動的な集合のためのデータ構造

アルゴリズム論

中野

Note 6　動的な集合2

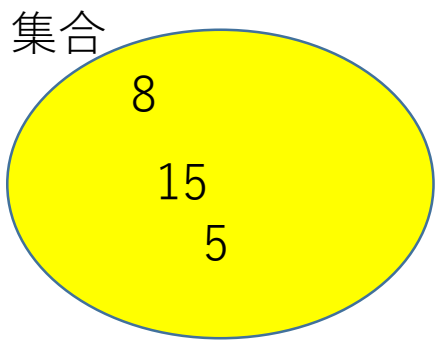
2020.4.20 作成

20205.28update　2021.5.20　　5.24

動的な集合

データの動的な集合Sがある。
各データは (1)固有のkeyと(2)key以外の部分(satellite date)からなるとする。
これを効率的に格納するデータ構造を考えよう。

動的 = データの追加や削除がある。

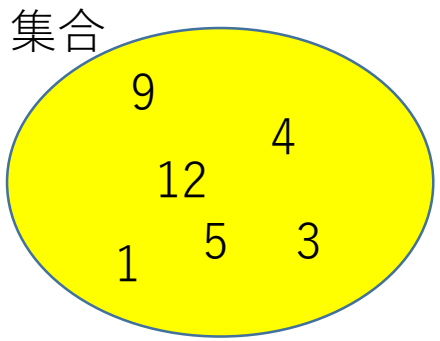


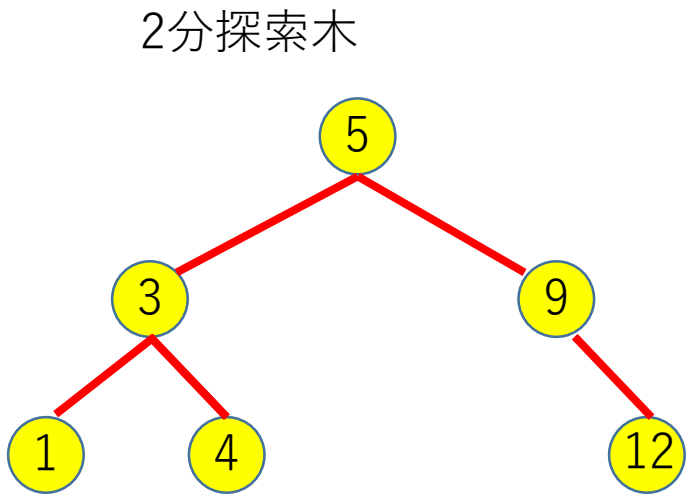
基本辞書操作

- Search(k)　keyがkであるデータの探索
- Insert(x)　新データxの追加
- Delete(x)　データxの削除(Delete(key)の場合もあり)
- Minimum　最小のkeyのデータの探索
- Maximum 最大のkeyのデータの探索
- Successor(x) データxの次に大きなkeyを持つデータの探索
- etc.

(注意 データとkeyの区別がない場合、つまりsatellite dataがない場合もある。)

2分探索木(Binary Search Tree)





整数の集合Sを2分木に格納する。
各データ=key を2分木の各点に格納する。
(注意 satellite dataも扱えるように改造できる。)
各点に、keyの他に(satellite dataもあるかも)、
3つのポインタ(left,right,parent)を格納する。
ただし、根のparentポインタは常にNILとする。
さらに次の性質を満たすとする。

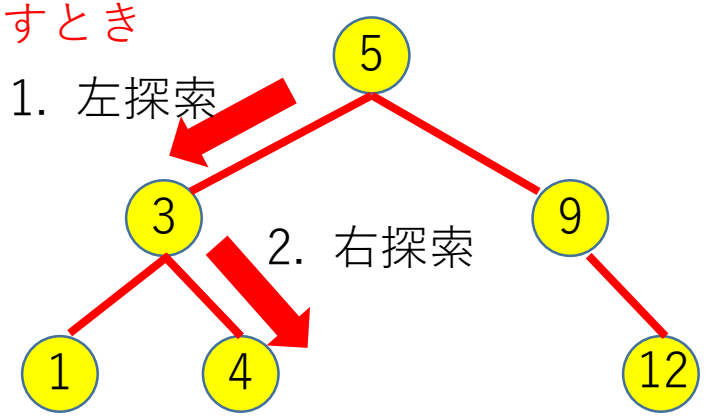
　任意の点xについて
　xのkeyは、任意のxの左の部分木中の点のkeyよりも大きい、また、
　xのkeyは、任意のxの右の部分木中の点のkeyよりも小さい。

木の点をinorderで並べると、keyは昇順に並ぶ。

データの探索・追加

2分探索木

例　4 を探すとき



基本辞書操作

木の高さをhとすると

Search(k)　keyがkであるデータの探索　O(h) time

Insert(x)　新データxの追加　O(h) time

Detete(x)　データxの削除　O(h) time

Minimum　最小のkeyのデータの探索　O(h)time

Maximum　最大のkeyのデータの探索　O(h)time

Searchの時間は　O(h) だけど。。

(クイズ 1)

n個のデータを順次、Insertしたとき、最大のhはいくつかな？ 最小のhはいくつかな？

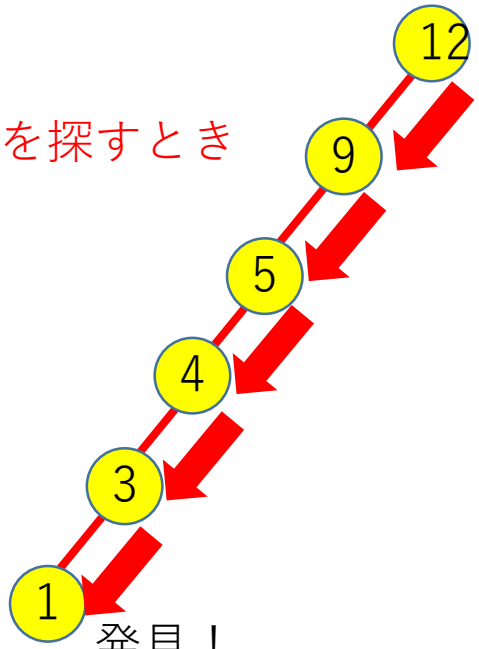
(クイズ 2)

Successor(x) はどのようにすればいいのかな？ 時間はどのくらいかかるかな？

(クイズ 3)

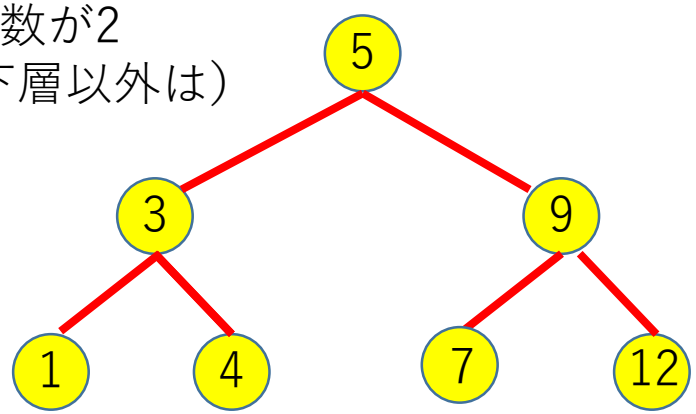
Delete(x)はどのようにすればいいのかな？

例　1を探すとき



データの探索・追加

完全2分木
子の数が2
(最下層以外は)



基本辞書操作

木の高さをhとすると

Search(k)　keyがkであるデータの探索　O(h) time

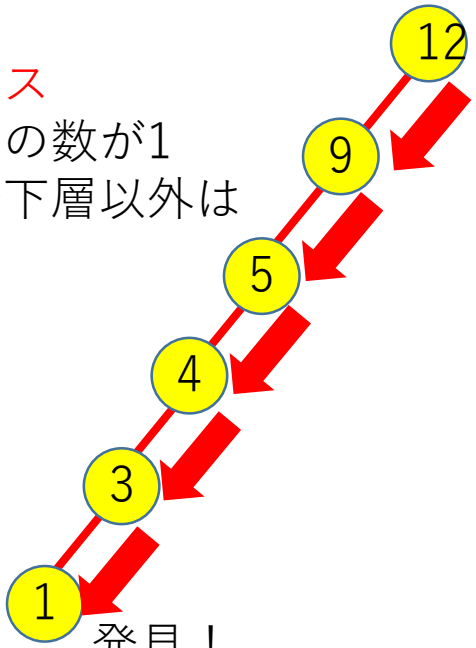
Insert(x)　新データxの追加　O(h) time

Detete(x)　データxの削除　O(h) time

Minimum　最小のkeyのデータの探索　O(h)time

Maximum　最大のkeyのデータの探索　O(h)time

パス
子の数が1
最下層以外は



発見！

(クイズ 1)

n個のデータを順次、Insertしたとき、最大のhはいくつかな？ 最小のhはいくつかな？

最大　高さhは　n

最小　高さhは　log n

n	10^3	10^6	10^9
log n	10	20	30

データの探索・追加

基本辞書操作

木の高さをhとすると

Search(k)　keyがkであるデータの探索　O(h) time

Insert(x)　新データxの追加　O(h) time

Detete(x)　データxの削除　O(h) time

Minimum　最小のkeyのデータの探索　O(h)time

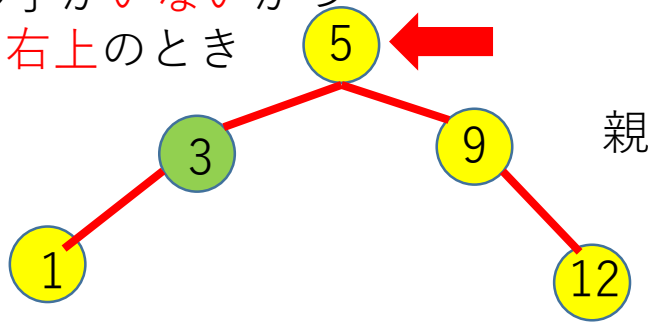
Maximum　最大のkeyのデータの探索　O(h)time

(クイズ 2)

Successor(x) はどのようにすればいいのかな？

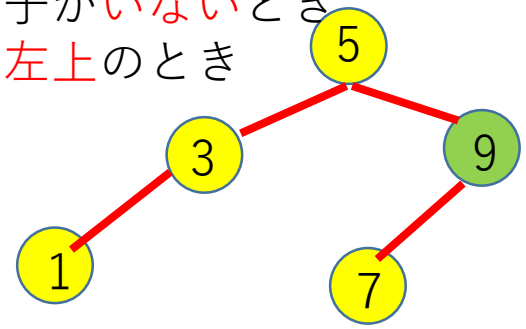
時間はどのくらいかかるかな？

場合1　右の子がいないかつ
親は右上のとき



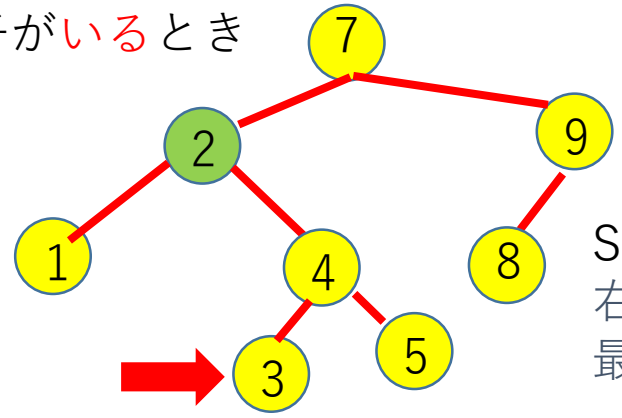
親がSuccessor

場合2　右の子がいないとき
親は左上のとき



Successorなし
そのデータが
最大！

場合3　右の子がいるとき



Successorは
右の子の子孫の
最小データ

データの削除

子の個数で場合わけ

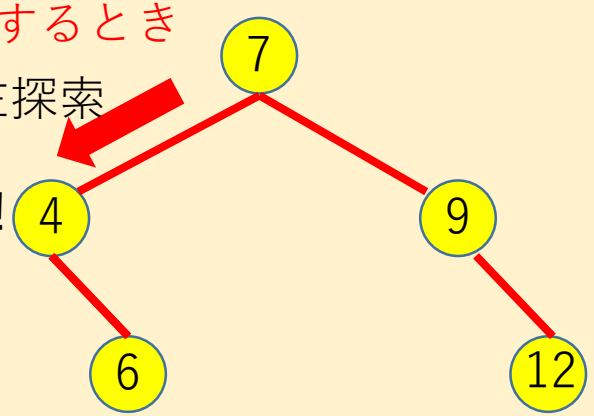
基本辞書操作

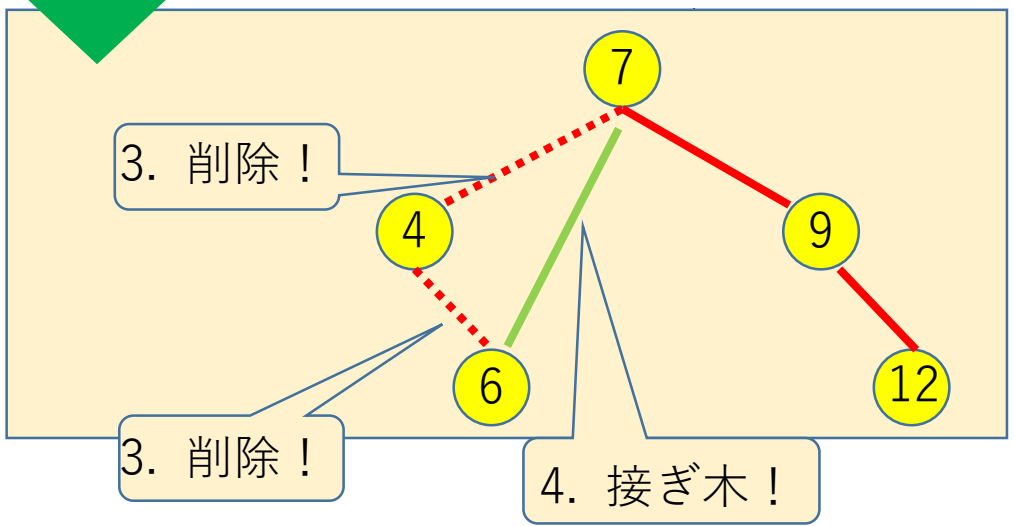
Detete(x)　データxの削除　　O(h) time

場合1　4 を削除するとき

1. 左探索

2. 発見！
子は1個





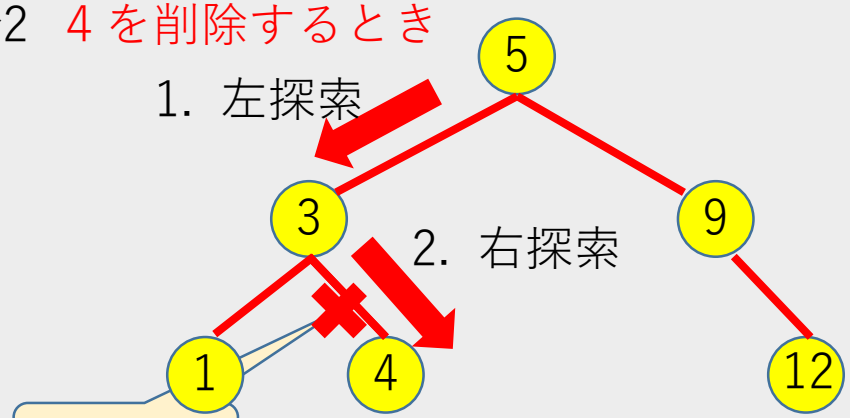
場合2　4 を削除するとき

1. 左探索

2. 右探索

4. 削除！

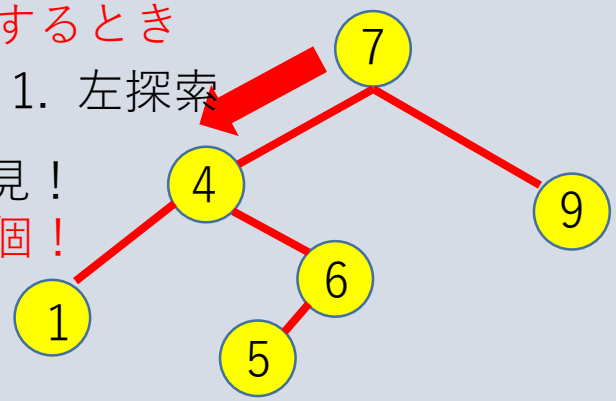
3. 発見！子はゼロ個



場合 3　4 を削除するとき

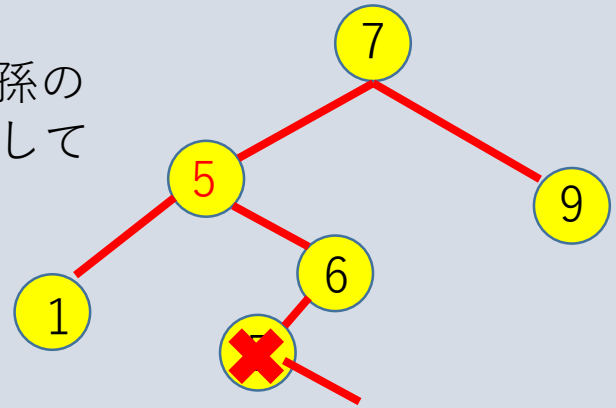
1. 左探索

2. 発見！
子は2個！

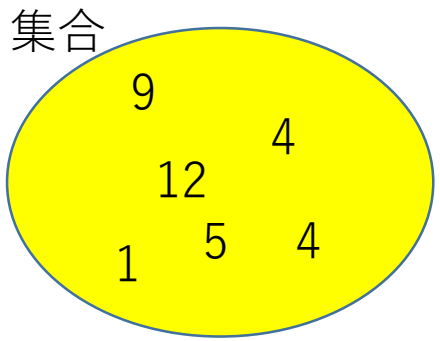


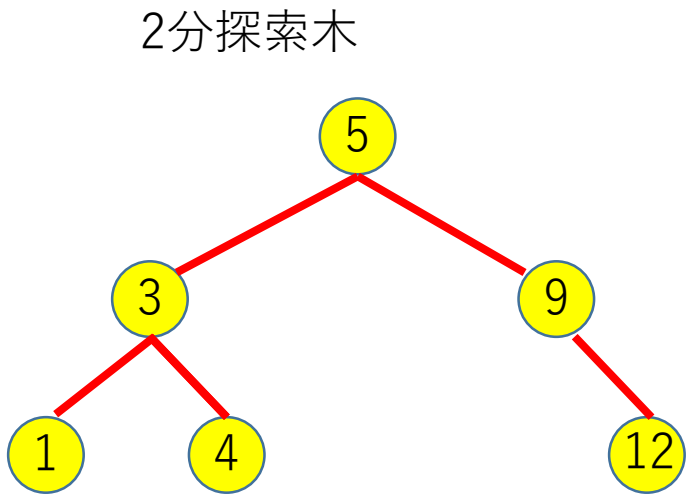
3. 4の右の子の子孫の
最小データを削除して
4に上書き

4. 必要ならば
再帰的に削除



2分探索木(Binary Search Tree)





基本辞書操作

木の高さをhとすると

Search(k)　keyがkであるデータの探索　O(h) time

Insert(x)　新データxの追加　O(h) time

Detete(x)　データxの削除　O(h) time

Minimum　最小のkeyのデータの探索　O(h)time

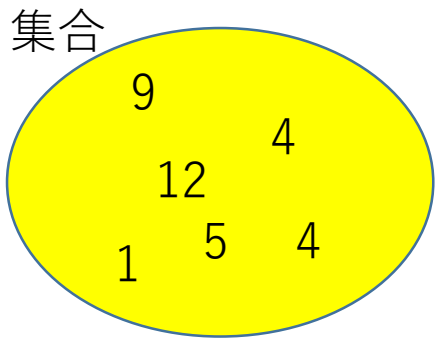
Maximum　最大のkeyのデータの探索　O(h)time

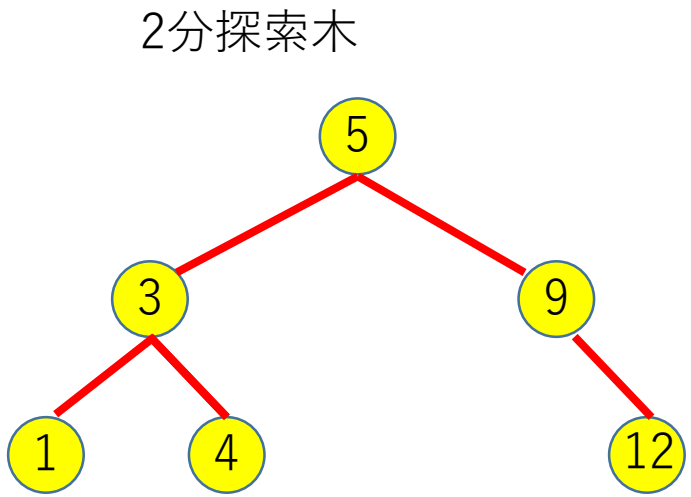
もしhが小さいと　高速

もしhが大きいと　低速

n	10^3	10^6	10^9	10^{12}
log n	10	20	30	40

バランス木(balanced tree)





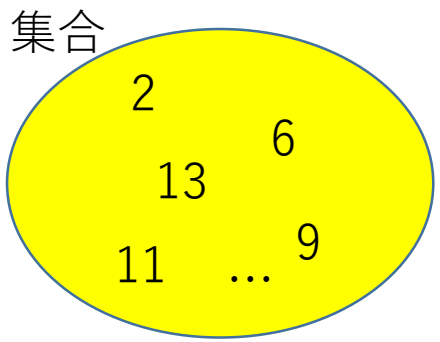
バランス木 ＝
2分探索木＋高さがO(log n)になるように工夫

2-3 treeとかAVL木とかいろいろあります。
本講義では、赤黒木を解説します。

n	10^3	10^6	10^9	10^{12}
log n	10	20	30	40

赤黒木

(1)2分木の各点を赤か黒で塗る
(2)葉は省略する。黒とみなす。
(3)根は黒である
(4)赤点の子は黒（黒多め！）
(5)各点から任意の子孫までの
パス上に常に同数の黒を含む



n	10^3	10^6	10^9	10^{12}
log n	10	20	30	40

補題 1

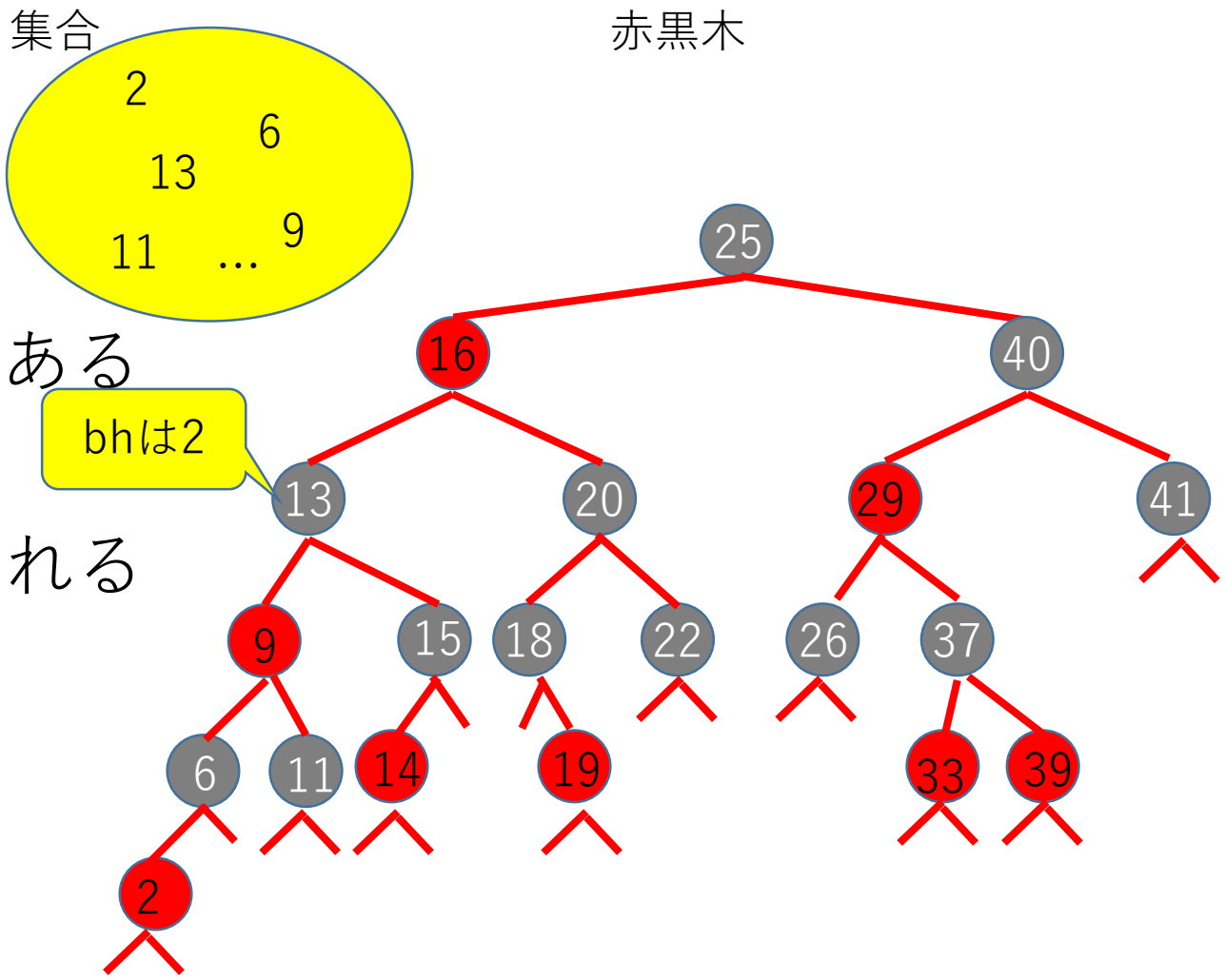
赤黒木の高さは高々 2 log (n+1)である

(証明)

bh(x): 点xから子孫までの道に含まれる黒点の個数

(xは含まない、

x以外の省略した葉は含む)



Claim 1

任意の点 x を根とする部分木は少なくとも $2^{bh(x)}-1$個の内点を含む。

(base) xが葉

$2^{bh(x)}-1 = 0$ なのでOK

(induction)

xの2つの子yとzまでＯＫとする

bh(y)はbh(x)かbh(x)-1のいずれか

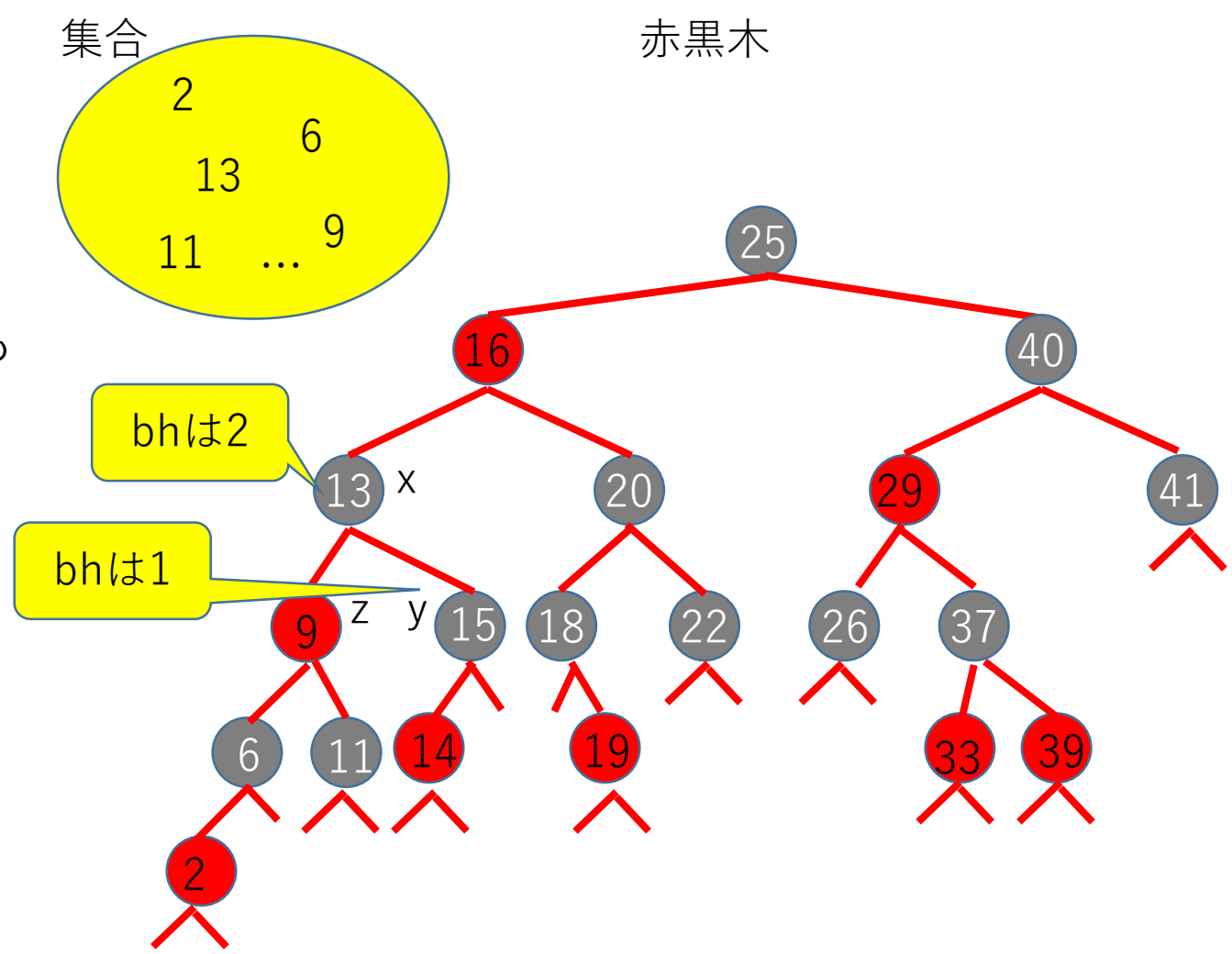
bh(z)はbh(x)かbh(x)-1のいずれか

yは少なくとも$2^{bh(y)}-1$個の子をもつ

zは少なくとも$2^{bh(z)}-1$個の子をもつ

よってxを根とする部分木は少なくとも

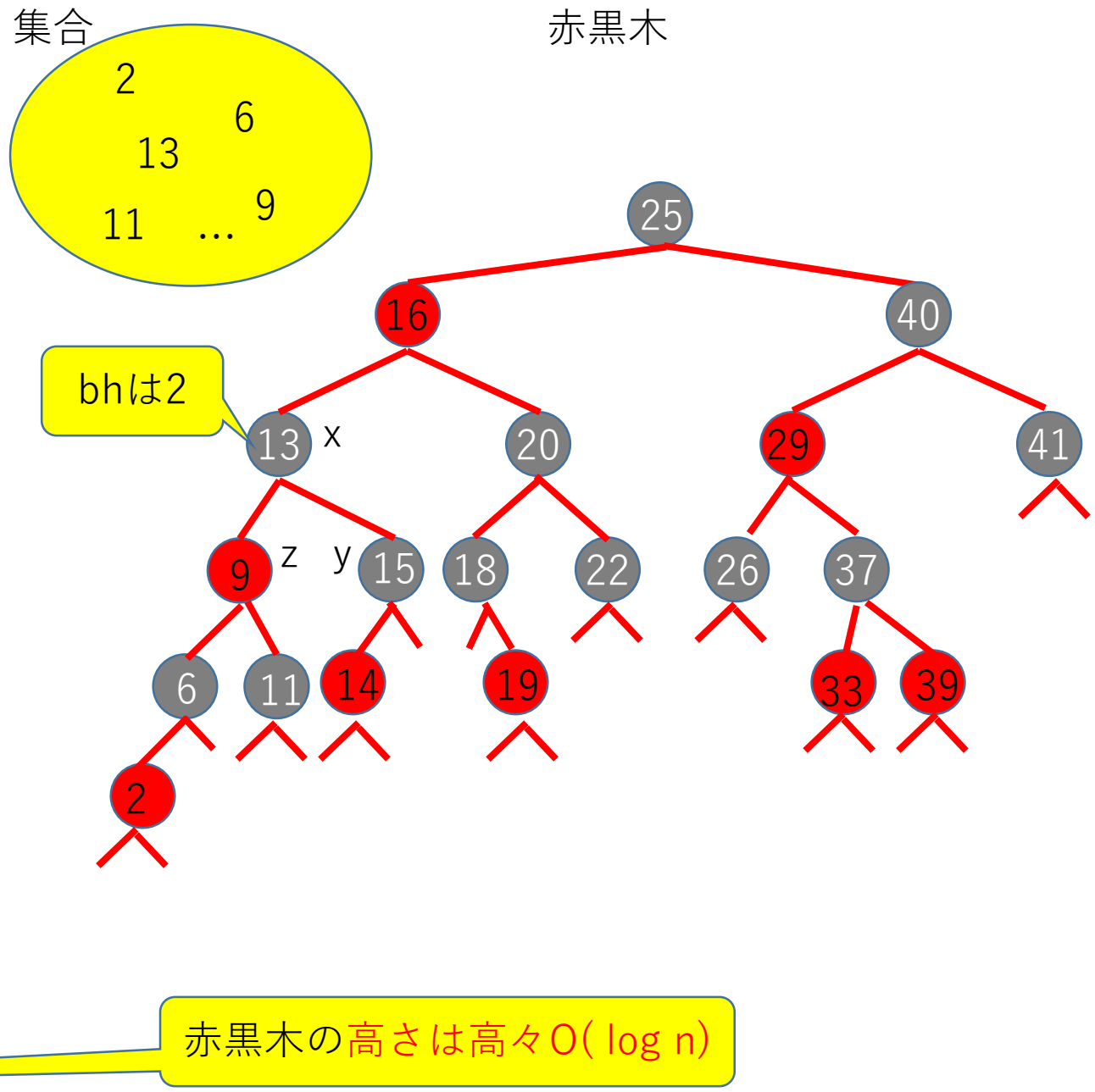
$(2^{bh(x)-1}-1)+ (2^{bh(x)-1}-1)+ 1= 2^{bh(x)}-1$ 個の内点を含む



Claim 1
任意の点 x を根とする部分木は少なくとも
$2^{bh(x)}$-1個の内点を含む。

Claim 2
赤黒木の高さをhとし、rを根とすると
bh(r) ≧　(h-1)/2
(赤の子は必ず黒 and 根は黒なので)

Claim 1 と2より
赤黒木は少なくとも
$2^{(h-1)/2}$ -1 個の内点をもつ
n ≧ $2^{(h-1)/2}$ -1　これを解くと
log (n+1) ≧ (h-1)/2 より
h ≦ 2 log (n+1)+1 である



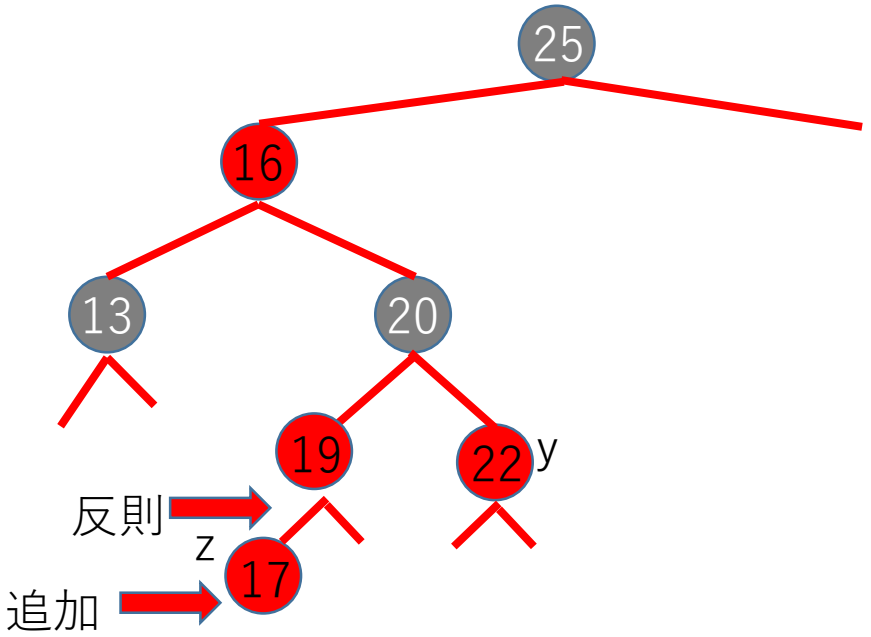
赤黒木の高さは高々O(log n)

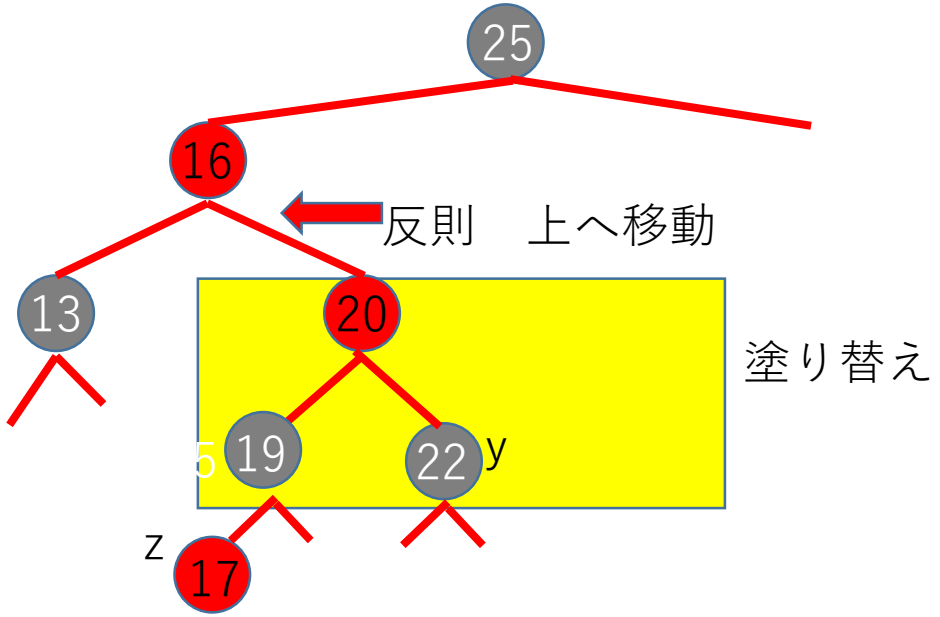
データ追加

Case 1　自分z赤　叔父赤

Case 2　自分赤右子　叔父黒

Case 3　自分赤左子　叔父黒





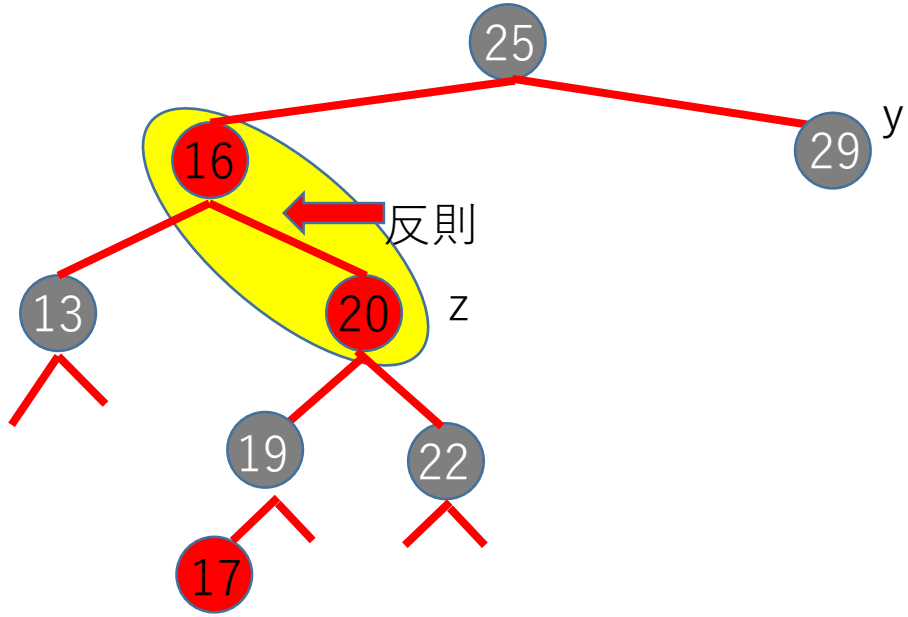
データ追加

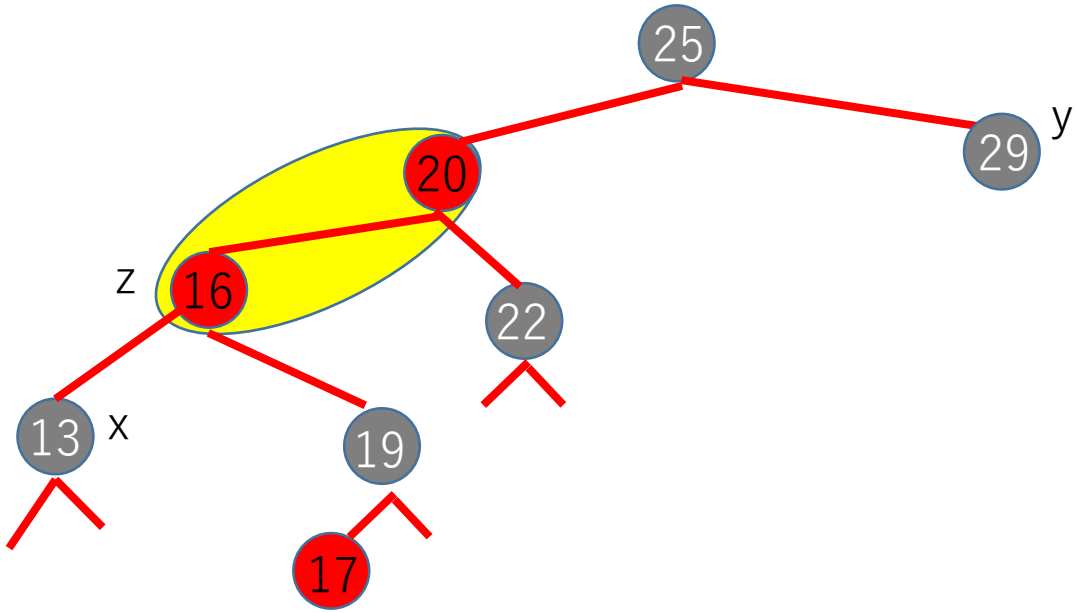
Case 1　自分z赤　叔父赤

Case 2　自分赤右子　叔父黒

　Case 3へ

Case 3　自分赤左子　叔父黒



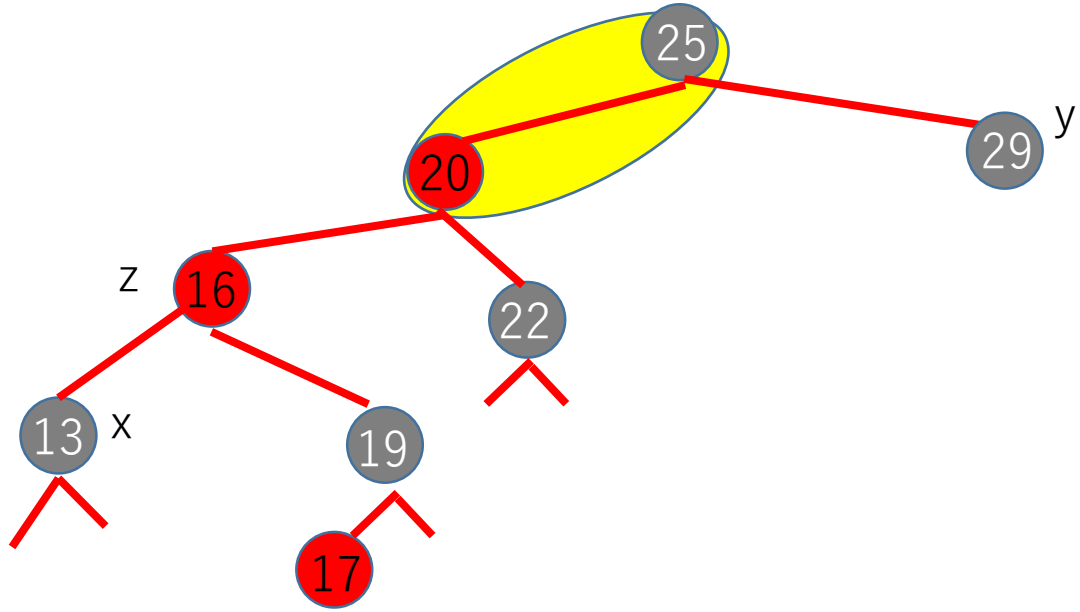


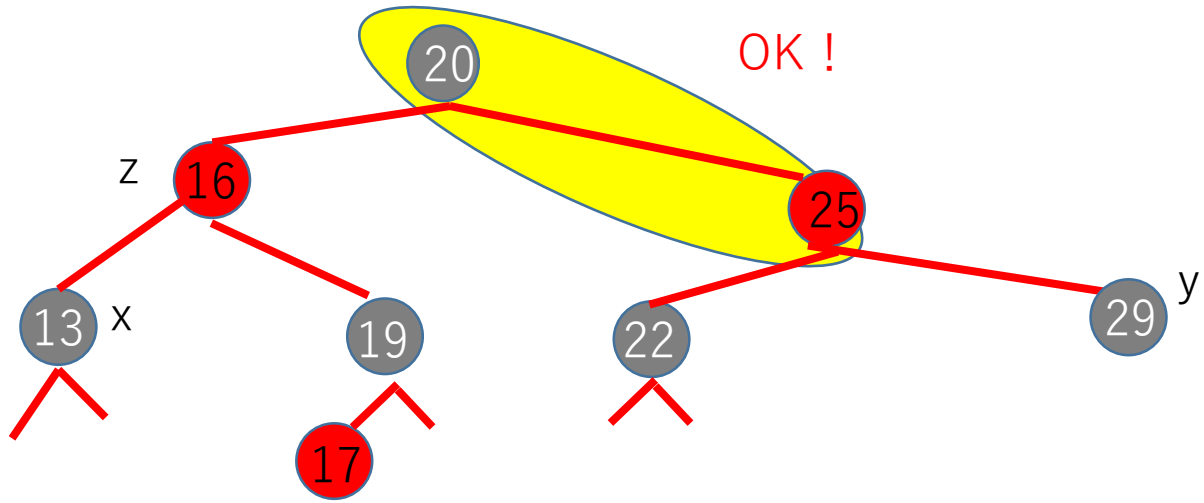
データ追加

Case 1　自分z赤　叔父赤

Case 2　自分赤右子　叔父黒

Case 3　自分赤左子　叔父黒





赤黒木

基本辞書操作

木の高さをhとすると

Search(k)　keyがkであるデータの探索
Insert(x)　新データxの追加

Detete(x)　データxの削除
Minimum　最小のkeyのデータの探索
Maximum　最大のkeyのデータの探索

すべて O(log n) 時間！