

# アルゴリズムとデータ 構造

## 第5回探索のためのデータ構造(1)

# 第5回 探索のためのデータ構造(1)

## ■ 今日の内容:

- 辞書と順序辞書
- 順序辞書の実装1: 整列された配列
- 順序辞書の実装2: 二分探索木
  - 探索, 挿入, 削除演算
  - 演算の計算量
- 予習: 平衡探索木

## ■ ポイント

- 「探索」問題とは何かを理解しよう
- 「二分探索木」とは何か。その条件を理解しよう
- 二分探索木での探索と挿入の操作

## ■ 付録(別ファイル):

- 二分探索木の実装のC言語によるソースコードと解説  
(第3回演習の準備. 自習教材)

# 辞書とは？

次の3つの基本操作を伴う集合 $S$ を**辞書(dictionary)**という。

1. 探索  $\text{search}(S, x) : x \in S$  ならばyes,  $x \notin S$  ならばnoを出力
2. 挿入  $\text{Insert}(S, x) : S$  を  $S \cup \{x\}$  に更新
3. 削除  $\text{delete}(S, x) : S$  を  $S - \{x\}$  に更新

ここでは、辞書に適したデータ構造について学ぶ。

正確には、次で説明する「順序辞書」を学ぶ

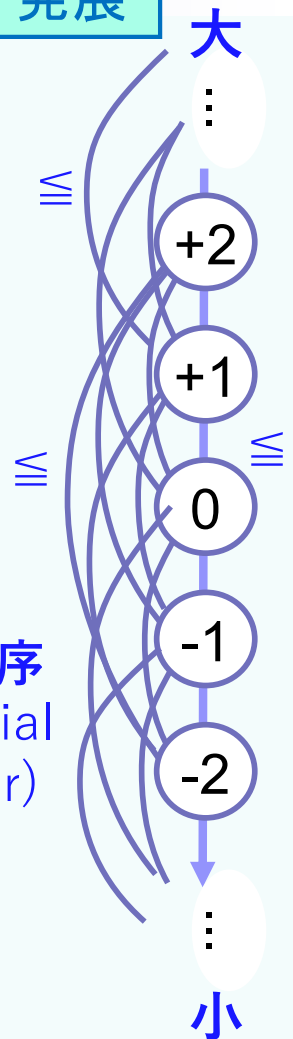
# 発展学習：全順序集合

- $x \leq y$ :  $x$  が  $y$  より小さいこと
- $x < y$ :  $x$  が  $y$  より真に小さいこと

- **定義：二項関係 $\leq$ が全順序 (total order) である**：次の4つの条件が成立すること

- 反射律： $(\forall x) x \leq x$ .
- 推移律： $(\forall x, y, z) x \leq y \text{ and } y \leq z \text{ then } x \leq z$ .
- 反対称律： $(\forall x, y) x \leq y \text{ and } y \leq x \text{ then } x = y$ .
- 比較可能性： $(\forall x, y) x \leq y \text{ or } y \leq x$ .

半順序  
(partial  
order)



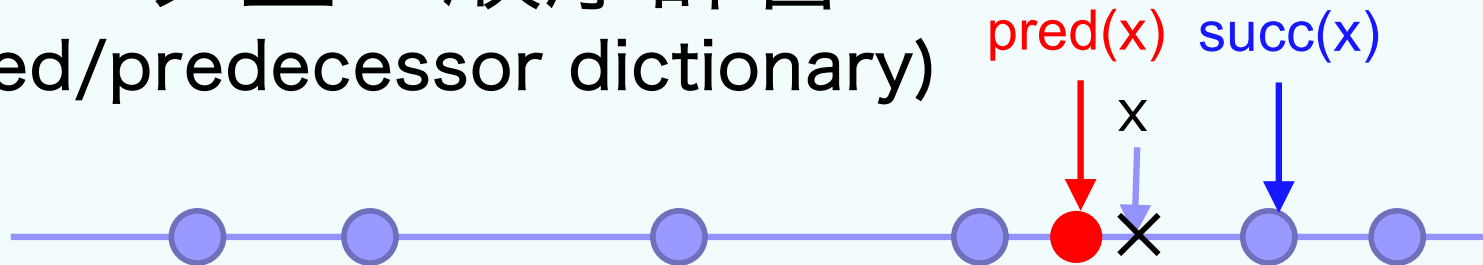
全順序集合 (totally  
ordered set) :

要素間に全順序 $\leq$ が定義されている集合  $A$ .

全順序集合の例

- 自然数の集合  $\mathbb{N}$  と数の大小関係 $\leq$ .
- 文字列の集合  $\Sigma^*$  と辞書式順序 $\leq$ .

# 抽象データ型：順序辞書 (ordered/predecessor dictionary)



全順序集合を格納する抽象データ型:

member, insert, deleteに加えて次の演算をもつ辞書

- 先行者演算 (predecessor, upperbound) :
  - **pred(x)**: 入力キーx以下の最大の値を返す  
(=キーの値か, 直前の値)
- 後続者演算 (successor, lowerbound)
  - **succ(x)**: 入力キーx以上の最小の値を返す  
(=キーの値か, 直後の値)
- データが含まれないときも, 直前の値を返す必要がある. 辞書よりも実現が難しい.
- Ordered map と呼ばれる
  - 例: C++やJavaのordered map

二分探索木は, 先行者辞書を実装するデータ構造

# 整列された配列 (辞書の素朴な実装)

S: 全順序集合

$A[0], A[1], \dots, A[n-1]$ : 配列AにSの要素を小さい順に格納

$\text{search}(S, x)$ : **2分探索(binary search)**が適用可能  
(最悪時間計算量  $O(\log n)$ )

$\text{insert}(S, x)$ ,  $\text{delete}(S, x)$ : 配列への挿入, 削除操作を伴う  
(最悪時間計算量  $O(n)$ )

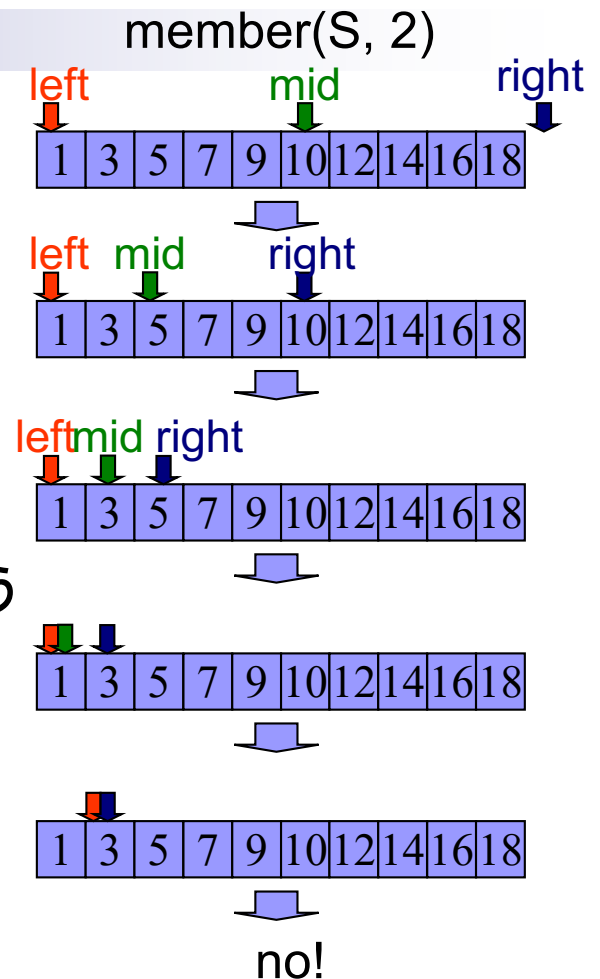
[2分探索による  $\text{search}(S, x)$  操作]

Step 1  $\text{left} \leftarrow 0, \text{right} \leftarrow n$

Step 2  $\text{left} \geq \text{right}$  であれば **no** を出力して停止

Step 3  $\text{mid} \leftarrow \lfloor (\text{left} + \text{right}) / 2 \rfloor$

Step 4 if  $x < A[\text{mid}]$  then  $\text{right} \leftarrow \text{mid}$   
 else if  $x = A[\text{mid}]$  then **yes** を出力して停止  
 else  $\text{left} \leftarrow \text{mid} + 1$   
 Step 2 へ



このループを最悪 高々  $(\log_2 n) + 1$  回まわる!

1回で範囲を半分以下に絞れる。  
 $k$ 回で範囲を  $n/2^k$  以下に絞れる。  
 範囲が空になったら終わりだから  
 $n/2^k < 1$ 。よって  $\log_2 n < k$ 。これを満たす最小の整数は  $(\log_2 n) + 1$  以下。

発展: 整列された配列を用いて, 順序辞書も実現できる. 考えてみよう

# 2分探索木

(辞書の実装)

S: 全順序集合

2分木(binary tree)とは

各節点が高々2つの子をもつ根付き木

2分木の各節点にSの要素を辞書に適した構造で格納することを考える。

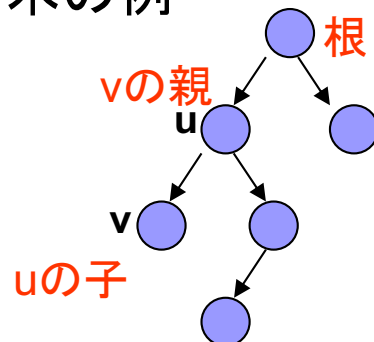
2分探索木(binary search tree)とは

任意の節点uに対して次の条件が成り立つ2分木

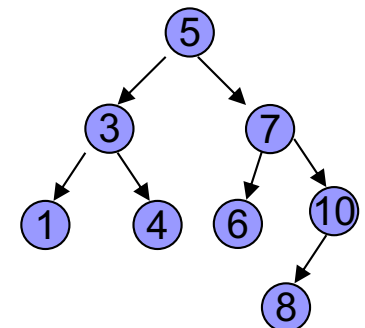
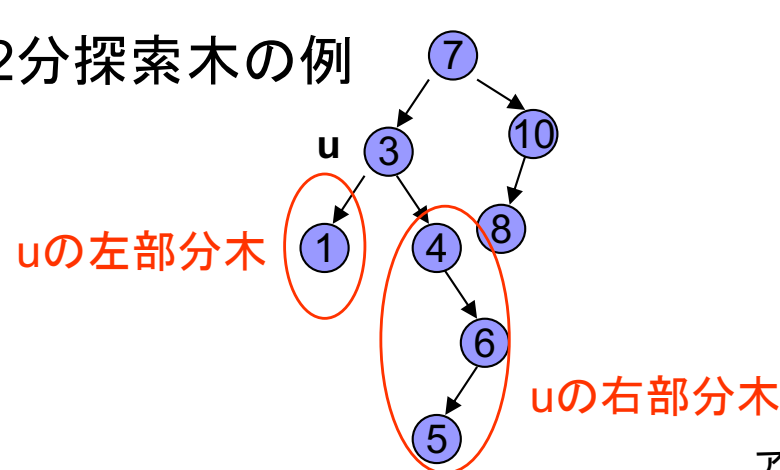
uの左部分木の任意の節点の要素 <

uの要素 < uの右部分木の任意の節点の要素

2分木の例



2分探索木の例



同じ要素が格納された  
異なる2分探索木  
アルゴリズムとデータ構造

# 2分探索木における操作

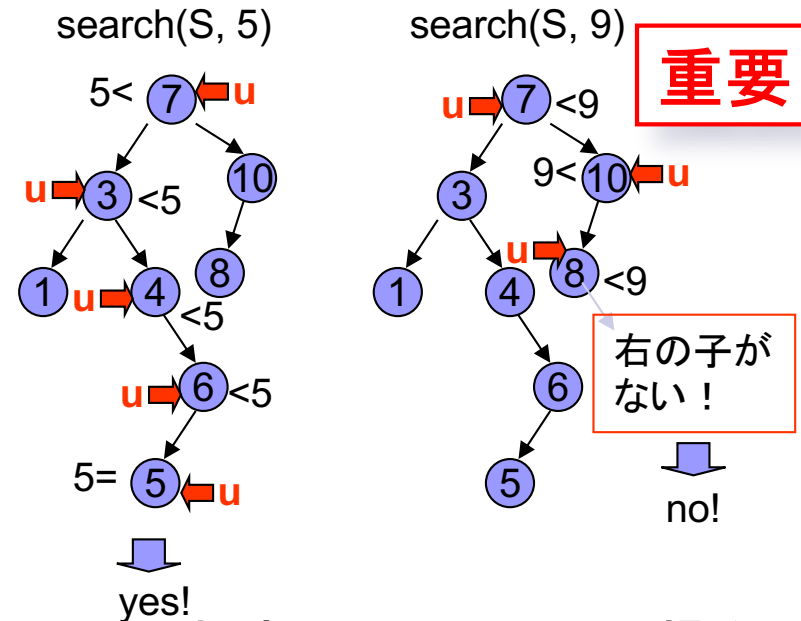
## [2分探索木のsearch(S, x)操作]

**Step 1**  $u \leftarrow$  根の節点

**Step 2**  $y \leftarrow$  節点 $u$ の要素

**Step 3** if  $x=y$  then yesを出力して停止  
 else if  $x>y$  then  
   if 右の子が存在 then  $u \leftarrow$  右の節点  
   else noを出力して停止  
 else  
   if 左の子が存在 then  $u \leftarrow$  左の節点  
   else noを出力して停止  
 Step 2へ

**重要**



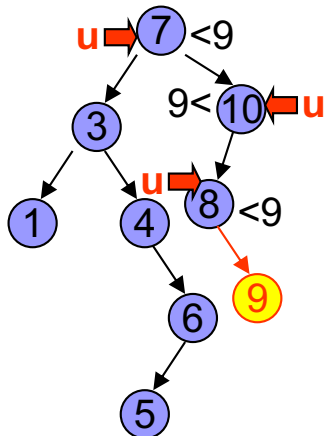
## [2分探索木のinsert(S, x)操作]

**Step 1**  $u \leftarrow$  根の節点

**Step 2**  $y \leftarrow$  節点 $u$ の要素

**Step 3** if  $x=y$  then 何もしないで停止  
 else if  $x>y$  then  
   if 右の子が存在 then  $u \leftarrow$  右の節点  
   else  $u$ の右の子として $x$ を要素とする節点を追加して停止  
 else  
   if 左の子が存在 then  $u \leftarrow$  左の節点  
   else  $u$ の左の子として $x$ を要素とする節点を追加して停止  
 Step 2へ

insert(S, 9)





# 2分探索木における操作

[2分探索木のdelete(S, x)操作]

**Step 1**  $u \leftarrow$  根の節点

**Step 2**  $y \leftarrow$  節点uの要素

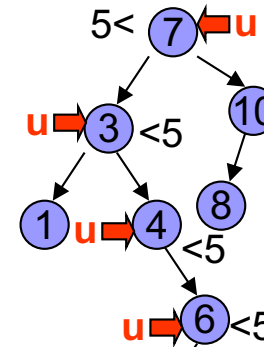
**Step 3** if  $x=y$  then Step 4  $\wedge$   
 else if  $x>y$  then  
   if 右の子が存在 then  $u \leftarrow$  右の節点  
   else 何もしないで停止  
 else  
   if 左の子が存在 then  $u \leftarrow$  左の節点  
   else 何もしないで停止  
 Step 2 $\wedge$

**Step 4** if uは葉 then uを木から除いて停止  
 else if uが1つの子をもつ then uの子をuの位置に上げて停止  
 else (uが2つの子をもつ場合)  $v \leftarrow$  uの右部分木の最小要素をもつ節点

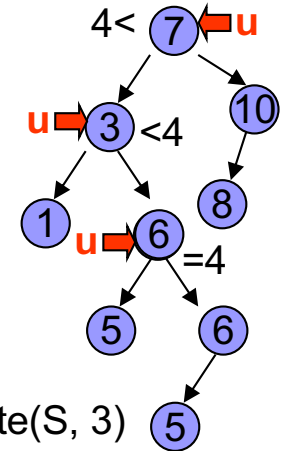
**Step 5** uの要素 $\leftarrow$ vの要素

**Step 6** if vは葉 then vを木から除いて停止  
 else (vが1つの子を持つ場合) vの子をvの位置に上げて停止

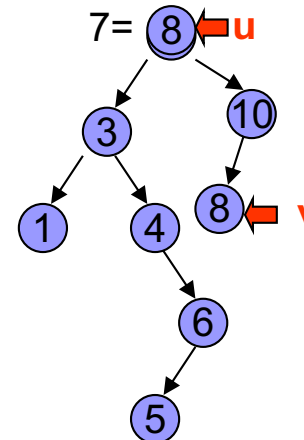
delete(S, 5)



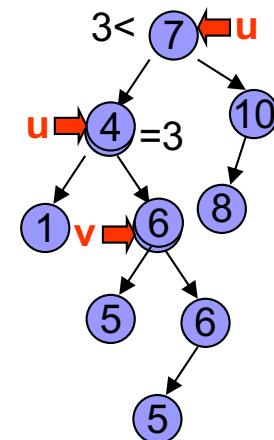
delete(S, 4)



delete(S, 7)

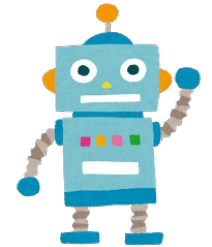


delete(S, 3)



vは部分木の最小要素をもつ節点なので子の数  
は高々1つ

# 演習：二分探索木



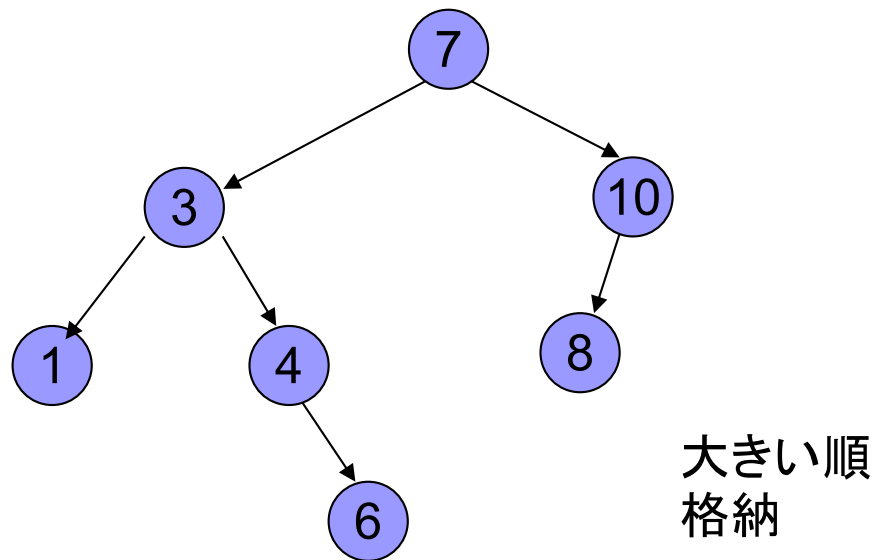
- 次の順序で要素を挿入した時にできる二分探索木をかけ

1.  $S_1 = 7, 3, 10, 1, 4, 8, 6$
2.  $S_2 = 4, 3, 2, 1$  (降順)
3.  $S_3 = 1, 2, 3, 4$  (昇順)

# 解答

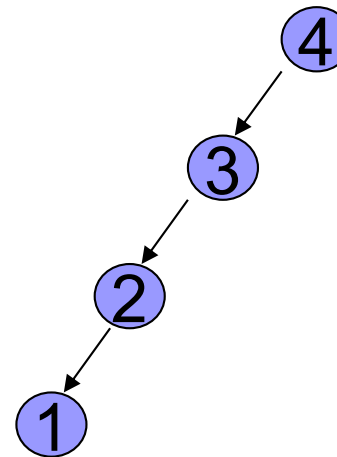
- 次の順序で要素を挿入した時にできる二分探索木をかけ

$S_1 = 7, 3, 10, 1, 4, 8, 6$



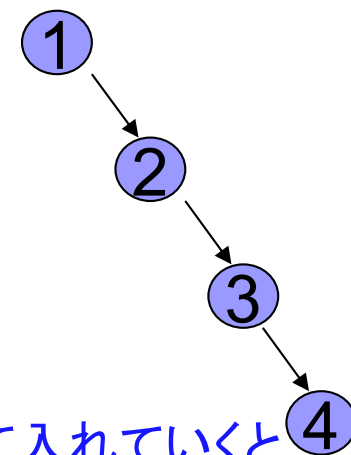
でたらめに入れると、葉の深さがだいたい同じくらいになりそう

$S_2 = 4, 3, 2, 1$  (降順)



順番に整列して入れていくと形が偏りそうだ

$S_3 = 1, 2, 3, 4$  (昇順)



# n要素の2分探索木に対する操作の時間計算量

n要素の2分探索木に対するsearch(S, x), insert(S, x), delete(S, x)操作の計算時間は、

$x \in S$ の場合、 $x$ を要素としてもつ節点の深さに比例する。

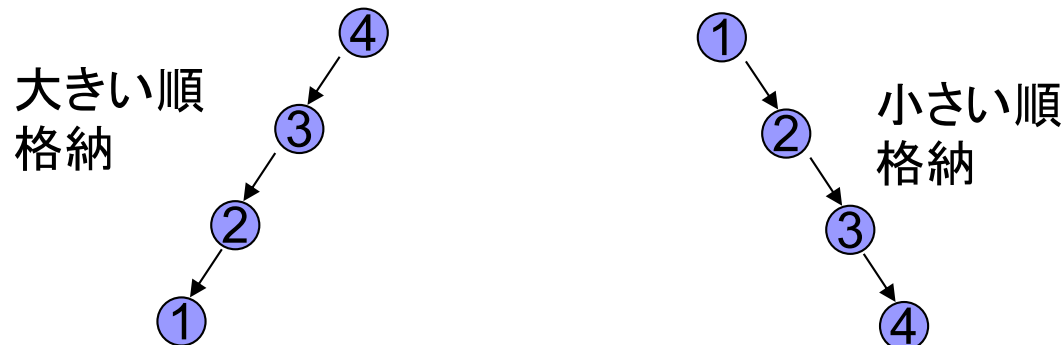
(2つの子をもつ節点のdeleteはその節点の右部分木の最小要素の節点の深さに比例する)

$x \notin S$ の場合は、insert(S, x)を行うことによってできる

$x$ を要素としてもつ節点の深さ(から1を引いた値)に比例する。

**最悪時間計算量  $O(n)$  (= 木の節点の深さの最大値)**

節点の深さが $n-1$ の場合( $S=\phi$ から小さい順、大きい順にinsert(S, x)を行った場合)



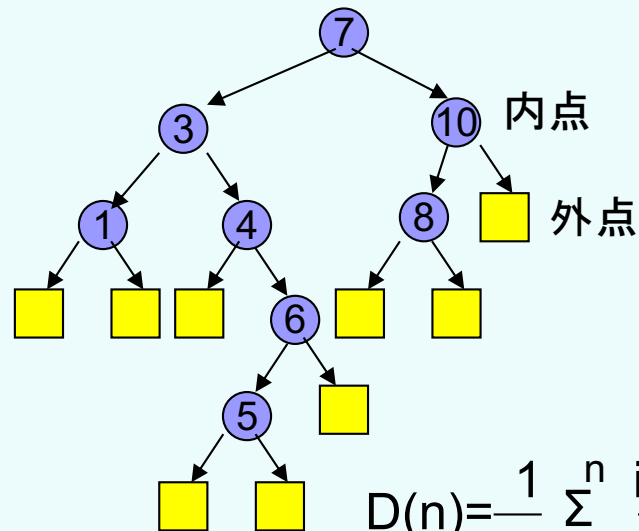
**平均時間計算量  $O(\log n)$**

節点の深さの期待値は $O(\log n)$

発展: 二分探索木を用いて, predecessorとsuccessorを実装することで, 順序辞書も実現できる. 考えてみよう

# 2分探索木の節点の深さの期待値は $O(\log n)$

(証明) 2分探索木の全ての節点がちょうど2つの子をもつように $n+1$ 個の節点を加える。  
 もとの節点を**内点**、新しく加えた節点を**外点**と呼ぶ。



外点の深さの平均 $\geq$ 内点の深さの平均

であるから外点の深さの平均が $O(\log n)$ であることを示せばよい。

$D(n)$ を外点の深さの平均とする。

最初に格納されるものが $i$ 番目の大きさである確率を $1/n$ (等確率)とすれば

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

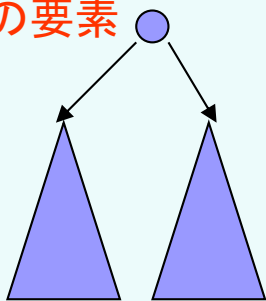
外点の数=内点の数+1

$$= \frac{2}{n(n+1)} \sum_{i=1}^n iD(i-1) + 1$$

$$= \frac{2}{n(n+1)} \left( \left( \frac{2}{n(n-1)} \sum_{i=1}^{n-1} iD(i-1) + 1 \right) \frac{n(n-1)}{2} - \frac{n(n-1)}{2} + nD(n-1) \right) + 1$$

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

$i$ 番目の要素



$i-1$ 個  
の要素  
(内点)

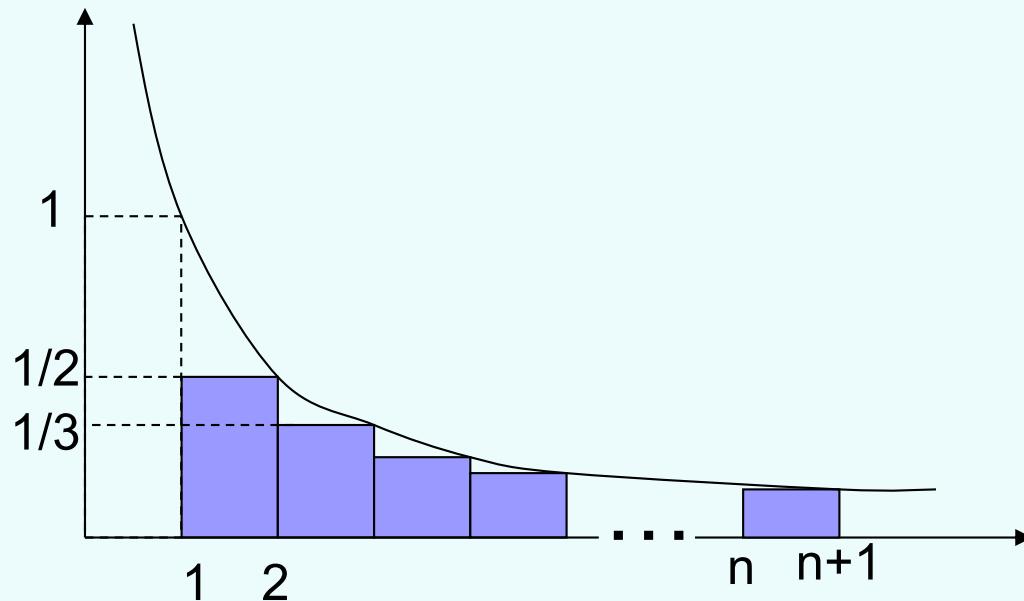
$n-i$ 個  
の要素  
(内点)

# 証明つづき

よって  $D(n) - D(n-1) = 2/(n+1)$ 、 $D(0) = 0$ であるから

$$D(n) = 2 \sum_{i=2}^{n+1} \frac{1}{i} \leq 2 \int_1^{n+1} (1/x) dx = 2 \log_e (n+1)$$

したがって  $D(n) = O(\log n)$ である。 (証明終わり)



# 予習：平衡探索木

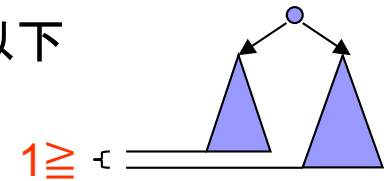
平衡探索木(balanced search tree)とは

各節点において、その子節点を根とする全ての部分木の高さが  
ほぼ平衡(バランス)している探索木

[主な平衡探索木]

次回は基本のAVL木を勉強します

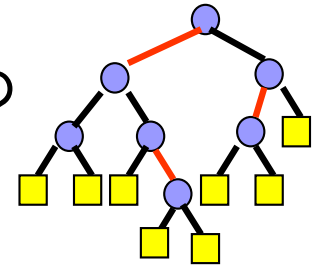
**AVL木** どの節点においても、その左部分木と右部分木の高さの差が1以下である2分探索木。AVLは2人の提唱者の頭文字 (G. M. Adel'son-Vel'skii and Y. M. Landis)



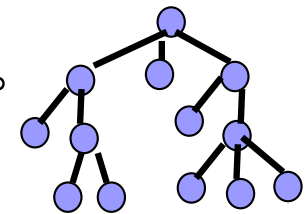
**2色木** 2分探索木の各辺に次の条件を満たすように赤か黒の色を塗れるもの

1. 外点に接続する辺の色は黒。
2. 根から外点に至るどの路の上でも、赤い辺が連続することはない。
3. 根から外点に至るどの路も、含む黒色の辺の本数は同じ。

例: C++やJavaのordered mapの実装



**B木** 根と葉を除く各節点が  $m/2$  個以上、 $m$ 個以下の子をもつ探索木。  
ただし、 $m$ は自然数。例: ファイルシステムの索引など



**2-3 木**  $m=3$ の場合のB木。

回のポイント: 挿入と削除において、どのようにすれば常に木を平衡に保てるか?

### R4年度

発展内容: 第5回に時間あればやります.  
なければ次回の最初に説明します.  
結果だけで良いですが, 結構重要です.

この部分の授業の受け方

- 前半の平衡探索木は, 「バランスをとる」ための, 基本アイデアが, 説明できるようになればよいです
- くわしい手続き(アルゴリズム)は, 理解しなくて良いです([発展]で興味ある人は考えてみよう)

今日の前半

# 平衡探索木



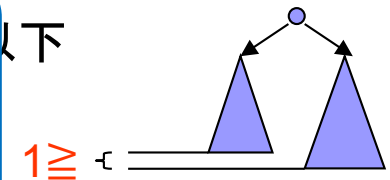
平衡探索木(balanced search tree)とは

各節点において、その子節点を根とする全ての部分木の高さが  
ほぼ平衡(バランス)している探索木

[主な平衡探索木]

次回は基本のAVL木を勉強します

**AVL木** どの節点においても、その左部分木と右部分木の高さの差が1以下である2分探索木。AVLは2人の提唱者の頭文字  
(G. M. Adel'son-Vel'skii and Y. M. Landis, 1962)

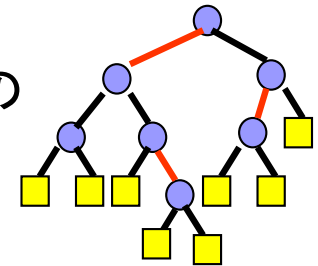


**2色木(赤黒木)** (Bayer 1972, Guibas, Sedgwick, FOCS 1978)

2分探索木の各辺に次の条件を満たすように赤か黒の色を塗れるもの

1. 外点に接続する辺の色は黒。
2. 根から外点に至るどの路の上でも、赤い辺が連続することはない。
3. 根から外点に至るどの路も、含む黒色の辺の本数は同じ。

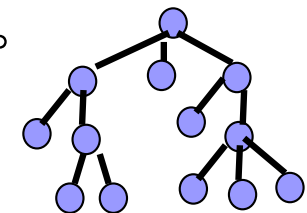
例: C++やJavaのordered mapの実装で広く利用



**B木** 根と葉を除く各節点が  $m/2$  個以上、 $m$ 個以下の子をもつ探索木。

ただし、 $m$ は自然数。例: ファイルシステムの索引などで広く利用

(Bayer, McCreight, SIGMOD Wks. 1970)



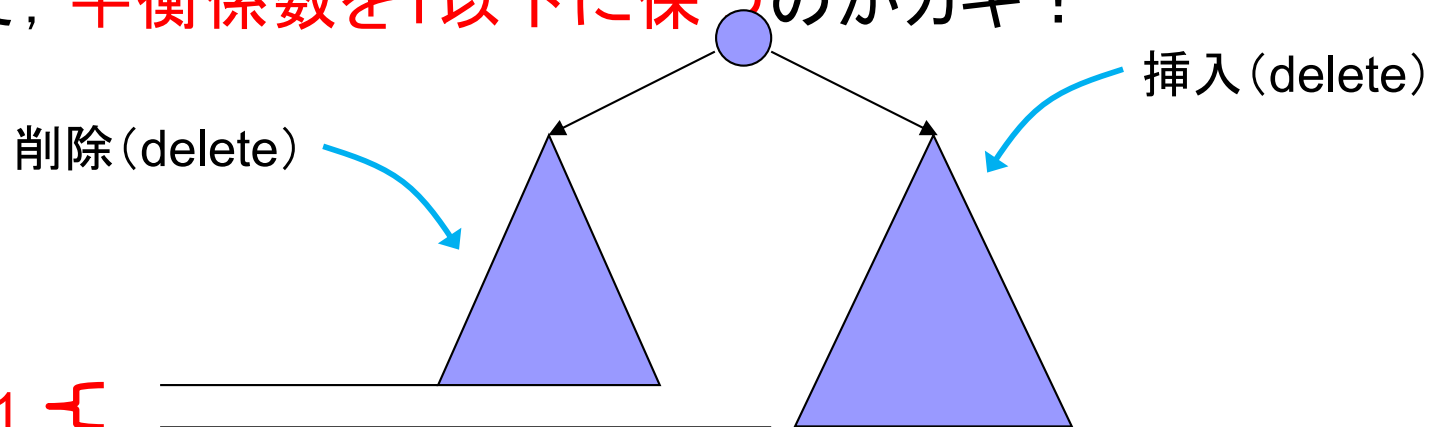
**2-3 木**  $m=3$ の場合のB木 (Hopcroft, 1970, AHU 1974)

**2-3-4 木**  $m=4$ の場合のB木 (Bayer 1972). 赤黒木はこれを2分木で模倣  
(正確にはsymmetric B-tree).

ポイント: 挿入と削除で、どのように常に木を平衡に保つか? アルゴリズムとデータ構造

# 基本アイディア: AVL木

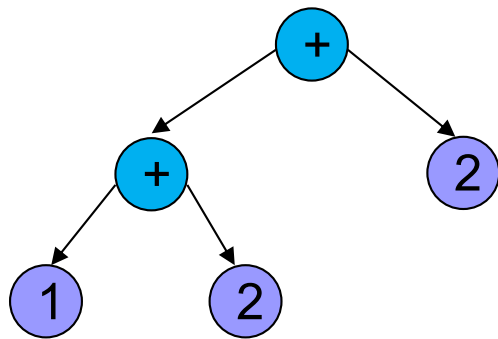
- 基本的な平衡2分木の一つ。
  - 1962年に, ロシアの情報科学者であるG. M. Adel'son-Vel'skii and Y. M. Landisによって提案された
- どの節点においても、その平衡係数が1以下である2分探索木。
  - 節点の「**平衡係数**」とは, 2分木において, その頂点の左部分木と右部分木の高さの差のこと。
- 基本的なアイディアとして, 挿入 (insert) と削除 (delete) において, **平衡係数を1以下に保つ**のがカギ！



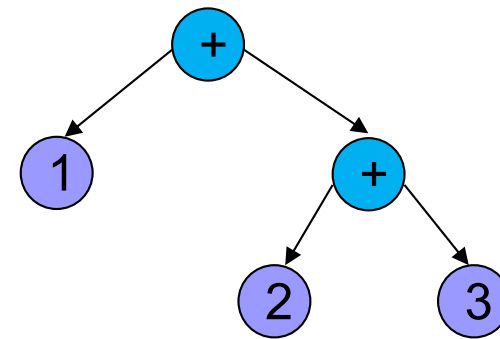
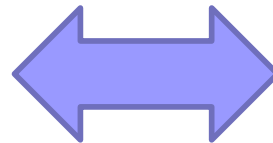
平衡係数  $\leq 1$  

# 基本アイディア: AVL木

- 挿入 (insert) と削除 (delete) において, 任意の節点で **平衡係数を1以下に保つ** のがカギ!
- 議論: 基本的なアイディアとして, 木の形を組みかえても, それが表す要素の順序 (の列) を同じにできる
  - 議論: 情報科学 (数学) でよく知られた例として, 数と二項演算 (+, \* など) からなる数式では, 同じ式を表すように数の木構造 (入れ子構造) を組み替えることができる.
  - 演算 + の結合則 (associative law):  $(a + b) + c = a + (b + c)$
  - 他にもいろいろできる



$$s = (1 + 2) + 3$$



$$s = 1 + (2 + 3)$$

# AVL木における操作

## [AVL木のinsert(x,S)操作]

$T_L^u$  : 節点uの左部分木

$T_R^u$  : 節点uの右部分木

$s(u)$  : 節点uの状態。

挿入前には以下のように設定されている。

$$s(u) = \begin{cases} L & \text{if } T_L^u \text{ の高さ} > T_R^u \text{ の高さ} \\ E & \text{if } T_L^u \text{ の高さ} = T_R^u \text{ の高さ} \\ R & \text{if } T_L^u \text{ の高さ} < T_R^u \text{ の高さ} \end{cases}$$

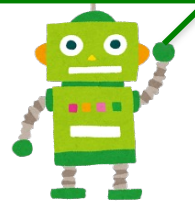
**Step 1** 一般の2分木の同じように挿入を行う。

挿入した節点の親節点をuとする。

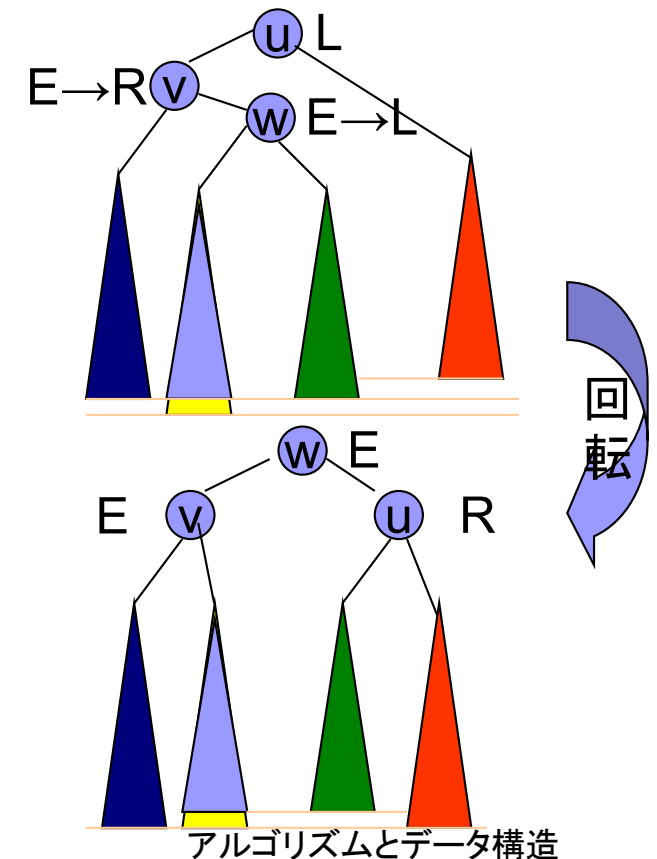
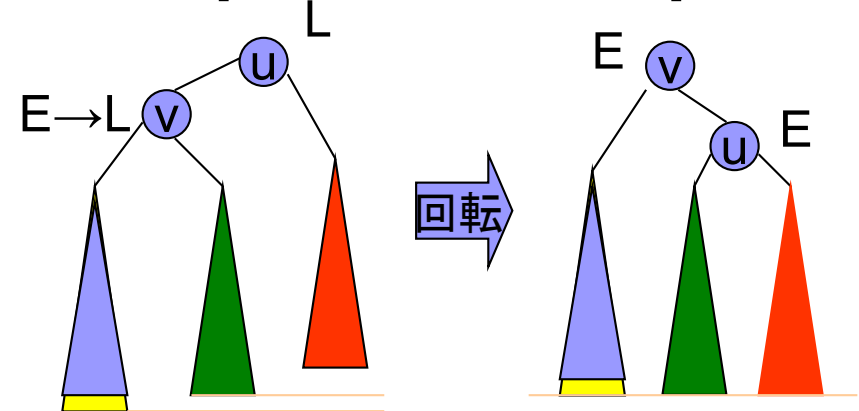
**Step 2**  $s(u) \neq E$ となるまで次のことを繰り返す。

1.  $s(u)$ の更新(挿入して高くなった方(L or R)に更新)
2.  $u \leftarrow u$ の親

**Step 3** if  $s(u)=R$  かつ  $T_L^u$  が高くなった or  
 $s(u)=L$  かつ  $T_R^u$  が高くなった then  
 $s(u) \leftarrow E$ として停止  
 else 回転して停止



[挿入に伴う回転操作]



## [AVL木のdelete(x,S)操作]

**Step 1** 一般の2分木の同じように削除を行う。

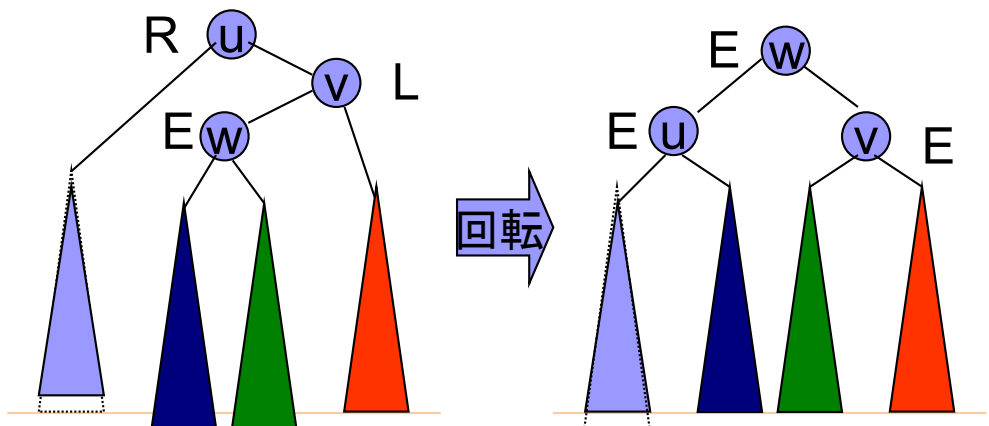
削除した最も下の節点の親節点をuとする。

**Step 2**  $s(u)=E$ となるまで次のことを繰り返す。

1. if  $s(u)=L$ かつ $T_{L^u}^u$ が低くなった or  
 $s(u)=R$ かつ $T_{R^u}^u$ が低くなった then  
 $s(u) \leftarrow E$   
 else if  $s(u)=R$ かつ $T_{L^u}^u$ が低くなった or  
 $s(u)=L$ かつ $T_{R^u}^u$ が低くなった then  
 (1) 回転  
 (2)  $s(u) \neq E$ ならば停止

2.  $u \leftarrow u$ の親

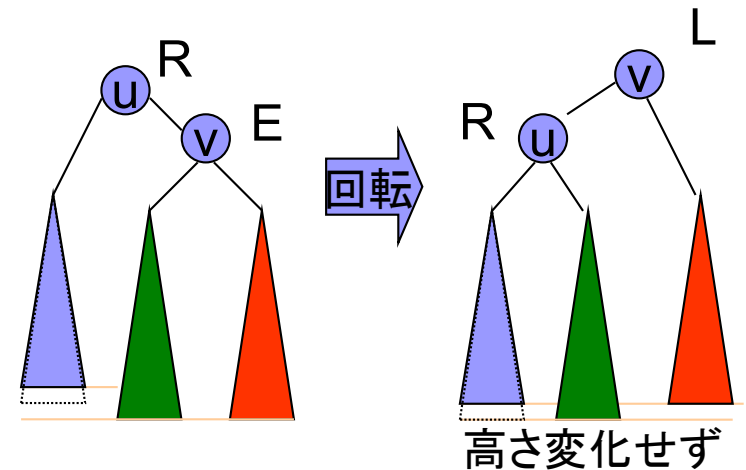
**Step 3**  $s(u)$ の更新(低くなった方の逆(L or R)に更新)



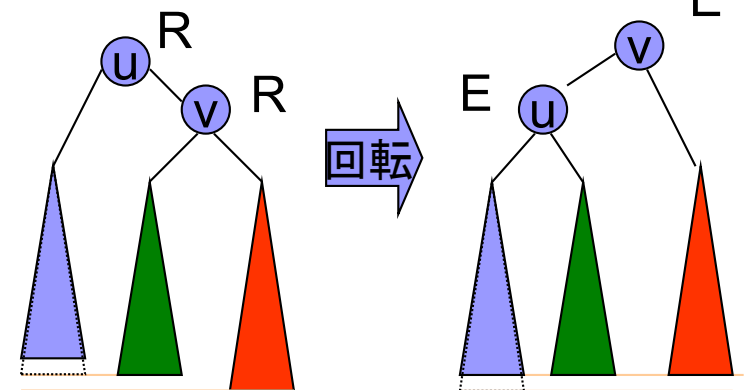
高さ1減少

## AVL木における操作(2)

[削除に伴う回転操作]



高さ変化せず



高さ1減少

やや難しい



# n要素のAVL木に対する操作の時間計算量

n要素のAVL木に対するmember(x,S), insert(x,S), delete(x,S)操作の計算時間は、

**重要**

**最悪時間計算量  $O(\log n)$**  ← n節点のAVL木の高さは $O(\log n)$

**平均時間計算量  $O(\log n)$**

## [n節点のAVL木の高さは $O(\log n)$ の証明]

f(h)を高さhのAVL木の最小節点数とすれば、以下の漸化式が成り立つ。

$$f(h) = f(h-1) + f(h-2) + 1, \quad f(0) = 1, \quad f(1) = 2$$

F(h)=f(h)+1とすれば

$$F(h) = F(h-1) + F(h-2), \quad F(0) = 2, \quad F(1) = 3$$

F(h)はフィボナッチ数列だから

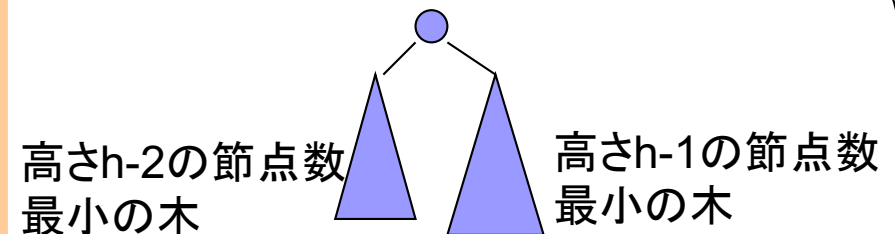
$$F(h) = (\varphi_1^{h+3} - \varphi_2^{h+3}) / \sqrt{5}$$

ただし、 $\varphi_1 = (1 + \sqrt{5})/2$ ,  $\varphi_2 = (1 - \sqrt{5})/2$

高さhのAVL木の節点数をnとすれば、

$$f(h) \leq n$$

高さhの節点数最小の木



$$\varphi_1^{h+3} - \varphi_2^{h+3} \leq \sqrt{5}(n+1)$$

$|\varphi_2| < 1$ だから

$$\varphi_1^{h+3} - 1 \leq \sqrt{5}(n+1)$$

$$h \leq \frac{\log(\sqrt{5}(n+1)+1)}{\log \varphi_1} - 3$$

したがって $h = O(\log n)$

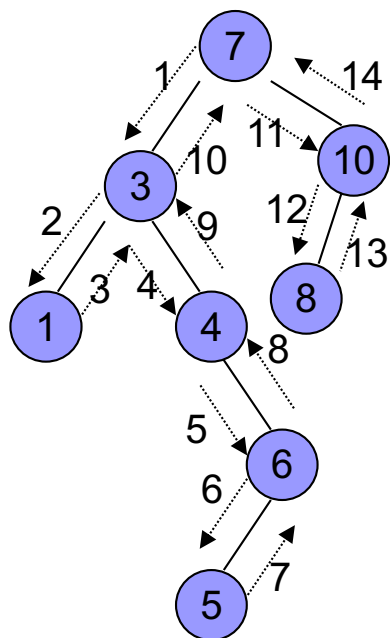
# 木のなぞり

## 木のなぞり(traverse)とは、

木のすべての節点を組織だった方法で訪問すること

次回詳しく学びます  
演習にも出ます

## 深さ優先探索(depth-first search)による木のなぞり



# 重要

## 2分探索木を中順出力すると 整列された要素のリストが えられる！

すべての節点の要素を次の3種類の順番で出力できる。

## 前順(行きがけ順、preorder)

## 最初の訪問時に出力

7, 3, 1, 4, 6, 5, 10, 8

## 中順(通りがけ順、inorder)

左の部分木のなぞりが終わった後に出力

1, 3, 4, 5, 6, 7, 8, 10

## 後順 (帰りがけ順, postorder)

## 最後の訪問時に出力

1, 5, 6, 4, 3, 8, 10, 7

# 第5回 探索のためのデータ構造(1)

## ■ 今日の内容:

- 辞書と順序辞書
- 順序辞書の実装1: 整列された配列
- 順序辞書の実装2: 二分探索木
  - 探索, 挿入, 削除演算
  - 演算の計算量
- 予習: 平衡探索木

## ■ ポイント

- 「探索」問題とは何かを理解しよう
- 「二分探索木」とは何か。その条件を理解しよう
- 二分探索木での探索と挿入の操作

## ■ 付録(別ファイル):

- 二分探索木の実装のC言語によるソースコードと解説  
(第3回演習の準備. 自習教材)