数据结构
Data Structure

Xia Tian
Email: xiat(at)ruc.edu.cn

Renmin University of China

查找表

查找是许多应用系统中最消耗时间的一部分，一个好的查找算法会大大提高运行速度。计算机需要存储包含该特定信息的表，才可以高效查找。

查找表的分类

- 静态查找表
  - 仅作查询和检索操作的查找表。
- 动态查找表
  - 有时在查询之后，还需要将“查询”结果为“不在查找表中”的数据元素插入到查找表中；或者，从查找表中删除其“查询”结果为“在查找表中”的数据元素。

关键字

- 是数据元素（或记录）中某个数据项的值，用以标识（识别）一个数据元素（或记录）。
- 若此关键字可以识别唯一的一个记录，则称之谓“主关键字”。
- 若此关键字能识别若干记录，则称之谓“次关键字”。
- 若数据元素只有一个数据项时，其关键字就是数据元素的值。

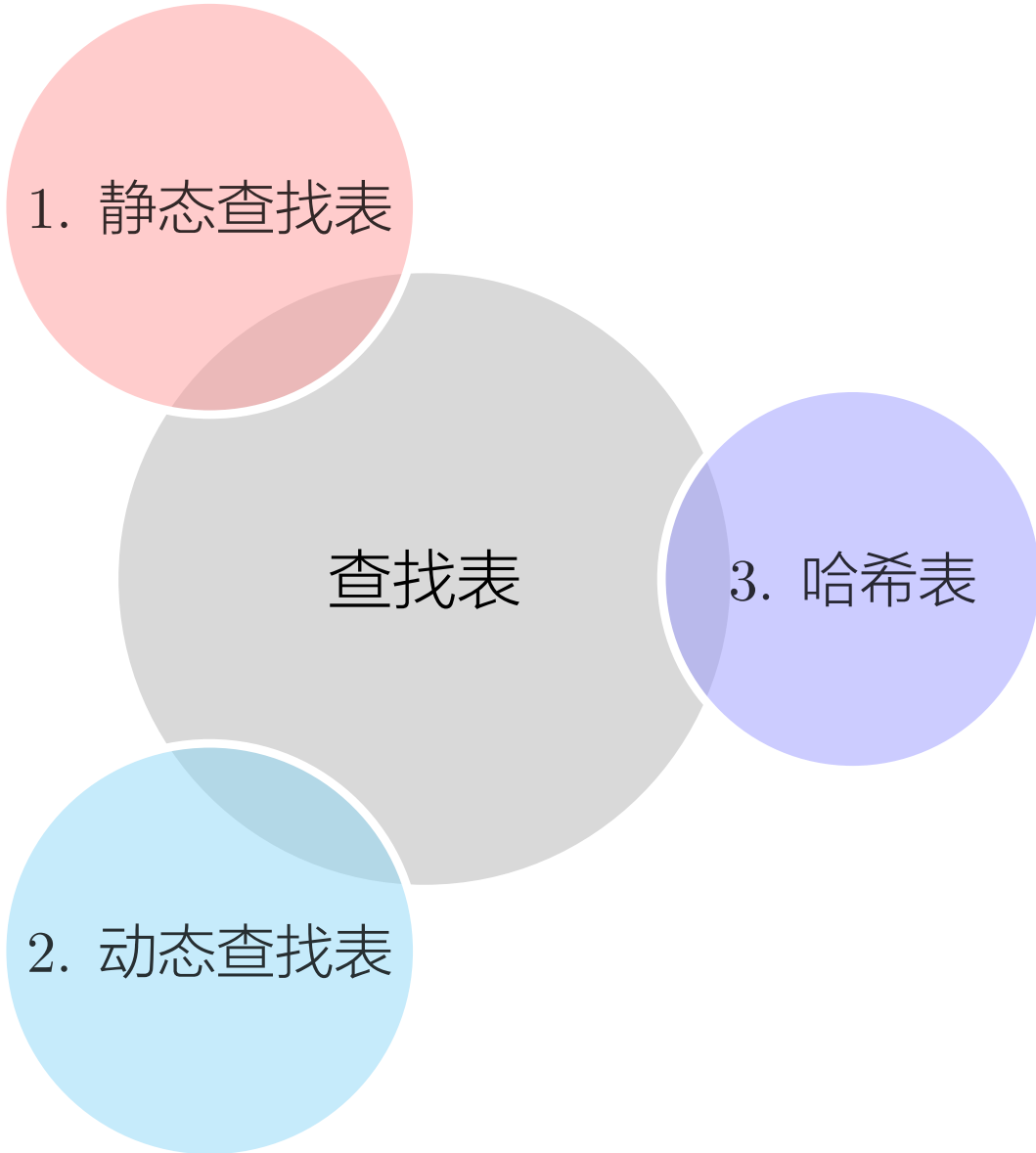
查找

- 根据给定的某个值，在查找表中确定一个其关键字等于给定值的数据元素或（记录）
- 若查找表中存在这样一个记录，则称“查找成功”：
  - 查找结果：给出整个记录的信息，或指示该记录在查找表中的位置；
- 否则称“查找不成功”，查找结果：
  - 给出“空记录”或“空指针”。

如何进行查找?

- 查找的方法取决于查找表的结构。
- 如果查找表中的数据元素之间不存在明显的组织规律，就不利于快速查找

本章大纲



1. 静态查找表

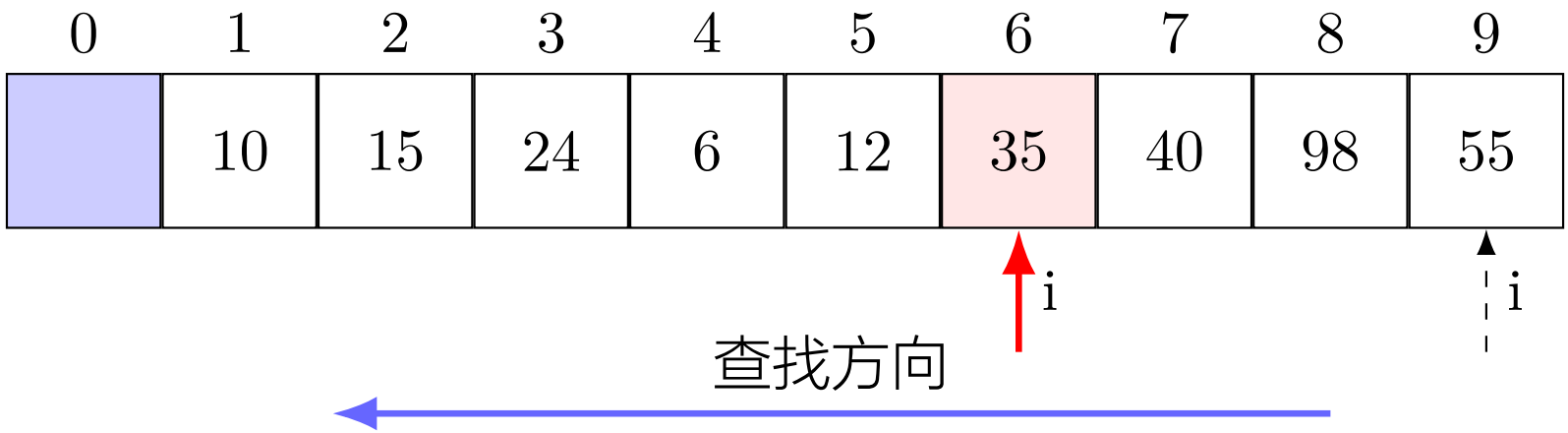
静态查找：对查找集合只进行查找，不涉及插入和删除操作。或者经过一段时间的查找之后，集中地进行插入和删除等修改操作。

包括：

- 顺序查找
- 折半查找
- 分块查找

顺序查找

- 又称线性查找, 是最基本的查找方法之一
- 从表的一端向另一端逐个按给定值与关键码进行比较, 若找到, 查找成功, 返回数据元素在表中的位置; 若未找到与 k 相同的关键码, 则返回失败信息。
- 例: 查找 $\mathrm{k} = 35$



注意：下标为 0 的位置，其哨兵用途。

顺序查找的性能分析

- 分析查找算法的效率, 通常用平均查找长度 ASL (Average Search Length) 来衡量, 即在查找成功时所进行的关键码比较次数的期望值。
  顺序查找 (等概率情况下):

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

  实际上, 数据的查找概率存在相当大的差别!
- 在查找概率不同的情况下, 应遵循查找表需依据查找概率越高, 比较次数越少; 查找概率越低, 比较次数就较多的原则来存储数据元素。

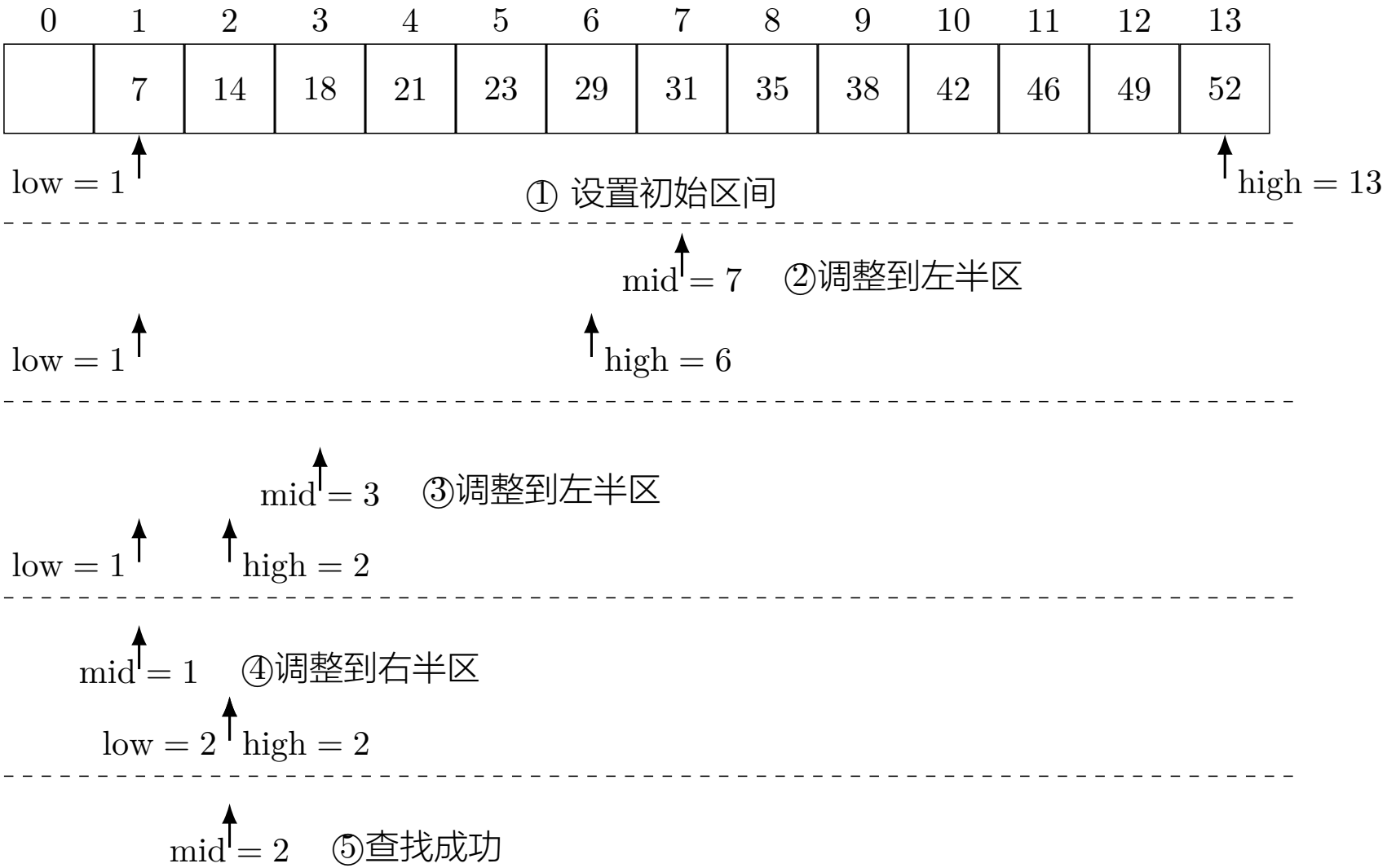
顺序查找总结

- 优点: 算法简单而且使用面广。
  - 对表中记录的存储没有任何要求, 顺序存储和链接存储均可 (当然, 链式也只能用顺序查找);
  - 对表中记录的有序性也没有要求, 无论记录是否按关键码有序均可。
- 缺点: 平均查找长度较大, 特别是当待查找集合中元素较多时, 查找效率较低。

有序表的折半查找

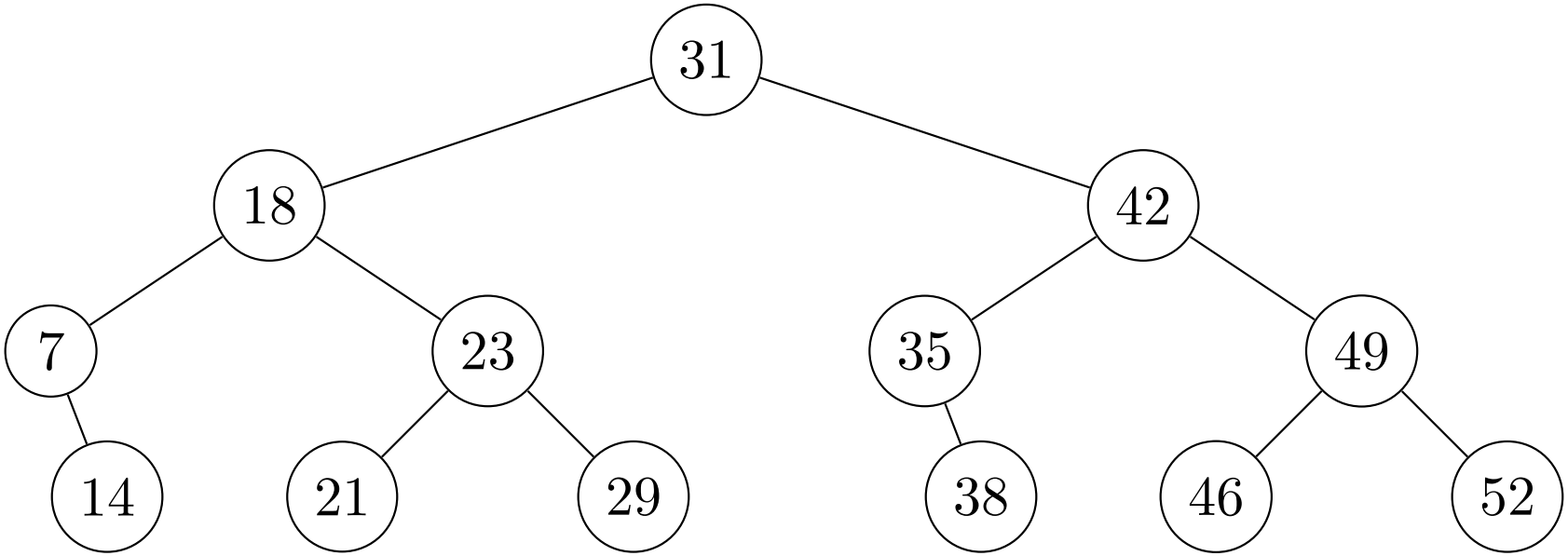
- 有序表是表中数据元素按关键码升序或降序排列。
- 适用于:
  - 线性表中的记录必须按关键码有序;
  - 必须采用顺序存储。

请查找 14


0 1 2 3 4 5 6 7 8 9 10 11 12 13
7 14 18 21 23 29 31 35 38 42 46 49 52
low = 1
high = 13
① 设置初始区间
mid = 7
②调整到左半区
low = 1
high = 6
mid = 3
③调整到左半区
low = 1
high = 2
mid = 1
④调整到右半区
low = 2
high = 2
mid = 2
⑤查找成功

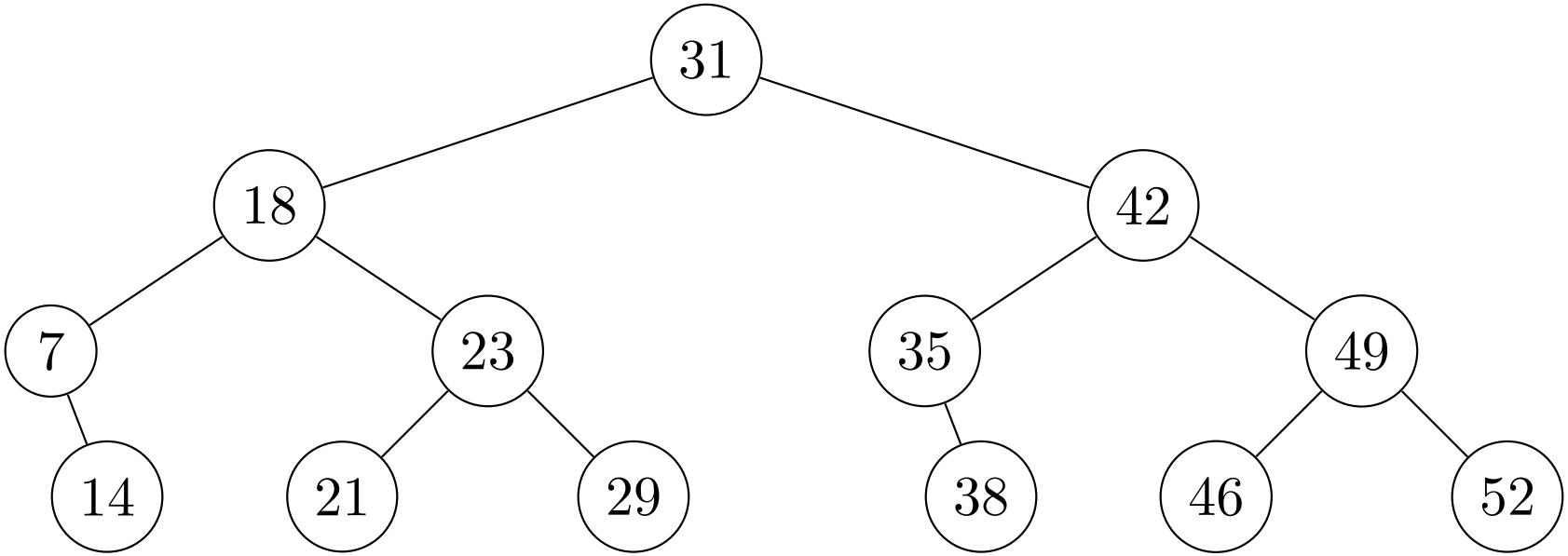
课堂练习：请查找 22

0	1	2	3	4	5	6	7	8	9	10	11	12	13
	7	14	18	21	23	29	31	35	38	42	46	49	52



从折半查找过程看, 以表的中点为比较对象, 并以中点将表分割为两个子表, 对定位到的子表继续这种操作。所以, 对表中每个数据元素的查找过程, 可用二叉树来描述。

- 折半查找在查找成功时, 所进行的关键码比较次数至多为?
- 请问平均查找长度 (ASL) 是多少?



- 折半查找在查找成功时, 所进行的关键码比较次数至多为?

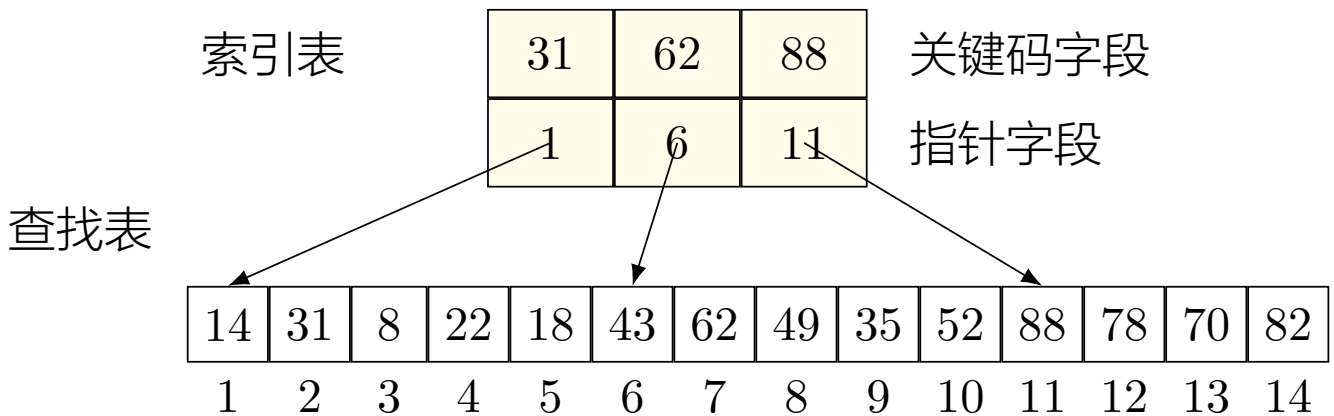
$$\lfloor \log_2 n \rfloor + 1$$

- 请问平均查找长度 (ASL) 是多少?

$$\mathrm{ASL} = \frac{1}{n}[1 \times 2^0 + 2 \times 2^1 + \cdots + k \times 2^{k-1}] \approx \frac{n+1}{n}\log_2(n+1) - 1$$

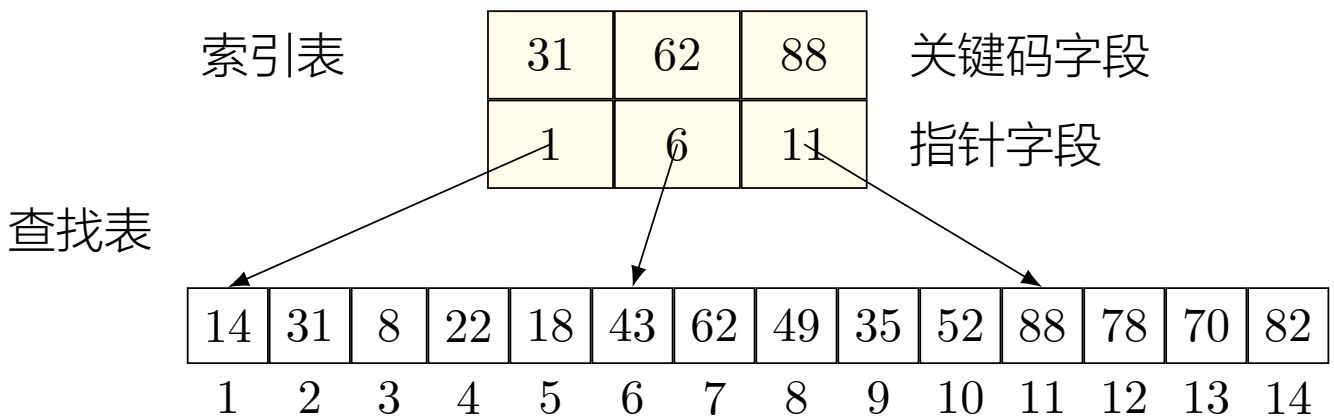
分块查找/索引顺序查找

- 分块查找又称索引顺序查找, 是对顺序查找的一种改进。适用于表有序或者分块有序 (后面的子表中所有记录的关键码均大于前一个子表的最大关键码) 的情形。
- 例: 对某集合按关键码值 31,62,88 分为三块建立的查找表及其索引表如下:



分块查找

- 分块查找要求将查找表分成若干个子表, 并对子表建立索引表, 查找表的每一个子表由索引表中的索引项确定。
- 索引项
  - 关键码字段 (存放对应子表中的最大关键码值) ;
  - 指针字段 (存放指向对应子表的指针) , 并且要求索引项按关键码字段有序。
- 如何根据索引表和查找表进行查找?

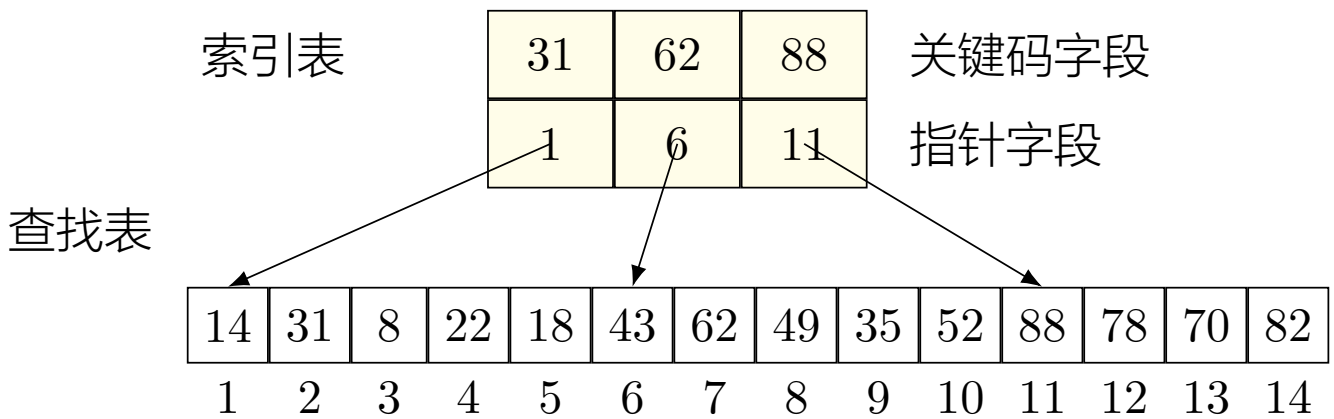


分块查找性能分析

- 分块查找含索引表查找和子表查找。
- 设 n 个数据元素的查找表分为 b 个相同大小的块，每块含有 s 个记录，即：$b = \left\lceil \frac{n}{s} \right\rceil$
- 则分块查找的平均查找长度为:

$$ASL = \frac{b+1}{2} + \frac{s+1}{2} = \frac{1}{2}\left(\frac{n}{s} + s\right) + 1$$

- 可见, 平均查找长度和表的总长度 n、每块的记录个数 s 有关。



2. 动态查找表

动态查找表的特点是, 表结构本身是在查找过程中动态生成的, 即对于给定的 key, 若表中存在其关键字等于 key 的记录, 则查找成功返回, 否则插入关键字等于 key 的记录。

包括:

- 二叉排序树
- 平衡二叉树

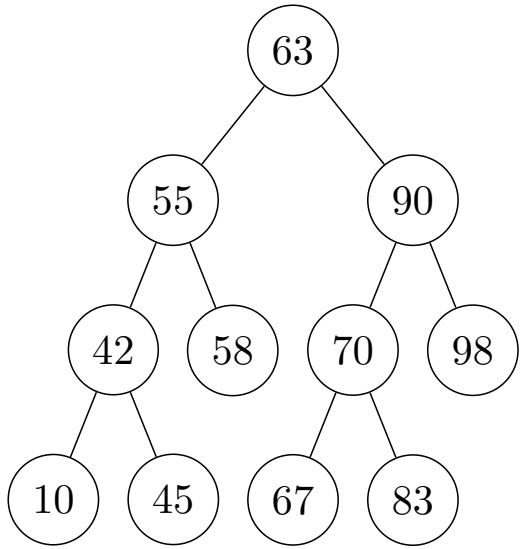
二叉排序树

- 二叉排序树 (Binary Sort Tree) 或者是一棵空树; 或者是具有下列性质的二叉树:

  ① 若左子树不空, 则左子树上所有结点的值均小于根结点的值; 若右子树不空, 则右子树上所有结点的值均大于根结点的值。

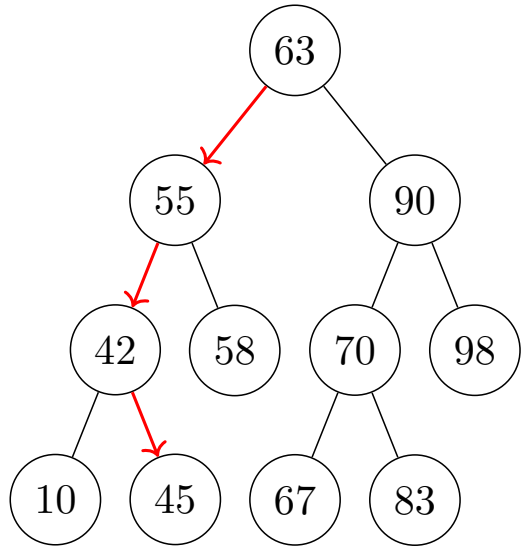
  ② 左右子树也都是二叉排序树。

- 对二叉排序树进行中序遍历, 可以得到一个按关键码有序的序列, 因此, 一个无序序列可通过构造二叉排序树而成为有序序列。



二叉排序树的查找

- 若查找树为空, 查找失败; 否则将 key 与查找树的根结点比较
  ① 若相等, 查找成功, 否则,
  ② 如果 key< 根结点关键码, 继续在以左子树上进行查找
  ③ 如果 key> 根结点关键码, 继续在以右子树上进行查找
- 例如在右图所示的树上查找 45

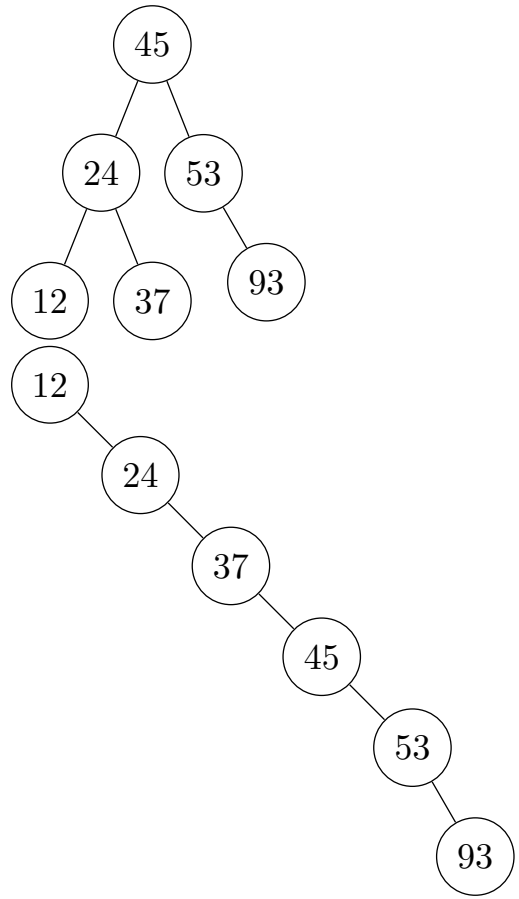


二叉排序树的查找 (cont.)

- 两树的平均查找长度分别为:

$$\mathrm{ASL_a} = \frac{1}{6} \times [1+2+2+3+3+3] = \frac{14}{6}$$

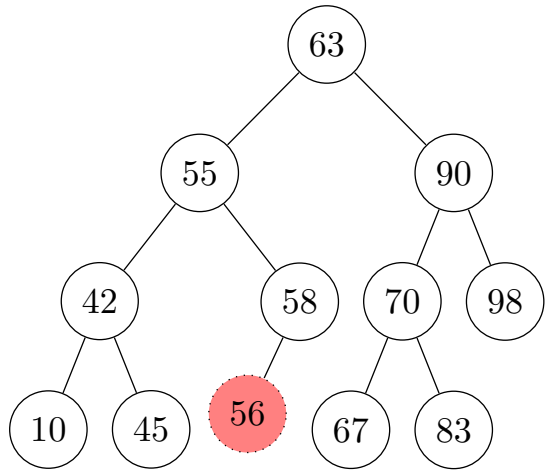
$$\mathrm{ASL_b} = \frac{1}{6} \times [1+2+3+4+5+6] = \frac{21}{6}$$



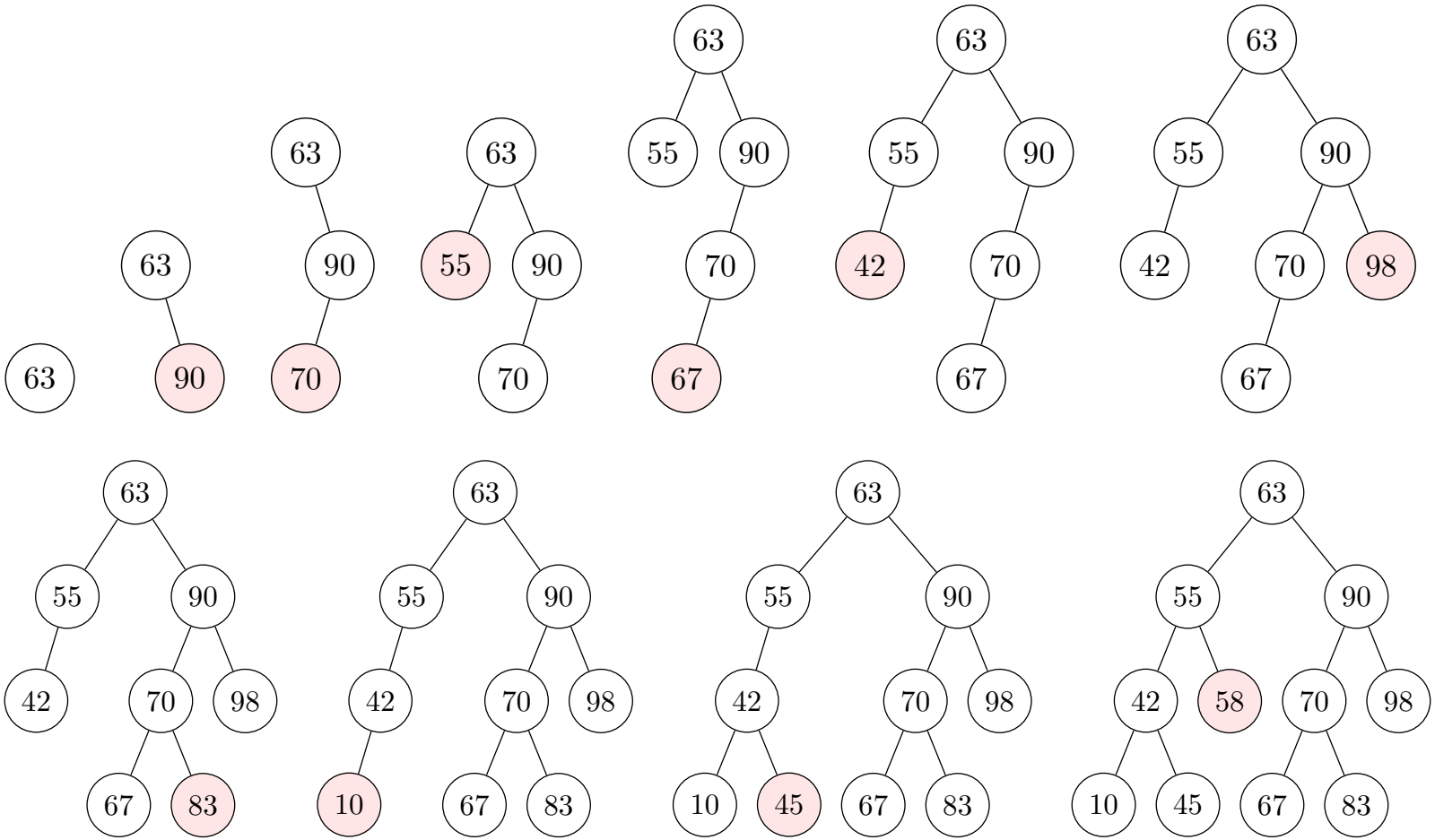
- 二叉排序树的平均查找长度和树的形态有关! 最好情况是 $\mathrm{O}(\log_2 \mathrm{n})$.

二叉排序树的构建 — 插入节点

- 在查找不成功时, 插入该 key
  - 新插入结点一定是作为叶子结点添加的
  - 插入位置在查找过程中得到
- 例如查找 56

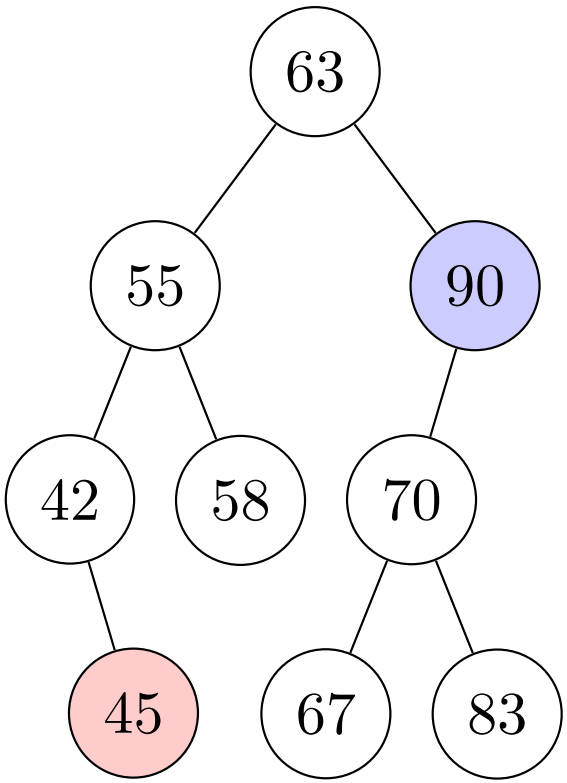


序列: 63, 90, 70, 55, 67, 42, 98, 83, 10, 45, 58



二叉排序树的删除操作

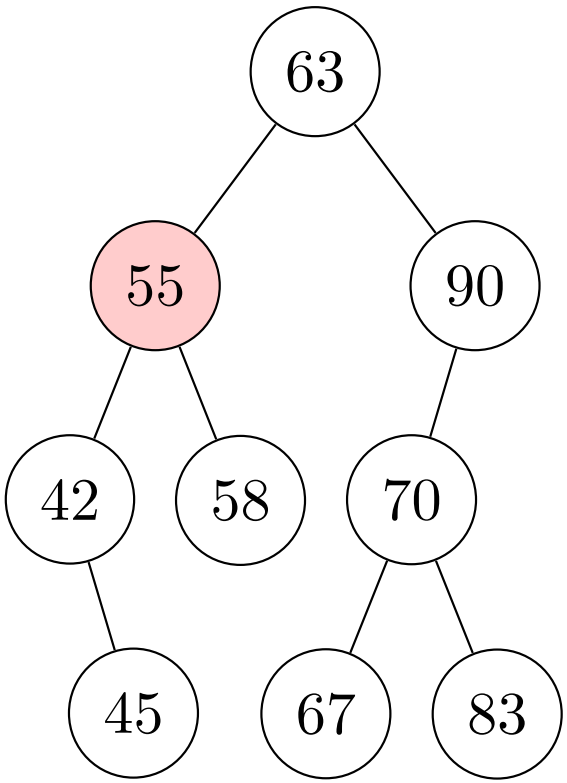
依次删除结点 45、90, 仍要使树保持二叉排序树的特性



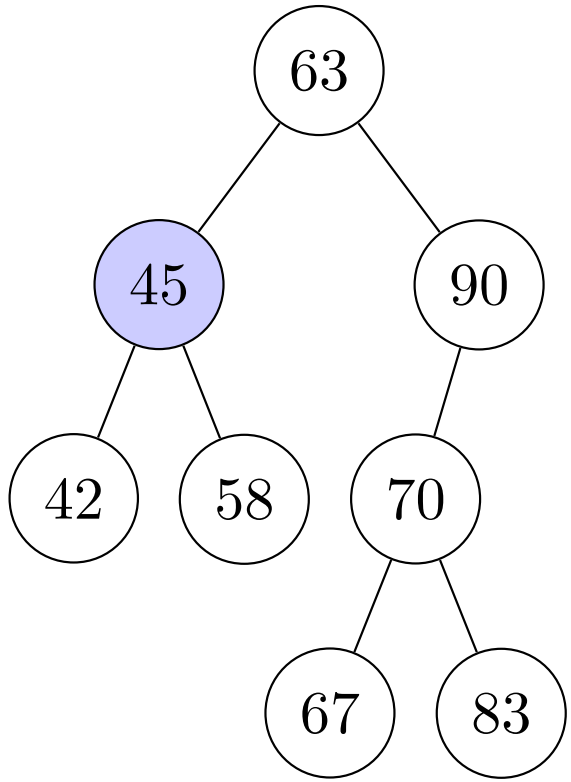
- 待删结点 p 为叶结点
  直接删除即可（如节点 45)
- 待删结点 p 只有右子树或只有左子树
  用子树的根代替之（如节点 90)
- 待删结点 p 有右子树也有左子树?
  原则: 保持中序遍历序列不变!

二叉排序树的删除操作

删除节点 55:

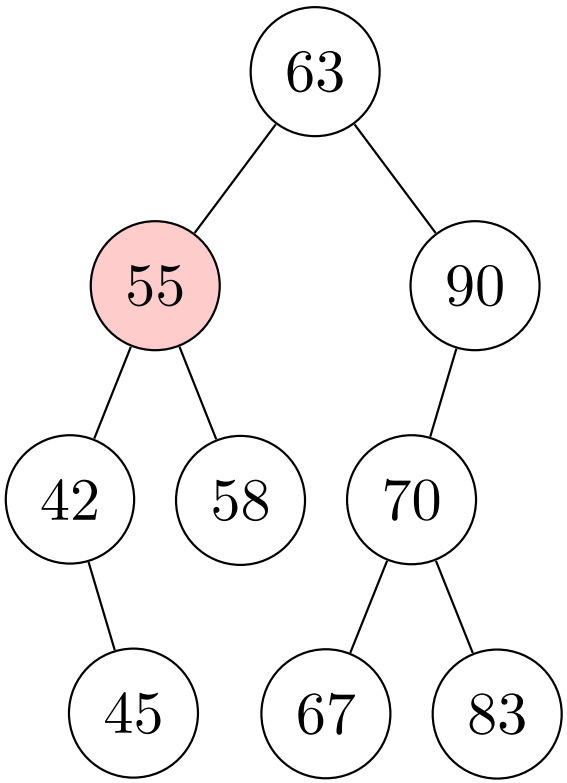


方法 1: 用前驱代替

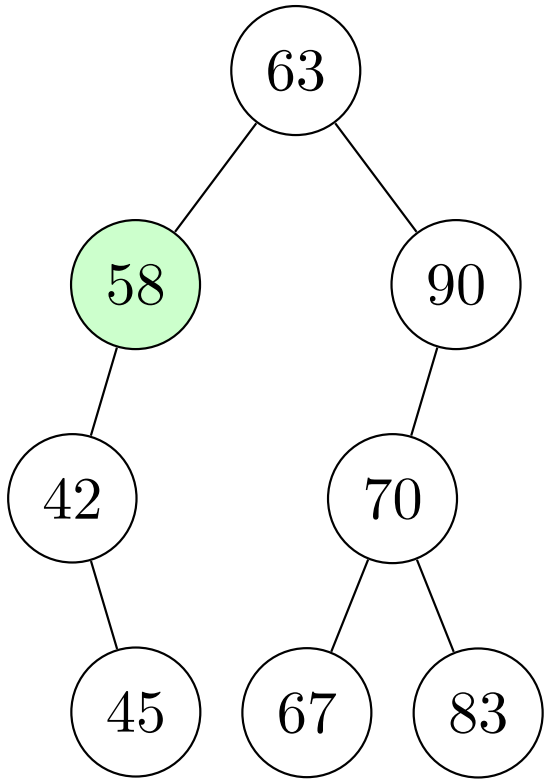


二叉排序树的删除操作

删除节点 55:

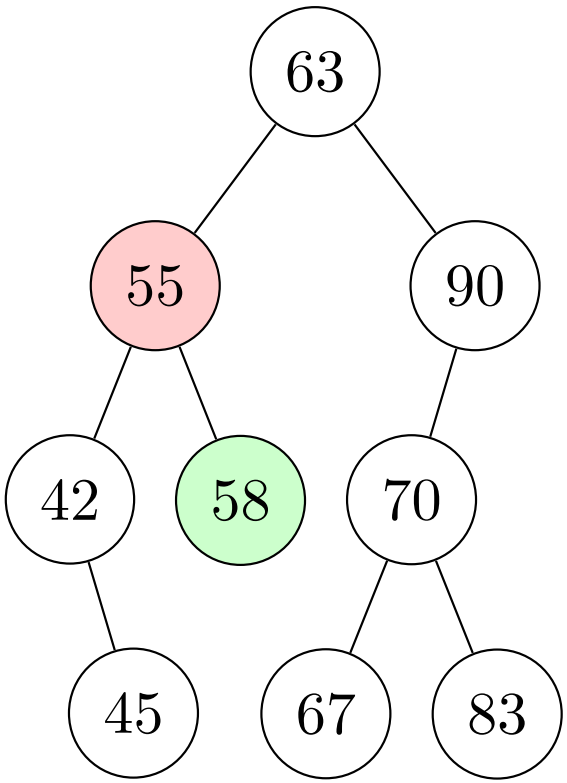


方法 2: 用后继代替

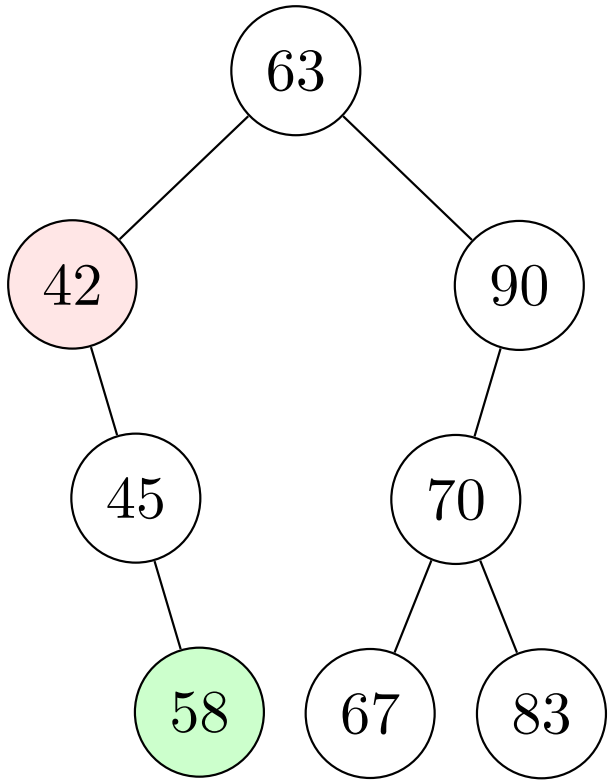


二叉排序树的删除操作

删除节点 55:



方法 3: 用左子树的根代替之, 并将右子树作为删除结点的前驱的右子树



课堂练习

给定关键字序列:63, 90, 70, 55, 67, 42, 98, 83, 10, 45, 58

- 构建二叉排序树
- 对该树中序遍历, 显示其序列
- 依次删除 10,42,63
- 再次对该树中序遍历, 显示其序列

平衡二叉树

在二叉查找树中, 若输入元素的顺序接近有序, 那么二叉查找树将退化为链表, 从而导致二叉查找树的查找效率大为降低。如何使得二叉查找树无论在什么样情况下都能使它的形态最大限度地接近满二叉树以保证它的查找效率呢?

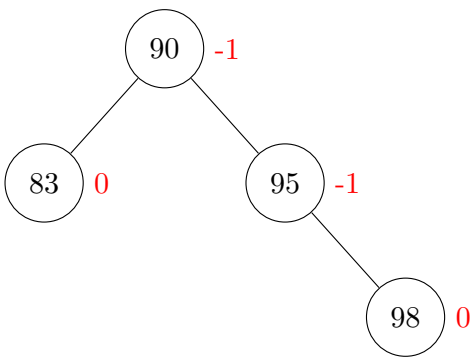
前苏联科学家 G.M. Adelson-Velskii 和 E.M. Landis 在 1962 年发表的一篇名为 An algorithm for the organization of information 的文章中提出了一种自平衡二叉查找树 (self-balancing binary search tree)。

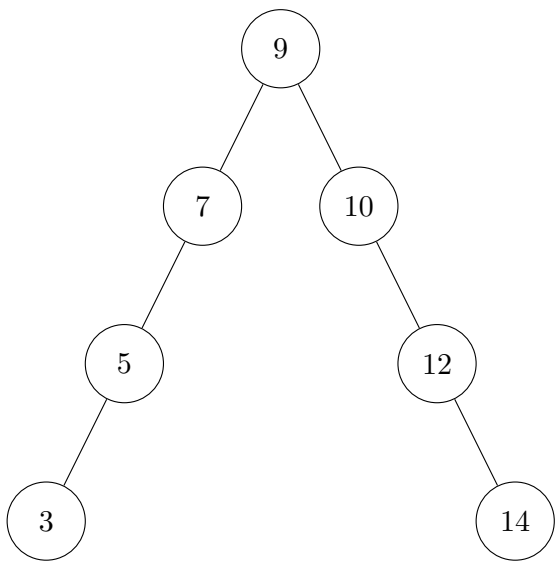
该树在插入和删除操作中, 通过一系列的旋转操作来保持平衡, 从而保证了二叉查找树的查找效率。最终这种二叉查找树以他们的名字命名为“AVL-Tree”。

平衡二叉树 (AVL)

平衡二叉树 (Balanced binary tree) 或者是一棵空树, 或者是具有下列性质的二叉排序树:

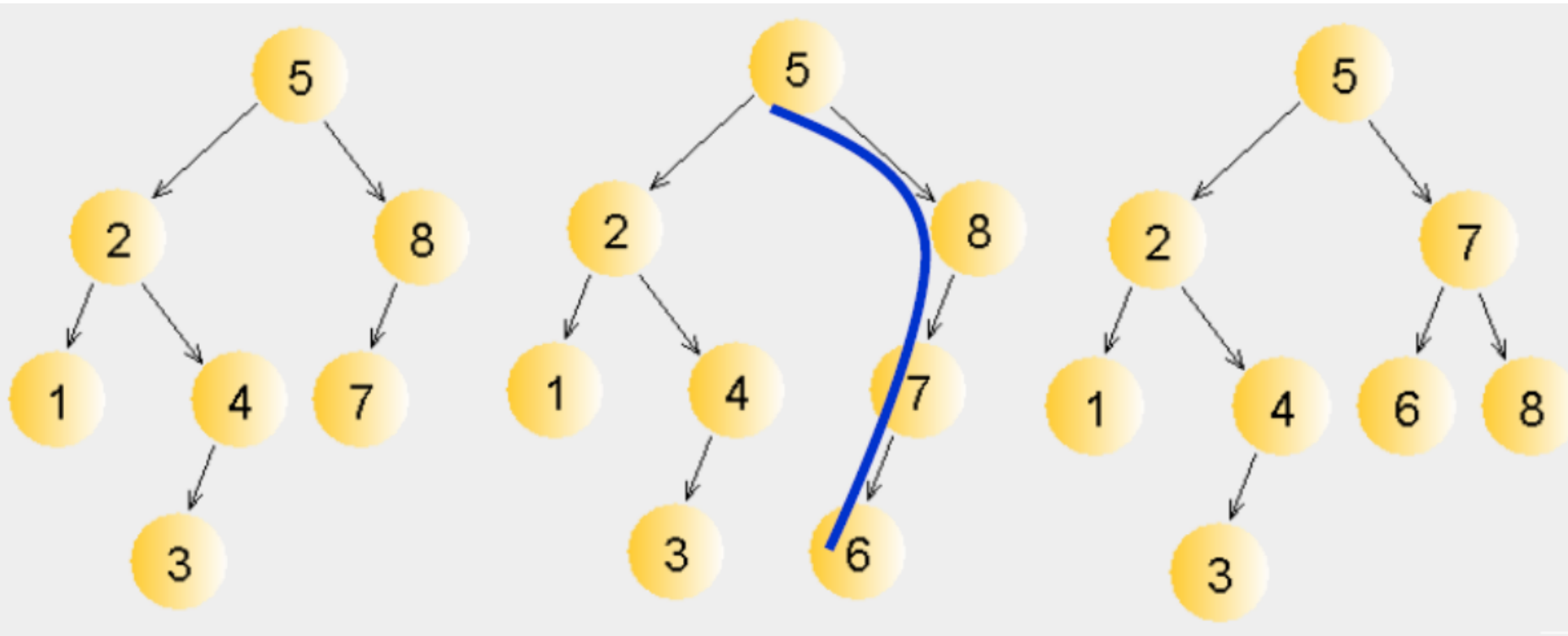
- 它的左子树和右子树都是平衡二叉树, 且左子树和右子树高度之差的绝对值不超过 1。
- 平衡因子 = 左子树高度-右子树高度





二叉子树的平衡化

- 在平衡二叉树上插入新结点，可能会导致不平衡!
- 请问哪些结点的平衡因子发生变化?



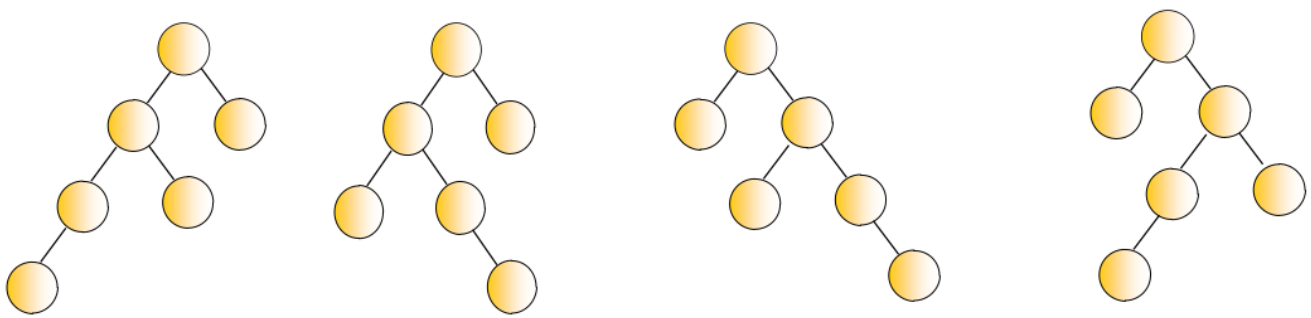
二叉子树的平衡化 (cont.)

平衡化调整的原则有二:

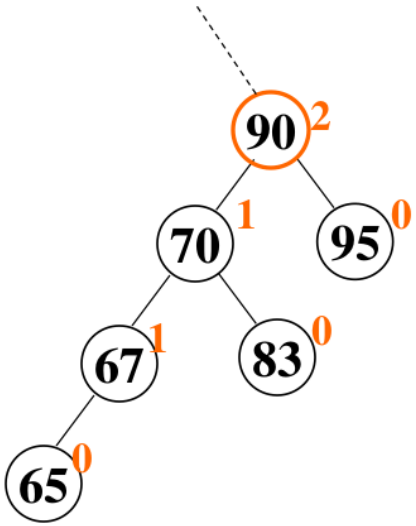
- 转换后的二叉树的中序遍历不变;
- 每次转换都要平衡。

四种情形:

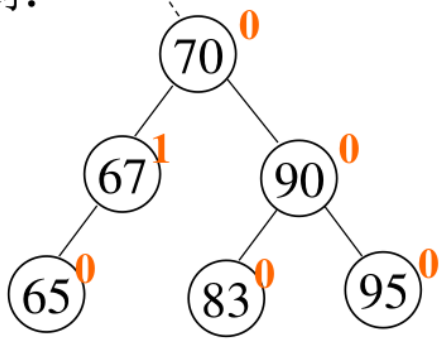
单向右旋　先左后右双向旋转　单向左旋　先右后左双向旋转



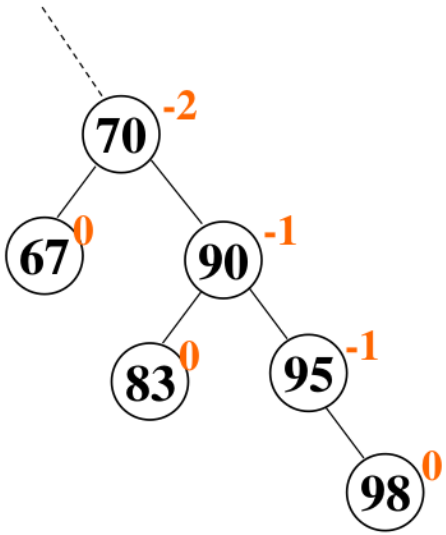
单向右旋



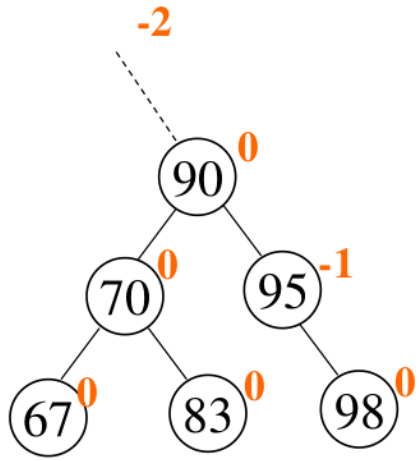
- 插入65导致不平衡！
- 结点90距离插入点最近，且平衡因子绝对值超过1；
- 结点90平衡因子为2，进行右旋得到：



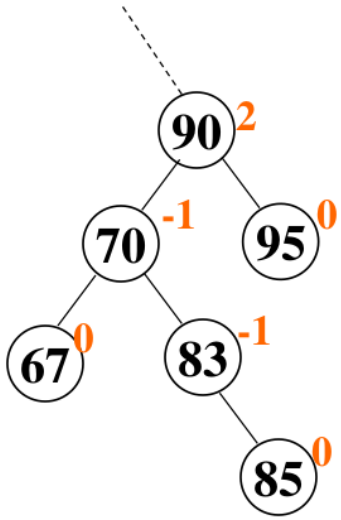
单向左旋



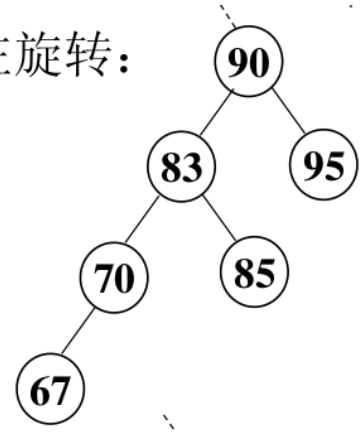
- 结点70平衡因子为-2，进行左旋得到：



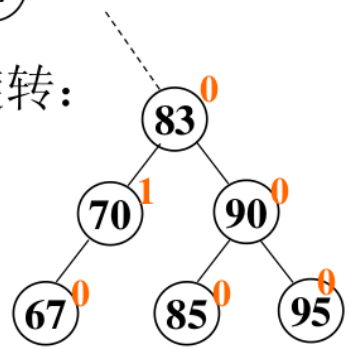
先左后右双向旋转



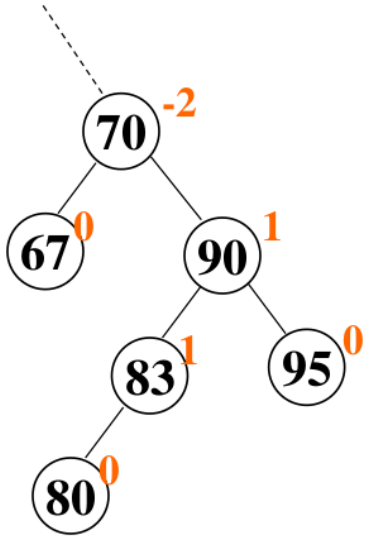
- 先进行左旋转：



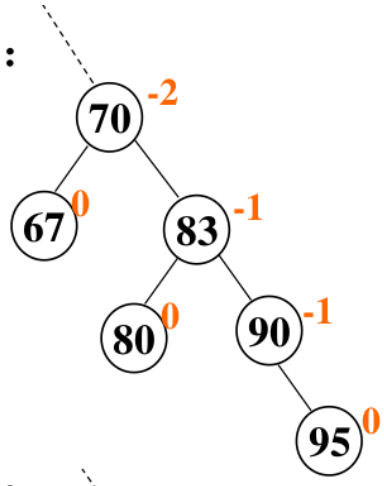
- 再进行右旋转：



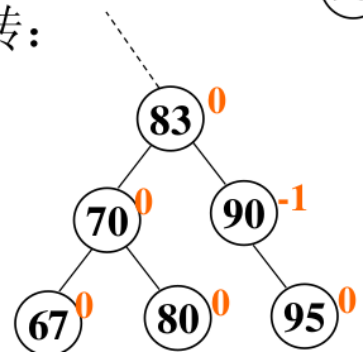
先右后左双向旋转



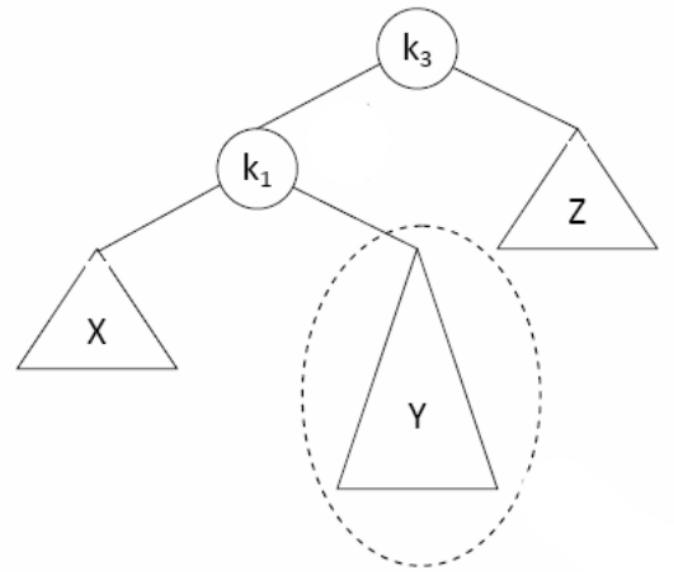
- 先进行右旋转：

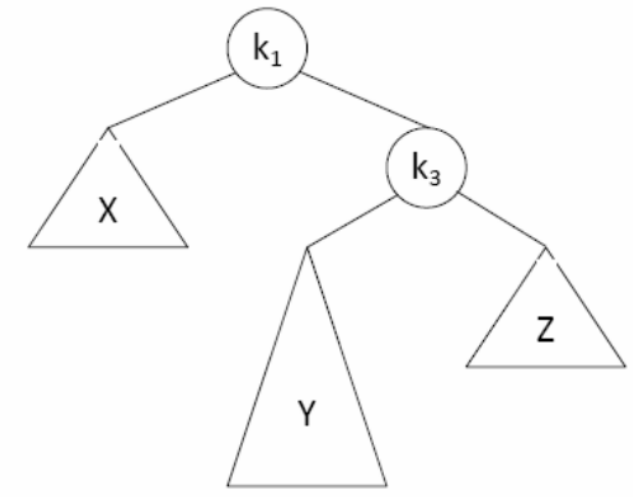


- 再进行左旋转：



Why double rotation?





练习

- 输入关键字 16,3,7,11,9,26,18,14,15,
- 构造一个 AVL 树

结果