アルゴリズムとデータ構造

第6a回探索のためのデータ構造（2）

第5回　探索のためのデータ構造（1）

- 今日の内容：
  - 復習：辞書、二分探索木
  - 平衡探索木とは何か
  - 発展：平衡探索木の実装
- ポイント
  - 二分探索木の最悪時の演算時間を改善できること
  - 「平衡係数」とは何かを理解しよう
  - 発展：挿入と削除ごとに、どうやって平衡した形を保てばよいか？

前回の復習

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

2分探索木　（辞書の実装）

重要

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

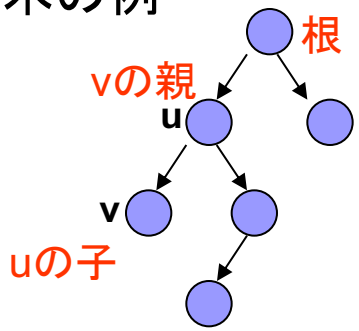
2分木の各節点にSの要素を辞書に適した構造で格納することを考える。

2分探索木(binary search tree)とは

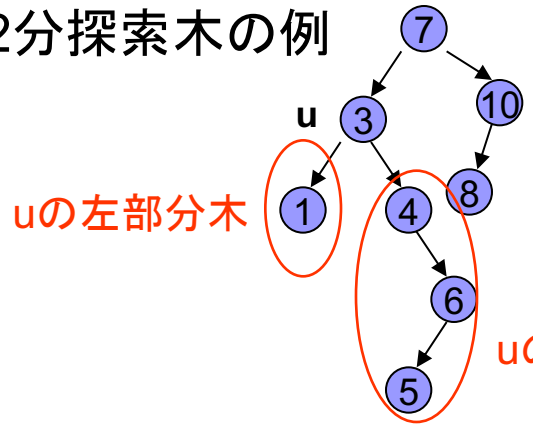
任意の節点uに対して次の条件が成り立つ2分木

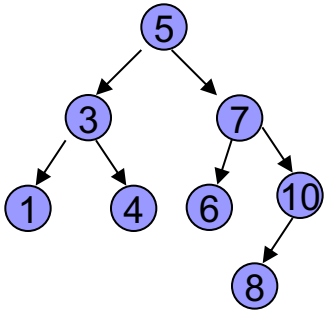
uの左部分木の任意の節点の要素<
uの要素<uの右部分木の任意の節点の要素

2分木の例



2分探索木の例



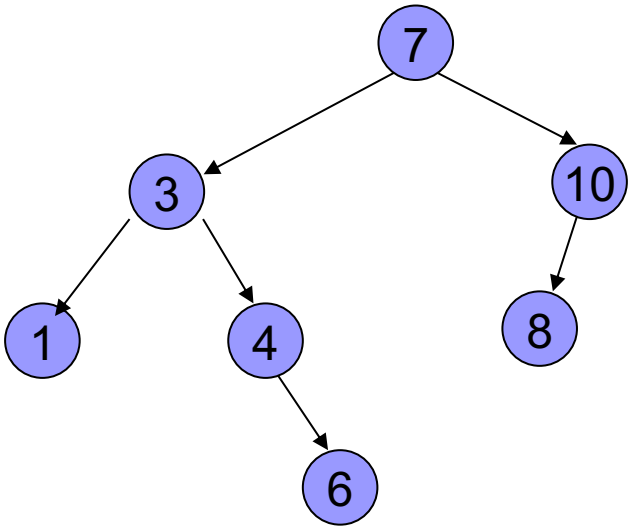


同じ要素が格納された異なる2分探索木

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

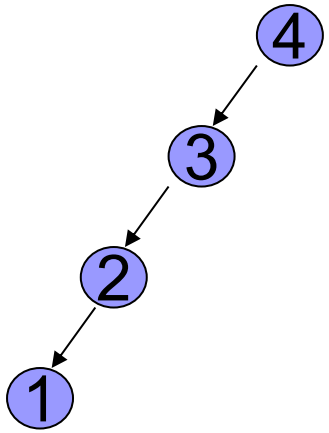
S_1 = 7, 3, 10, 1, 4, 8, 6



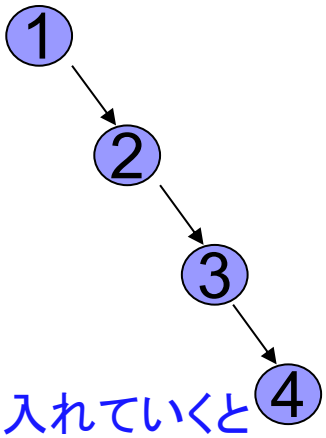
大きい順
格納

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）



S_3 = 1, 2, 3, 4（昇順）



順番に整列して入れていくと
形が偏りそうだ

重要

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がSの場合、 xを要素としてもつ節点の深さに比例する。

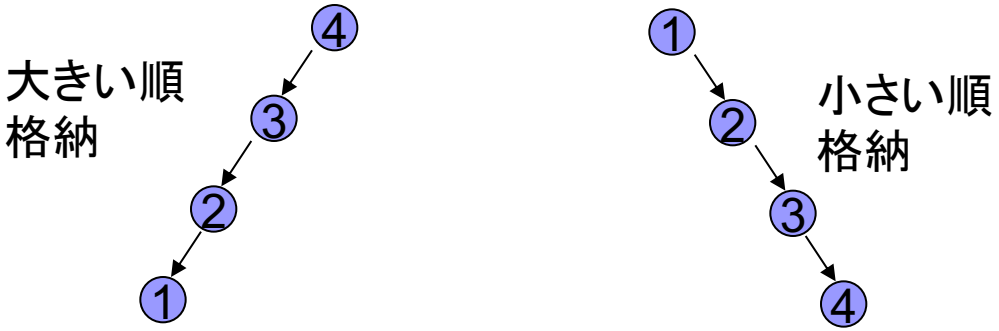
(2つの子をもつ節点のdeleteはその節点の右部分木の最小要素の節点の深さに比例する)

xが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=φ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)

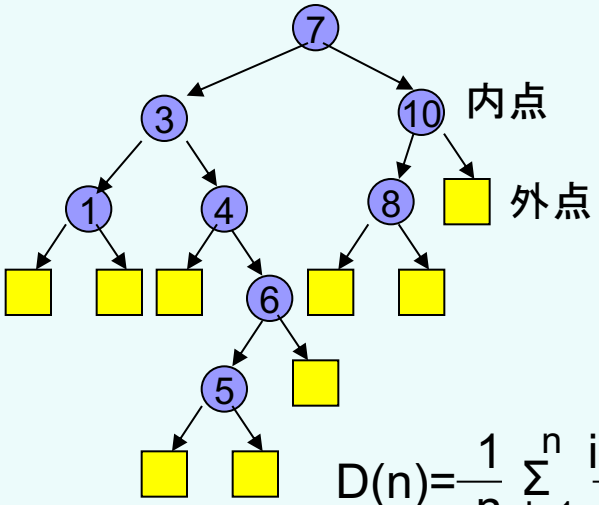
節点の深さの期待値はO(log n)

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

発展

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。

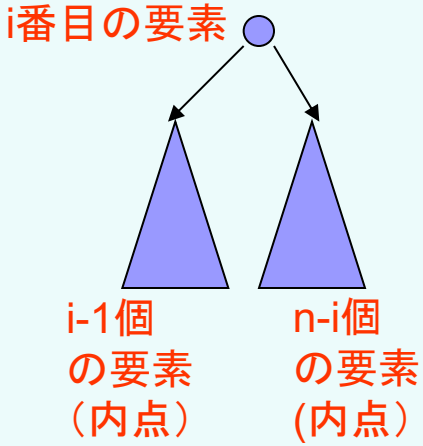


外点の深さの平均≧内点の深さの平均

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



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

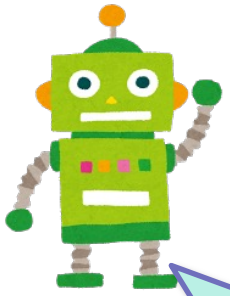
$$=\frac{2}{n(n+1)}\left(\left(\frac{2}{n(n-1)}\sum_{i=1}^{n-1}iD(i-1)+1\right)\frac{n(n-1)}{2}-\frac{n(n-1)}{2}+nD(n-1)\right)+1$$

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

議論

まとめと自然な疑問 (natural question)

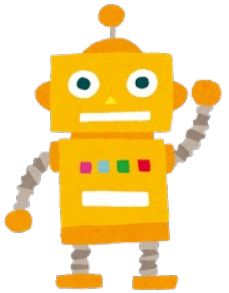
二分探索木 入力としてn個の要素を辞書に格納すると仮定する



2分探索木では，
探索・挿入・削除演算を
O(n)最悪時間とO(log n)平均時間で行う

ふつうは速いけど，苦手な入力だと，ときどき，すごく遅くなってしまいます!

平衡(二分)探索木



探索・挿入・削除演算を
O(log n)最悪時間にできるか？

どんな入力でも，高速処理O(log n)を保証します!

今日の前半

平衡探索木

この部分の授業の受け方

- この回は、発展内容です。結果だけが理解できれば良いです。ただし、アイディア（考え方）は重要ですので、理解しましょう。
- 平衡探索木は，「バランスをとる」ための，基本アイディアが，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。興味ある人は、[発展]のスライドで考えてみるとよいです。

平衡探索木

（文献追補）

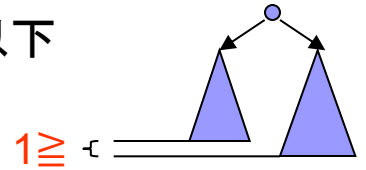
平衡探索木(balanced search tree)とは

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

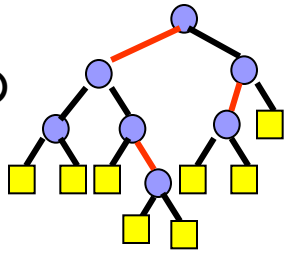
[主な平衡探索木]

授業では、基本となるAVL木を勉強します

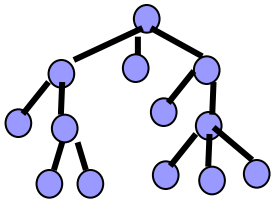
AVL木　どの節点においても、その左部分木と右部分木の高さの差が1以下である2分探索木。AVLは2人の提唱者の頭文字
(G. M. Adel'son-Vel'skii and Y. M. Landis, 1962)



2色木（赤黒木）(Bayer 1972, Guibas, Sedgewick, 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)



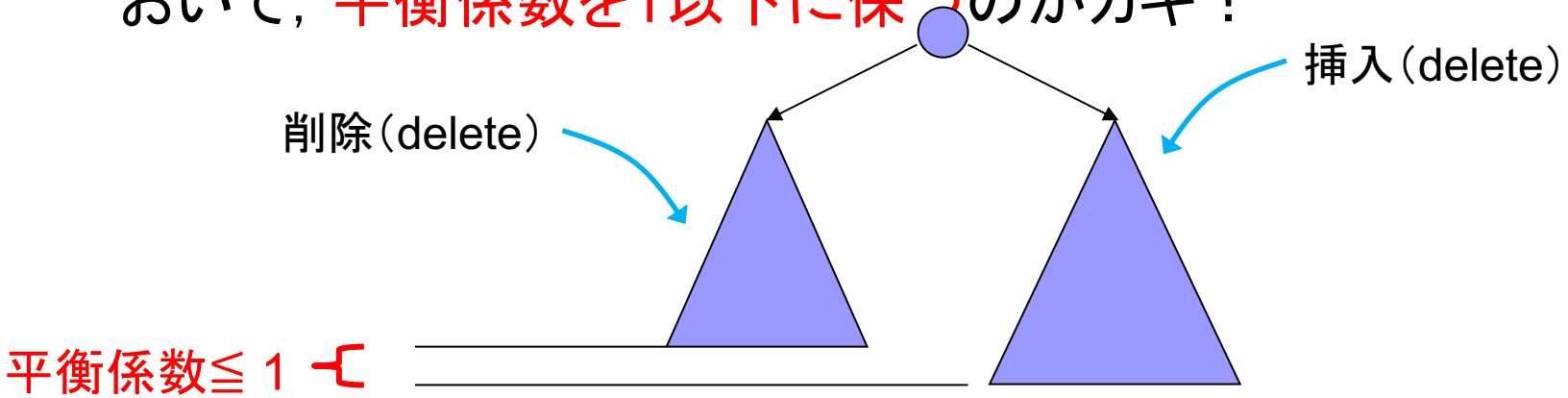
2-3 木　m=3の場合のB木 (Hopcroft, 1970, AHU 1974)

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？

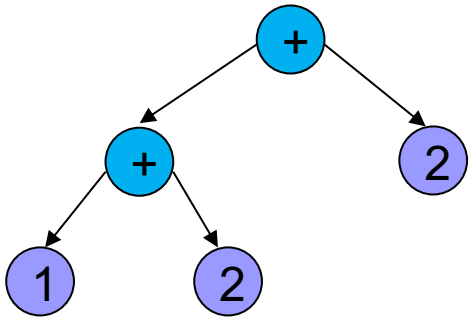
基本アイディア：AVL木

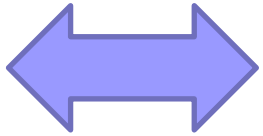
- 基本的な平衡2分木の一つ.
  - 1962年に，ロシアの情報科学者であるG. M. Adel'son-Vel'skii and Y. M. Landisによって提案された
- どの節点においても、その平衡係数が1以下である2分探索木。
  - 節点の「平衡係数」とは，2分木において，その頂点の左部分木と右部分木の高さの差のこと.
- 基本的なアイディアとして，挿入（delete）と削除（delete）において，平衡係数を1以下に保つのがカギ！

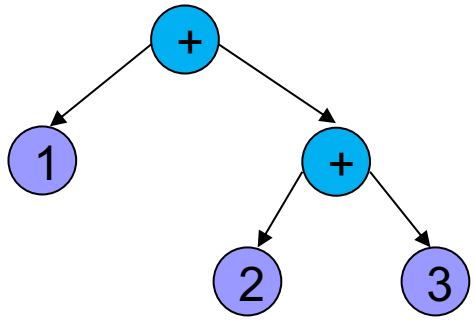


基本アイディア: AVL木

- 挿入(delete)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



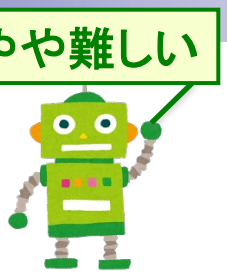




s = (1 + 2) + 3

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。

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

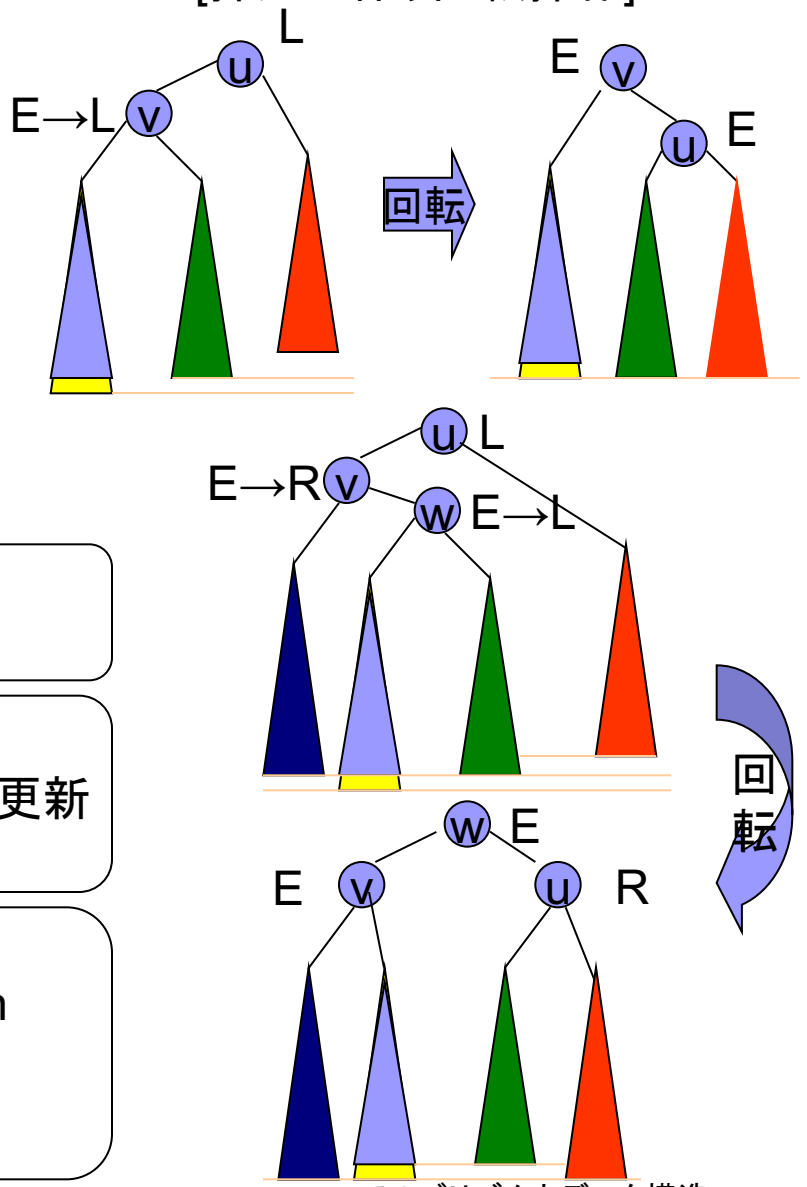
Step 1 一般の2分木の同じように挿入を行う。
挿入した節点の親節点をuとする。

Step 2 s(u)≠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)

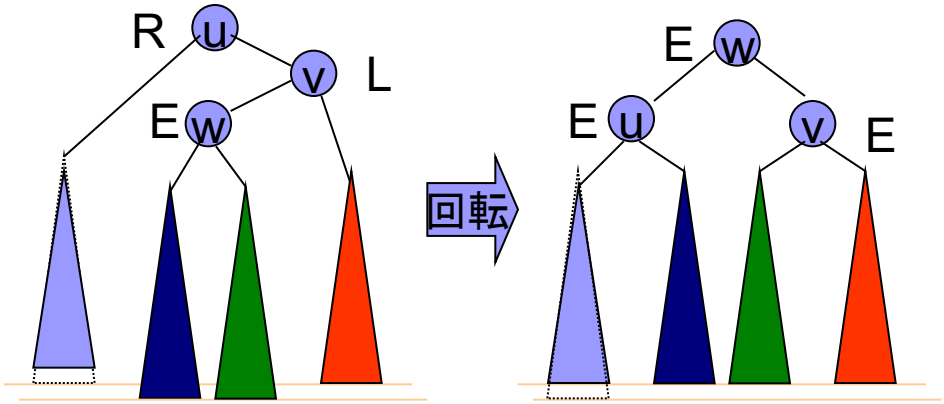
[AVL木のdelete(x,S)操作]

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。

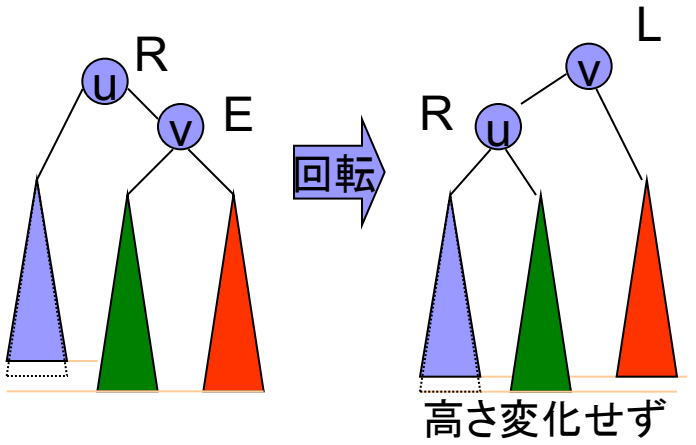
1. if s(u)=LかつT_L^uが低くなった or
   s(u)=RかつT_R^uが低くなった then
   s(u)←E
   else if s(u)=RかつT_L^uが低くなった or
   s(u)=LかつT_R^uが低くなった then
   (1) 回転
   (2) s(u)≠Eならば停止
2. u←uの親

Step 3 s(u)の更新（低くなった方の逆(L or R)に更新）

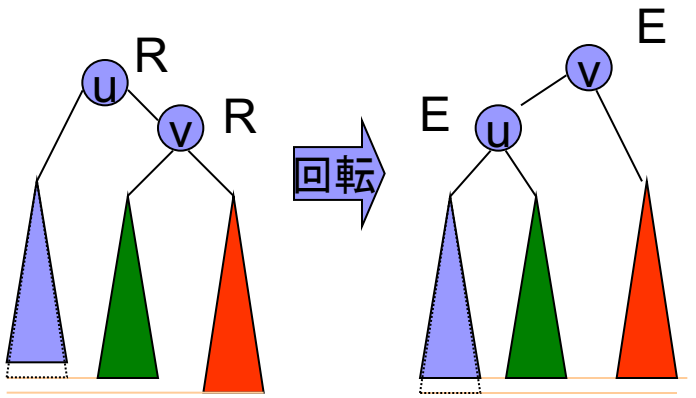


高さ1減少

[削除に伴う回転操作]



高さ変化せず



高さ1減少

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。

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

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

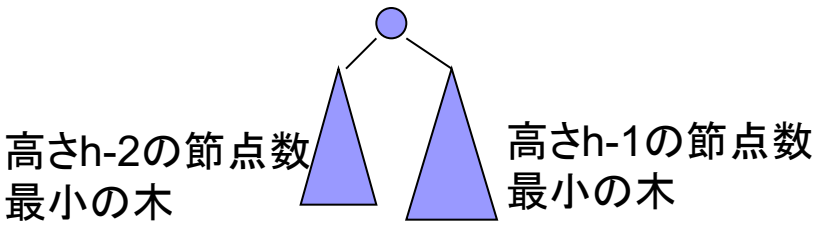
$$F(h)=(\varphi_1^{h+3}-\varphi_2^{h+3})/\sqrt{5}$$

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

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

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

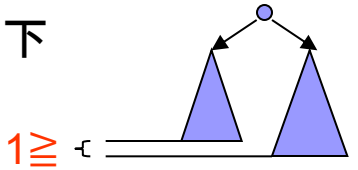
したがってh=O(log n)

いろいろな平衡探索木

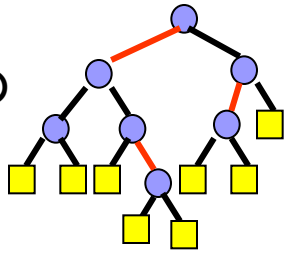
（文献追補）

[主な平衡探索木]

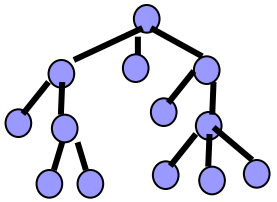
AVL木　どの節点においても、その左部分木と右部分木の高さの差が1以下である2分探索木。AVLは2人の提唱者の頭文字
(G. M. Adel'son-Vel'skii and Y. M. Landis, 1962)



2色木（赤黒木）(Bayer 1972, Guibas, Sedgewick, 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？

第5回　探索のためのデータ構造（1）

- 今日の内容：
  - 復習：辞書、二分探索木
  - 平衡探索木とは何か
  - 発展：平衡探索木の実装
- ポイント
  - 二分探索木の最悪時の演算時間を改善できること
  - 「平衡係数」とは何かを理解しよう
  - 発展：挿入と削除ごとに、どうやって平衡した形を保てばよいか？