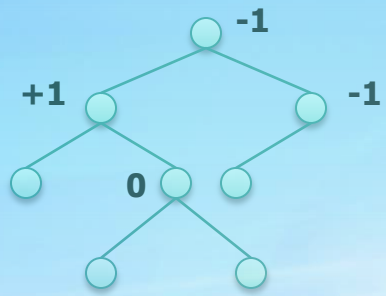
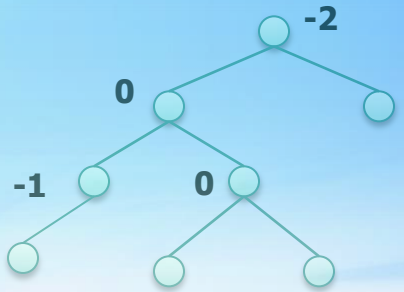
《数据结构》

平衡树（上）

主讲人：李清



平衡树



非平衡树

教学目标和要求

1.能够准确阐述平衡树的定义、性质

2.能够通过图示准确分析描述平衡树的插入删除算法

3.能够准确图示给定平衡树的插入删除过程

平衡树的基本概念

二叉平衡检索树（简称平衡树）

树的高度h限制在结点数n的对数阶范围内

$$h=O(\log n)$$

平衡树是一类树的总称（又称为平衡树模式）

平衡树的基本概念

AVL树

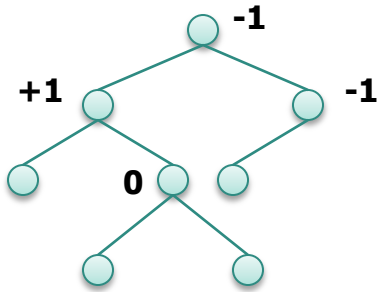
前苏联学者Adelson-Velskii和Landis的名字命名。

- h ≤1.45logn
- 二叉树的右、左子树的高度差不超过1
- 二叉平衡树

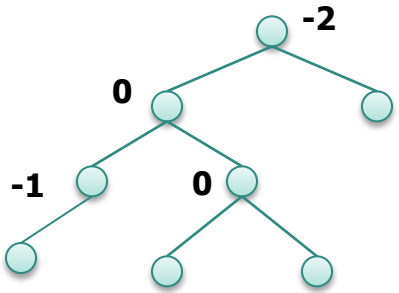
平衡树的基本概念

平衡因子

右、左子树高度之差。



平衡树



非平衡树

二叉平衡检索树

既是AVL树，也是检索树。

结点的平衡因子：0、1、-1

平衡树的基本概念

- 平衡树的查找、插入、构造、删除同检索树。
- 平衡树上查找、插入、删除一个结点所需时间不超过 O(1.45logn)。
- 当插入、删除破坏平衡条件时，要调整树结构，使它保持平衡（同时保中序）。

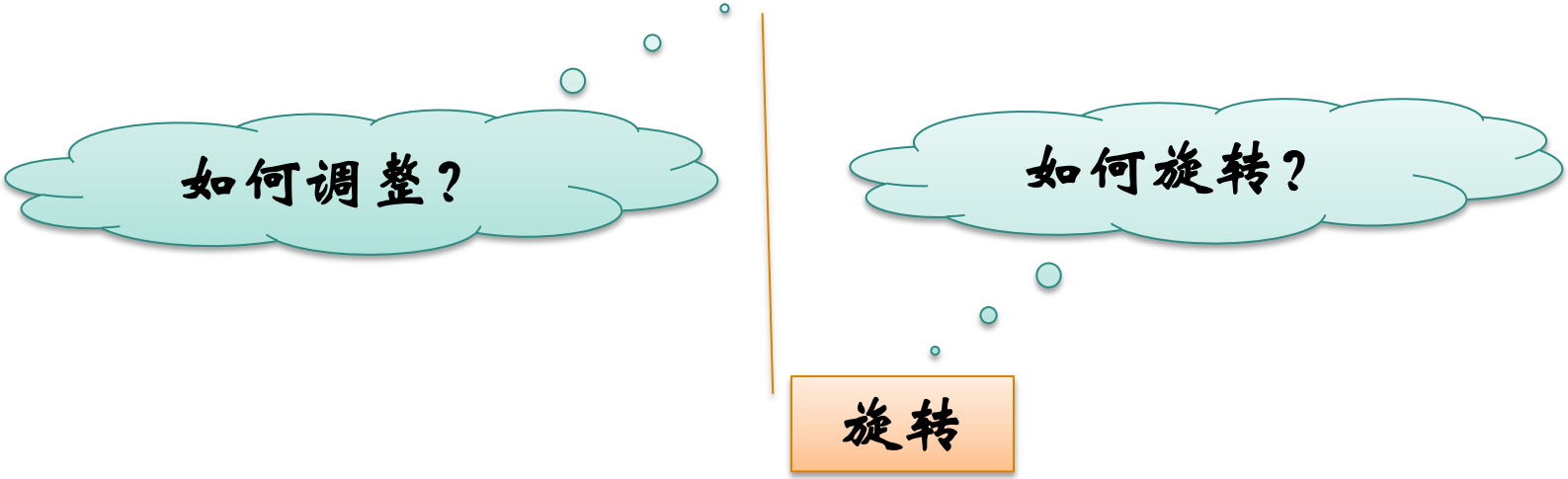
还能保持平衡吗？

插入、删除时平衡树不再平衡了，如何保持平衡？

平衡树的基本概念

分析

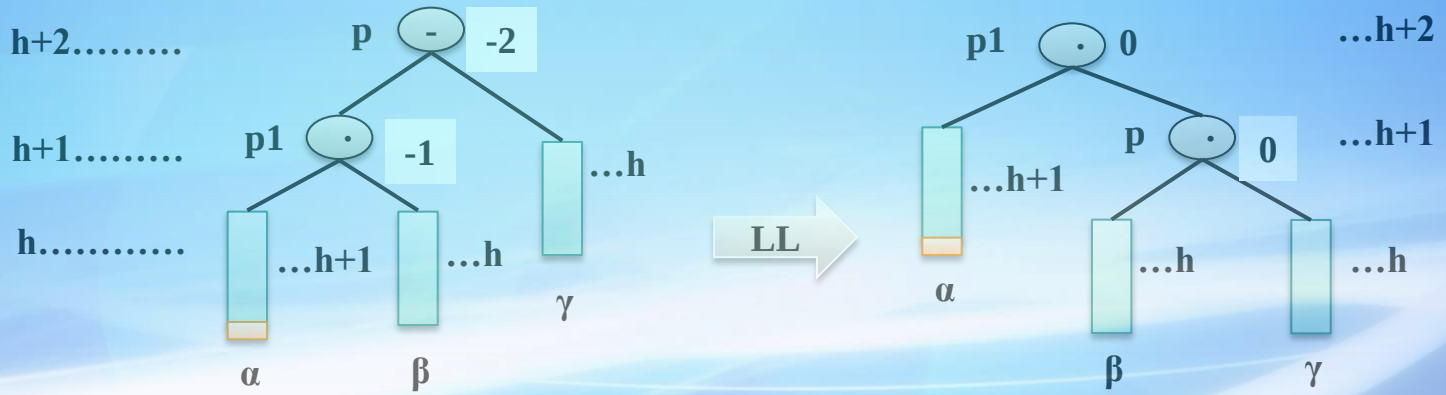
设p指向树中某结点，p->bal为其平衡因子，则插入、删除时， p->bal由1→2， 或 p->bal由-1→-2，也即以p为根的子树失衡，则要调整p子树使之平衡。



平衡树的插入

《数据结构》

主讲人：李清



平衡树的插入

- 像一般的**检索树**那样**插入**x（新叶）；
- 沿插入x的路径**返回**，**修改**x祖先的平衡因子；
- **回溯**中，一旦发现x的祖先p失衡

  由p->bal=1变成p->bal=2

  由p->bal=-1变成p->bal=-2

  旋转以p为根的子树，使之平衡

问题

如何旋转呢？

旋转后又会怎样呢？

平衡树的插入

旋转　　LL，LR，RR，RL。

设 P 是离插入结点最近的失衡祖先结点。

- 左子树增高：即　p->bal 由 -1 → -2 ，
  LL、LR旋转。
- 右子树增高：即　p->bal 由1 → 2，
  RR，RL旋转。

平衡树的插入----插入结点在左子树上

x插在p的左子树上，且使p的左子树高度增1

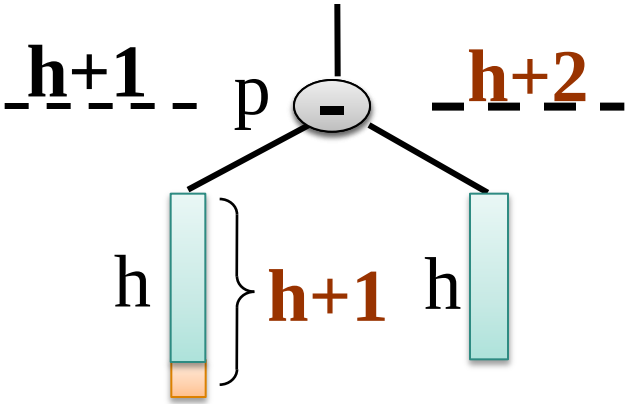
插入前：

- **p->bal=0（$h_R=h_L=h$）** ⊙
- **p->bal=1（$h_R=h$，$h_L=h-1$）** ⊕
- **p->bal=-1（$h_R=h$，$h_L=h+1$）** ⊖

右子树树高　　左子树树高

平衡树的插入----插入结点在左子树上

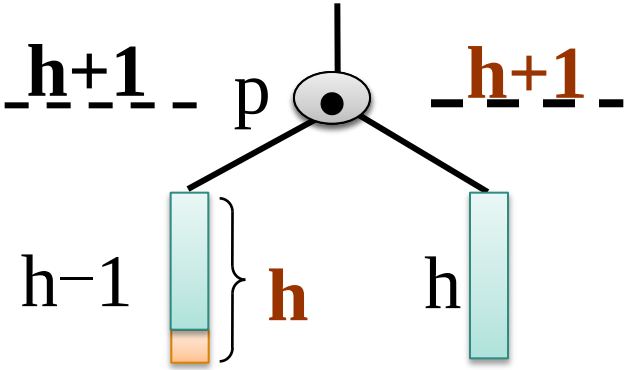
情况一：p->bal=0（$h_R=h_L=h$）



仍平衡，但树高增加，向上回溯

平衡树的插入----插入结点在左子树上

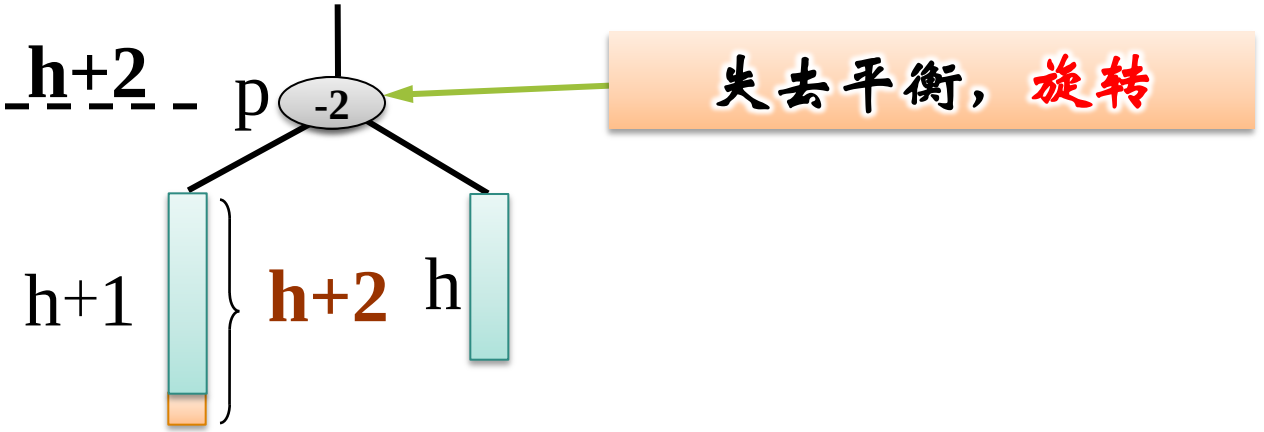
情况二： p->bal=1（h_R=h，h_L=h-1 ）



树高不变，平衡

平衡树的插入----插入结点在左子树上

情况三： p->bal=-1（$h_R=h$，$h_L=h+1$）



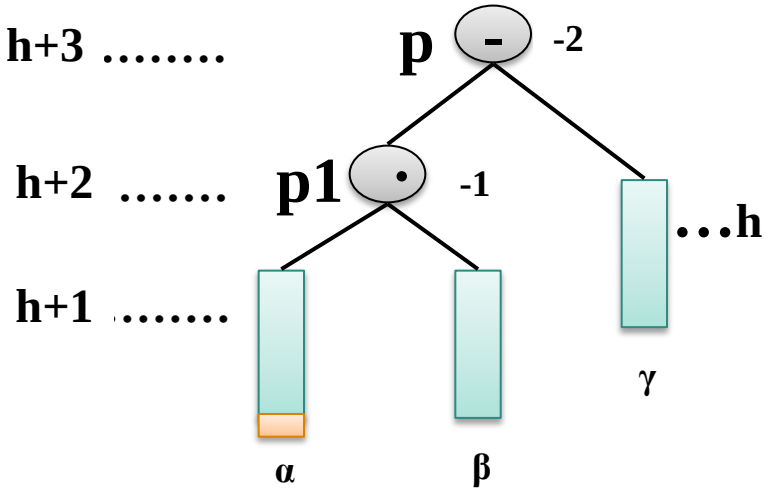
平衡树的插入----插入结点在左子树上

插入结点在失衡结点的左子树上

- LL旋转：在失衡结点的左儿子的左子树上
- LR旋转：在失衡结点的左儿子的右子树上

平衡树的插入----LL旋转，失衡点p的左儿子作根

插入前，p->bal=-1 且 p1->bal=0



旋转方式：

（1）使p1作为子树之根；

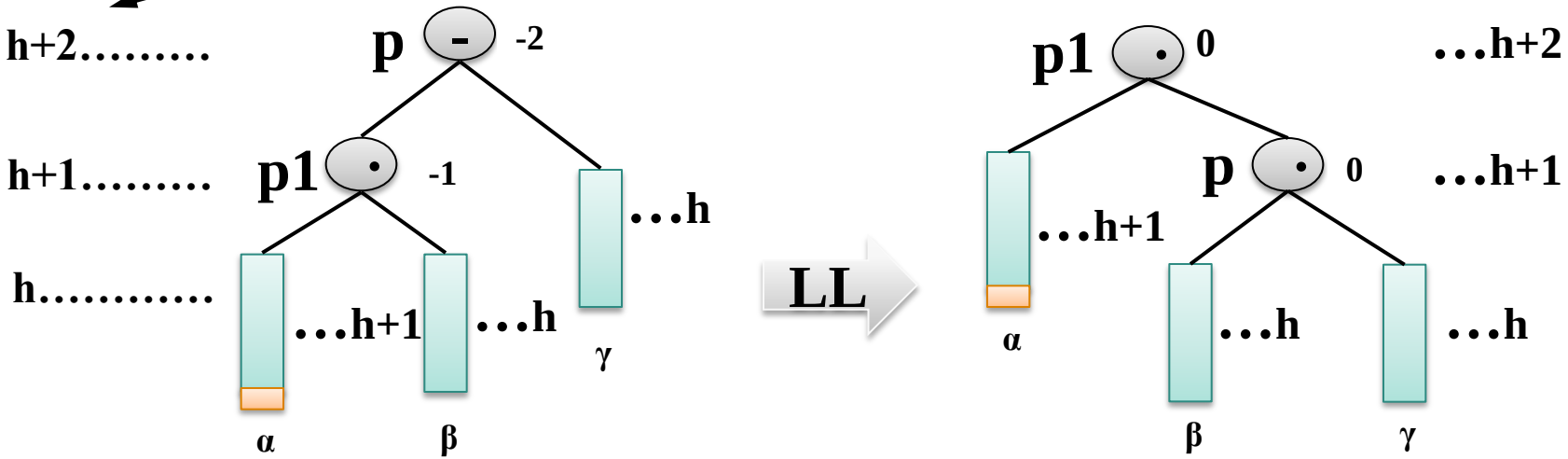
（2）p1的右儿子作p的左儿子；

（3） p作p1的右儿子；

插入点在左儿子的左子树上

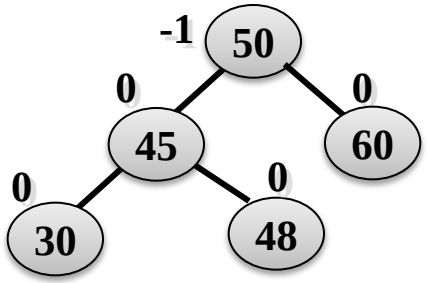
平衡树的插入----LL旋转，失衡点p的左儿子作根

平衡，且树高度不变，旋转结束



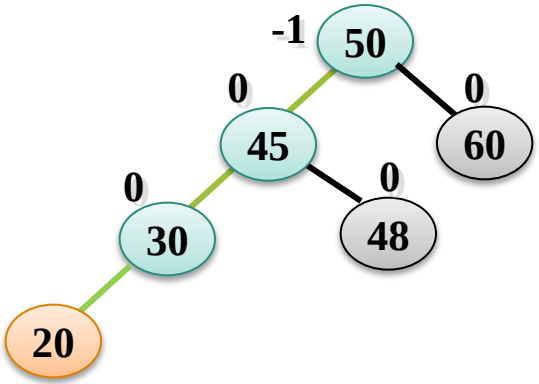
平衡树的插入----LL旋转，失衡点p的左儿子作根

示例： 插入20



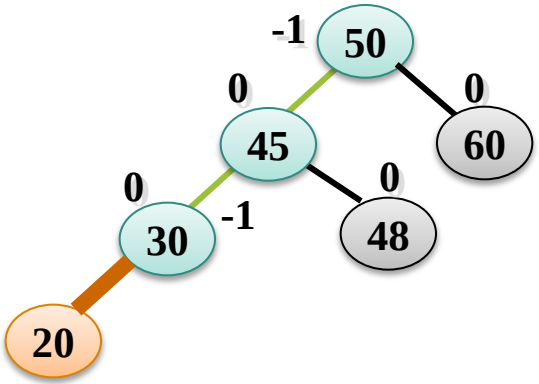
平衡树的插入----LL旋转，失衡点p的左儿子作根

示例： 插入20



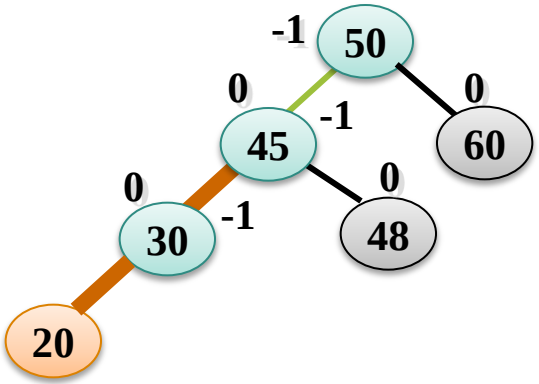
平衡树的插入----LL旋转，失衡点p的左儿子作根

示例： 插入20



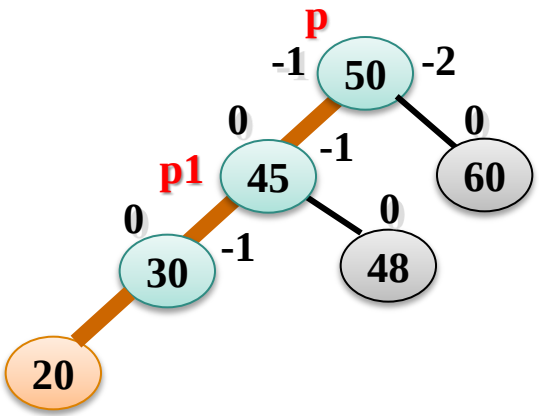
平衡树的插入----LL旋转，失衡点p的左儿子作根

示例： 插入20



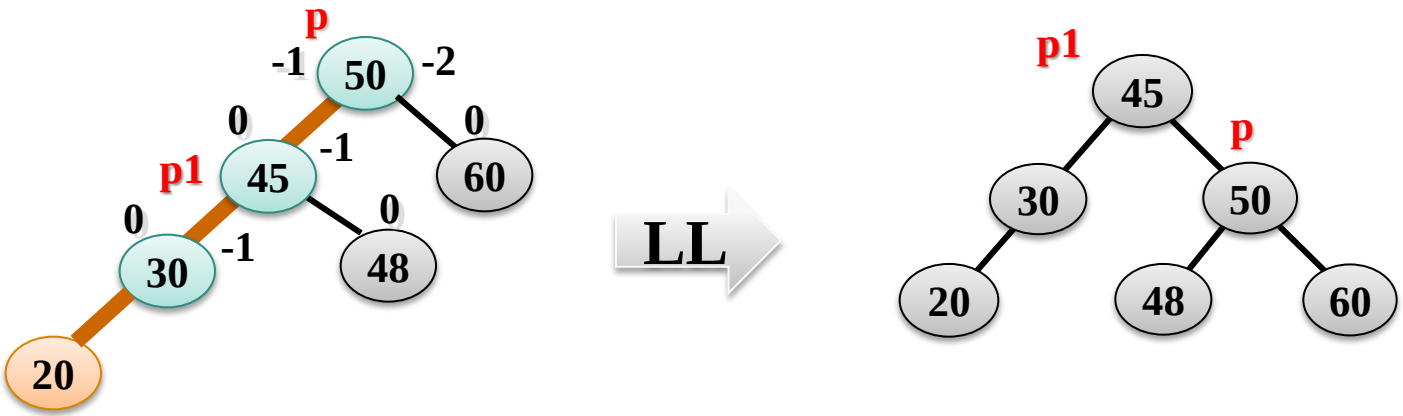
平衡树的插入----LL旋转，失衡点p的左儿子作根

示例： 插入20

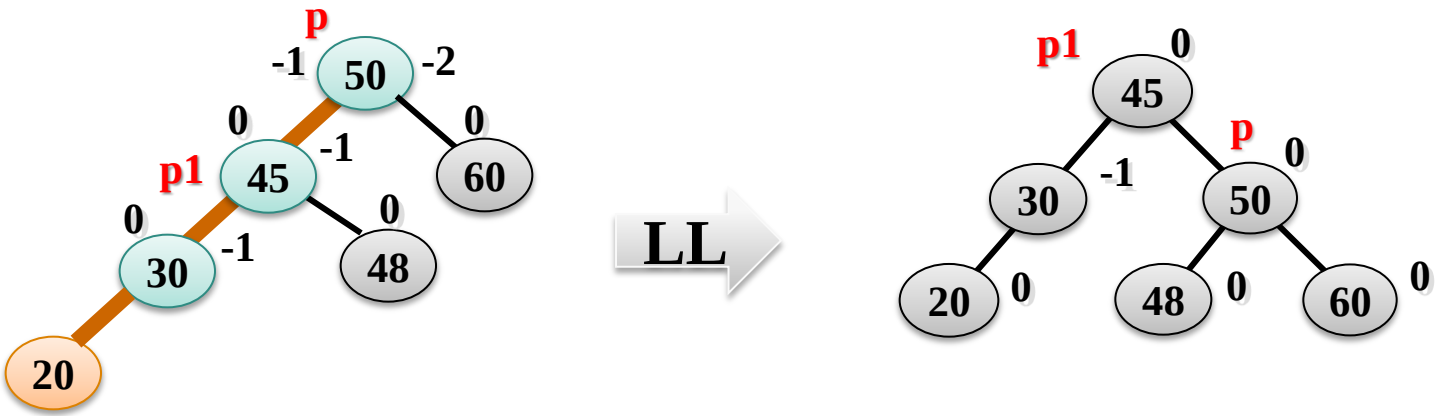


平衡树的插入----LL旋转，失衡点p的左儿子作根

示例： 插入20



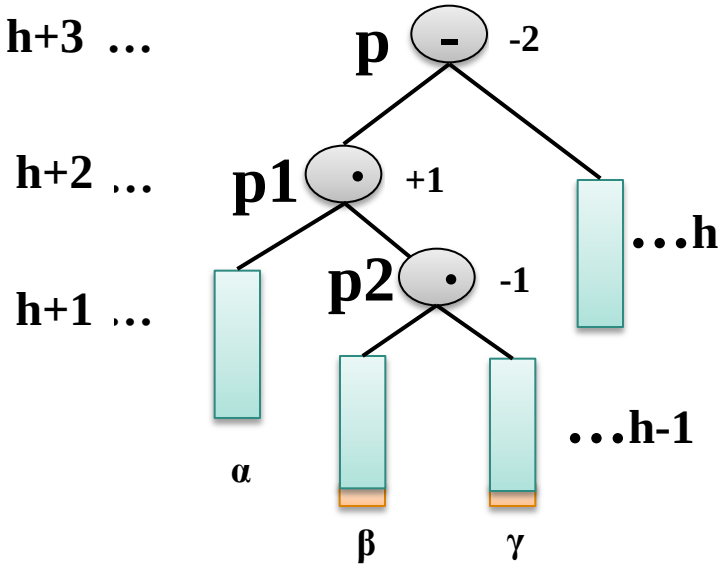
示例： 插入20



子树高度没有增加，停止回溯

平衡树的插入--LR旋转，失衡点p的左儿子的右儿子作根

插入前，p->bal=-1 且 p1->bal=p2->bal=0



旋转方式：

（1）使p2作为子树之根；

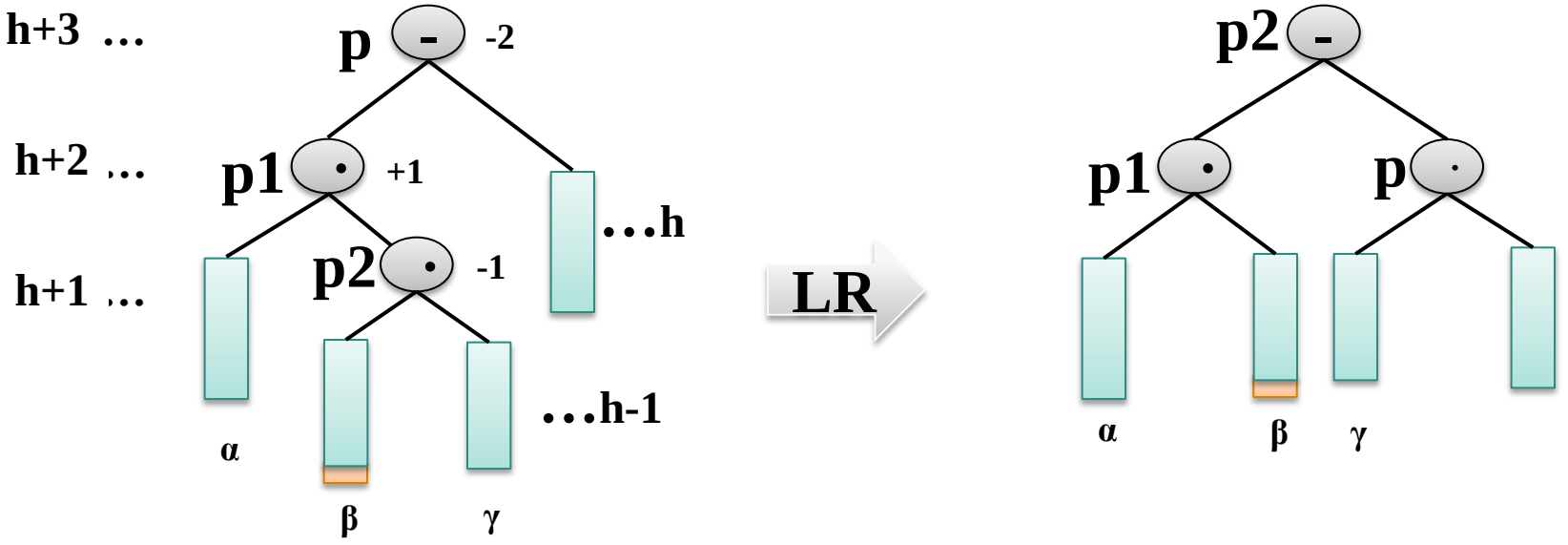
（2）p作p2的右儿子；

（3） p1作p2的左儿子；

（4）p2原来的左右儿子分别作p1的右儿子，p的左儿子；

插入点在左儿子的右子树上

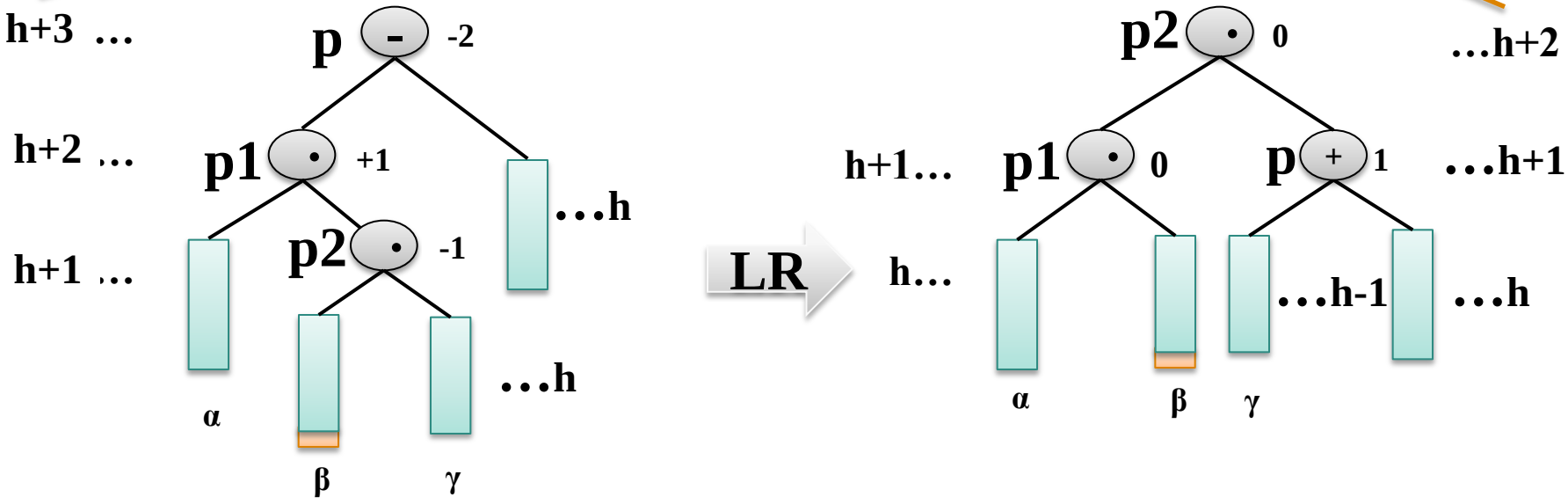
平衡树的插入--LR旋转，失衡点p的左儿子的右儿子作根



插入点在左儿子的右子树上

平衡树的插入--LR旋转，失衡点p的左儿子的右儿子作根

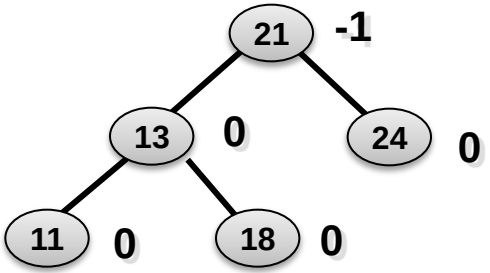
高相等，无需回溯，旋转结束。



插入点在左儿子的右子树上

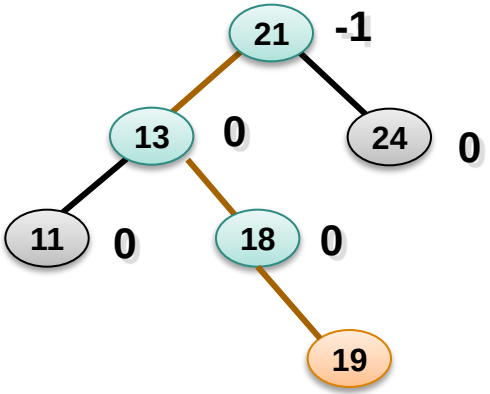
平衡树的插入--LR旋转，失衡点p的左儿子的右儿子作根

示例：　　插入19



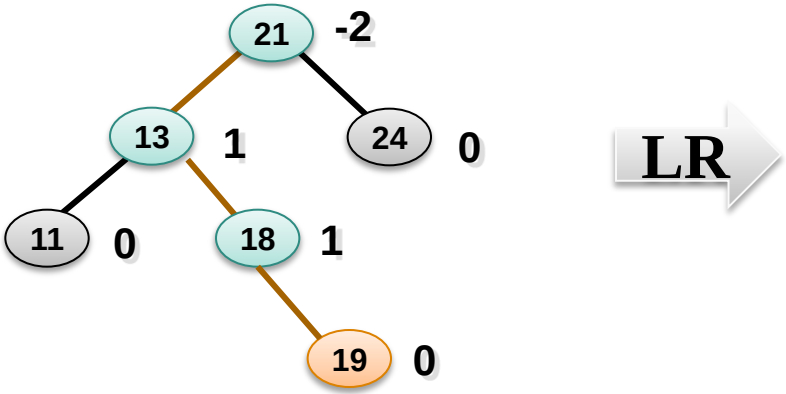
平衡树的插入--LR旋转，失衡点p的左儿子的右儿子作根

示例：　　插入19



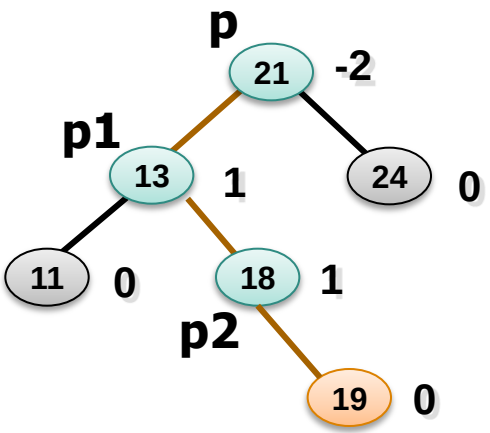
平衡树的插入--LR旋转，失衡点p的左儿子的右儿子作根

示例：　　插入19



平衡树的插入--LR旋转，失衡点p的左儿子的右儿子作根

示例：　　插入19



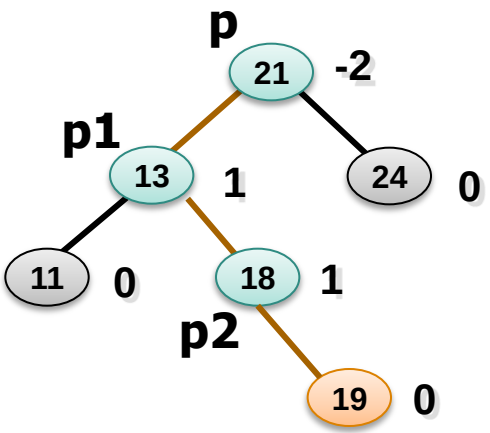
LR

p的孙子p2作根

平衡树的插入--LR旋转，失衡点p的左儿子的右儿子作根

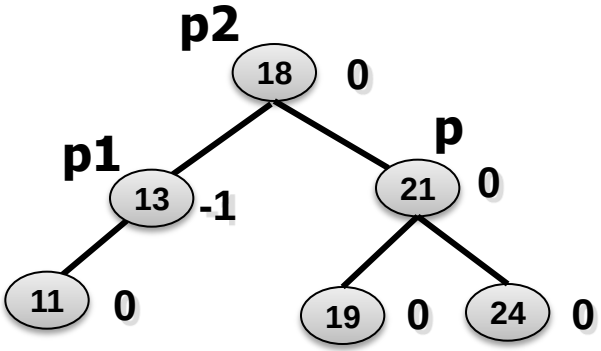
示例：　插入19

子树高度没有增加，停止回溯。



LR

p的孙子p2作根



平衡树的插入----LL旋转

总结

旋转前：

- x插在p的左儿子p1的左子树上，使p1的左子树升高，继而引起p1，p点升高;
- 插入x前，p1->bal=0, p->bal=-1；
- 插入x后，p1->bal=-1, p->bal=-2，p失衡 。

LL旋转：

- 使p1作为子树之根，p作p1的右儿子；
- p1的右儿子作p的左儿子。

旋转后：新的以p1为根的子树高度没有增加，停止回溯。

平衡树的插入----LR旋转

总结

旋转前：

- x插在p的左儿子p1的右子树上，设p1的右儿子为p2，使p2升高，继而引起p1，p点升高;
- 插入x前，p2->bal=p1->bal=0, p->bal=-1；
- 插入x后，p1->bal=-1, p->bal=-2，p失衡 。

LR旋转：

- 使p2作为子树之根，p作p2的右儿子，p1作p2的左儿子；
- p2的左、右儿子分别作p1的右儿子和p的左儿子。

旋转后：新的以p2为根的子树高度没有增加，停止回溯。

平衡树的插入----RR旋转、RL旋转

在结点p的右子树上插入一个结点，从右路返回到p的处理与在其左子树上插入结点完全**对称**：

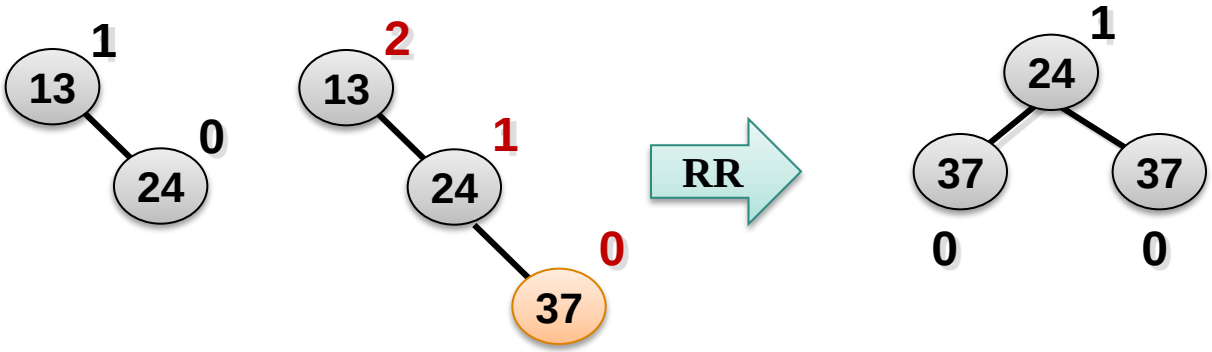
（1）旋转：RR单旋、RL双旋；

（2）“左”都换成“右”，“右”都换成“左”；

（3）正号都换成负号，负号都换成正号。

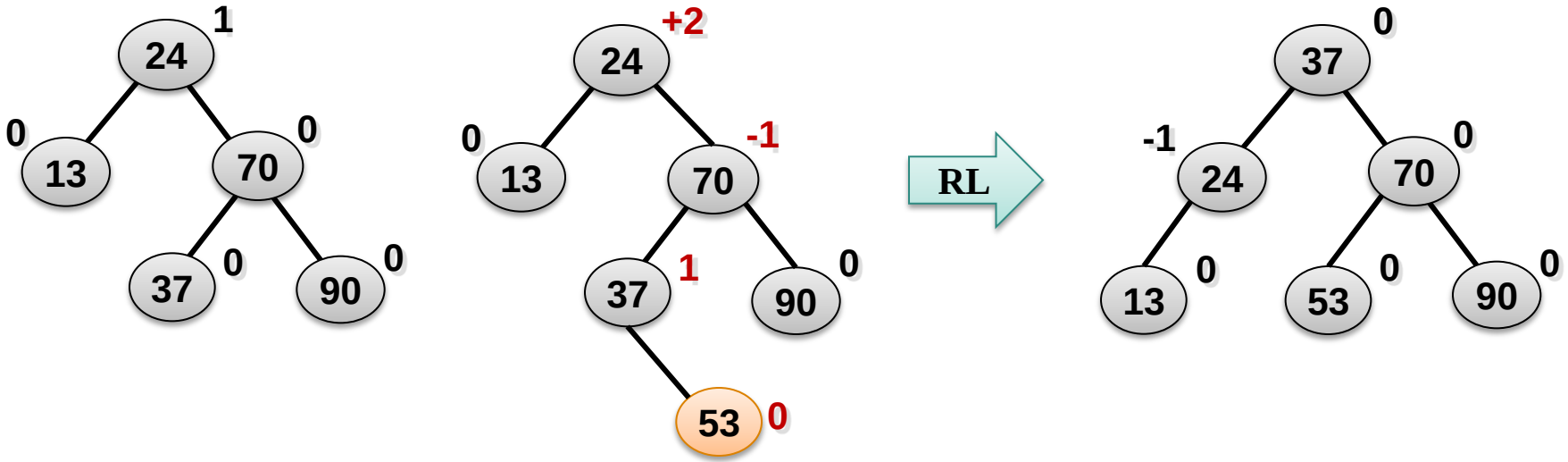
平衡树的插入----RR旋转

示例：插入37



平衡树的插入----RL旋转

示例：插入53

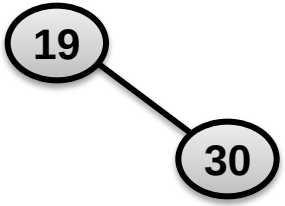


平衡树的构造

输入序列：19，30，43，16，13，18，37

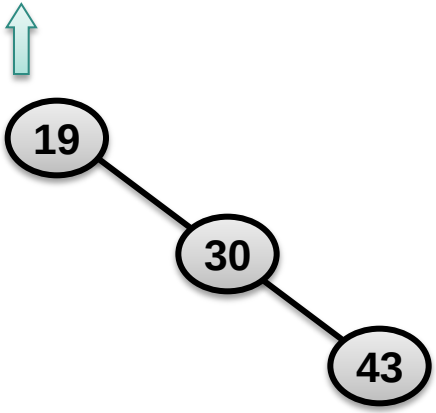
平衡树的构造

输入序列：19，30，43，16，13，18，37



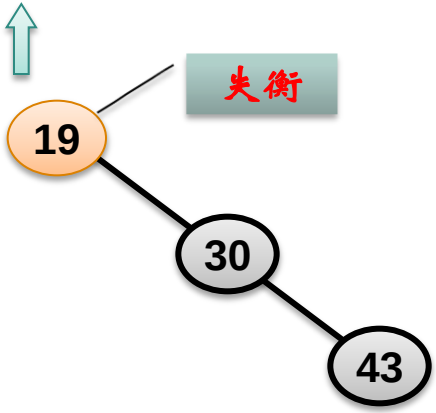
平衡树的构造

输入序列：19，30，43，16，13，18，37



平衡树的构造

输入序列：19，30，43，16，13，18，37

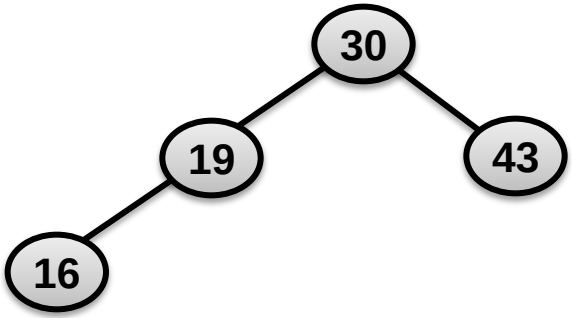


平衡树的构造

输入序列：19，30，43，16，13，18，37

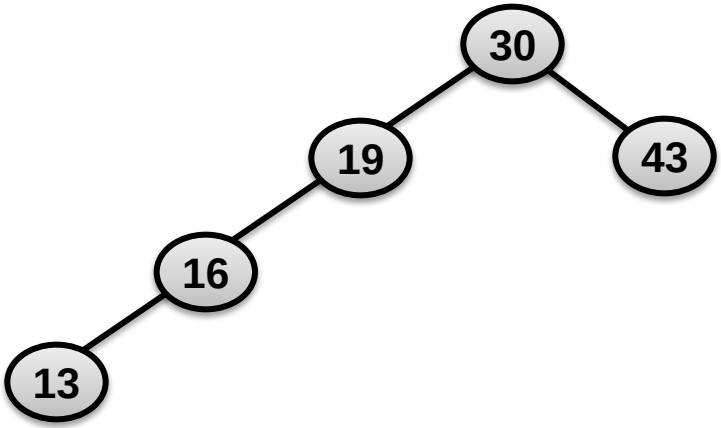
平衡树的构造

输入序列：19，30，43，16，13，18，37



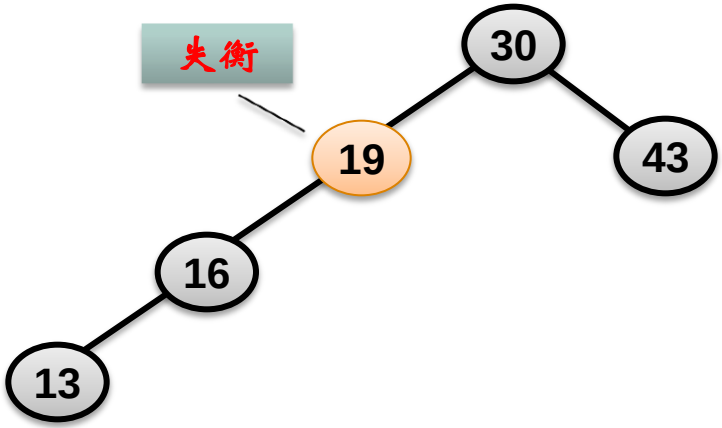
平衡树的构造

输入序列：19，30，43，16，13，18，37



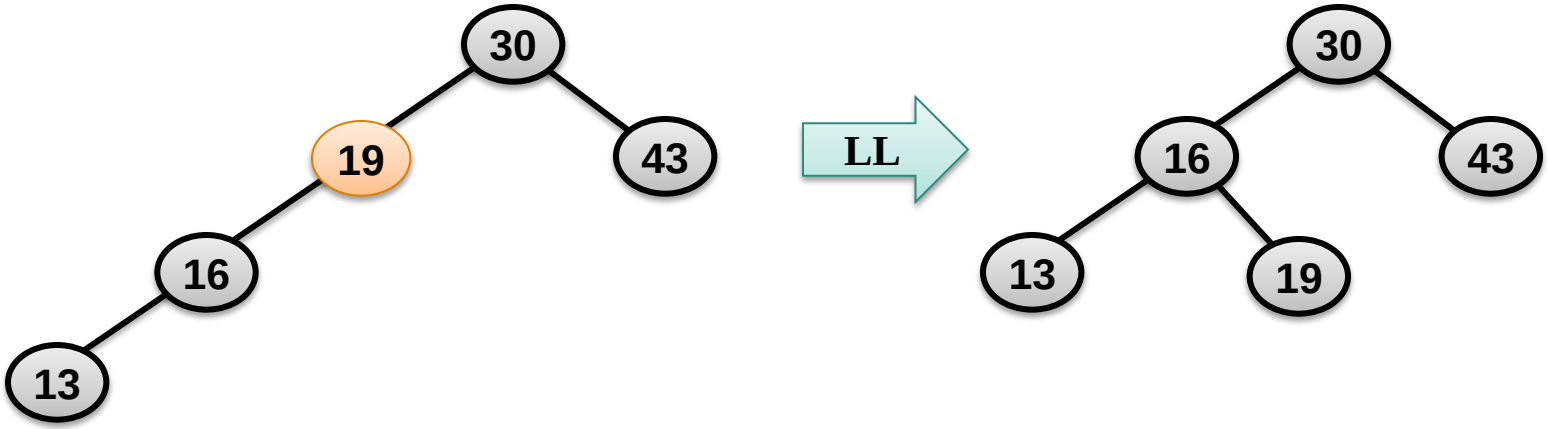
平衡树的构造

输入序列：19，30，43，16，13，18，37



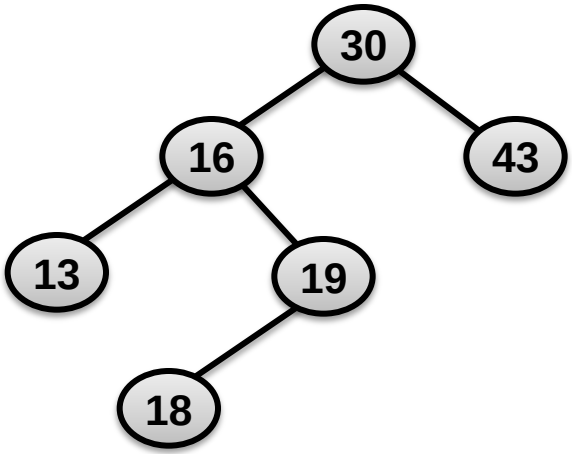
平衡树的构造

输入序列：19，30，43，16，13，18，37



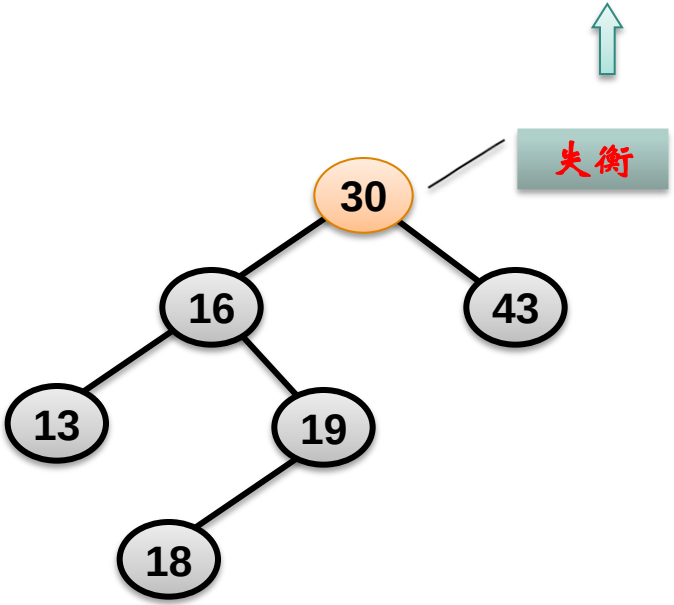
平衡树的构造

输入序列：19，30，43，16，13，18，37



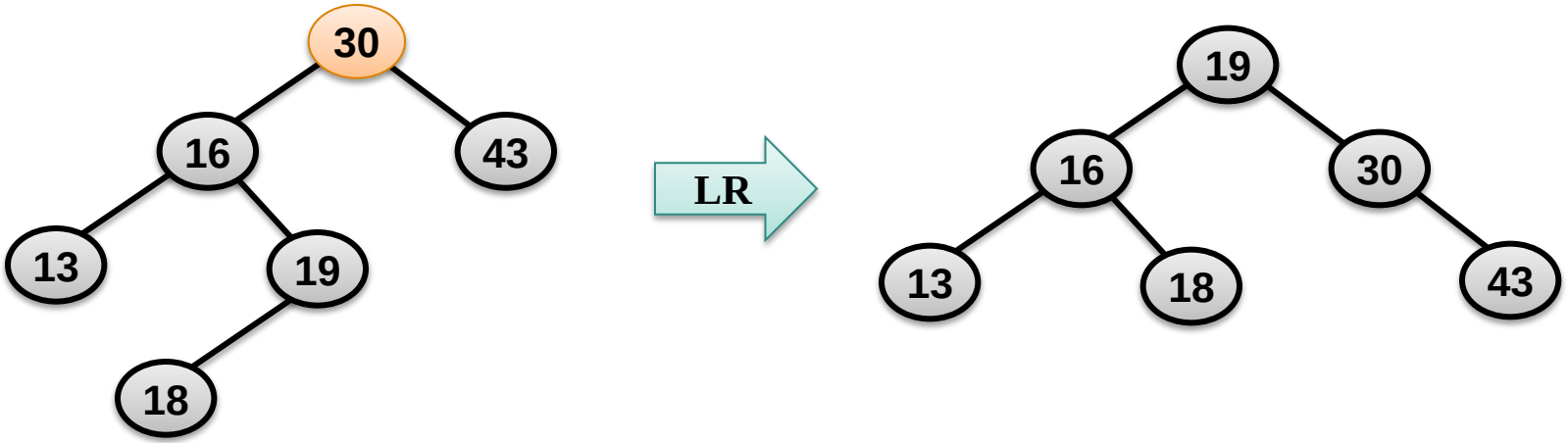
平衡树的构造

输入序列：19，30，43，16，13，18，37



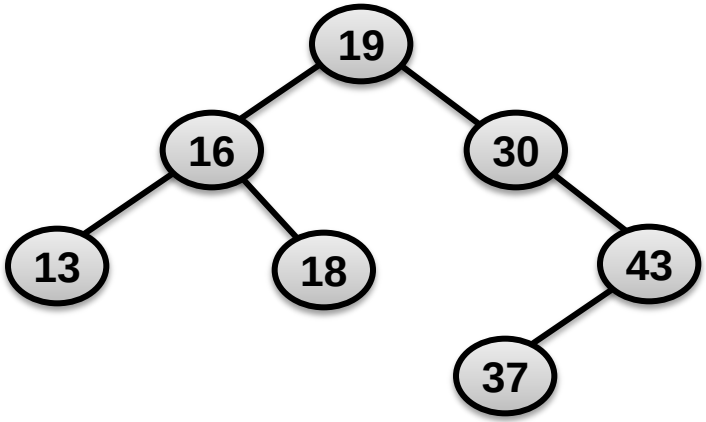
平衡树的构造

输入序列：19，30，43，16，13，18，37



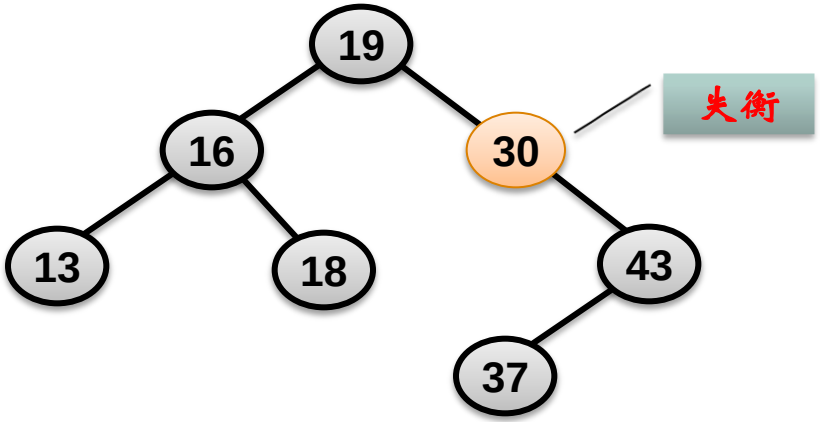
平衡树的构造

输入序列：19，30，43，16，13，18，37



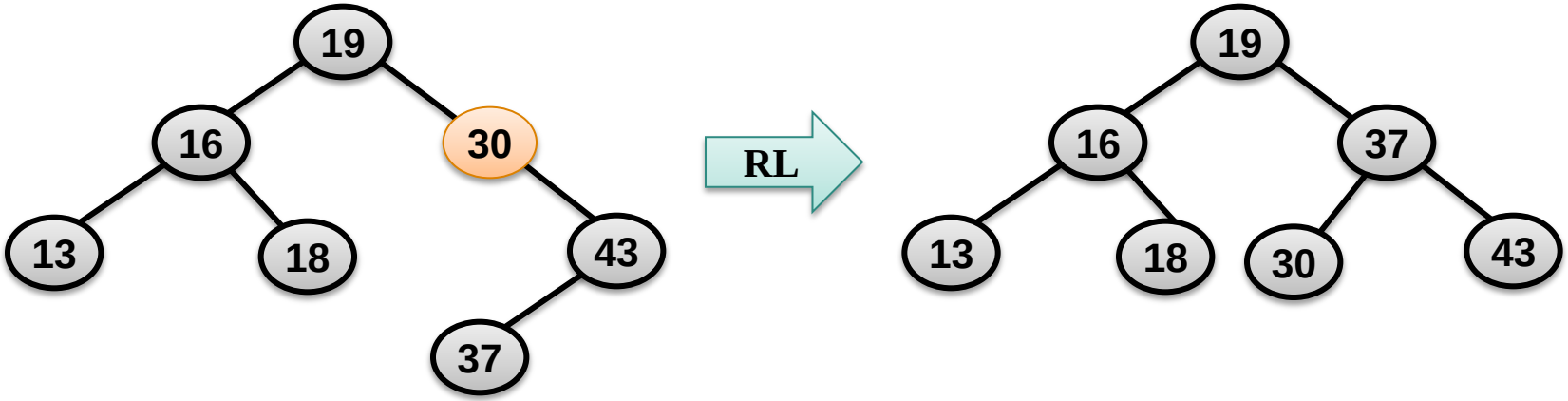
平衡树的构造

输入序列：19，30，43，16，13，18，37



平衡树的构造

输入序列：19，30，43，16，13，18，37



平衡树的插入----算法实现（递归）

在一般检索树的插入insert()中，添加平衡处理步骤（结点加平衡因子域bal）

平衡树的插入算法

主调语句：

```
b=0;
insert_B(x,root, b );//b=0表示平衡树的高不变，=1表示高增高了
```

平衡树的插入----算法实现（递归）

```
void insert_B(element_type x,Bptr &p,int &b)
 {              //平衡树插入函数
1.  if(!p)
2.   { p=newBnode;
3.     p->data=x;
4.     p->Lson=p->Rson=NULL; p->bal=0;
5.     b=1;  //p点升高
6.     return;
     }
7.  if(x<=p->data)
      {
8.     insert_B(x,p->Lson,b);
9.     if(b) Ibalance_L( p,b); //“左”平衡处理
      }
    else
      {
10.   insert_B(x,p->Rson,b)
11.   if(b) Ibalance_R( p,b); //“右”平衡处理
      }
 }
```

平衡树的插入----算法实现（递归）

```
void Ibalance_L(Bptr &p,int &b)
      //左平衡树处理函数
 { Bptr p1,p2;
12. if(p->bal==1) //不再向上层回溯
    { p->bal=0;    b=0;       return;    }
   else
13.  if(p->bal==0) //向上层回溯
     { p->bal=-1;       return;      }
     else    //旋转以p为根的子树
     {
```

```
14.     p1=p->Lson;  //取p的左儿子p1
15.     if(p1->bal= = -1)  //LL 旋转
        {
16.       p->Lson=p1->Rson;  p->bal=0;
17.       p1->Rson=p;
18.       p1->bal=0;
19.       p=p1;  //将p的父亲的链域改为p1
          b=0;   //置停止回溯标记
          return;
        }
        else  // LR双旋
```

平衡树的插入----算法实现（递归）

```
        // LR旋转
        {
20.       p2=p1->Rson;
21.       p1->Rson=p2->Lson;
22.       p2->Lson=p1;
23.       p->Lson=p2->Rson;
24.       p2->Rson=p;
```

```
25.       if(p2->bal==0)
26.         p->bal=p1->bal= 0;  //p2是新叶
          else
27.         if(p2->bal= = - 1)  //x插在p2的左子树上
28.           { p1->bal= 0;   p->bal=1; }
           else   //x插在p2的右子树上
29.           { p1->bal= -1;  p->bal= 0;  }
30.       p=p2;  //改将p之父的链域改为p2
31.       p->bal=0;
32.       b=0; //置停止回溯标记
33.       return;
        }
      }
    }
```