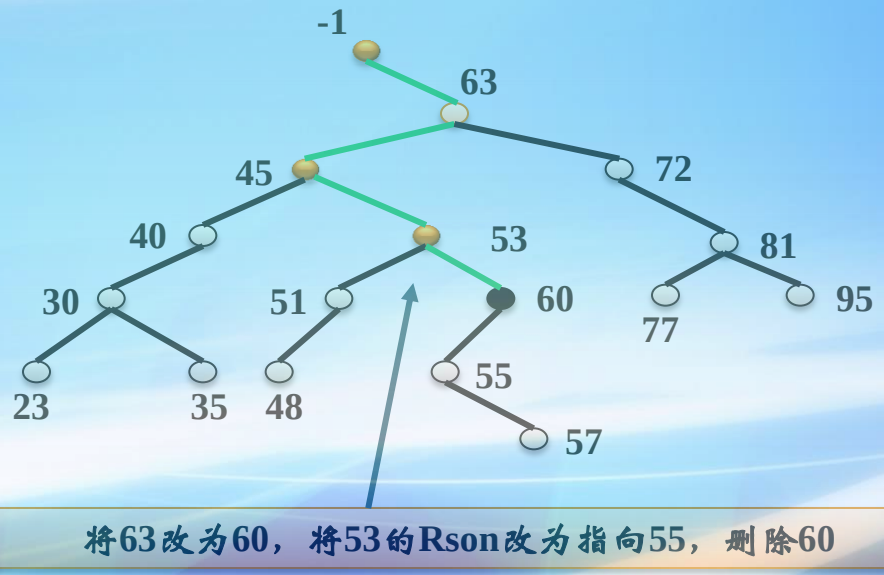
检索树（下）

《数据结构》

主讲人：李清

删除63

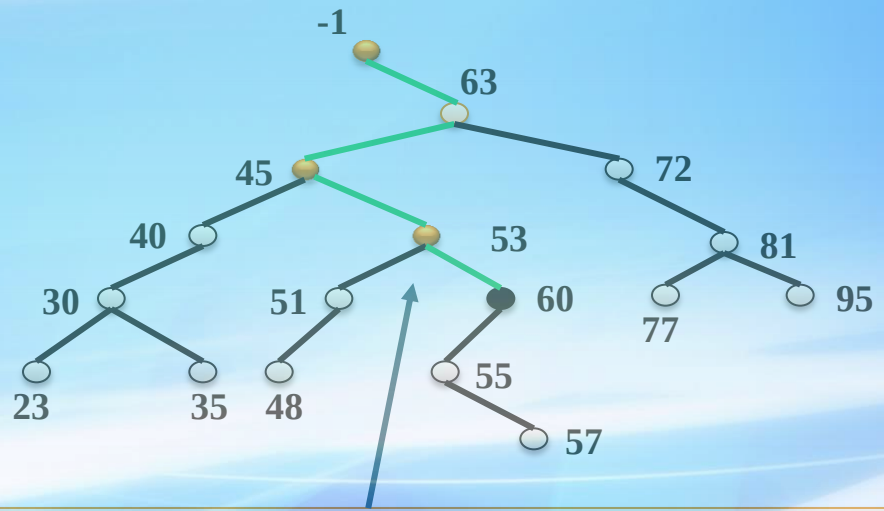


检索树的删除

《数据结构》

主讲人：李清

删除63



问题

通常删除以此元素为根的子树

如何删除二叉树的元素？

如何删除检索树中的元素？

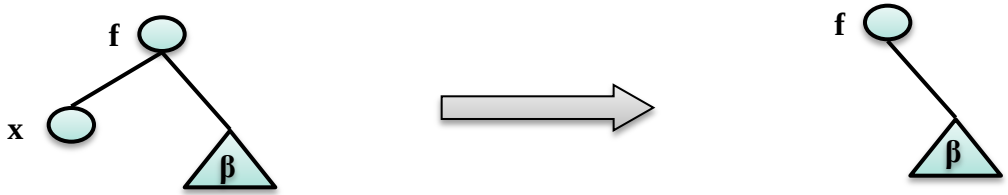
通常仅删除此元素且保中序有序

检索树的删除（1）

先找到要删除的结点x，并记下结点x的父亲f；

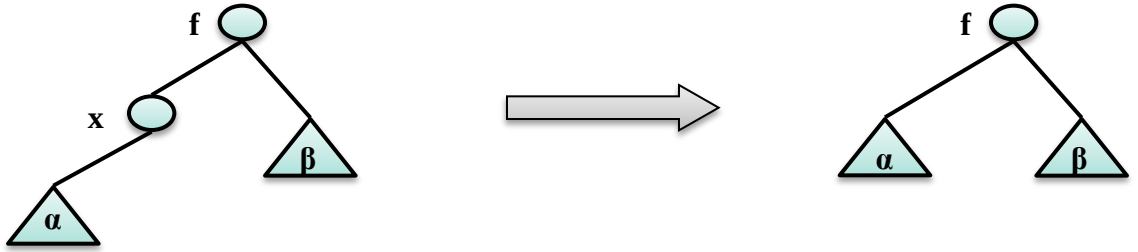
（1）若x是叶，把f指向x的链域（Lson或Rson）置空，

就删除了x；



检索树的删除（2）

（2）若x只有一个儿子，将f指向x的链域（Lson或Rson）改为指向x的儿子；



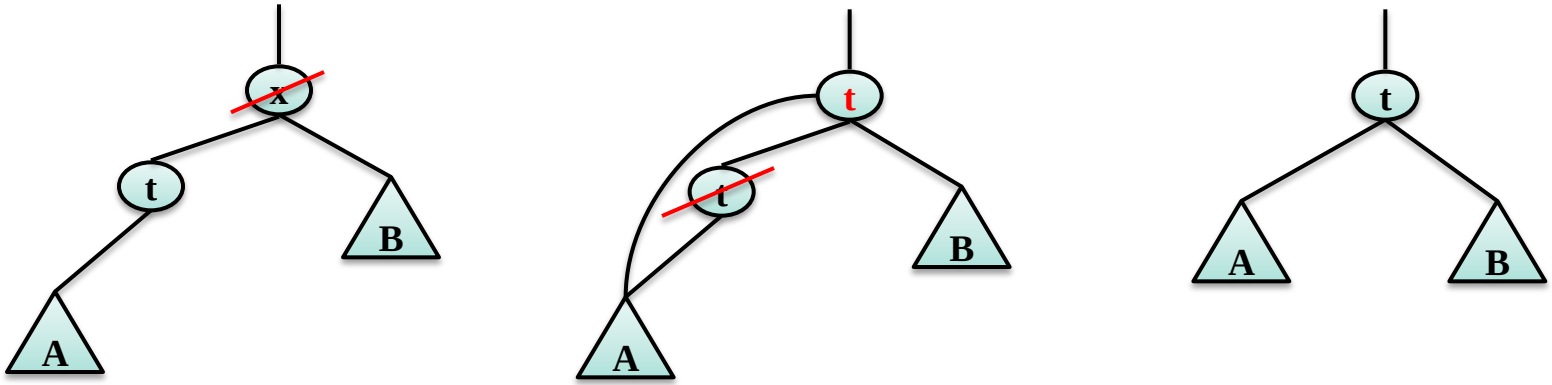
检索树的删除（3）

（3）若x有两个儿子，先找到x的中序前趋y，用结点y代替结点x，并删除结点y；

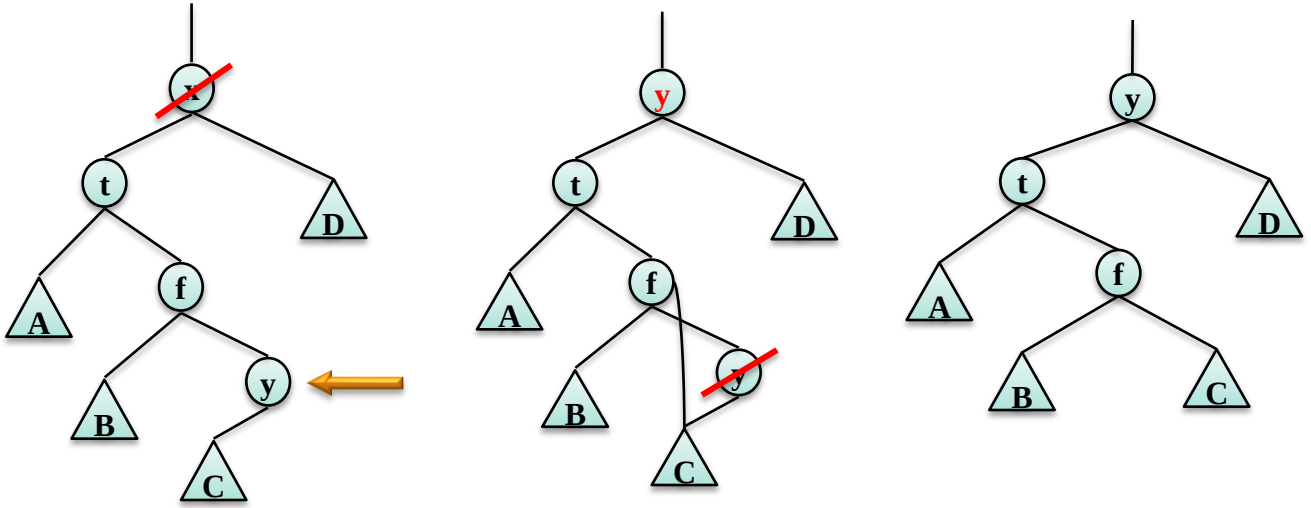
真正删除的是其中序前驱结点

检索树的删除（3）

检索树的删除----用中序前驱替换

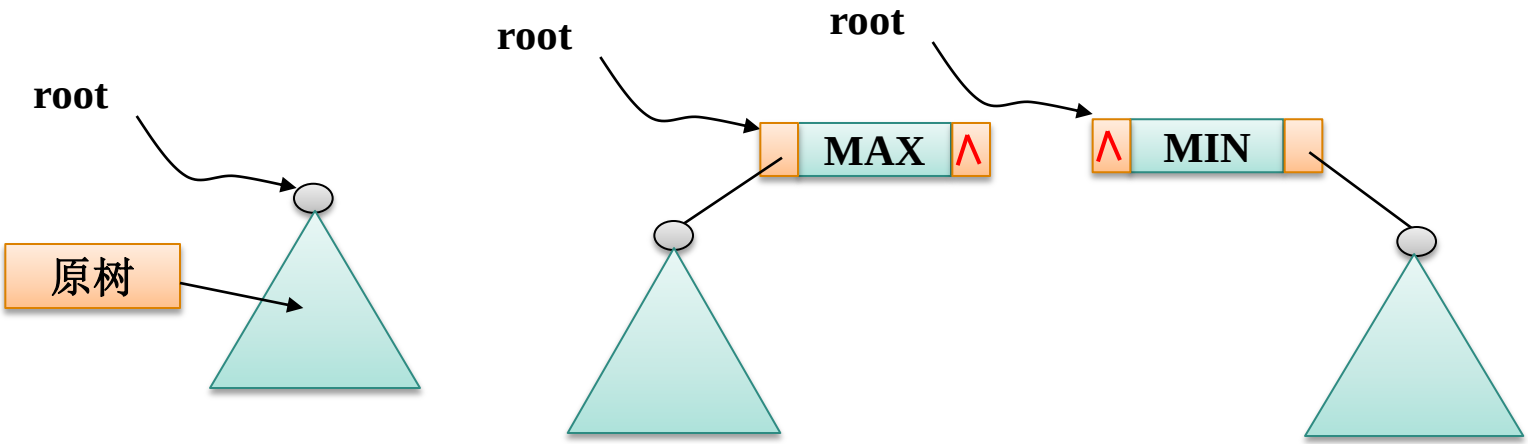


检索树的删除----用中序前驱替换



检索树的删除

避免删除根结点，增加一个监督元，值为“无穷大”或“无穷小”。



MIN、MAX的值视具体情况而定

算法：带监督元结点检索树的删除（非递归）

```
int deleteT(element_type x, Bptr root)
  {  Bptr f,p,q,s,r; //f：要删除结点的父结点
1.   p=NULL;  //p将指向要删除结点
2.   f=root;   //f的初值指向虚根（无穷小）
3.   q=root->Rson;  //q搜索指针
4.   while(q)  //循环查找x
5.     if(x==q->data) { p=q;  q=NULL;} //找到x
6.    else if(x<q->data) { f=q; q=q->Lson;} //向左搜索
7.          else {f=q; q=q->Rson;}   //向右搜索
8.   if(!p) return  0; //没找到x,结束
9.   if (!p->Rson) // p无右儿子，用左儿子代替p
10.    if(p==f->Lson){ f->Lson=p->Lson; delete p; }
11.    else { f->Rson=p->Lson; delete p; }
     else
12.    if (!p->Lson) //p无左儿子，用右儿子代替p
13.      if (p==f->Lson){ f->Lson=p->Rson; delete p; }
14.      else  { f->Rson=p->Rson; delete p; }
     else //p有两个儿子，用中序前驱代替p
15.  { s=p->Lson;  //s是p的左儿子
16.    if (s->Rson==NULL)  //s没有右儿子，用s代替p
17.     { p->data=s->data;  //用s的值域代换p的值域
18.       p->Lson=s->Lson;  //删去s
19.       delete  s;
        }
       else //s有右儿子，查找p的左儿子的最右子孙r
20.    {  r=s->Rson;
21.       while(r->Rson) { s=r; r=r->Rson; }
22.       p->data=r->data;  //用r的值域代换p的值域
23.       s->Rson=r->Lson; //删去r
24.       delete r;
        }
25.  }
26.  return  1; //返回删除成功信息
  } //函数结束
```

检索树删除算法

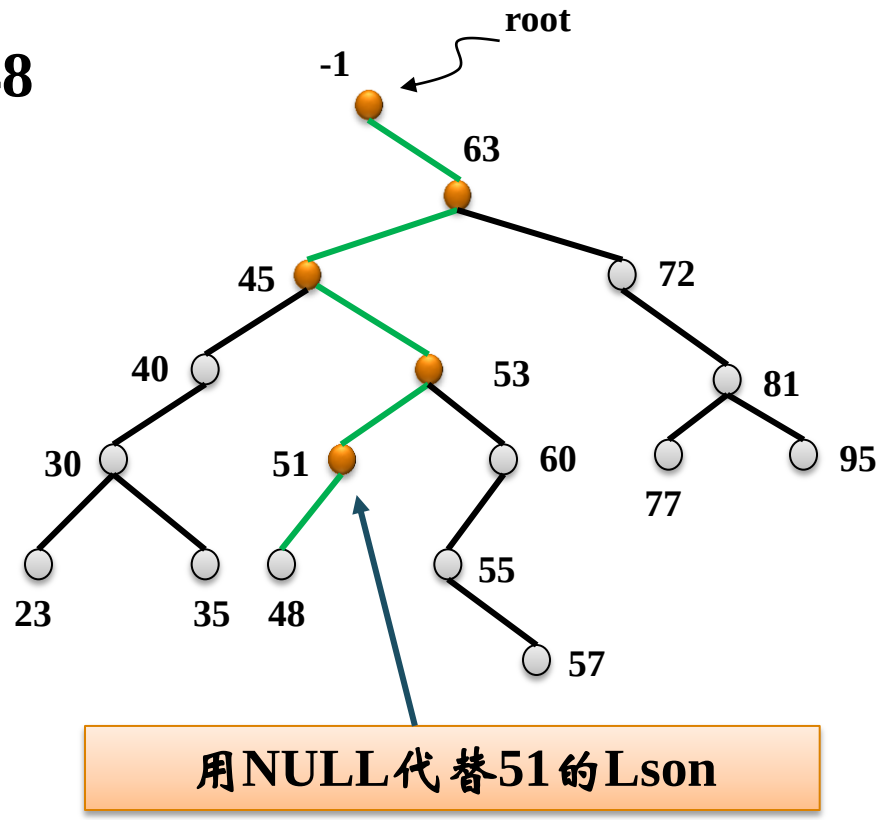
时间复杂度：

- 查找要删除的结点；
- 查找要删除结点的中序前驱结点

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

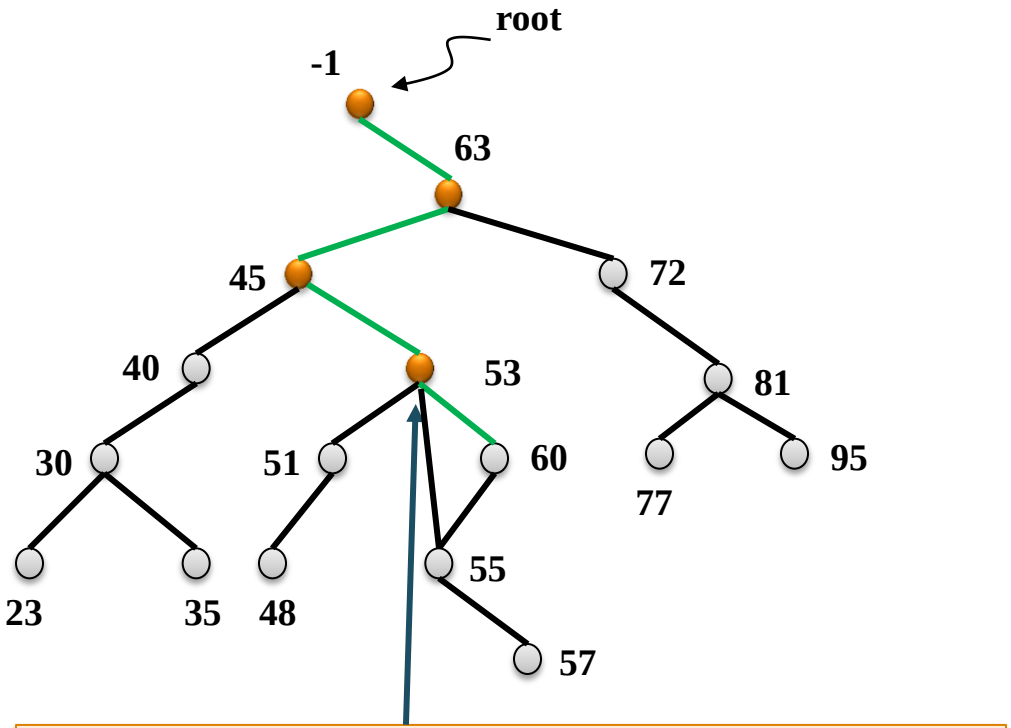
实例：检索树的删除

删除叶子48



实例：检索树的删除

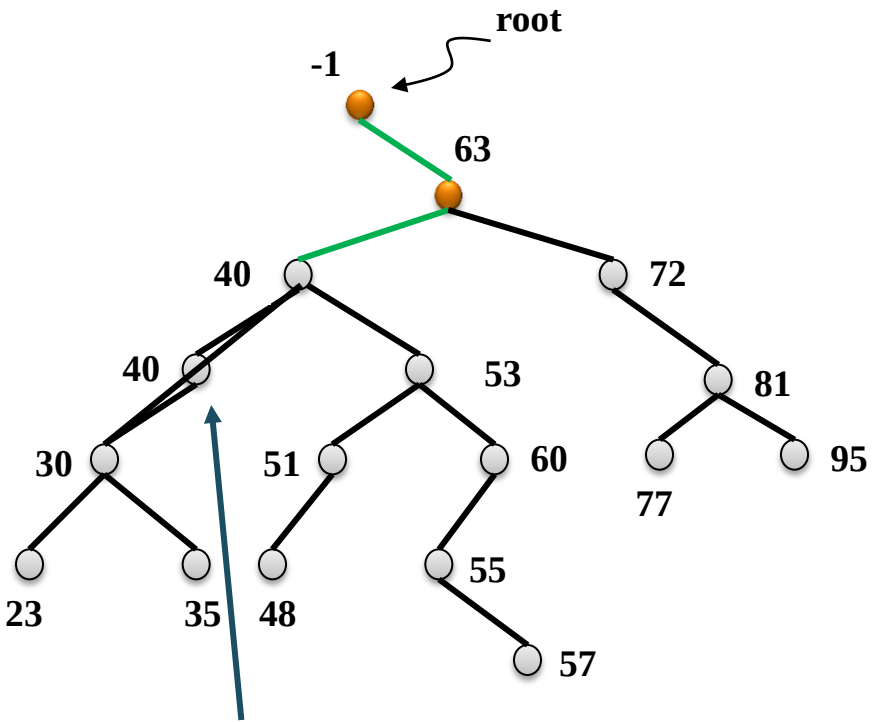
删除60



将结点53的Rson置为60的Lson（55）

实例：检索树的删除

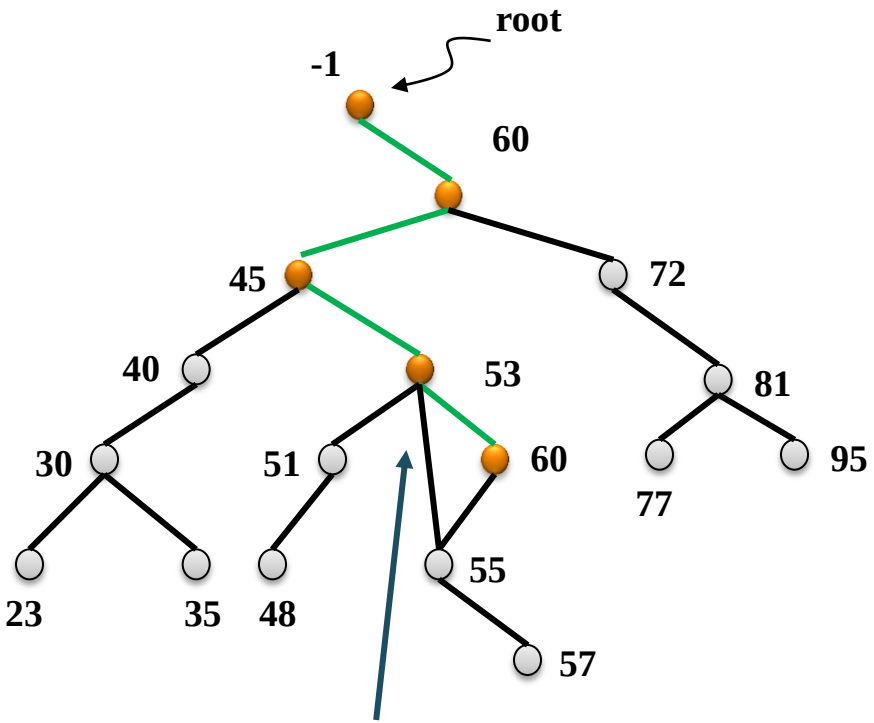
删除45



将45改为40，将新40的Lson改为指向30，删除原40

实例：检索树的删除

删除63



将63改为60，将53的Rson改为指向55，删除60

小结

- 左小右大，中序有序。
- 检索树的查找可采用递归和非递归算法实现。
- 检索树插入的新结点都是作为新的叶子。
- 检索树删除结点分叶结点、单枝结点和双枝结点分别处理。
- 检索树查找、插入、删除算法的时间复杂度取决于树高，平均性能是O(logn)，最坏情况O(n)。

检索树具有“动态信息结构”的特点，查找优于一般的二叉树。