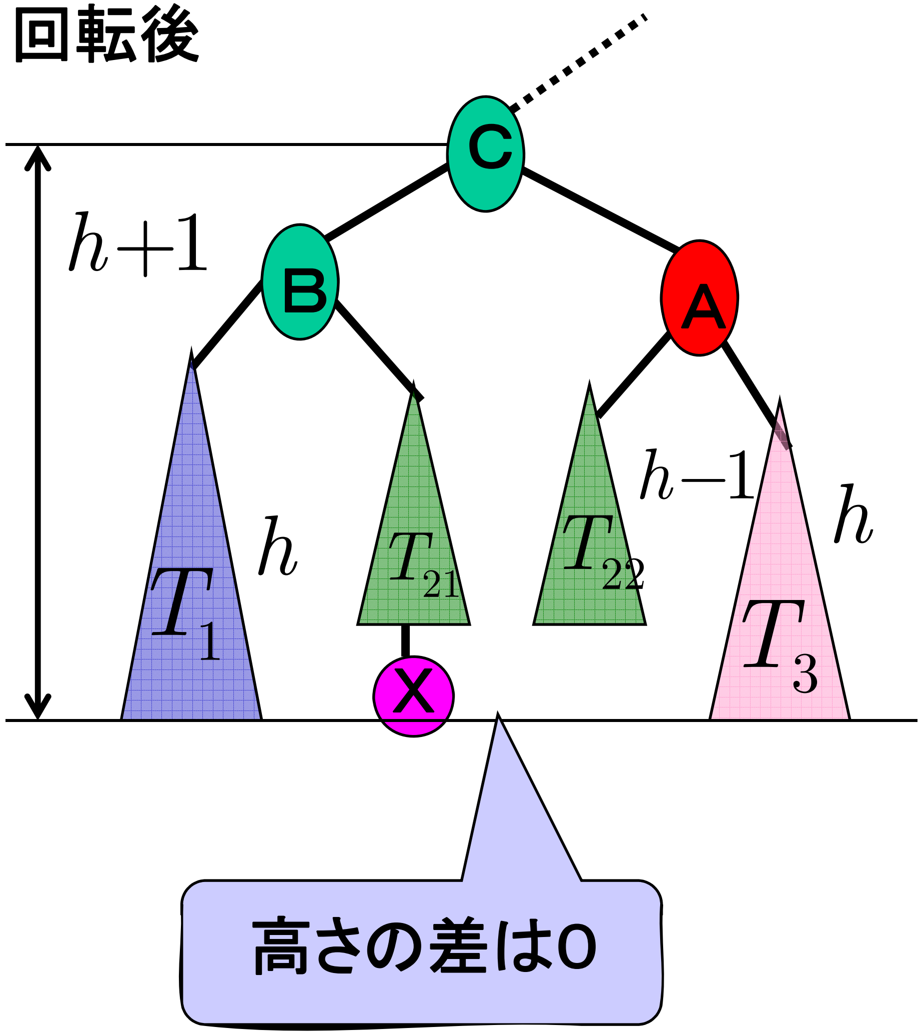


２重回転2

回転前

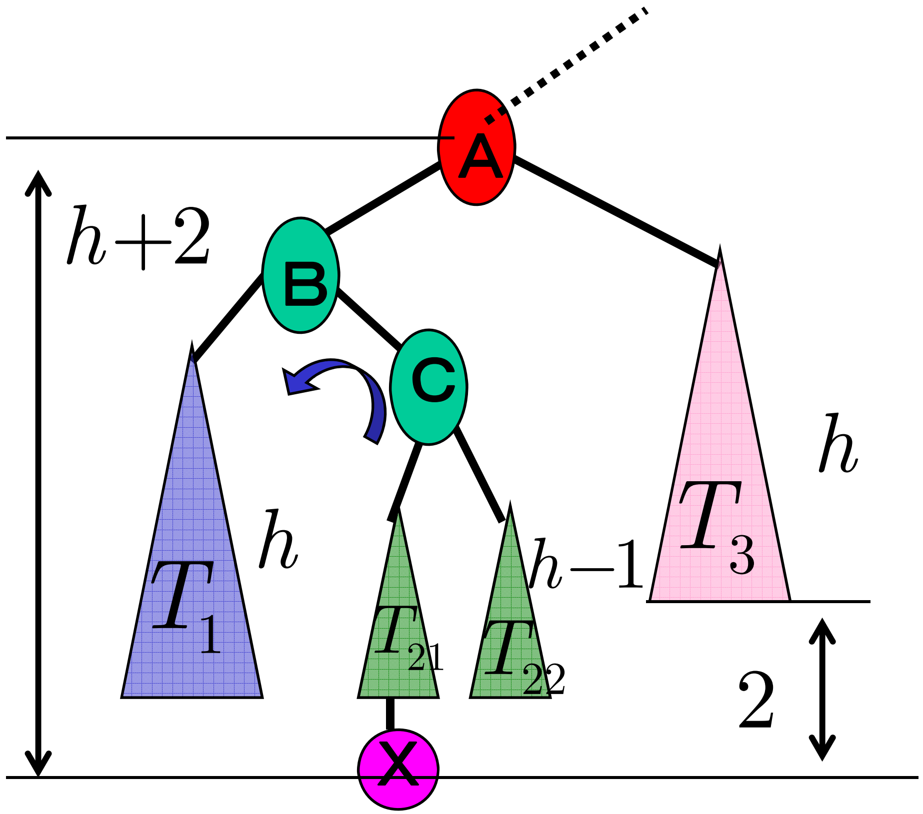


回転後

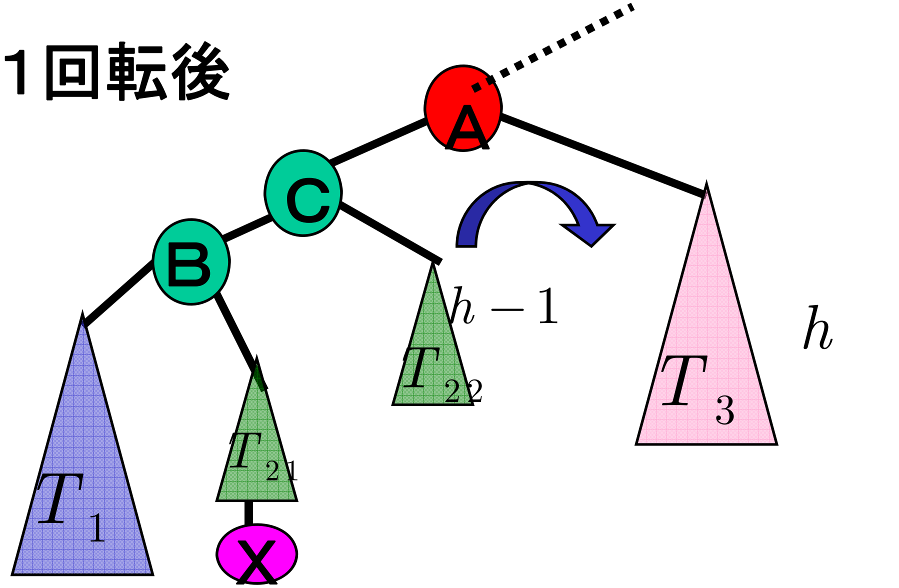


１重回転2回での２重回転の実現

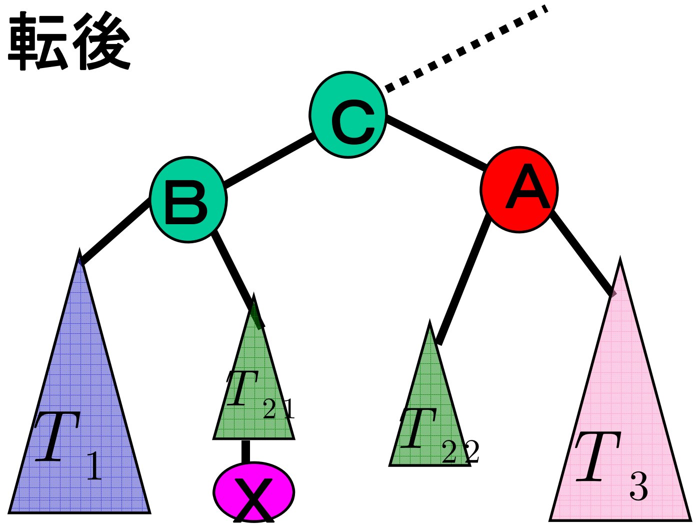
回転前



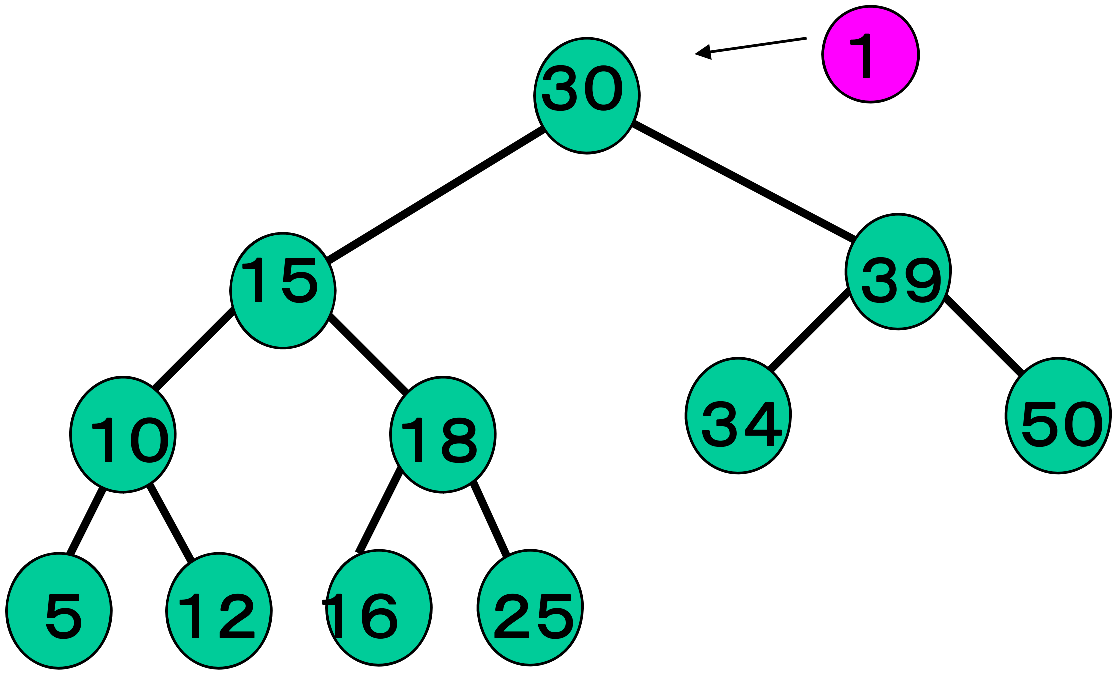
1回転後

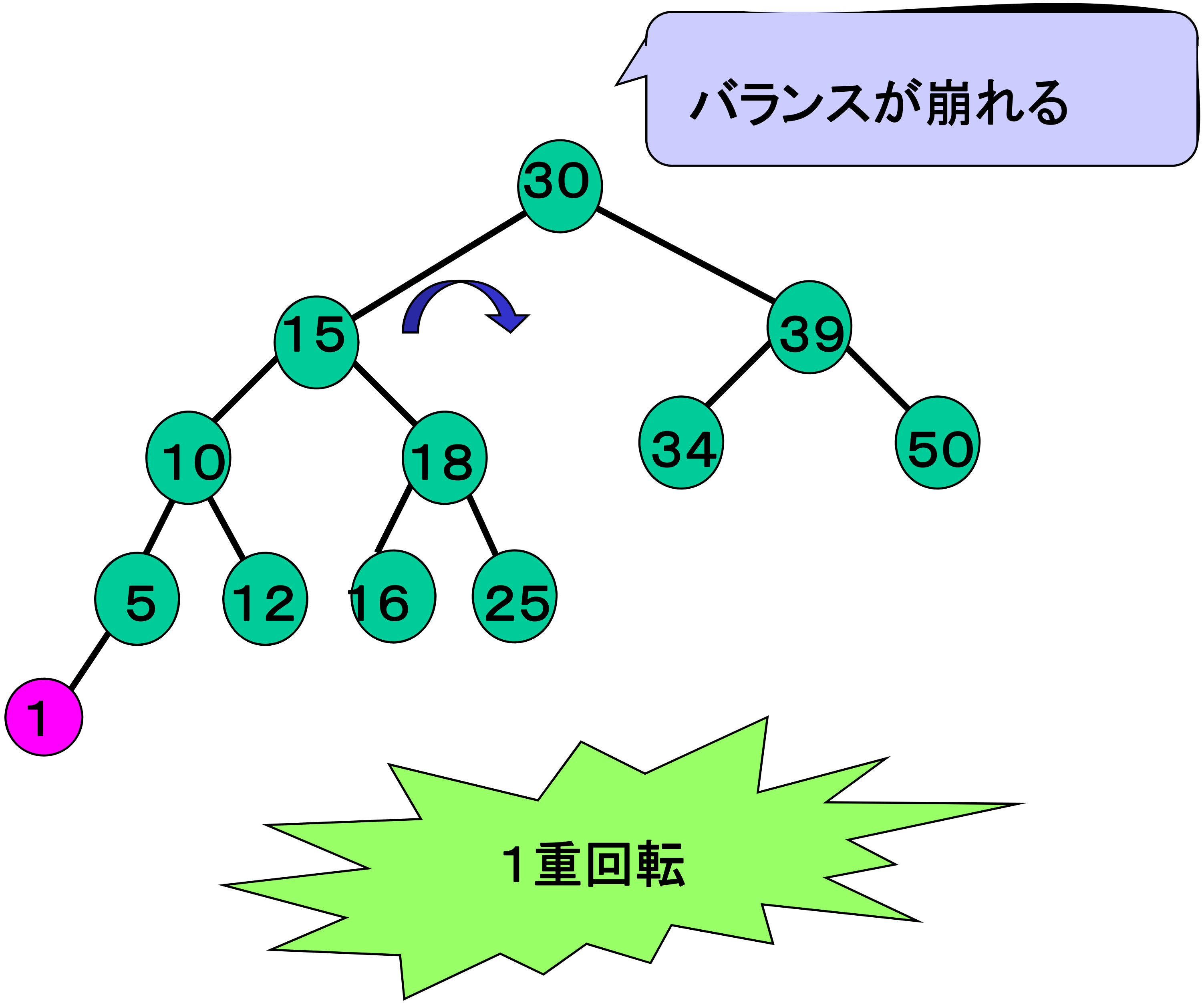


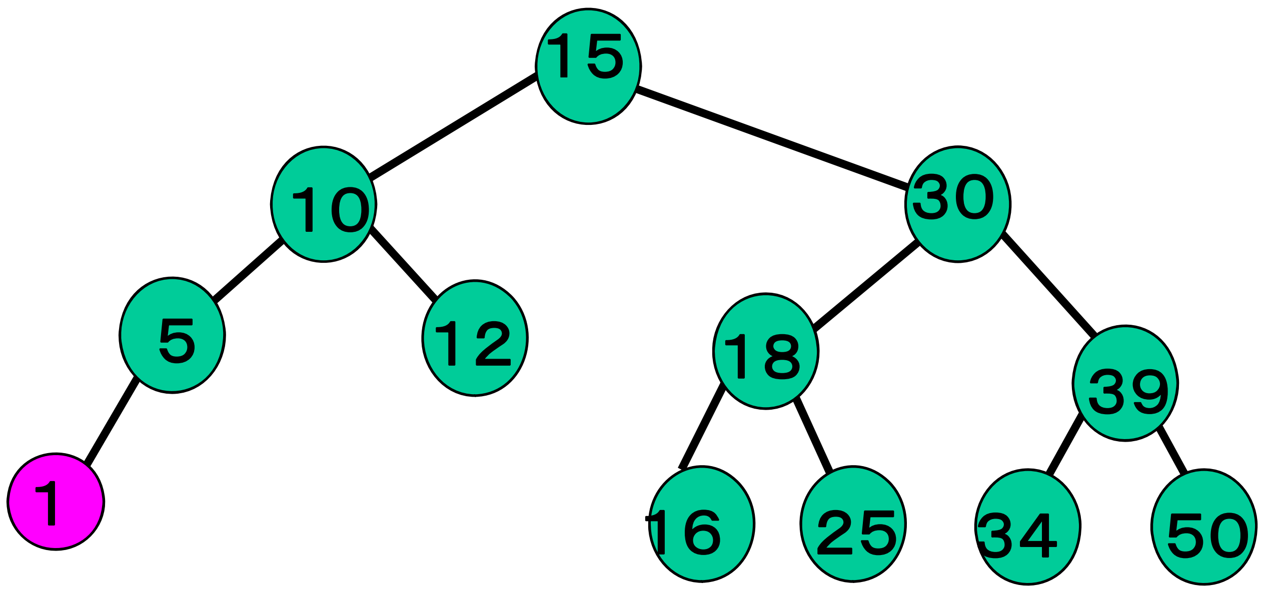
2回転後



AVL木への挿入例1

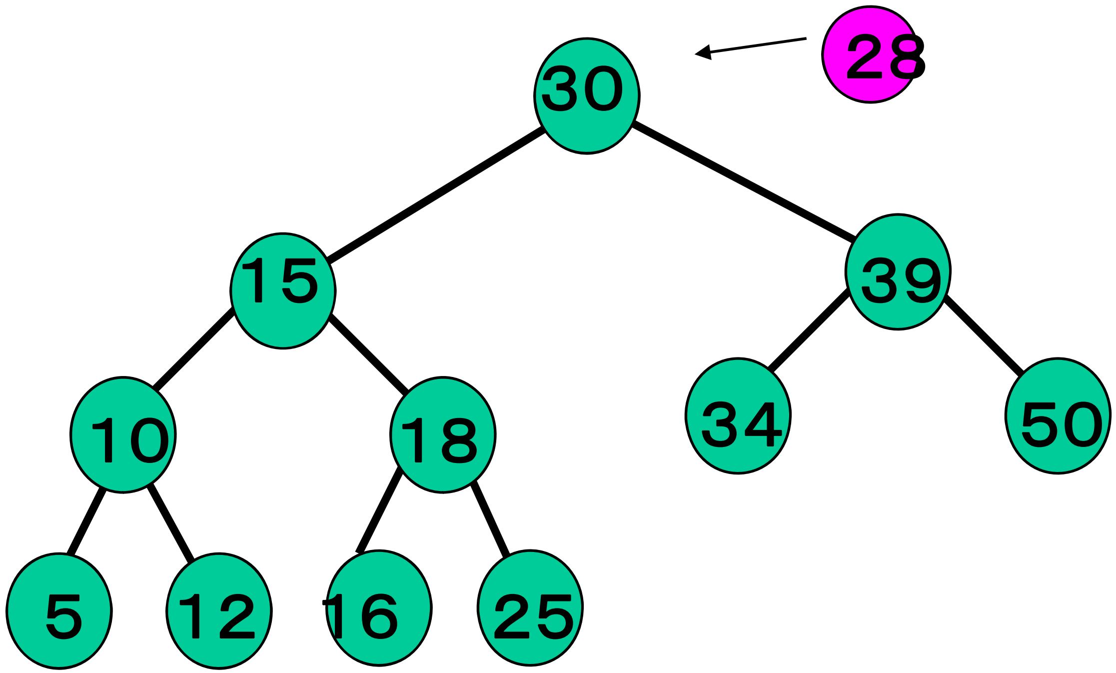


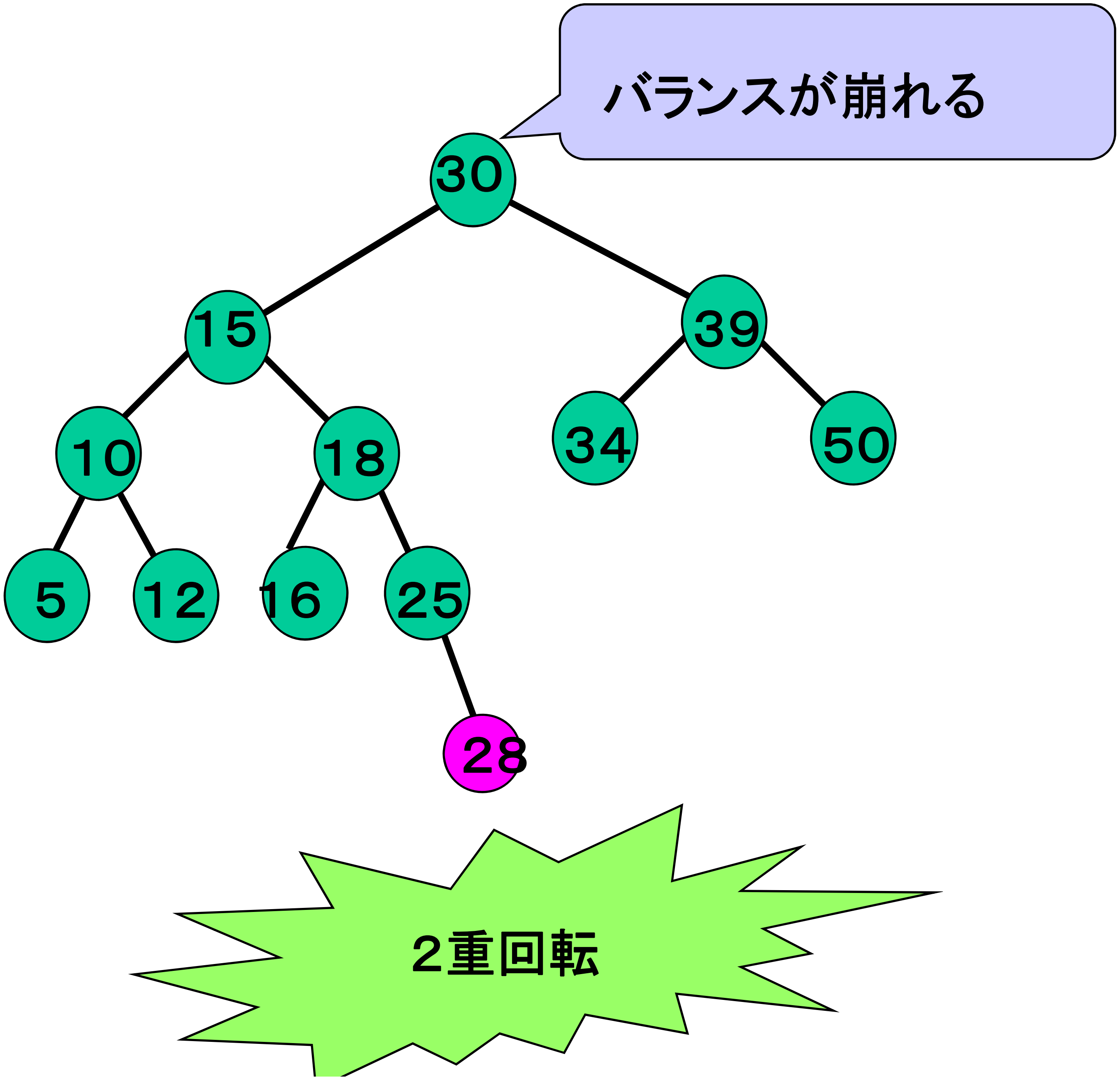

バランスが崩れる
30
15
39
10
18
34
50
5
12
16
25
1
１重回転


15
10
30
5
12
18
39
1
16
25
34
50

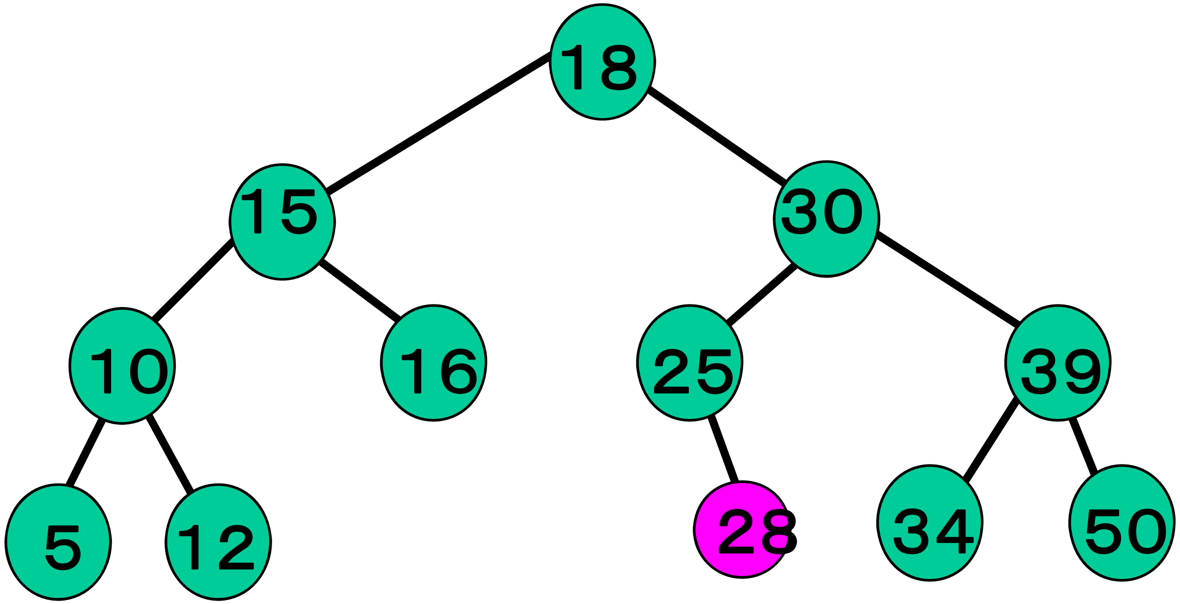

１重回転後

AVL木への挿入例2



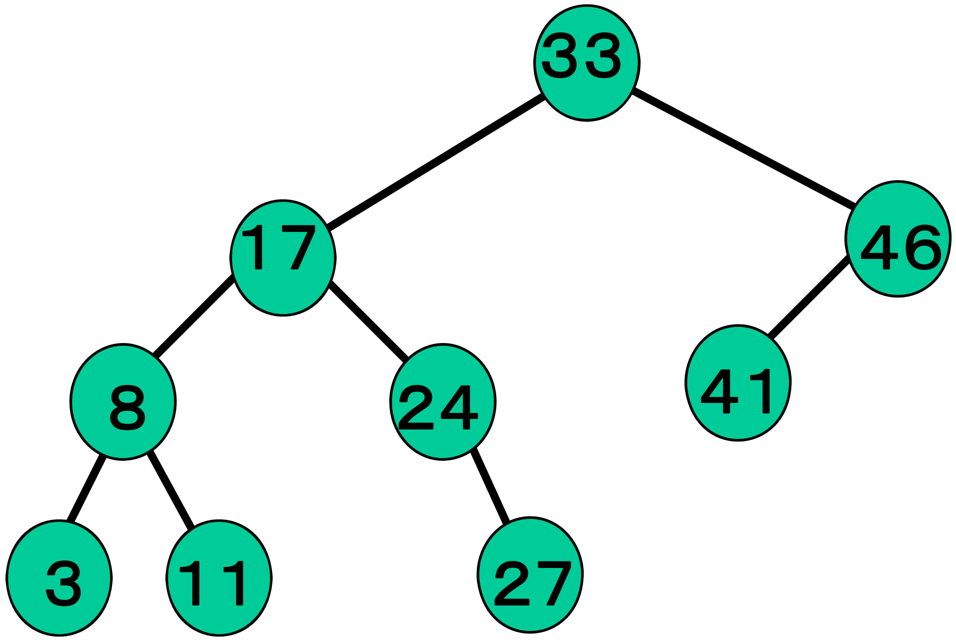

バランスが崩れる
30
15
39
10
18
34
50
5
12
16
25
28
2重回転

バランスが崩れる



練習

次のAVL木に、各要素を順に挿入した結果を示せ。



$$28 \to 10 \to 35 \to 23$$

AVLへの挿入の計算量

- 挿入位置の確認とバランス条件のチェックに、木の高さ分の時間計算量が必要である。
- また、回転操作には、部分木の付け替えだけであるので、定数時間($O(1)$ 時間)で行うことができる。
- 以上より、挿入に必要な最悪時間計算量は、

$$O(\log n)$$

である。

AVLへの削除の計算量

- 削除時に、バランス条件が崩された場合も、挿入時と同様に、回転操作によって、バランスを回復することができる。
- 削除位置を求めることと、バランス条件のチェックに、木の高さ分の時間計算量が必要である。
- 以上より、削除に必要な**最悪**時間計算量も、

$$O(\log n)$$

である。

B木の概略

- 多分木（ d 分木）を基にした平衡木
- 各ノードには、データそのものと、部分木へのポインタを交互に蓄える。
- 各葉ノードまでの道は全て等しい。

（したがって、明らかに平衡木である。）

- 部分木中の全てのデータは、親ノードのデータで範囲が限定される。

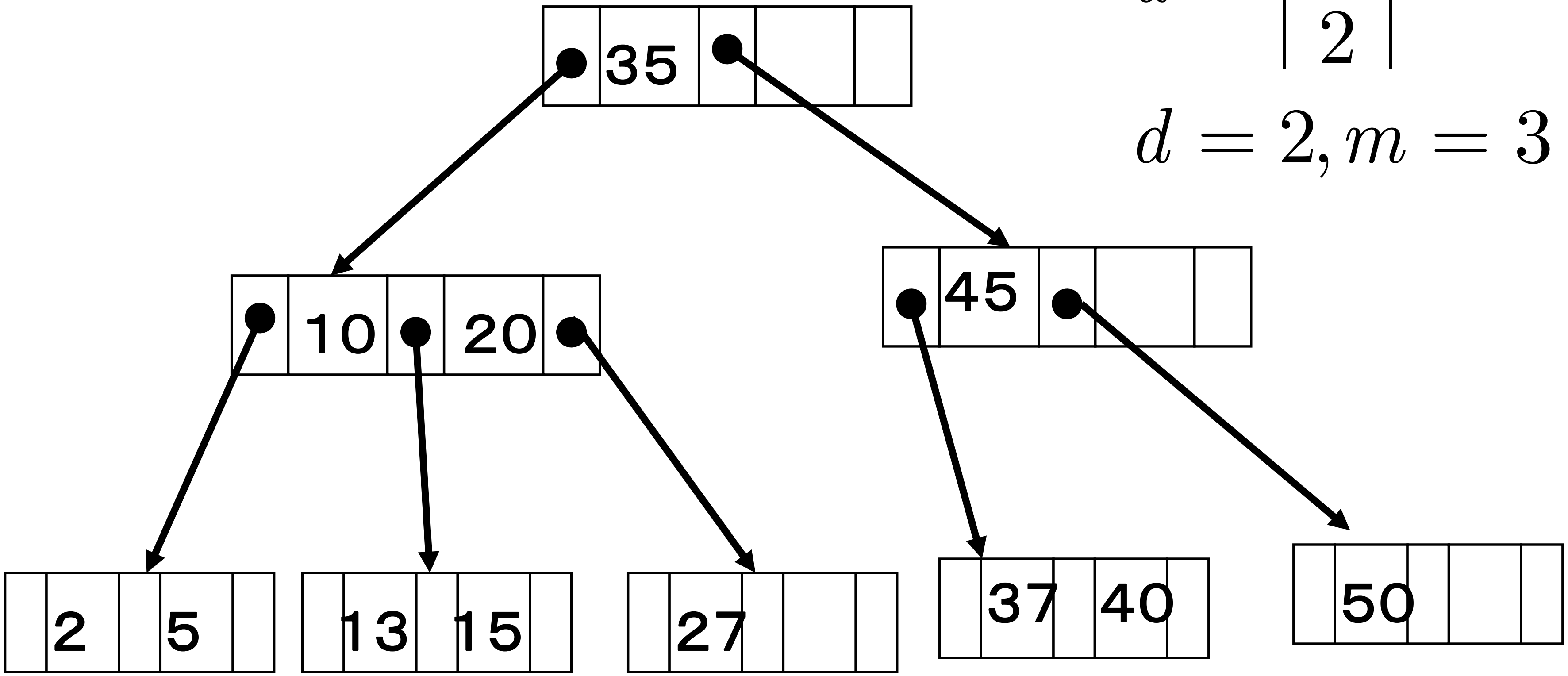
B木の満たすべき条件

- ①根は、葉になるかあるいは $2 \sim m$ 個の子を持つ。
- ②根、葉以外のノードは、$\left\lceil \frac{m}{2} \right\rceil \sim m$ 個の子を持つ。
- ③根からすべての葉までの道の長さは等しい。
- ④部分木全てのデータは、その部分木へのポインタを"はさんでいる"データにより、制限される。

Ｂ木の例

$$d = \left\lceil \frac{m}{2} \right\rceil$$

$$d = 2, m = 3$$



B木の高さ

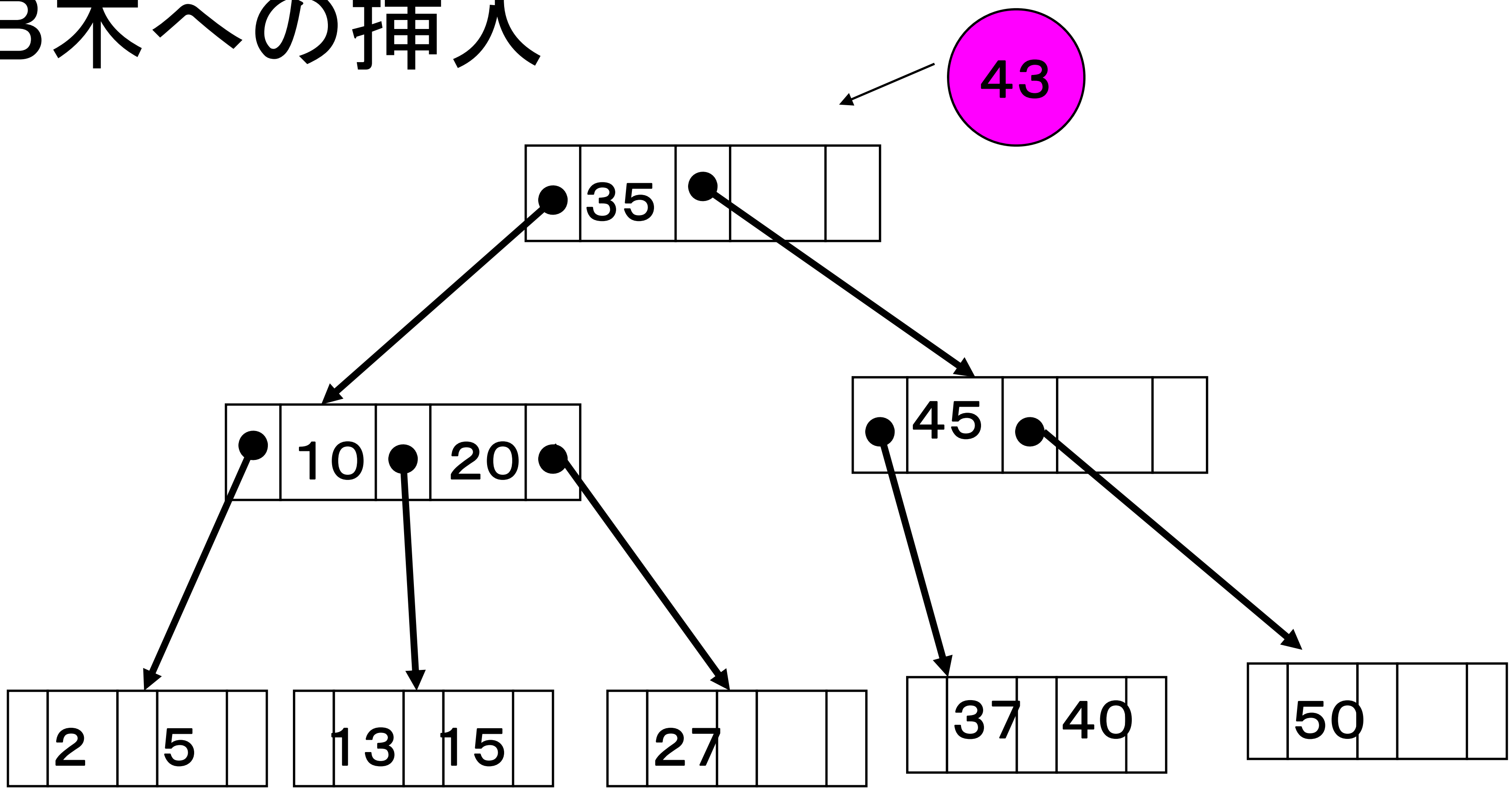
簡単のため、根以外は、 d 個以上の個があるとする。

このとき、高さ h のB木に含まれるノード数を $N(h)$ とする。このとき、次が成り立つ。

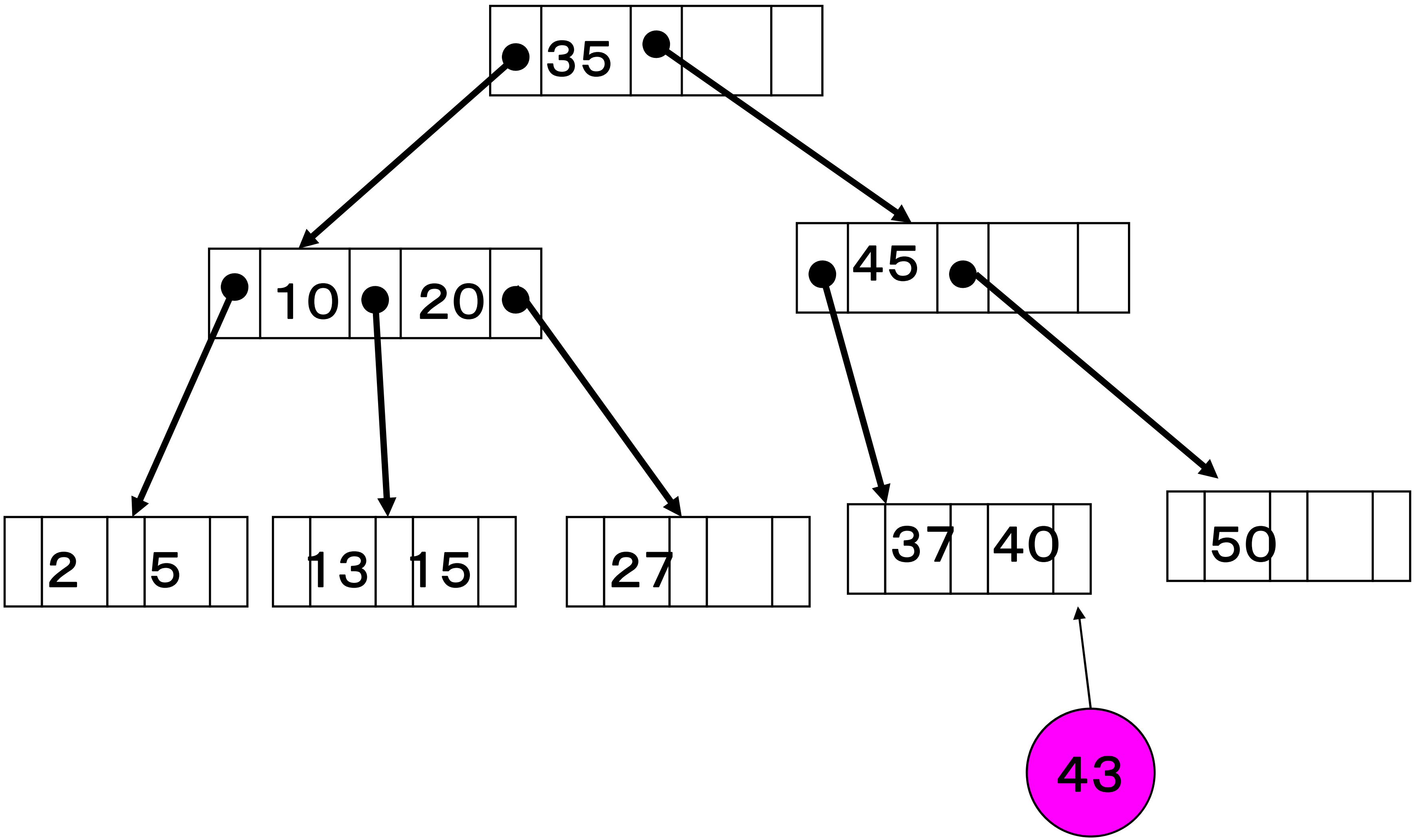
$$n = N(h) \geq \sum_{i=0}^{h} d^i = \frac{d^{h+1} - 1}{d - 1}$$

$$\therefore h = O(\log_d n)$$

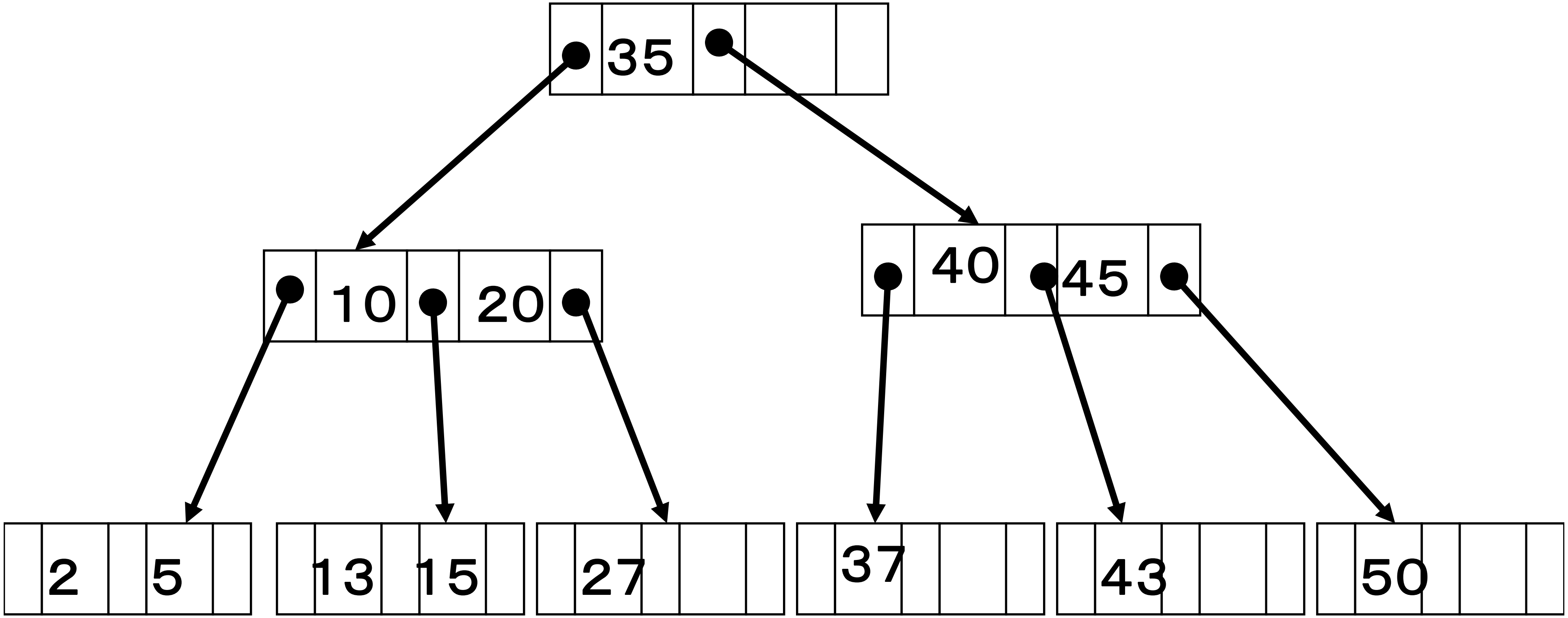
B木への挿入



オーバーフロー時のノード分割1

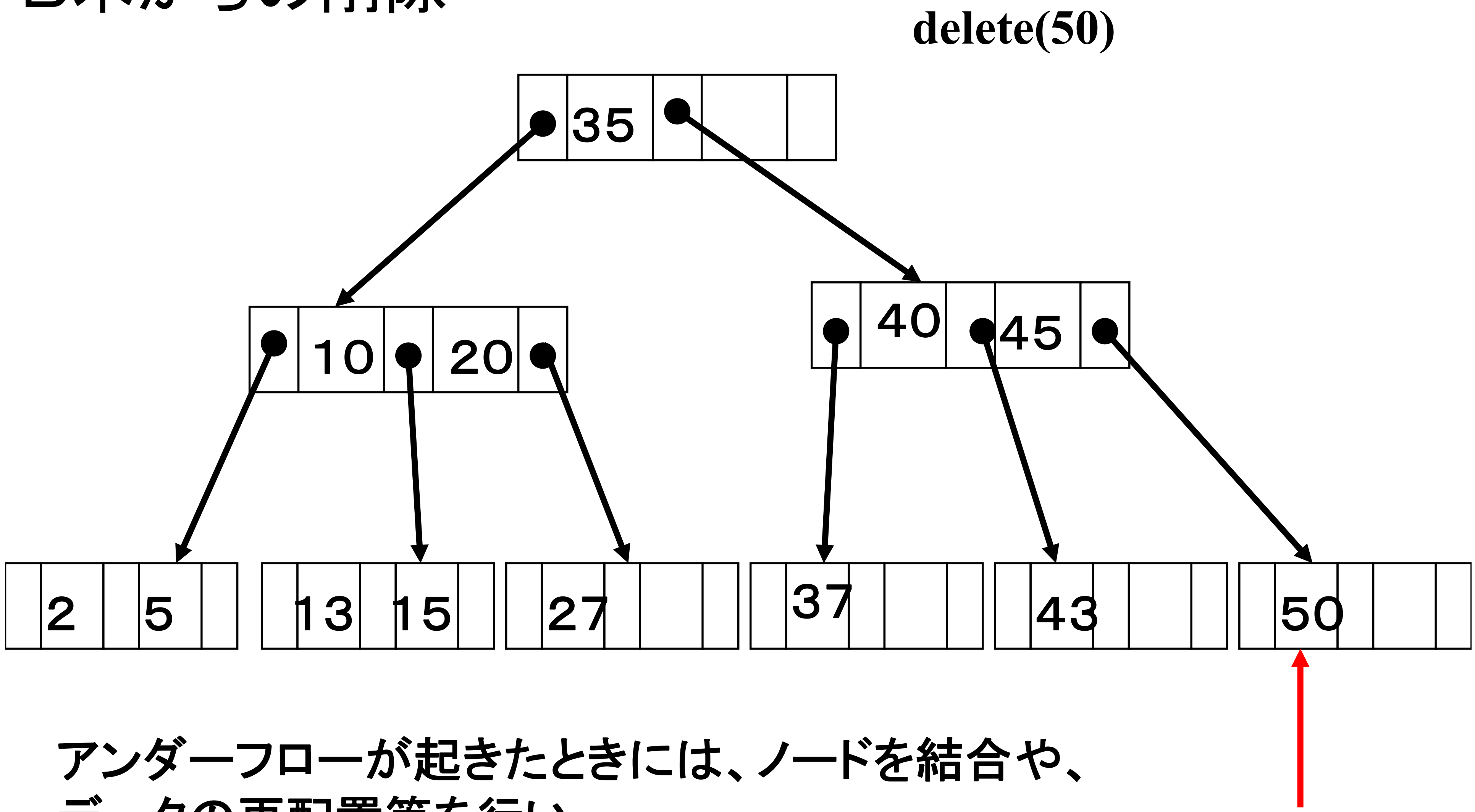


オーバーフロー時のノード分割2



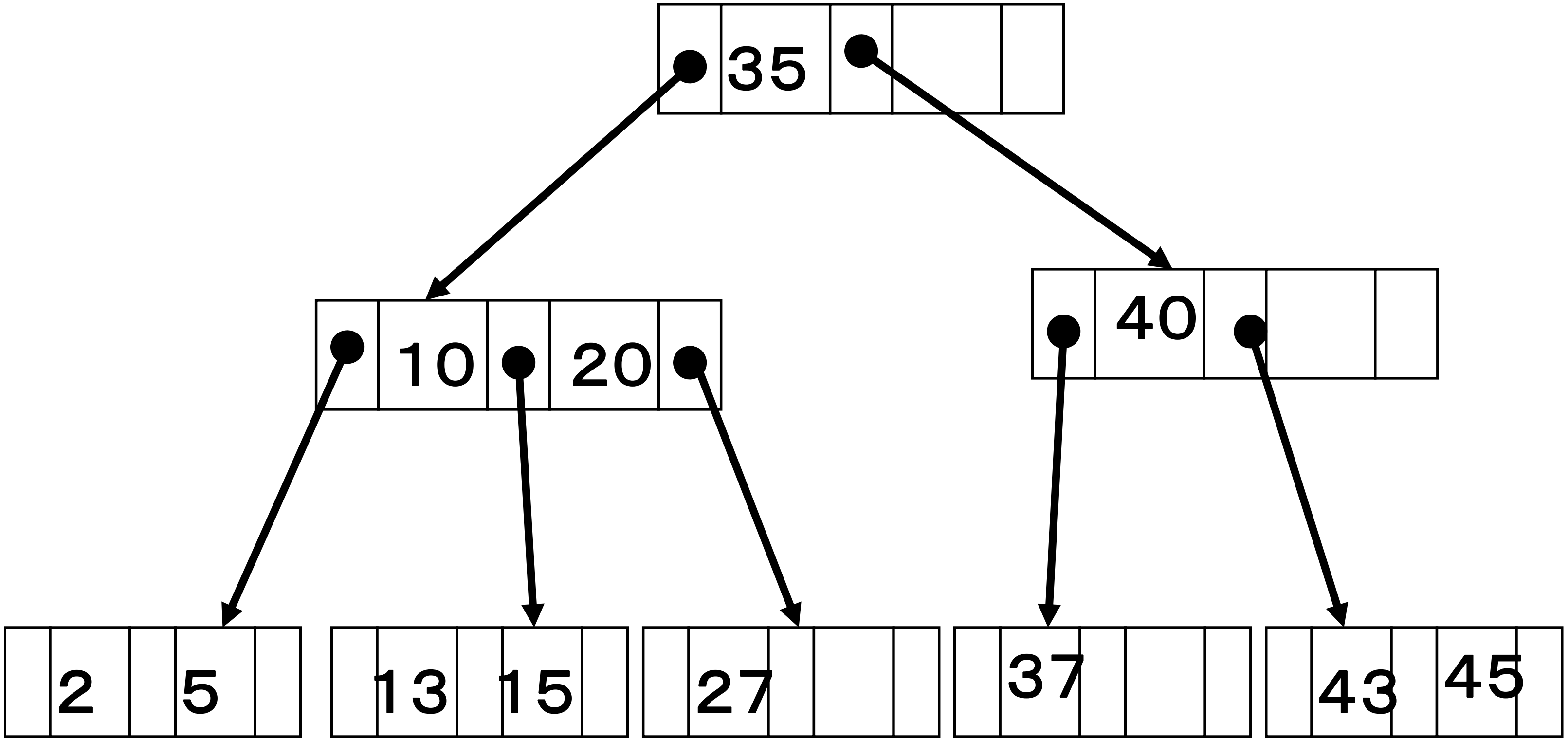
オーバーフローが起きたときには、ノードを分割して、親に向かって再帰的にB木の条件を満足するように更新していく。

B木からの削除



アンダーフローが起きたときには、ノードを結合や、
データの再配置等を行い、
再帰的にB木の条件を満足するように更新していく。

アンダーフローにおけるデータの再配置



アンダーフローが起きたときには、ノードを結合や、
データの再配置等を行い、
再帰的にB木の条件を満足するように更新していく。

Ｂ木の最悪計算量

- Ｂ木の高さが、$O(\log_d n)$ であることに注意する。
- また、1つのノードを処理するために、$O(m)$時間必要である。
- 以上より、各操作は、最悪時間計算量として、

$$O(m + \log_{\lceil \frac{m}{2} \rceil} n)$$

時間である。パラメータ m の値により性能に違いが生じる。$m = \Omega(n)$ とすると高速に動作しない

Ｂ木の応用

- ディスクアクセスは、メモリアクセスに比べて極端に遅い。したがって、ある程度もまとまったデータを１度の読み込んだ方が全体として高速に動作することが多い。
- よって、Ｂ木の各ノードに蓄えられているデータを、一度に読み込むようにすれば、ディスクアクセスの回数が軽減される。
- 各ノード内の処理は、メモリ上で効率よく実現できる。

平衡木のまとめ

- 平衡木の高さは、

$$O(\log n)$$

となる。

- 平衡を実現するための条件により、各種平衡木が定義される。
- 平衡状態を満足するために、各種バランス回復処理が行われる。