講義「アルゴリズムとデータ構造」

第６回　探索のためのデータ構造（１）

大学院情報科学研究院 情報理工学部門
情報知識ネットワーク研究室
喜田拓也

今日の内容

探索※のためのデータ構造

整列済み配列による辞書の実現

二分探索木による辞書の実現

平衡探索木による辞書の実現

※ 探索： データの中から，特定の要素を探し出すこと

探索のためのデータ構造： 辞書(dictionary)

次の3つの基本操作を伴う集合Sを辞書（dictionary）という

1. member(x,S): $x \in S$ ならば yes, $x \notin S$ ならば no を出力
2. insert(x,S): S を $S \cup \{x\}$ に更新
3. delete(x,S): S を $S - \{x\}$ に更新

調べたり

新しい項目を書き入れたり

項目を消したり（汗

整列済み配列による辞書の実現

以降，集合Sは全順序集合であると仮定する
配列AにSの要素を小さい順に格納する

Aは整列（ソート）済み配列

`member`(x,S)：配列上を二分探索する

最悪時間計算量 $O(\log n)$

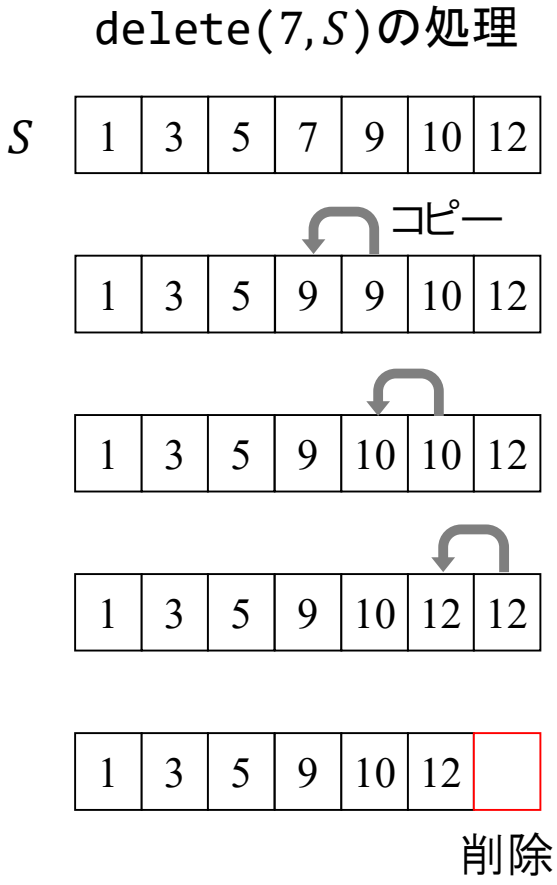
次で説明

`insert`(x,S)：xが入る位置を見つけたら，それ以降の要素を後ろにずらして，できた空きにxを入れる

$O(n)$

`delete`(x,S)：xの位置を見つけたら，それ以降の要素を一つずつ前に詰めていく．Sの最後の要素は空きとなる

$O(n)$

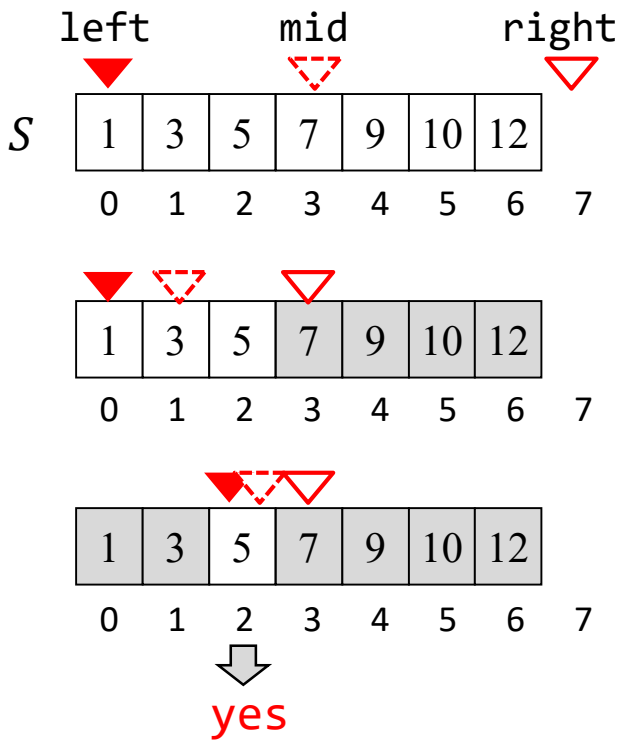


配列を二分探索するアルゴリズム

```
Step 1: left ← 0, right ← n
Step 2: left ≥ right ならnoを出力して停止
Step 3: mid ← ⌊(left + right)/2⌋
Step 4: if x < A[mid]
        then right ← mid
        else if x = A[mid]
          then yesを出力して停止
          else left ←mid +1
        Step 2 へ
```

このループは高々$\lfloor(\log_2 n) + 1\rfloor$ 回

member$(5, S)$の場合



1回で範囲を半分以下に絞れるので，k回で範囲を$n/2^k$以下に絞れる
範囲が空と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kは$\lfloor(\log_2 n) + 1\rfloor$

二分探索木による辞書の実現

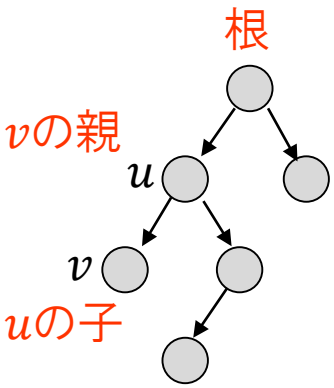
二分木（binary tree）とは，各節点が高々二つの子をもつ根付き木

二分探索木（binary search tree）とは，任意の節点uに対して次の条件が成り立つ二分木のこと

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

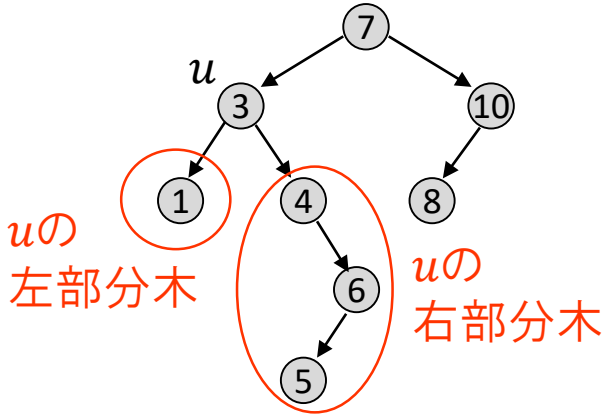
この二分探索木を使って全順序集合Sを管理し，辞書とする

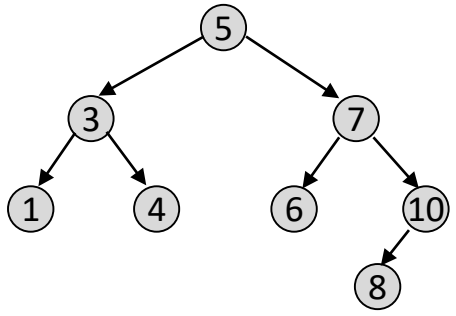
2分木の例



子が1個の場合もある

2分探索木の例



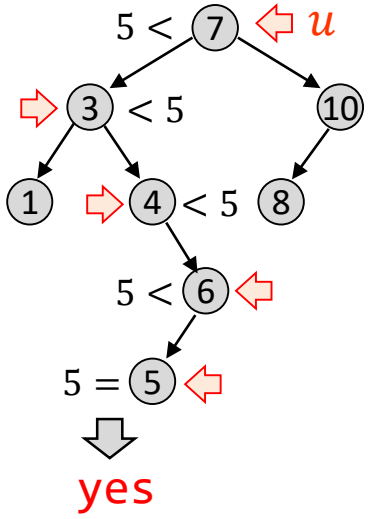


同じ要素が格納された異なる二分探索木

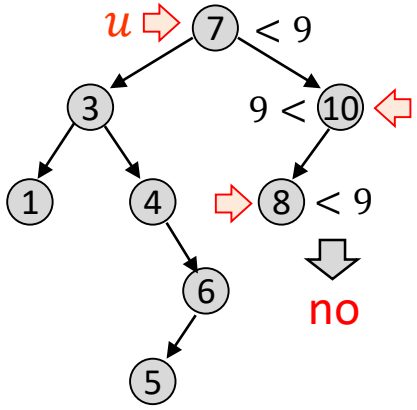
二分探索木における member(x, S) 操作

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 u の右の子が存在 **then**
　　$u \leftarrow u$ の右の子
　　else
　　noを出力して停止
$x \leq y$ **else if** u の左の子が存在 **then**
　　$u \leftarrow u$ の左の子
　else
　　noを出力して停止
　Step 2へ

member$(5, S)$の場合



member$(9, S)$の場合



二分探索木における$\mathtt{insert}(x, S)$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 u の右の子が存在 **then**

$u \leftarrow u$ の右の子

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 u の左の子が存在 **then**

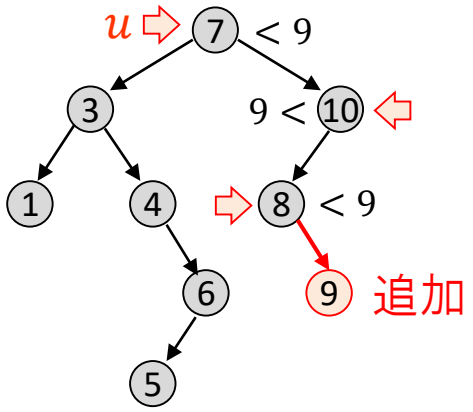
$u \leftarrow u$ の左の子

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

$\mathtt{insert}(9, S)$の場合



赤字の部分が
$\mathtt{member}(x, S)$
操作と違うところ

二分探索木におけるdelete(x, S)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 へ
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 へ

S 中に x が無ければ何もしない

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
  $v \leftarrow u$ の右部分木の最小要素をもつ節点

これ以降はu が2つの子をもつ場合

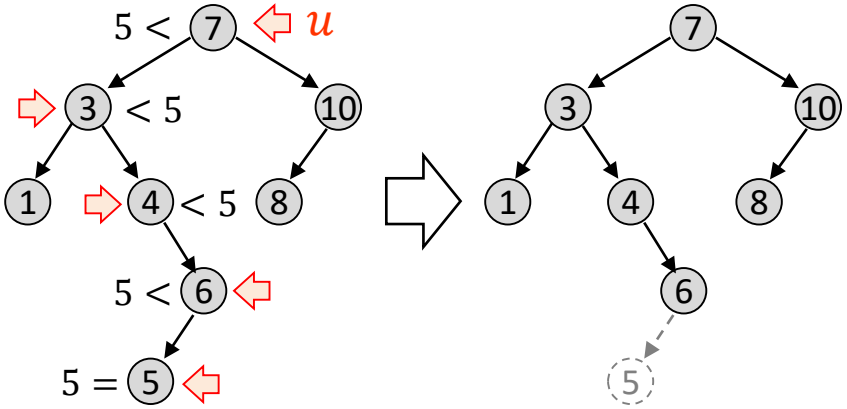
Step 5: u の要素 $\leftarrow v$ の要素

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 を u の位置に上げて停止

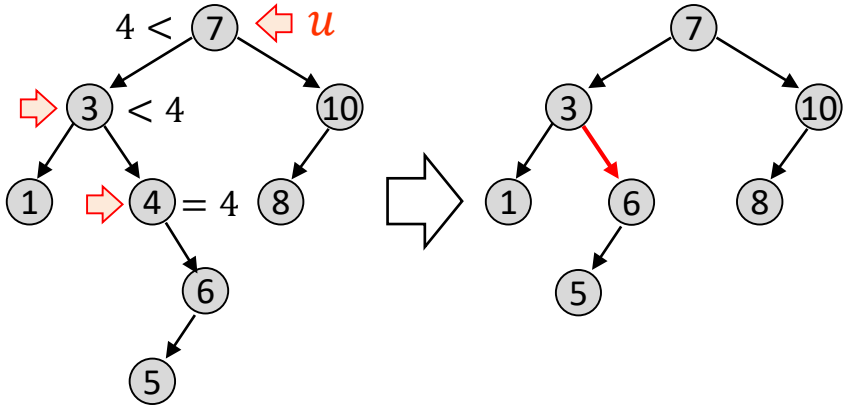
v は部分木の最小要素をもつ節点なので子の数は高々1つ

二分探索木における$\mathtt{delete}(x, S)$操作の例

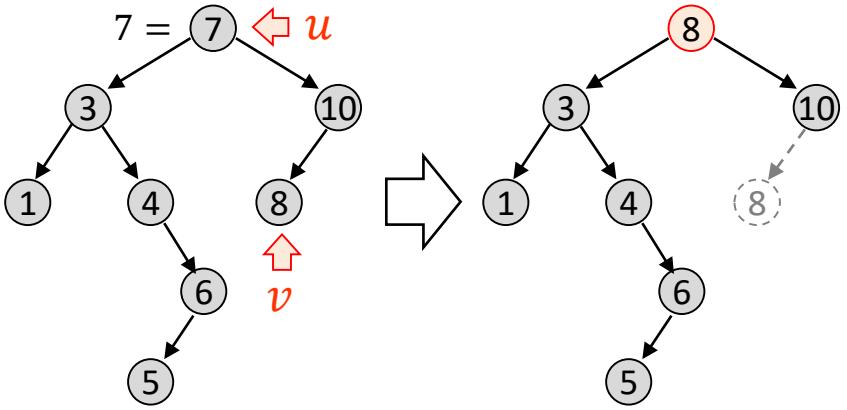
$\mathtt{delete}(5, S)$の場合



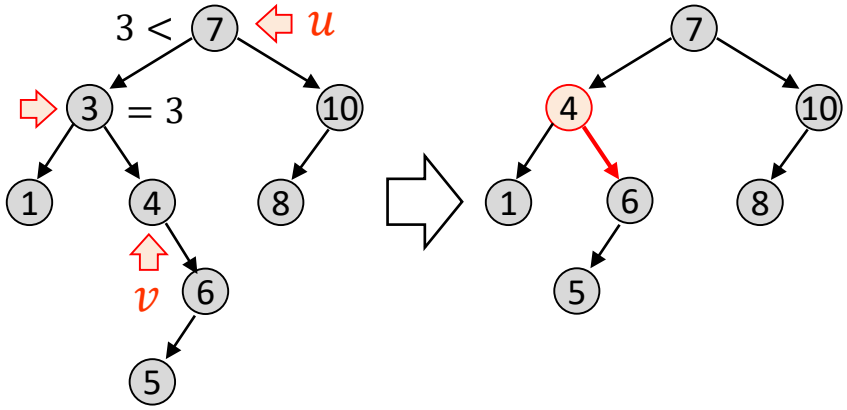
$\mathtt{delete}(4, S)$の場合



$\mathtt{delete}(7, S)$の場合



$\mathtt{delete}(3, S)$の場合



二分探索木に対する操作の時間計算量

n 要素の二分探索木に対する各操作の計算時間は，

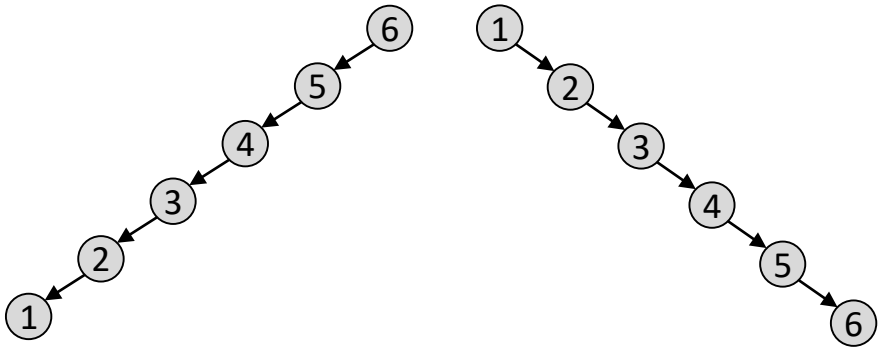
$x \in S$ の場合，x を要素としてもつ節点の深さ※に比例する

$x \notin S$の場合，insert(x, S)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 -1 に比例する

※ x が2つの子を持つ場合のdelete(x, S)は，x の右部分木の最小要素の節点 v の深さにも比例

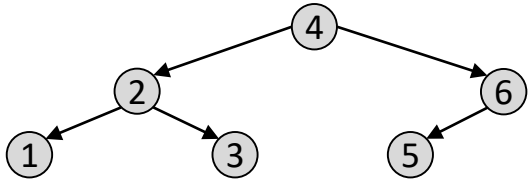
最悪時間計算量は$O(n)$，平均時間計算量は$O(\log n)$

最悪な場合（木の深さ$= n - 1$）



大きい順に格納　　小さい順に格納

最良の場合（木の深さ$= \log n$ ）



節点の深さの期待値は$O(\log n)$

深さの期待値が$O(\log n)$である証明

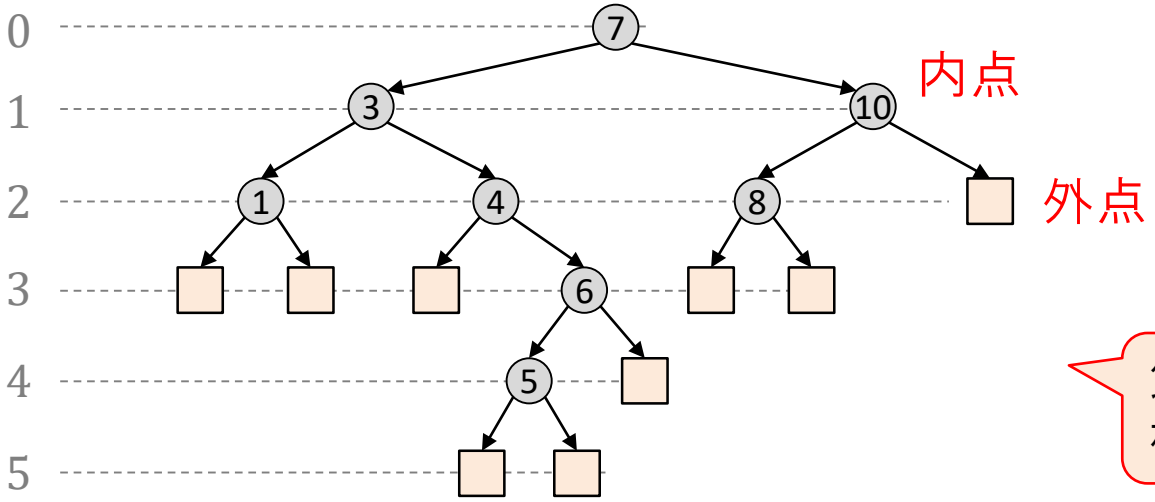
（証明）二分探索木の全ての節点がちょうど2つの子をもつように$n+1$個の節点を加える.

もとの節点を**内点**，新しく加えた節点を**外点**と呼ぶ.

トーナメントの試合数を考えてみよう

「外点の深さの平均」≥「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.

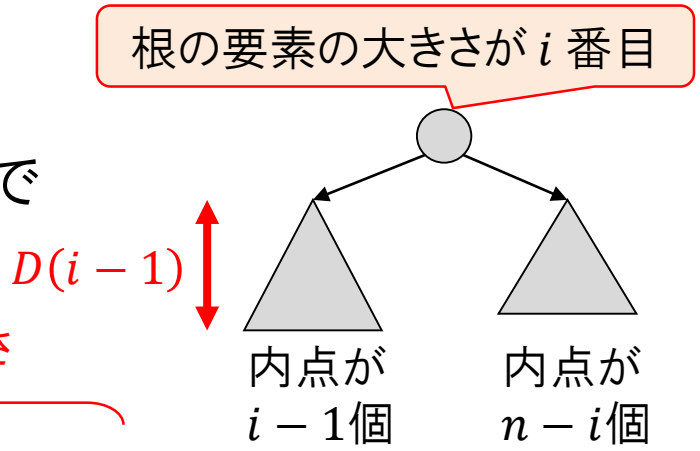


節点の深さ＝根からの路の長さ

証明つづき

$D(n)$ 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,



左側の総深さ　　右側の総深さ

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

外点の数 = 内点の数＋1

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

$iD(i-1)$ と $(n-i+1)D(n-i)$ は
i に対して対称なので

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

$D(n-1)$

$i = n$ のところを抽出して分解

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

証明つづき

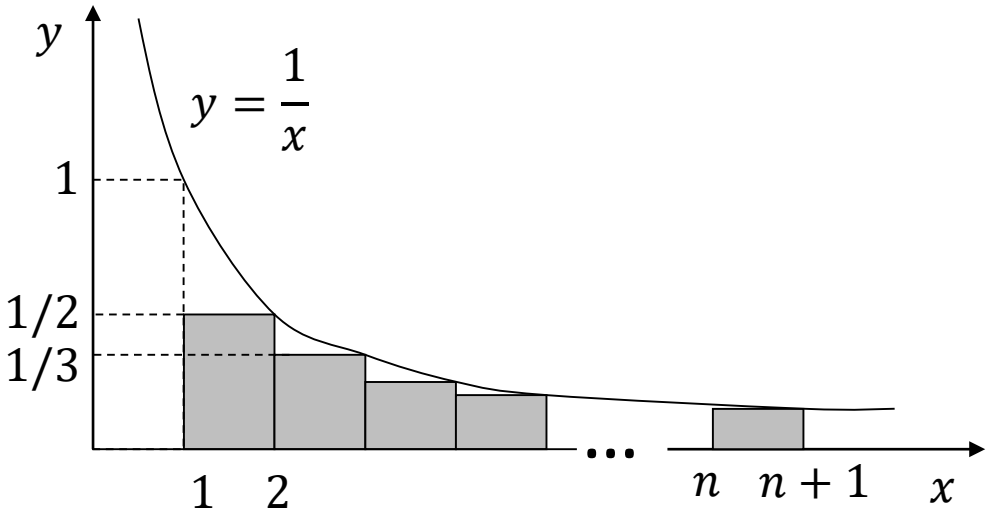
よって, $D(n) - D(n-1) = 2/(n+1)$ となる.

$D(0) = 0$ であるから,

調和級数の部分和

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

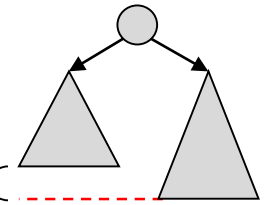
したがって $D(n) = O(\log n)$ である.　　【証明終わり】



平衡探索木（balanced search tree）

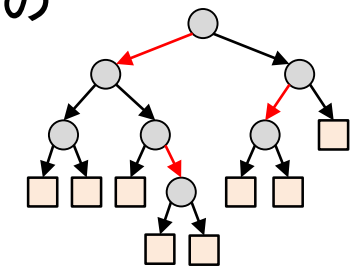
各節点において，その子を根とする全ての部分木の高さがほぼ平衡している探索木

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以下である二分探索木
AVLは提唱者（Adel'son-Vel'skii と Landis）の頭文字



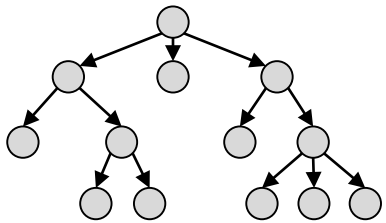
2色木　二分探索木の各辺に次の条件を満たすように赤か黒の色を塗れるもの

1. 外点に接続する辺の色は黒
2. 根から外点までのどの路も赤い辺が連続しない
3. 根から外点までのどの路も含む黒い辺の数が同じ



B木　根と葉を除く各節点が $m/2$個以上，m個以下の子を持つ探索木（mは自然数）

2-3 木　$m = 3$ の場合のB木



AVL木におけるinsert(x, S)操作

T_L^uを節点uの左部分木，T_R^uをuの右部分木，$s(u)$をuの状態とする
挿入前の$s(u)$は次のように設定されている．

$$S(u) = \begin{cases} L, & T_L^u\text{の高さ} > T_R^u\text{の高さの場合}, \\ E, & T_L^u\text{の高さ} = T_R^u\text{の高さの場合}, \\ R, & T_L^u\text{の高さ} < T_R^u\text{の高さの場合}. \end{cases}$$

左が高い

右が高い

Step 1: 二分木と同様に挿入を行う．挿入した節点の親を 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. $s(u)$を，挿入して高くなった方（Lか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

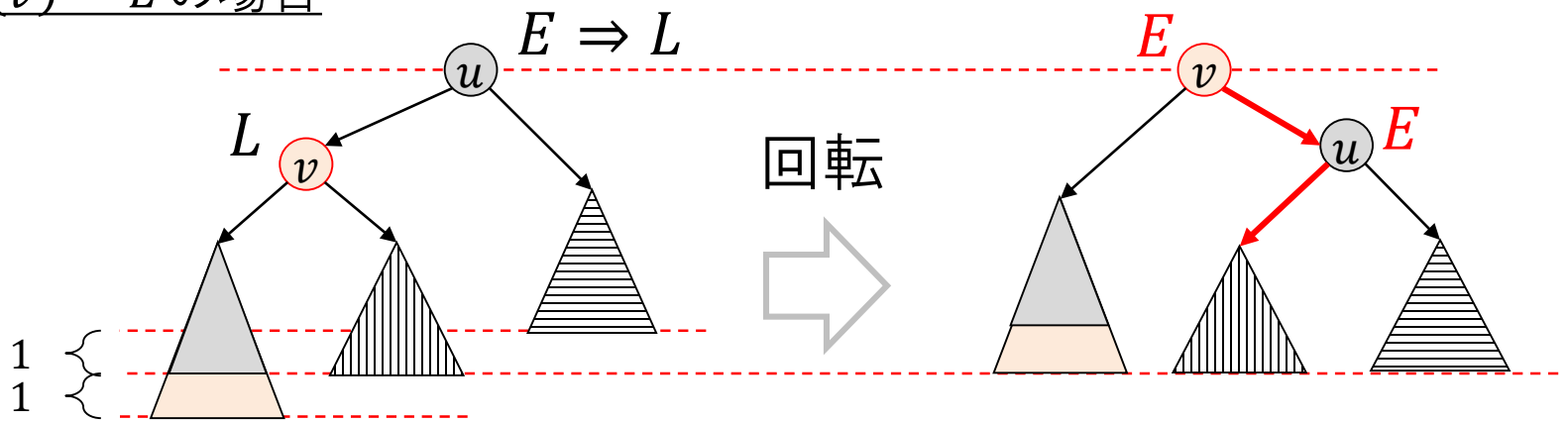
else 回転して停止

挿入したことによる祖先のラベル変更

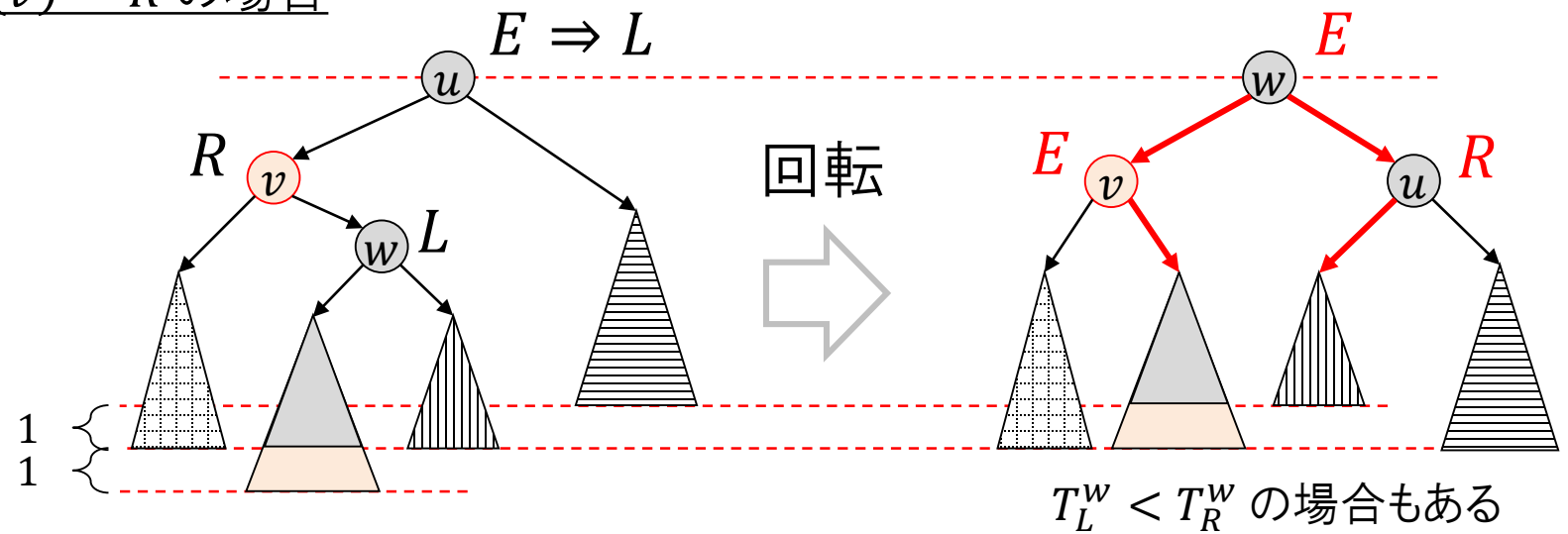
ここの処理は定数時間

AVL木における挿入(左)に伴う回転操作

$s(v) = L$ の場合



$s(v) = R$ の場合



$T_L^w < T_R^w$ の場合もある

AVL木におけるdelete(x, S)操作

Step 1: 一般の二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 が低くなった or
   $s(u) = R$ かつ T_R^u が低くなった **then**

   $s(u) \leftarrow E$

   else if $s(u) = R$ かつ T_L^u が低くなった or
   $s(u) = L$ かつ T_R^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

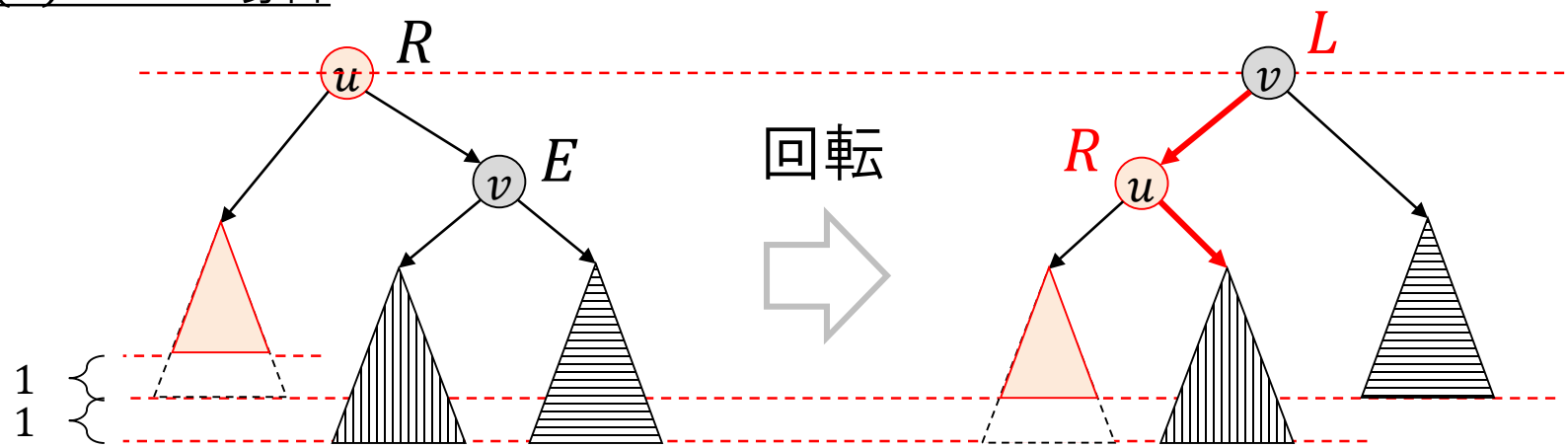
Step 3: $s(u)$ を, 低くなった方の逆(L or R)に更新

回転処理は定数時間

高々, 木の高さの回数しか繰り返さない

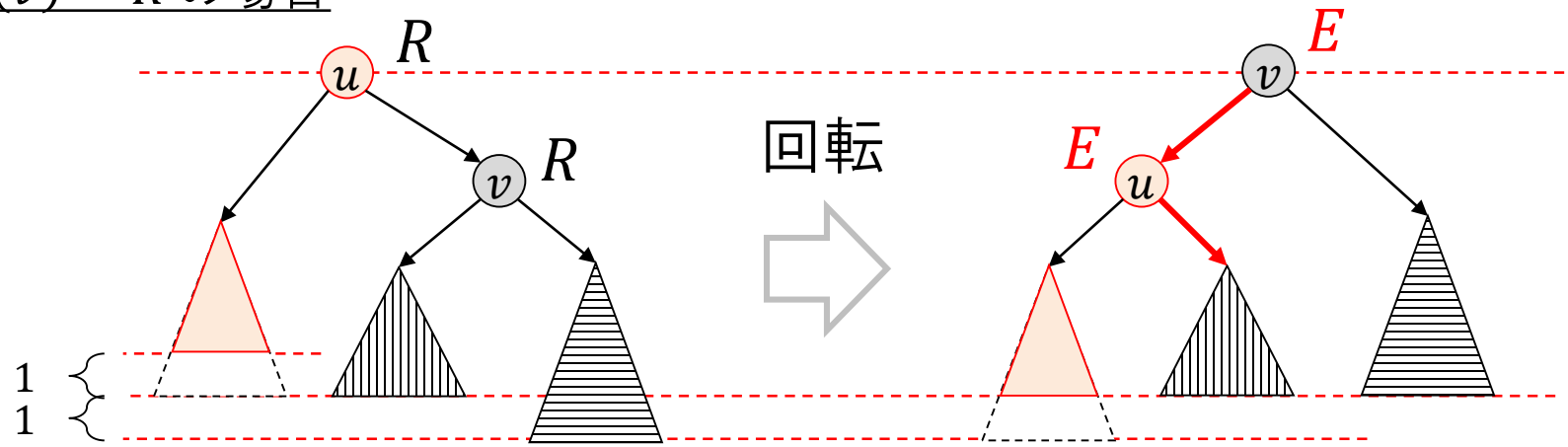
AVL木における削除(左)に伴う回転操作

$s(v) = E$ の場合



高さ変化せず

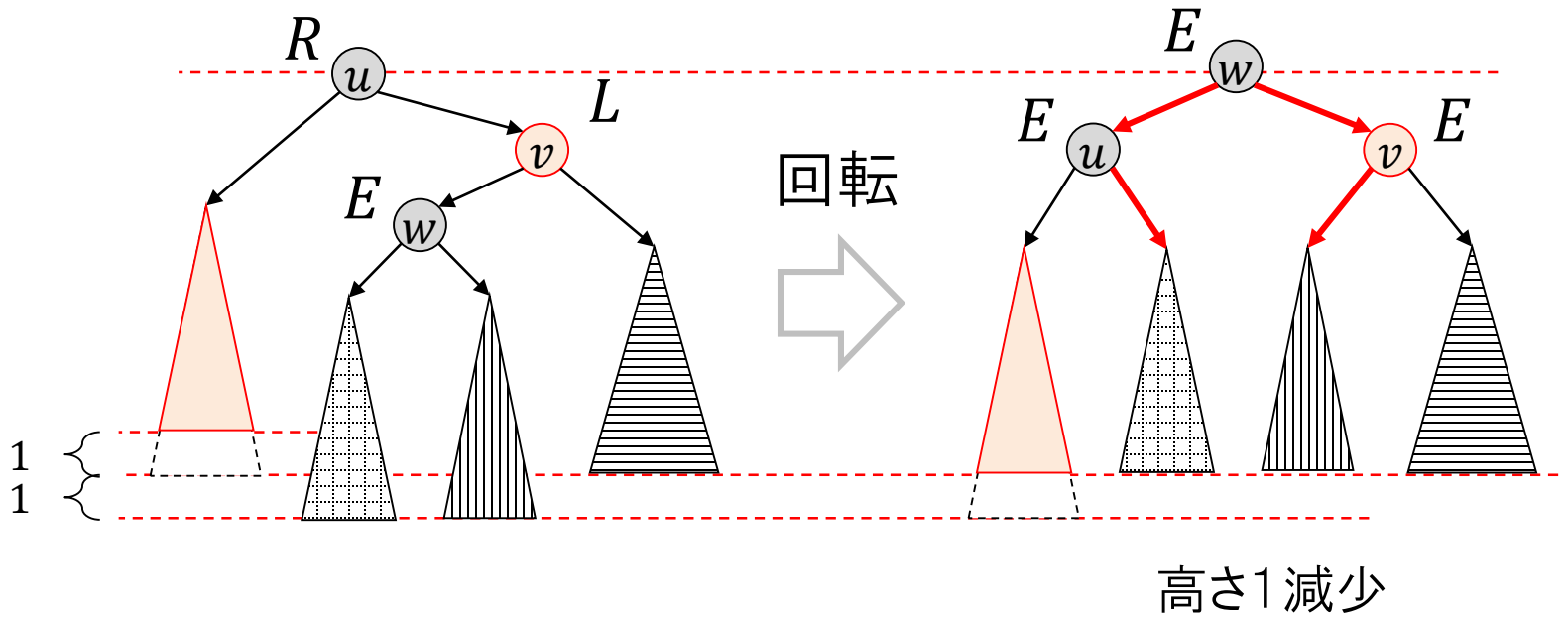
$s(v) = R$ の場合



高さ1減少

AVL木における削除(左)に伴う回転操作

$s(v) = L$ の場合



高さ1減少

AVL木に対する操作の時間計算量

n 要素のAVL木に対する各操作の計算時間は，

最悪時間計算量は$O(\log n)$，平均時間計算量は$O(\log n)$

AVL木のどの操作も，木の高さに比例した時間しかかからない．
n 節点のAVL木の高さは $O(\log n)$

AVL木の高さが $O(\log n)$ 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．

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

$F(h) = f(h) + 1$ とすれば，

根だけの木

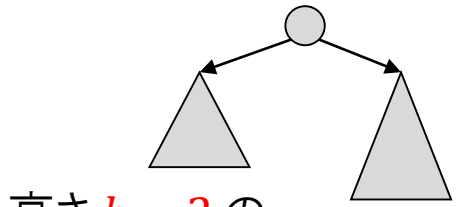
$$F(h) = F(h-1) + F(h-2),\ \ F(0) = 2,\ F(1) = 3.$$

すると，$F(h)$ はフィボナッチ数列なので，

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

ただし，$\varphi_1 = \frac{1+\sqrt{5}}{2}$，$\varphi_2 = \frac{1-\sqrt{5}}{2}$ である．

高さ h の節点数最小のAVL木



高さ $h-2$ の節点数最小のAVL木

高さ $h-1$ の節点数最小のAVL木

証明つづき

高さ h のAVL木の節点数を n とすれば，$f(h)$ はその下限なので，

$$f(h) \leq n$$

である．よって，

$\because \dfrac{\varphi_1^{h+3} - \varphi_2^{h+3}}{\sqrt{5}} = f(h) + 1$

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

$|\varphi_2| < 1$ なので，

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

よって，

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

したがって，$h = O(\log n)$ である．　【証明終わり】

今日のまとめ

探索のためのデータ構造である辞書の実現方法を学んだ

- 整列済み配列による辞書の実現
  - memberは最悪時間計算量$O(\log n)$
  - insertとdeleteは$O(n)$かかる
- 二分探索木による辞書の実現
  - 最悪時間計算量$O(n)$
  - 平均時間計算量$O(\log n)$
- 平衡探索木による辞書の実現
  - 最悪時計算量も平均時間計算量も$O(\log n)$