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）

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. member(x,S) : x∈Sならばyes, x∉Sならばnoを出力
2. Insert(x,S) : SをS∪{x}に更新
3. delete(x,S) : 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。

整列された配列

S: 全順序集合

A[0],A[1],…,A[n-1]: 配列AにSの要素を小さい順に格納

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

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

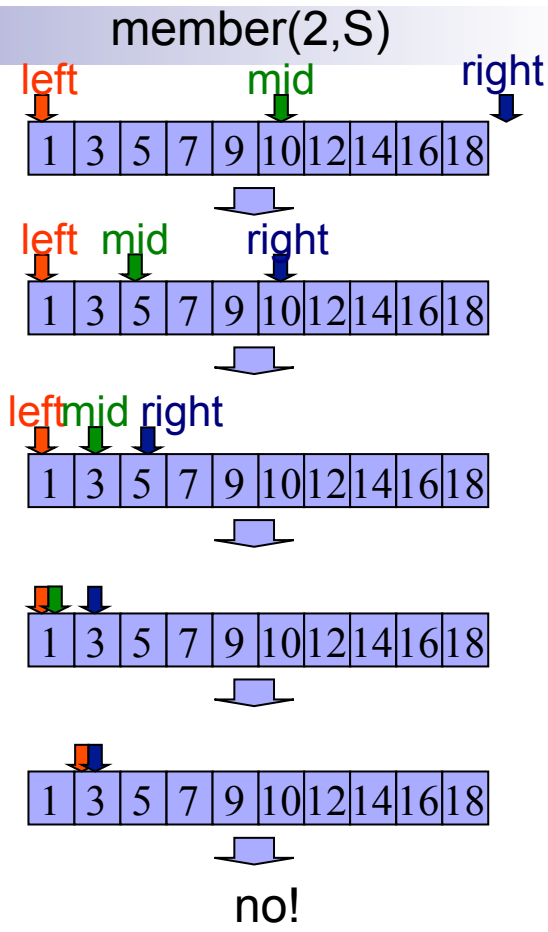
[2分探索によるmember(x,S)操作]

Step 1 left←0, right←n

Step 2 left≧rightであれば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は $(\log_2 n)+1$以下 。

2分探索木

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

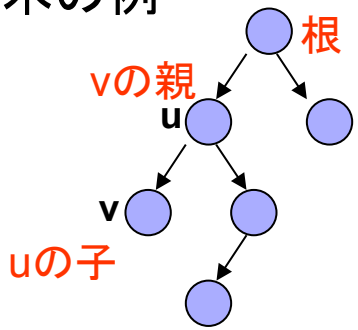
2分木の各節点にSの要素を辞書に適した構造で格納することを考える。

2分探索木(binary search tree)とは

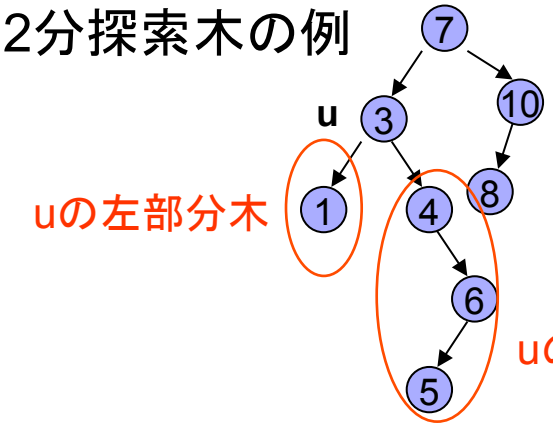
任意の節点uに対して次の条件が成り立つ2分木

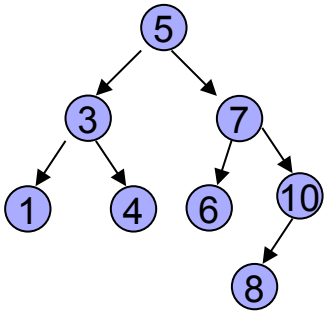
uの左部分木の任意の節点の要素<
uの要素<uの右部分木の任意の節点の要素

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

2分探索木における操作

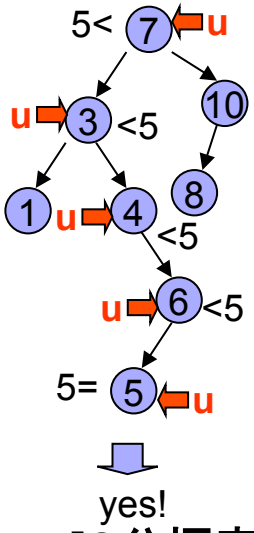
[2分探索木のmember(x,S)操作]

Step 1 u←根の節点

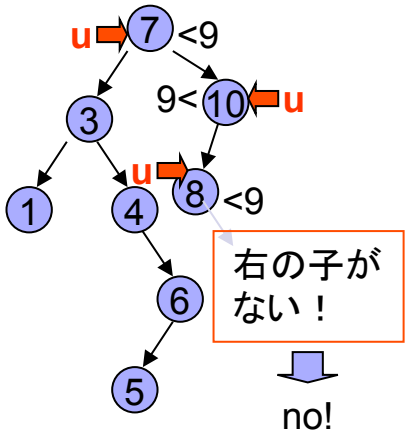
Step 2 y←節点uの要素

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←右の節点
else noを出力して停止
else
if 左の子が存在 then u←左の節点
else noを出力して停止
Step 2へ

member(5,S)



member(9,S)

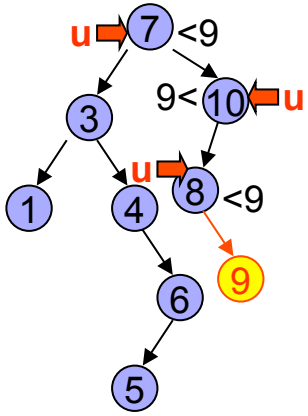


[2分探索木のinsert(x,S)操作]

Step 1 u←根の節点

Step 2 y←節点uの要素

insert(9,S)



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←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←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

2分探索木における操作

[2分探索木のdelete(x,S)操作]

Step 1 u←根の節点

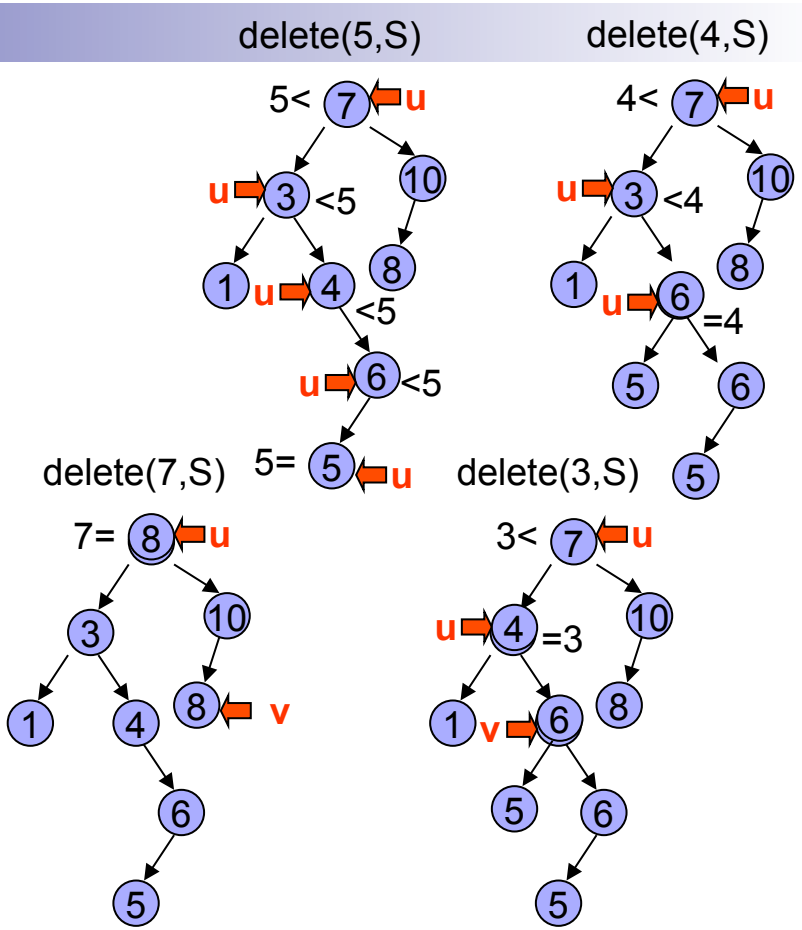
Step 2 y←節点uの要素

Step 3 if x=y then Step 4 へ
else if x>y then
if 右の子が存在 then u←右の節点
else 何もしないで停止
else
if 左の子が存在 then u←左の節点
else 何もしないで停止
Step 2へ

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←uの右部分木の最小要素をもつ節点

Step 5 uの要素←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



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

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member(x,S), insert(x,S), delete(x,S)操作の計算時間は、

x∈Sの場合、 xを要素としてもつ節点の深さに比例する。

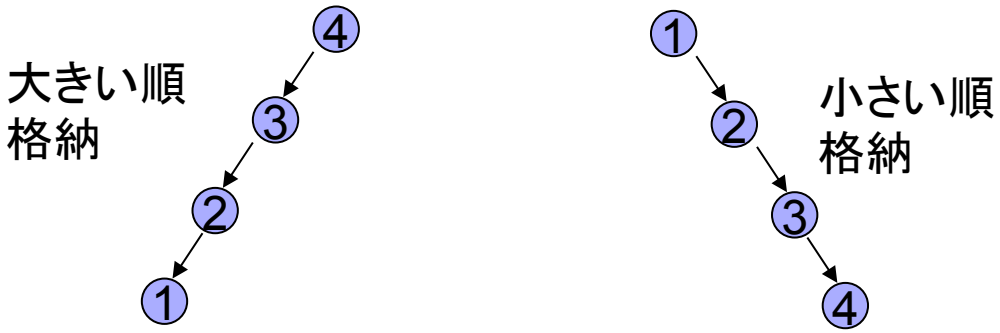
(2つの子をもつ節点のdeleteはその節点の右部分木の最小要素の節点の深さに比例する)

x∉Sの場合は、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(から1を引いた値)に比例する。

最悪時間計算量　O(n)

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

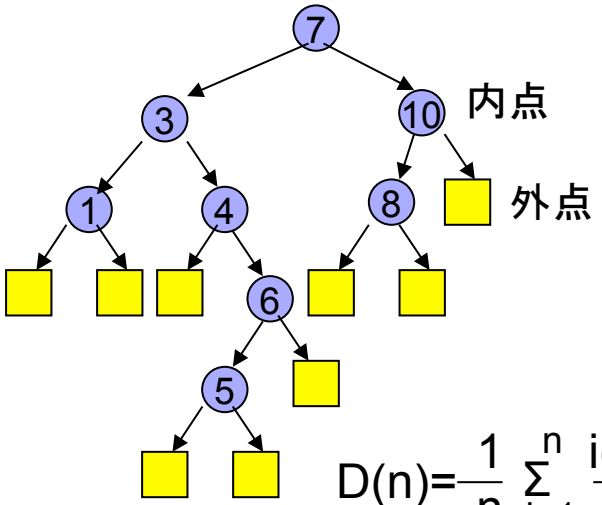


平均時間計算量　O(log n)

節点の深さの期待値はO(log n)

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。

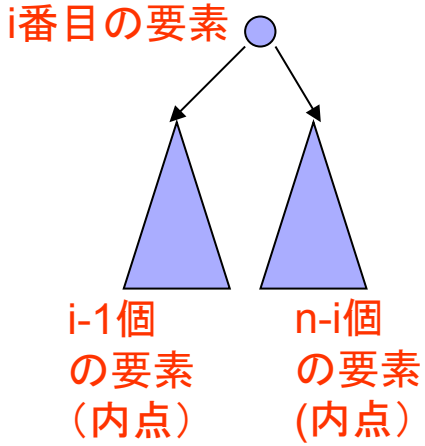


外点の深さの平均≧内点の深さの平均

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



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

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

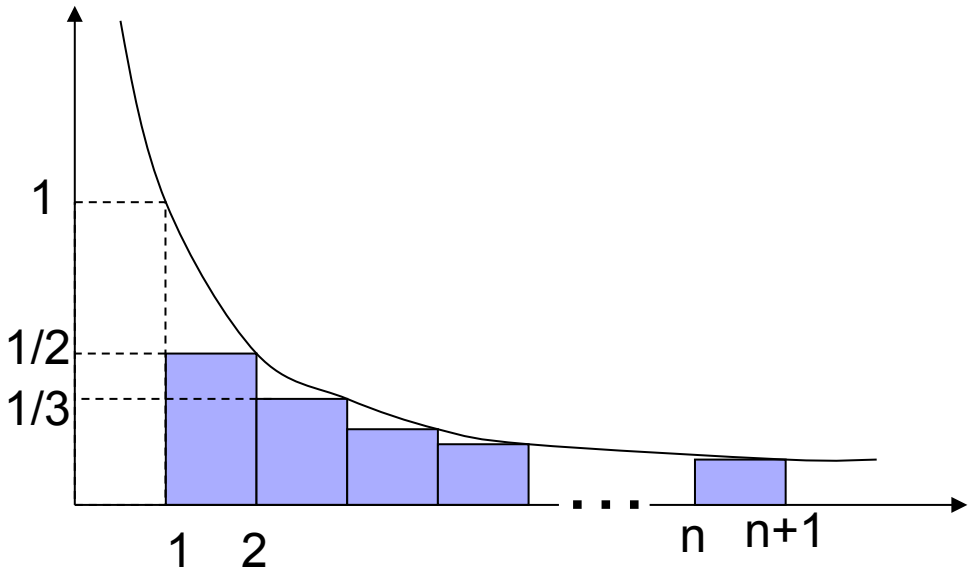
$$=D(n-1)+\frac{2}{n+1}$$

証明つづき

よってD(n)-D(n-1)=2/(n+1)、D(0)=0であるから

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

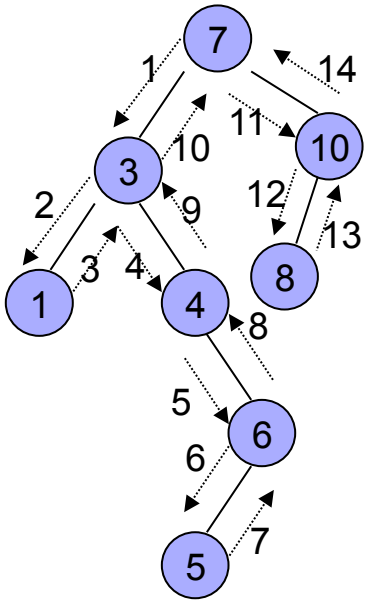
したがってD(n)=O(log n)である。　　(証明終わり)



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

深さ優先探索(depth-first search)による木のなぞり



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

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！

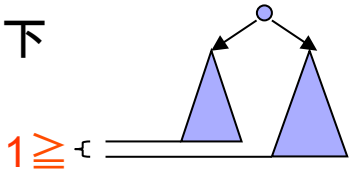
平衡探索木

平衡探索木(balanced search tree)とは

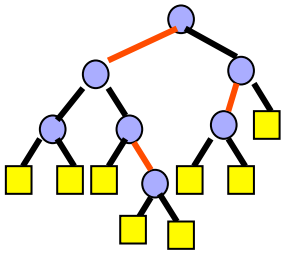
各節点において、その子節点を根とする全ての部分木の高さが
ほぼ平衡している探索木

[主な平衡探索木]

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)

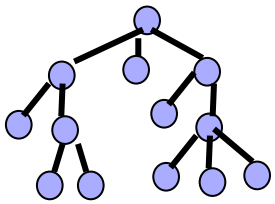


2色木　2分探索木の各辺に次の条件を満たすように赤か黒の色を塗れるもの
1. 外点に接続する辺の色は黒。
2. 根から外点に至るどの路の上でも、赤い辺が連続することはない。
3. 根から外点に至るどの路も、含む黒色の辺の本数は同じ。



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。

2-3 木　m=3の場合のB木。



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。

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

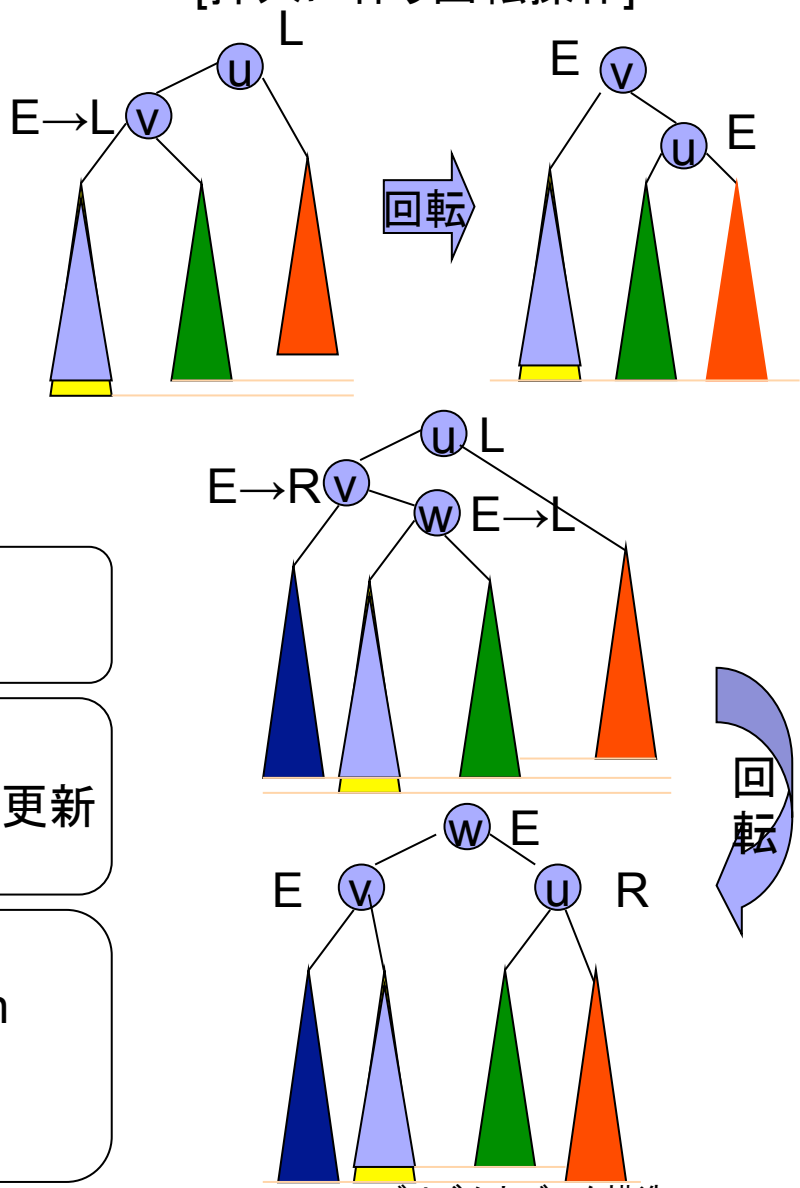
Step 1 一般の2分木の同じように挿入を行う。
挿入した節点の親節点をuとする。

Step 2 s(u)≠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←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)←Eとして停止
else 回転して停止

[挿入に伴う回転操作]



AVL木における操作(2)

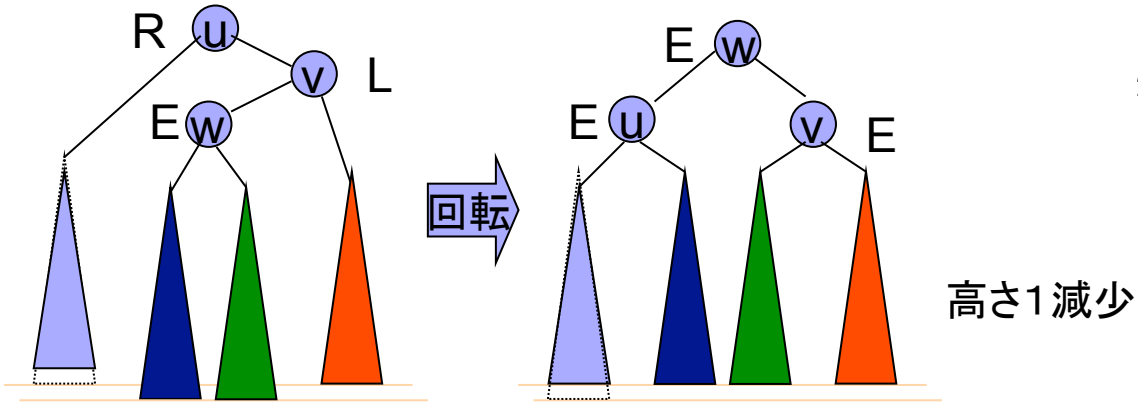
[AVL木のdelete(x,S)操作]

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。

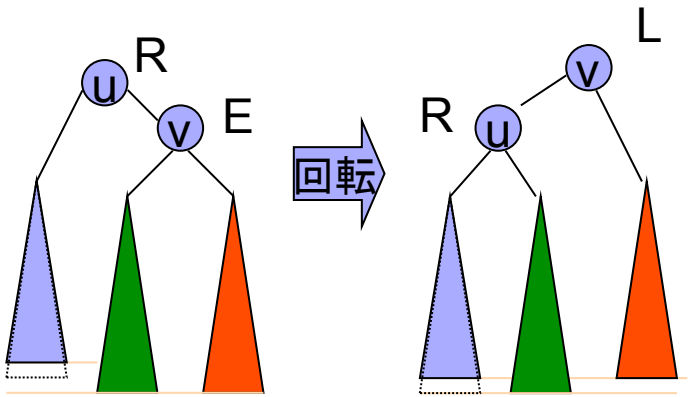
Step 2 s(u)=Eとなるまで次のことを繰り返す。

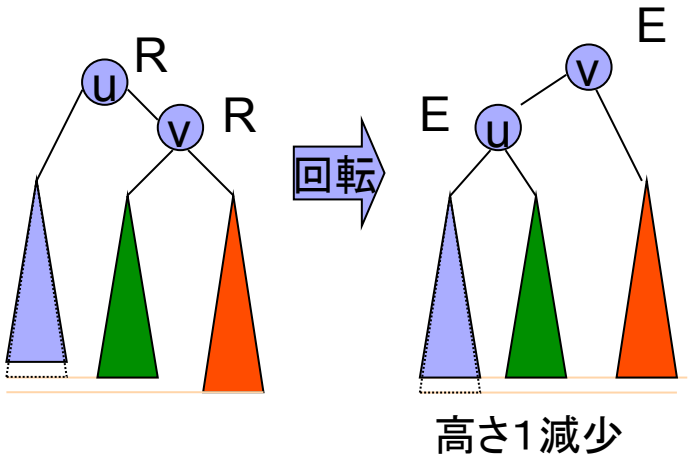
1. if s(u)=LかつT_L^u が低くなった or
   s(u)=RかつT_R^u が低くなった then
   s(u)←E
   else if s(u)=RかつT_L^u が低くなった or
   s(u)=LかつT_R^u が低くなった then
   (1) 回転
   (2) s(u)≠Eならば停止
2. u←uの親

Step 3 s(u)の更新（低くなった方の逆(L or R)に更新）



[削除に伴う回転操作]





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、

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。

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

$F(h)=f(h)+1$とすれば

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

F(h)はフィボナッチ数列だから

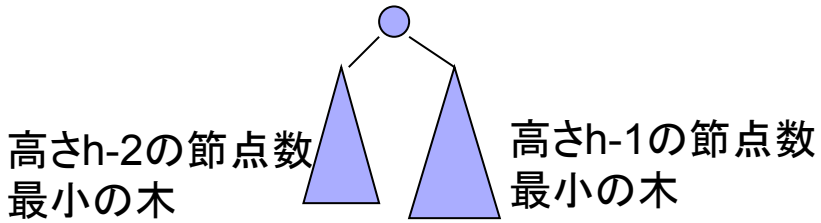
$$F(h)=(\varphi_1^{h+3}-\varphi_2^{h+3})/\sqrt{5}$$

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、

$$f(h) \leqq n$$

高さhの節点数最小の木



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

$|\varphi_2|<1$だから

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

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

したがってh=O(log n)