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。

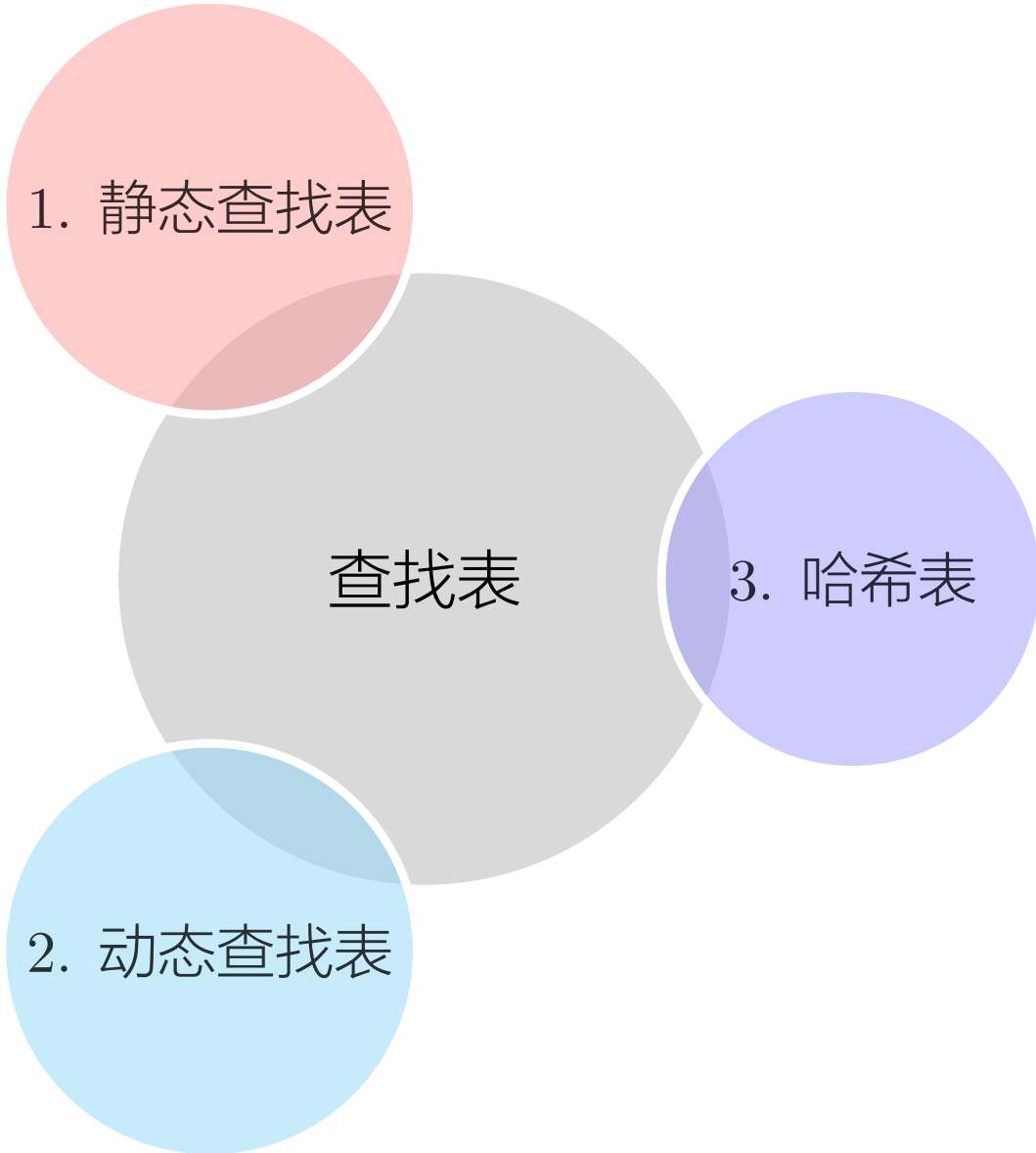
查找

- 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”。
  - 或者利用新的可选数据类型，例如 Java 中的 Optional<Int>，Scala 中的 Option[Int] 等

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

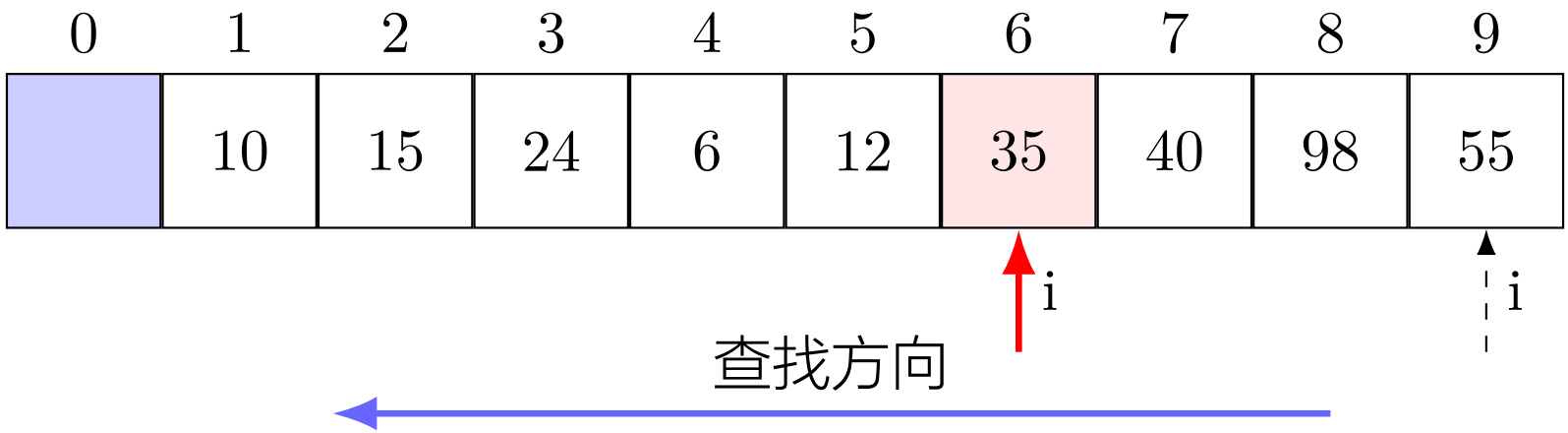
静态查找：对查找集合只进行查找，不涉及插入和删除操作。或者经过一段时间的查找之后，集中地进行插入和删除等修改操作。

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。

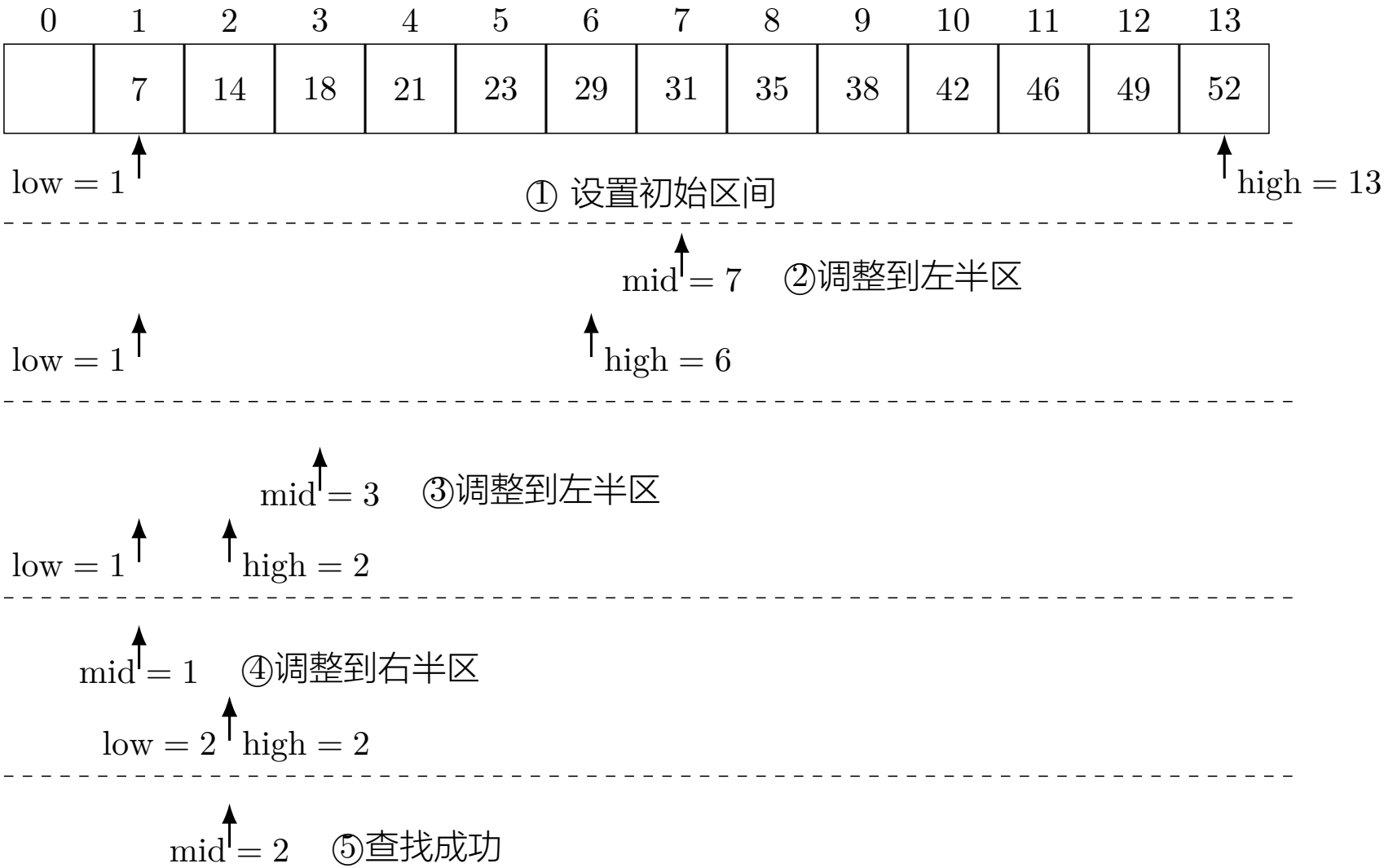
顺序查找总结

- 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

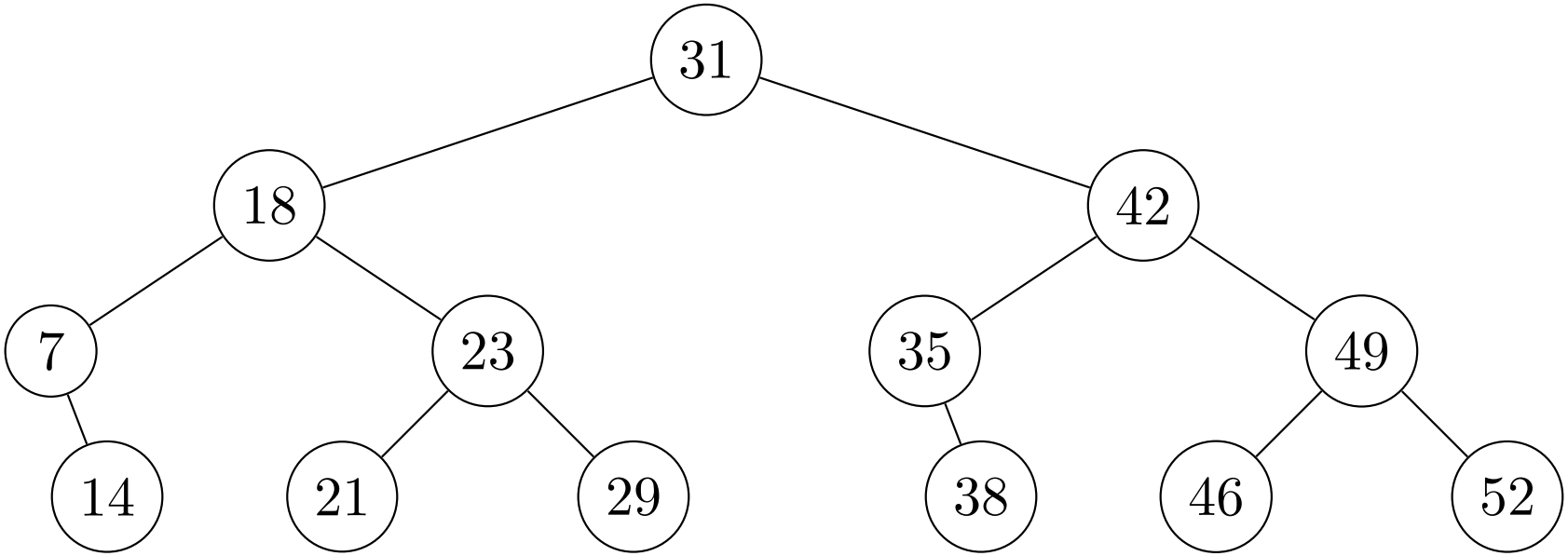
- 有序表是表中数据元素按关键码升序或降序排列。
- 适用于:
  - 线性表中的记录必须按关键码有序;
  - 必须采用顺序存储。

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

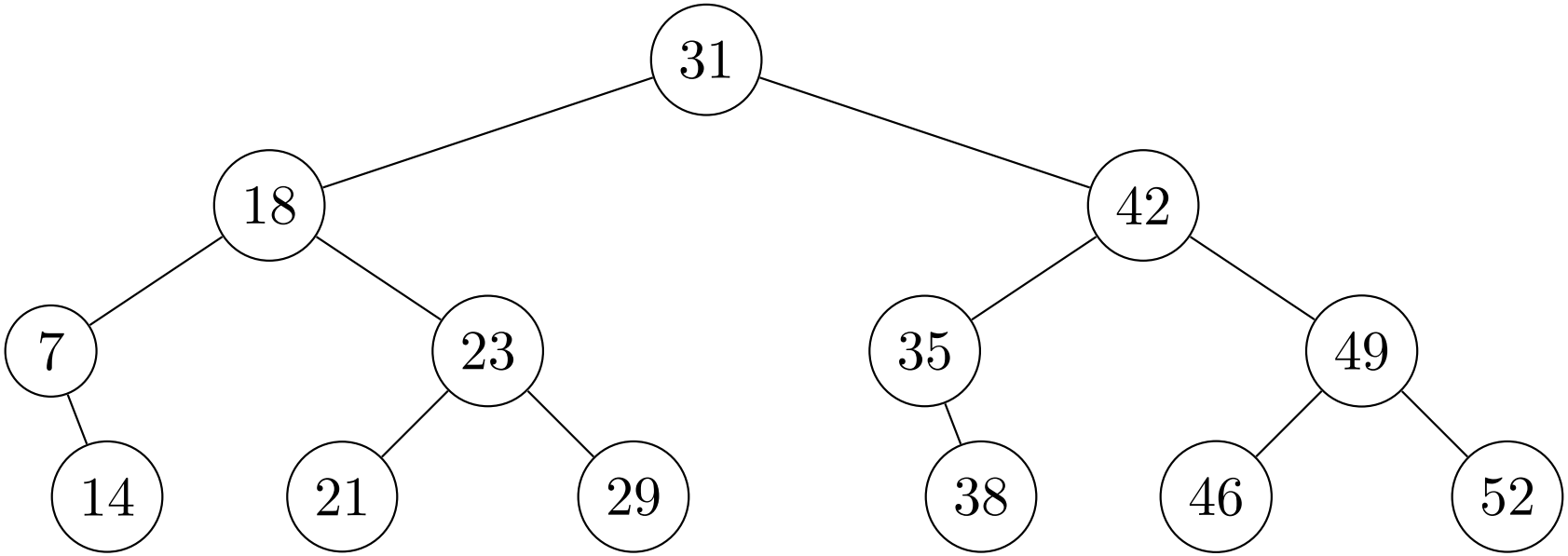
课堂练习：请查找 22

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?

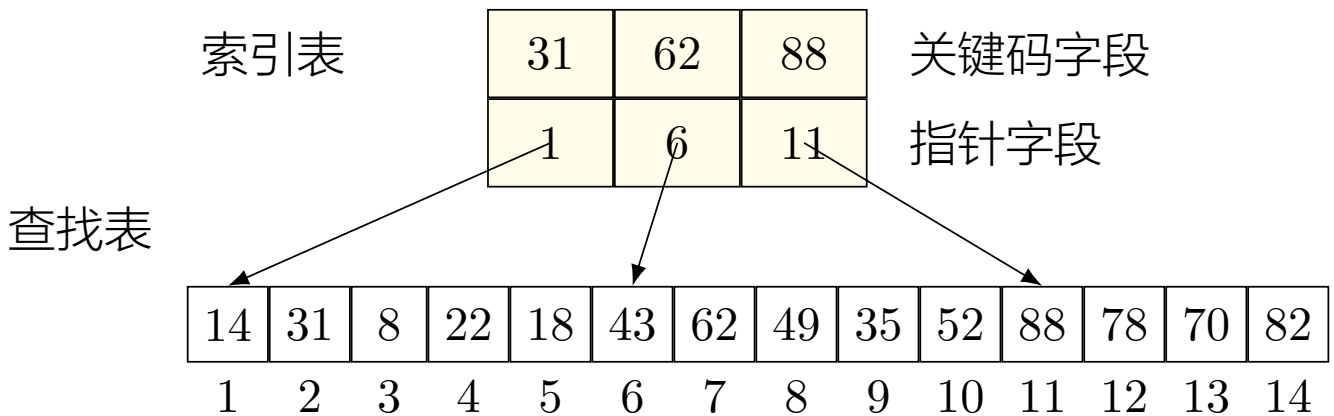
$$\lfloor \log_2 n \rfloor + 1$$

- 请问平均查找长度 (ASL) 是多少?

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

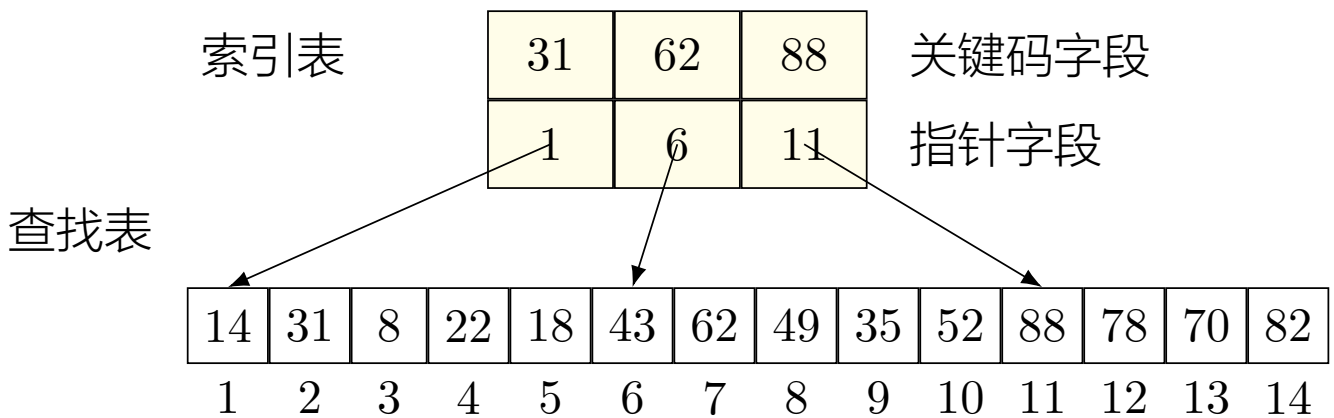
分块查找/索引顺序查找

- 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?

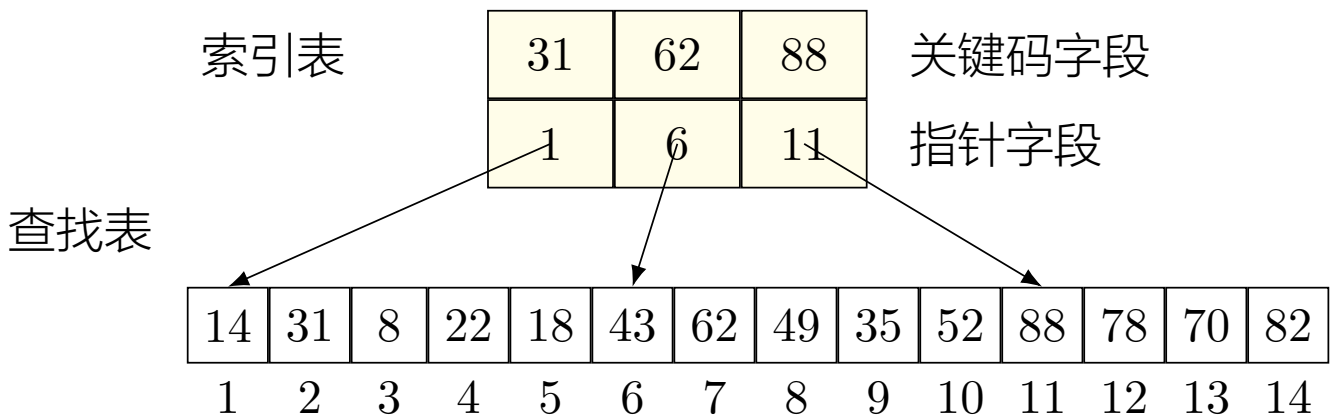


分块查找性能分析

- 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

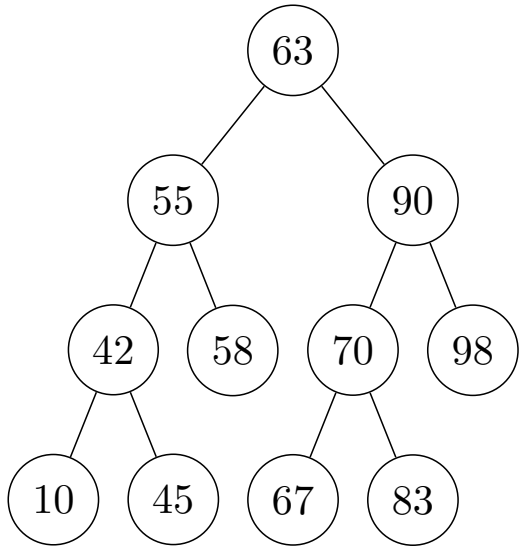
二叉排序树

- 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。

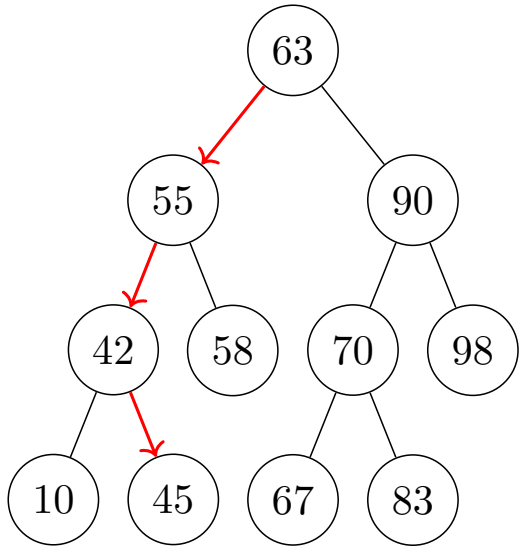
  ② 左右子树也都是二叉排序树。

- 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

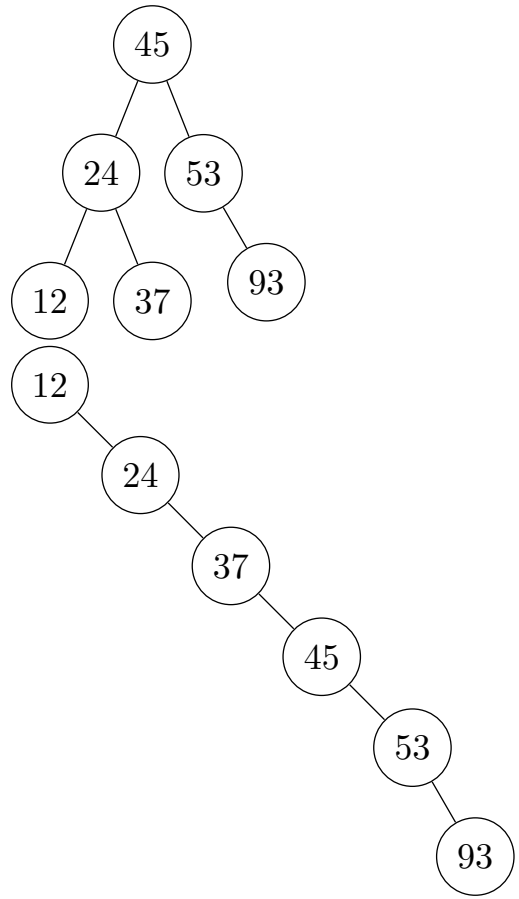


二叉排序树的查找 (cont.)

- 两树的平均查找长度分别为:

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

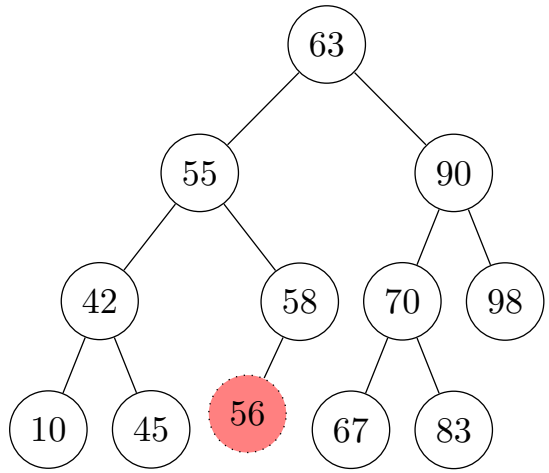
$$\mathrm{ASL_b} = \frac{1}{6} \times [1+2+3+4+5+6] = \frac{21}{6}$$



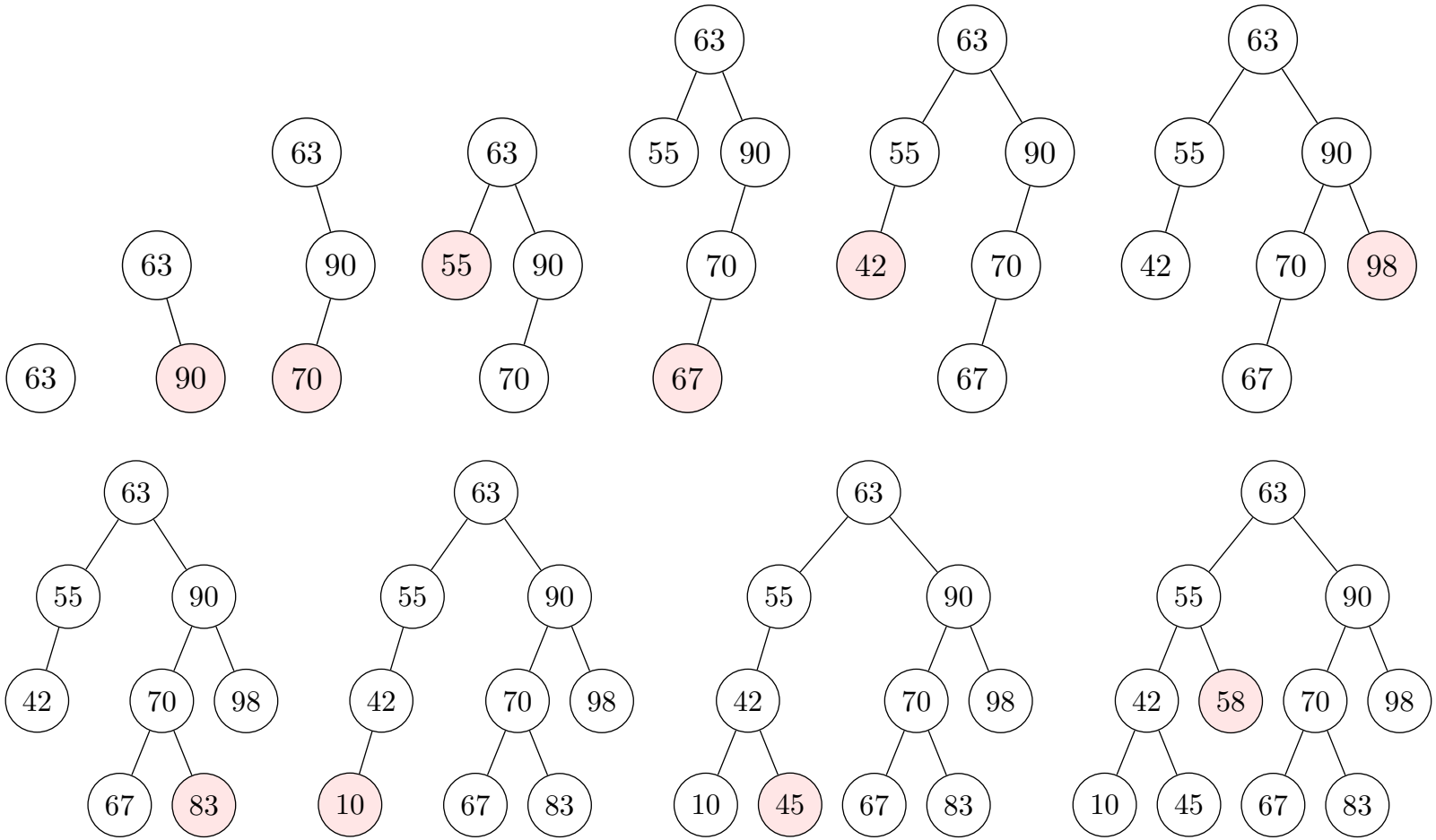
- 二叉排序树的平均查找长度和树的形态有关! 最好情况是 $\mathrm{O}(\log_2 \mathrm{n})$.

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

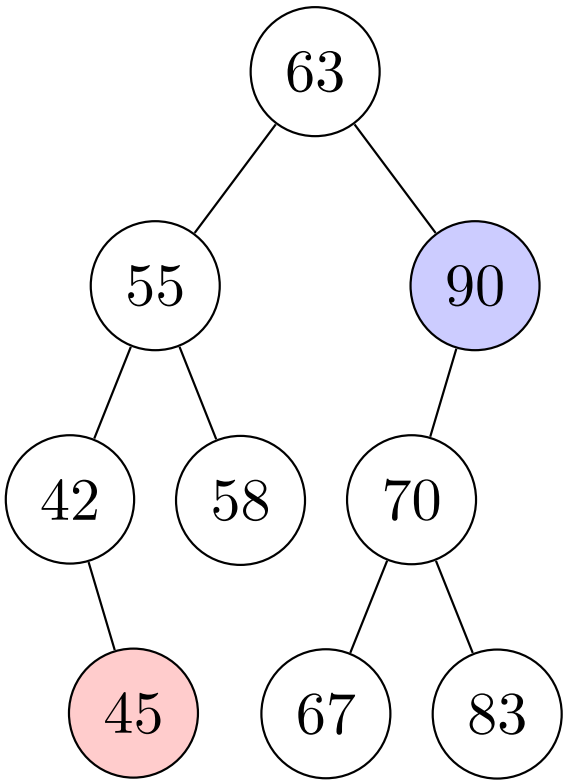


序列: 63, 90, 70, 55, 67, 42, 98, 83, 10, 45, 58



二叉排序树的删除操作

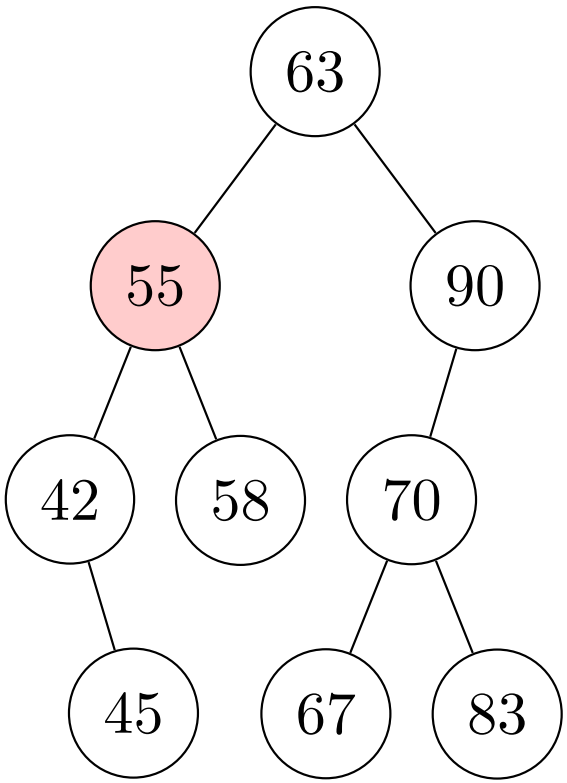
依次删除结点 45、90, 仍要使树保持二叉排序树的特性



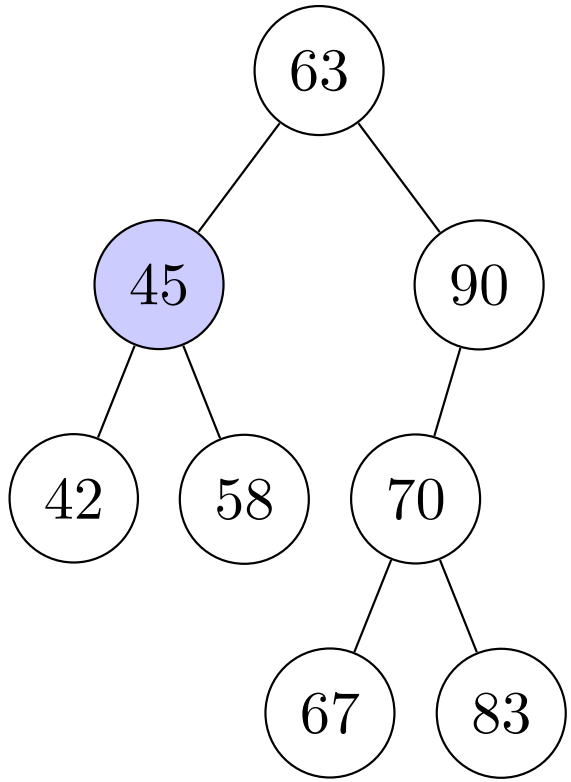
- 待删结点 p 为叶结点
  直接删除即可 (如节点 45)
- 待删结点 p 只有右子树或只有左子树
  用子树的根代替之 (如节点 90)
- 待删结点 p 有右子树也有左子树?
  原则: 保持中序遍历序列不变!

二叉排序树的删除操作

删除节点 55:

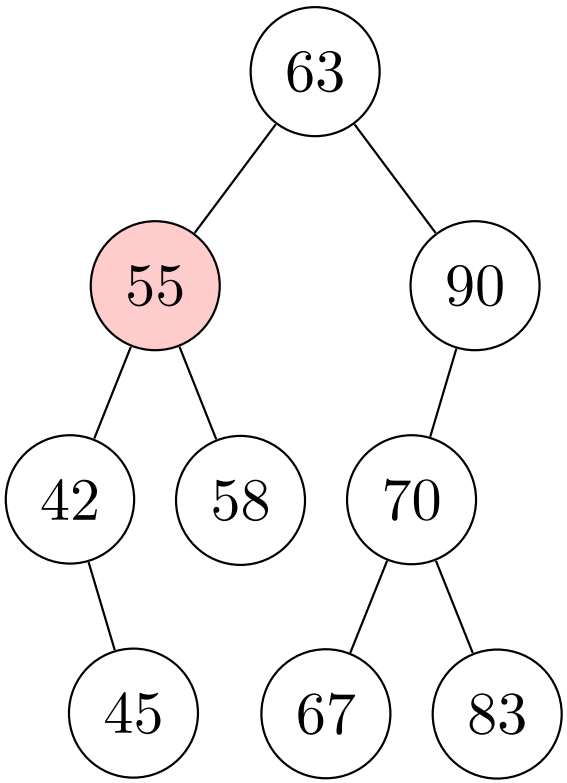


方法 1: 用前驱代替

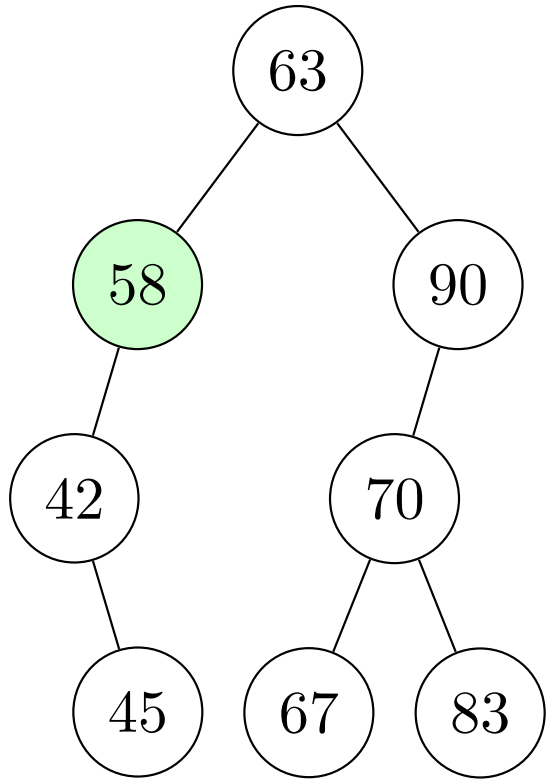


二叉排序树的删除操作

删除节点 55:

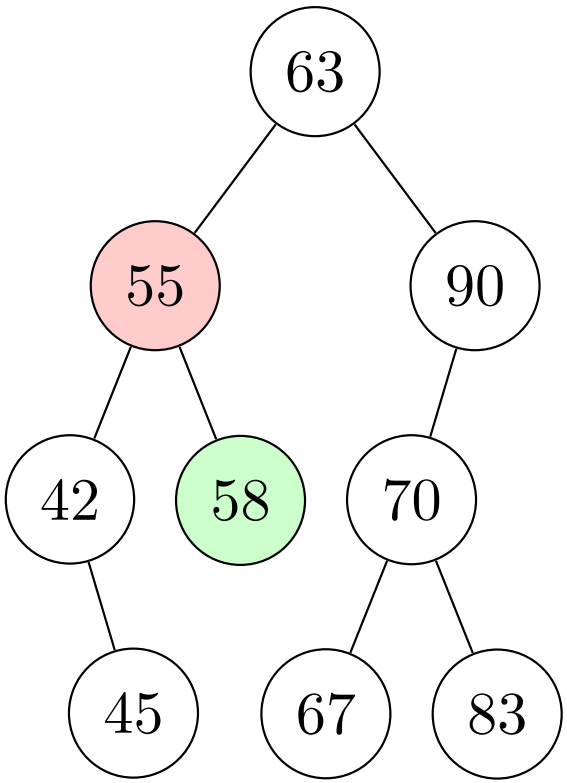


方法 2: 用后继代替

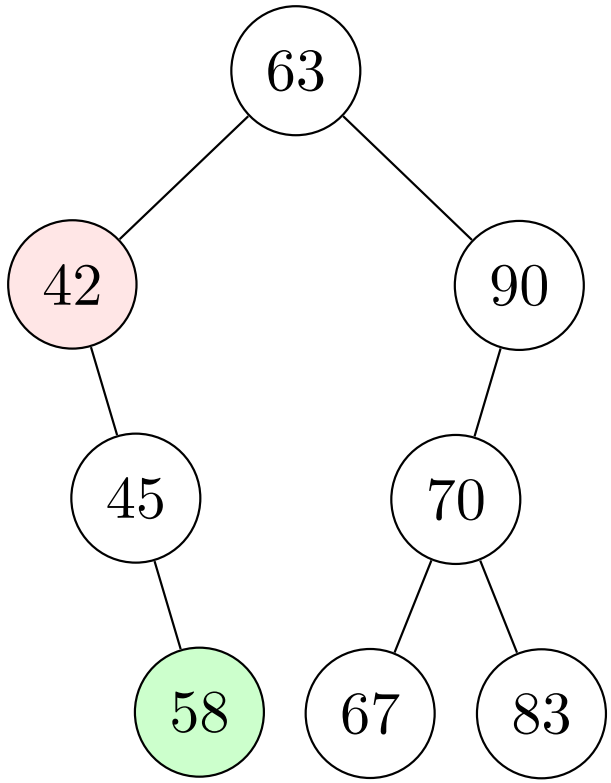


二叉排序树的删除操作

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?

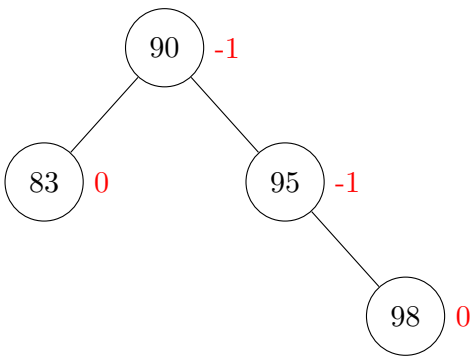
前苏联科学家 G.M. Adelson-Velskii 和 E.M. Landis 在 1962 年发表的一篇名为 An algorithm for the organization of information 的文章中提出了一种自平衡二叉查找树 (self-balancing binary search tree)。

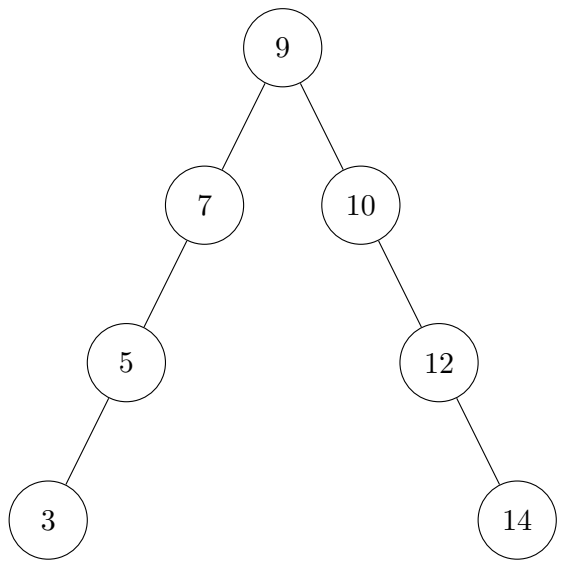
该树在插入和删除操作中, 通过一系列的旋转操作来保持平衡, 从而保证了二叉查找树的查找效率。最终这种二叉查找树以他们的名字命名为“AVL-Tree”。

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:

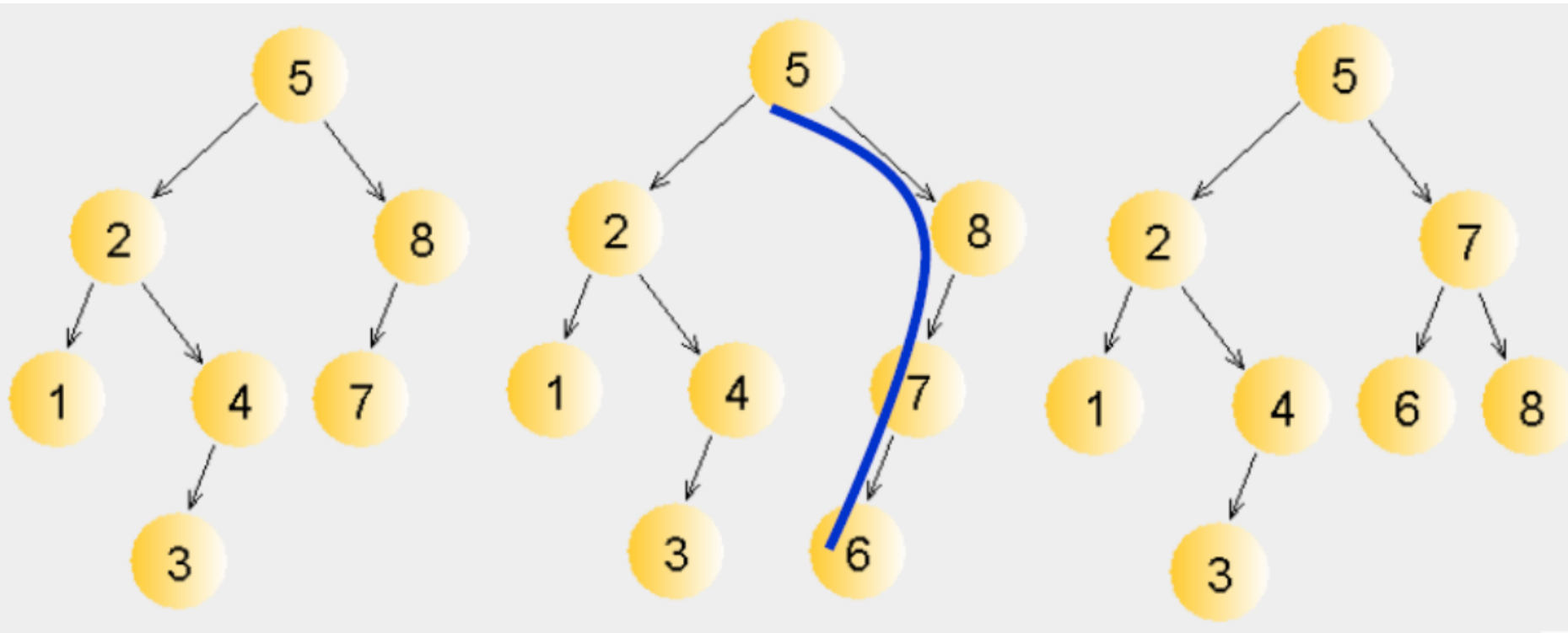
- 它的左子树和右子树都是平衡二叉树, 且左子树和右子树高度之差的绝对值不超过 1。
- 平衡因子 = 左子树高度-右子树高度





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?



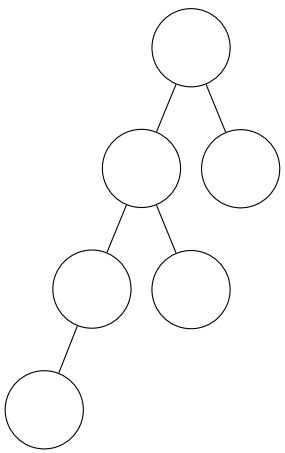
二叉子树的平衡化 (cont.)

平衡化调整的原则:

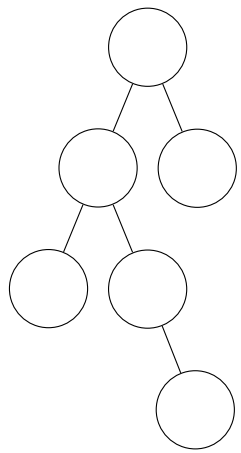
- 转换后的二叉树的中序遍历不变;
- 每次转换都要平衡。

四种情形:

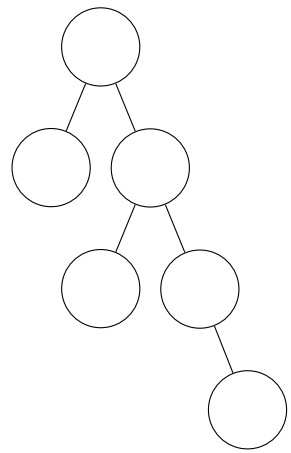
单向右旋



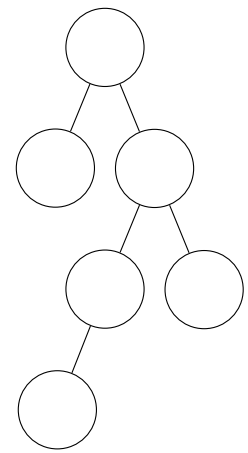
先左后右双向旋转



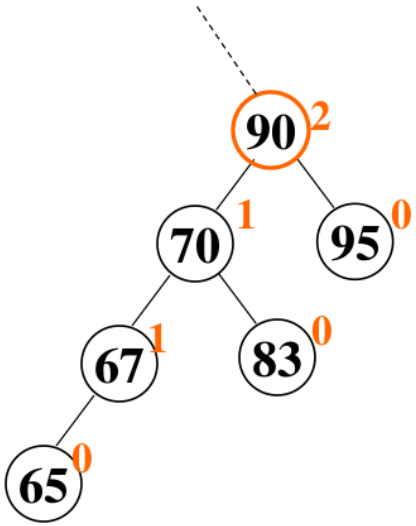
单项左旋



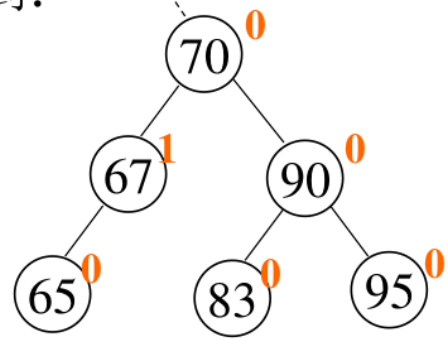
先右后左双向旋转



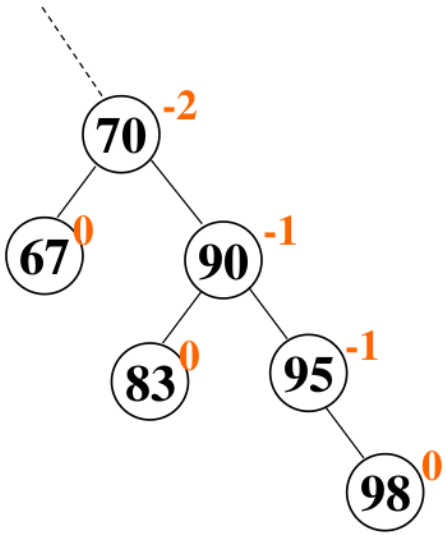
单向右旋



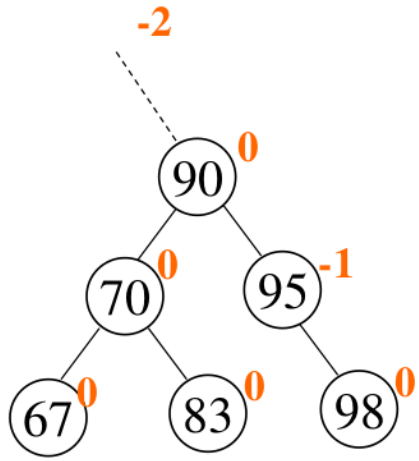
- 插入65导致不平衡！
- 结点90距离插入点最近，且平衡因子绝对值超过1；
- 结点90平衡因子为2，进行右旋得到：



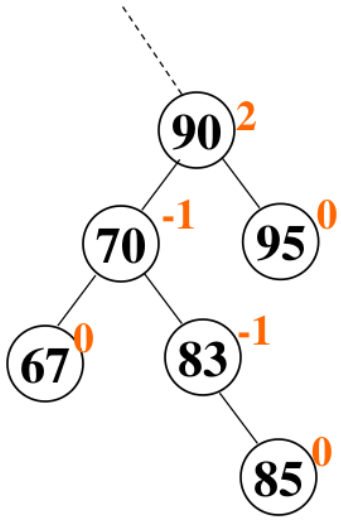
单向左旋



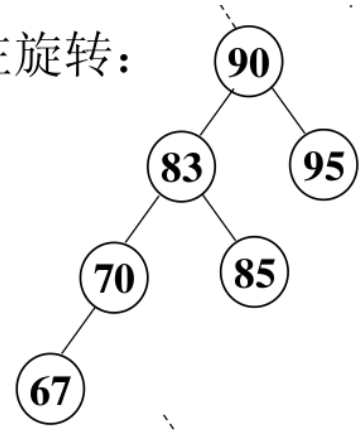
- 结点70平衡因子为-2，进行左旋得到：



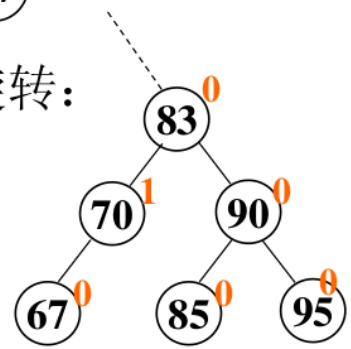
先左后右双向旋转



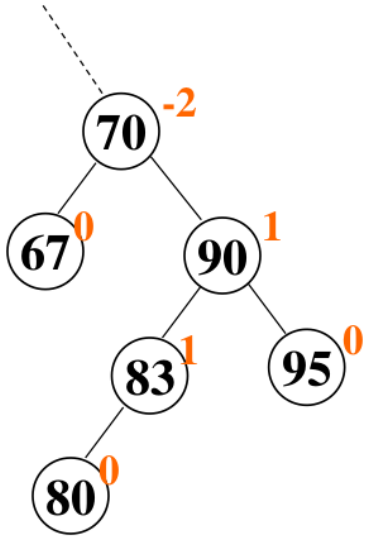
- 先进行左旋转：



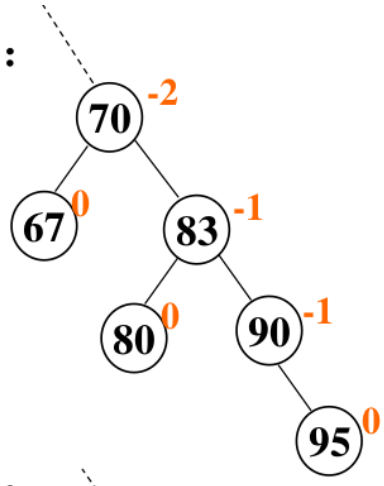
- 再进行右旋转：



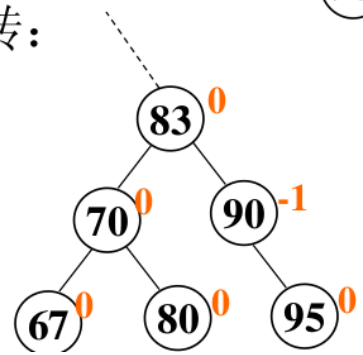
先右后左双向旋转



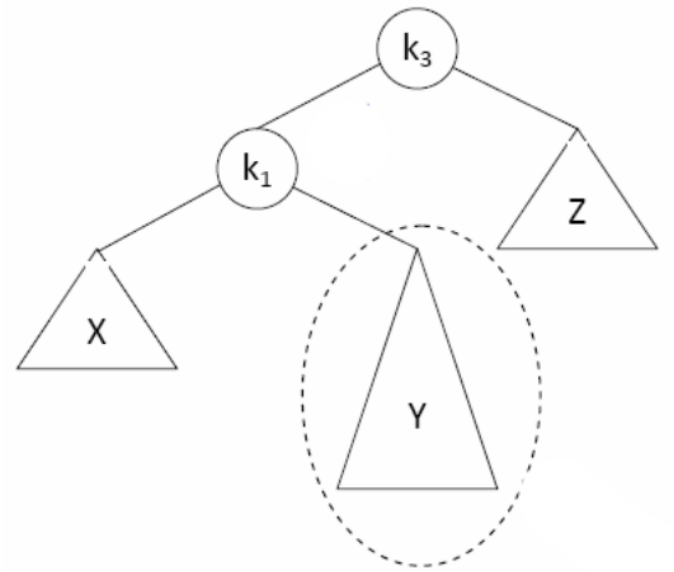
- 先进行右旋转：

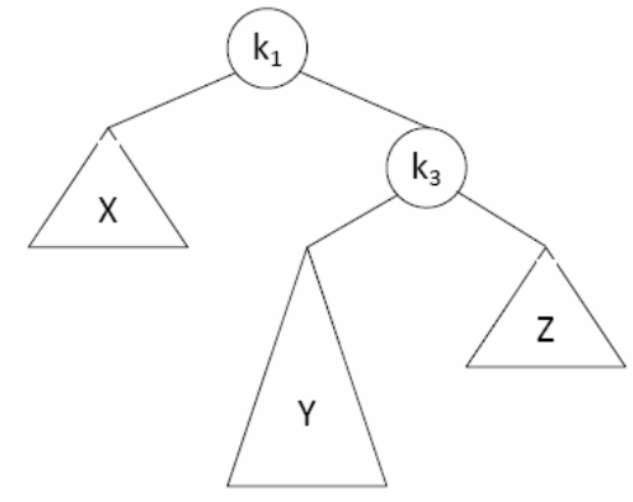


- 再进行左旋转：



Why double rotation?

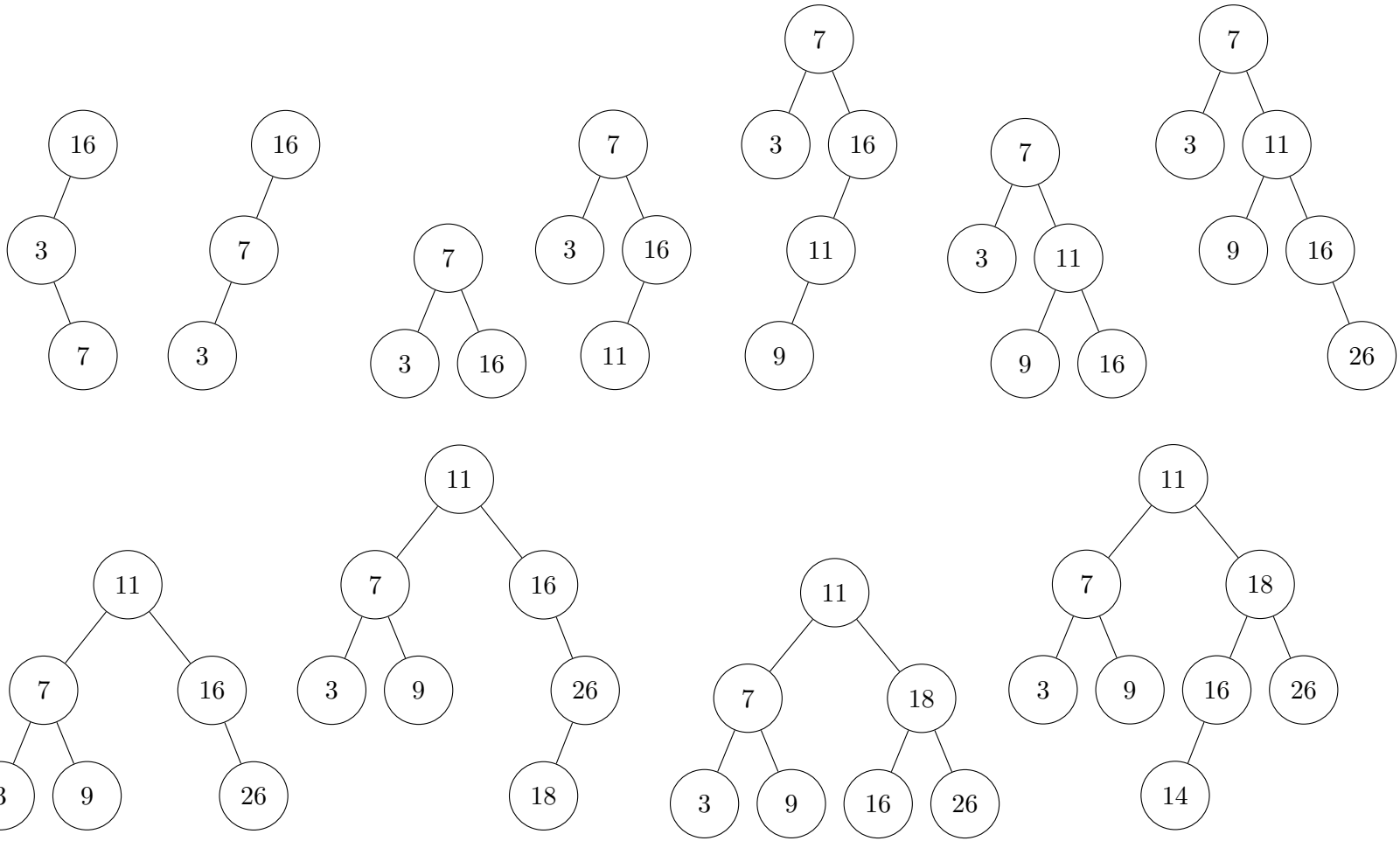




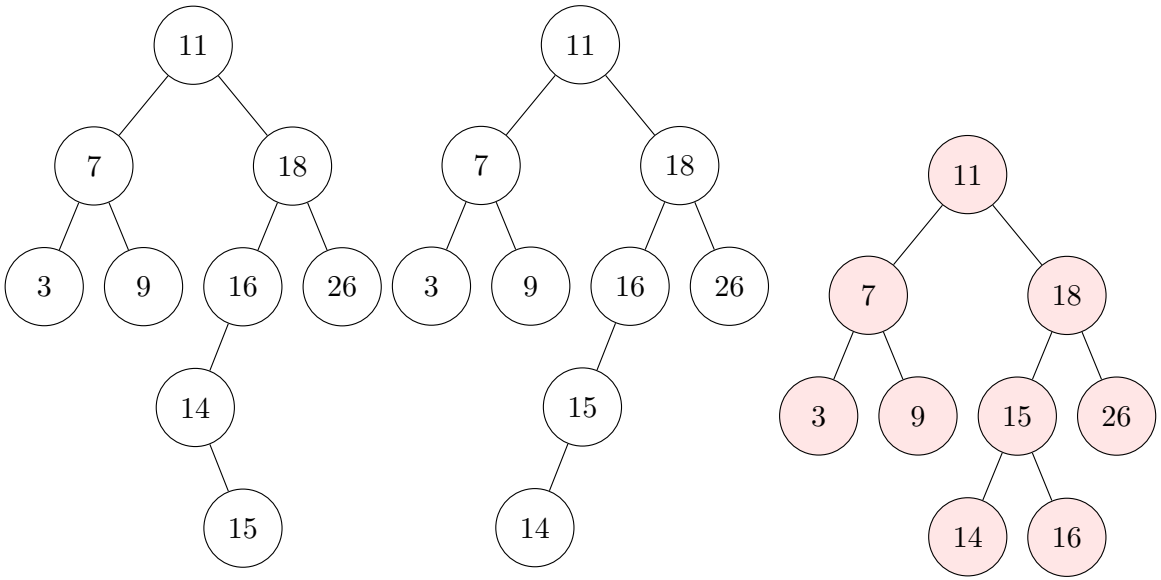
练习

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

序列：16,3,7,11,9,26,18,14,15



序列：16,3,7,11,9,26,18,14,15



3. 哈希表

哈希是一种重要的存储方法，也是一种重要的查找方法。其基本思想是以关键字为自变量，使用哈希函数映射到地址集合，那么根据哈希函数便可以直接找到包含该关键字的集合的存储地址。

Think

- 顺序查找
- 折半查找
- 分块查找
- 二叉查找树

对给定值和关键字进行比较，查找效率由比较一次缩小的查找范围决定。

能否直接定位到给定值的存储位置，不用逐步缩小查找范围?

哈希表

- 哈希方法：选取某个函数，依据关键字直接得到其对应的数据元素的存储位置。也有的将哈希译为散列。
- 哈希方法中使用的转换函数称为哈希函数。按这个思想构造的表称为哈希表。

Example

Name	Cellphone	Affiliate
Zhang	13800011234	RUC
Wang	13800021235	RUC
Zhao	13800038322	IRM
Qian	13800047322	IRM

- 该例子中我们可以直接利用红色部分作为存储单元的位置。
- 理想的哈希函数应该运算简单并保证不同的关键字映射到不同单元，但这是不可能的。冲突不可避免，只能尽量减少。
- 存储单元涉及的范围不要过大

哈希方法需要解决的两个问题：

- 构造好的哈希函数
- 制定解决冲突的方案

Hash 函数的构造方法

- Hash 地址 (函数值) 分布应均匀
  函数值尽量均匀散布在地址空间，保证空间有效利用并减少冲突
- Hash 函数计算应简单，保证转换的效率

1. 直接定址法

- Hash 函数是关键字的线性函数：
  - 这类函数是一一对应函数，对于不同的关键字不会产生冲突；但得到的地址集合与关键字集合大小相同，因此不适用于较大的关键字集合。

2. 数字分析法

- 根据关键码在各个位上的分布情况，选取分布比较均匀的若干位组成 Hash 地址。
- 适用情形：
  - 能预估出全部关键码每一位上各种数字出现的频度。

3. 平方取中法

- 对关键码平方后，按散列表大小，取中间的若干位作为 Hash 地址。
- 适用于：
  - 事先不知道关键码的分布，且关键码位数不是很大。

4. 折叠法

- 将关键码从左到右分割成位数相等的几部分，将这几部分叠加求和，取后几位作为散列地址。
- 适用于：
  - 事先不知道关键码的分布，且关键码位数很大。

5. 除留余数法

- 取关键字除以 p 的余数作为哈希地址，或者在关键字折叠、平方取中等后取模。

$$\mathrm{Hash(key)} = \mathrm{keyMODp}$$

- 除留余数法是一种最简单、也是最常用的构造散列函数的方法，并且不要求事先知道关键码的分布。
- p 最好接近表长；一般选取质数，或不包含小于 20 的质因数的合数。

Example: 给定表长 m = 2000，选 p = 1999，则 $\mathrm{Hash}(13810066001)\%1999$

处理冲突的方法

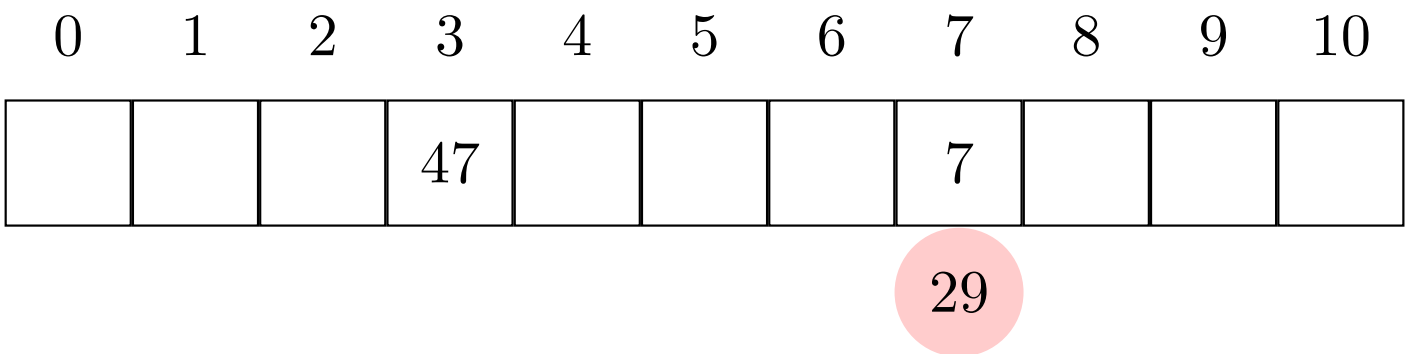
- Hash 地址 (函数值) 分布应均匀
  - 函数值尽量均匀散布在地址空间，保证空间有效利用并减少冲突
- Hash 函数计算应简单，保证查找效率

方法:

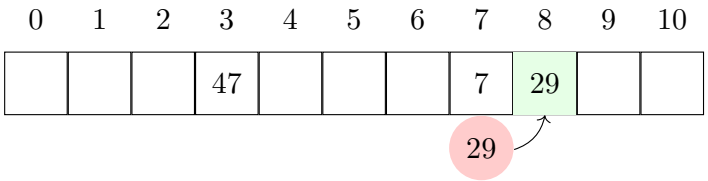
- 开放定址法
- 再哈希法
- 拉链法
- 建公共溢出区

1. 开放定址法

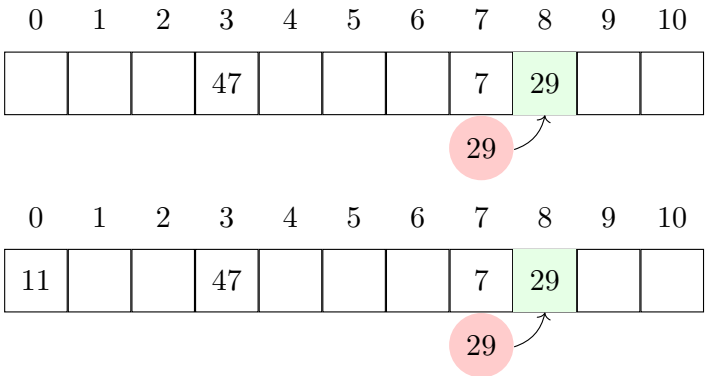
- 由关键码得到的散列地址一旦产生了冲突，就去寻找下一个空的散列地址，并将记录存入。
- 例如，关键码集合为 $\{47, 7, 29, 11, 16, 92, 22, 8, 3\}$，hash 表的表长 $\mathrm{m} = 11$，$\mathrm{Hash(key)} = \mathrm{keyMOD11}$，用线性探测法处理冲突如下：



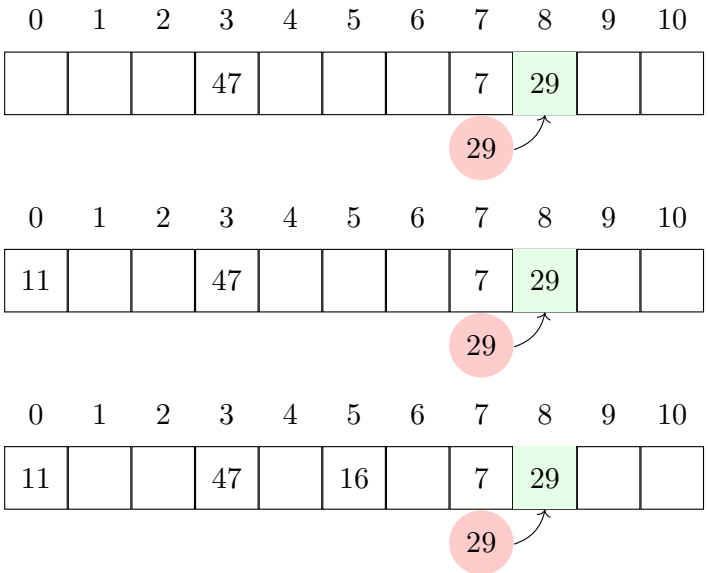
1. 开放定址法–示例: $\{47, 7, 29, 11, 16, 92, 22, 8, 3\}$



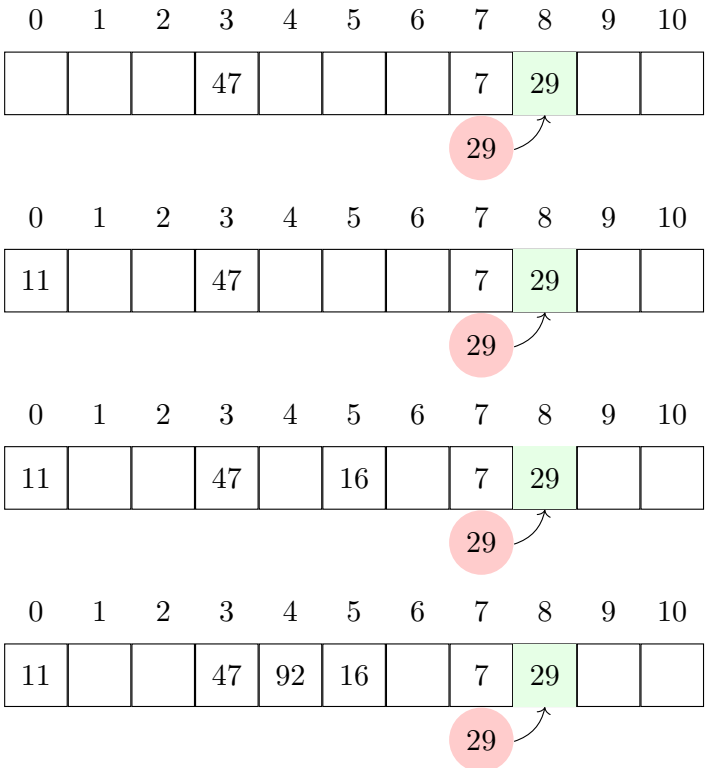
1. 开放定址法–示例: $\{47, 7, 29, 11, 16, 92, 22, 8, 3\}$

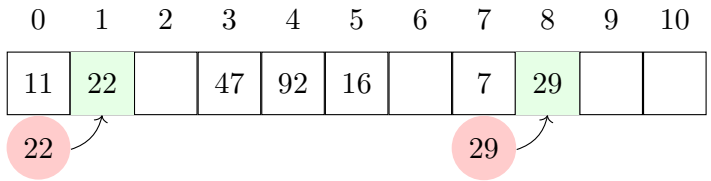


1. 开放定址法–示例: $\{47, 7, 29, 11, 16, 92, 22, 8, 3\}$

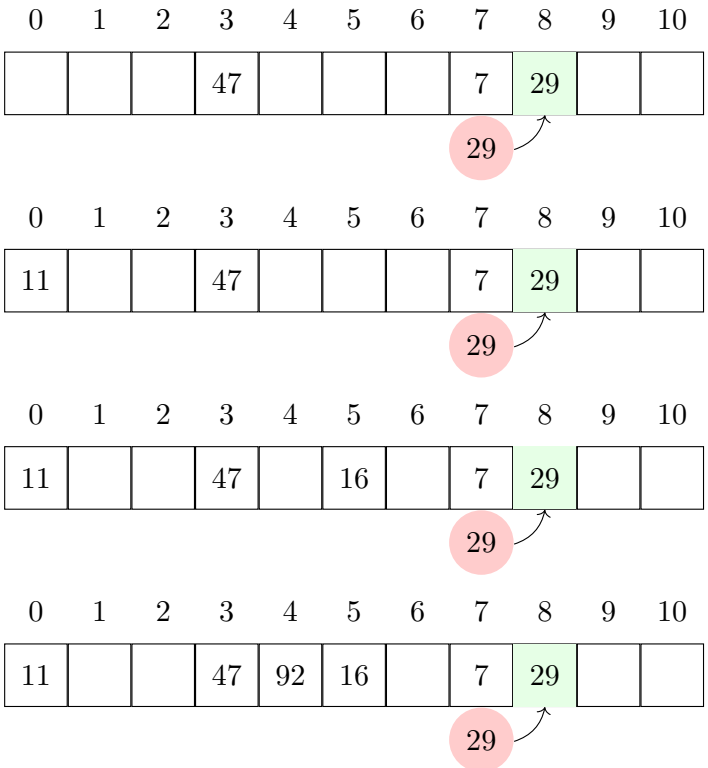


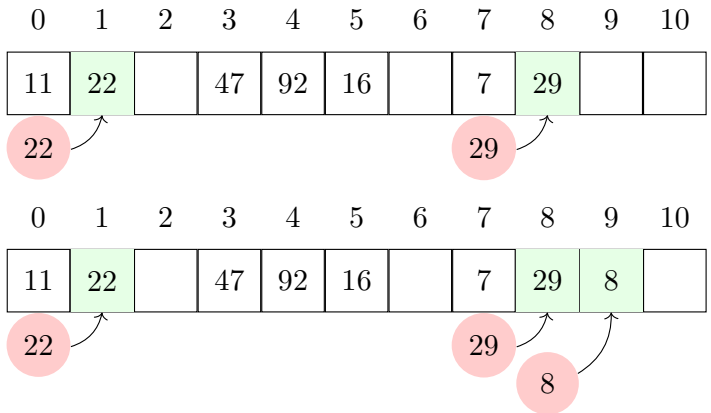
1. 开放定址法–示例: $\{47, 7, 29, 11, 16, 92, 22, 8, 3\}$



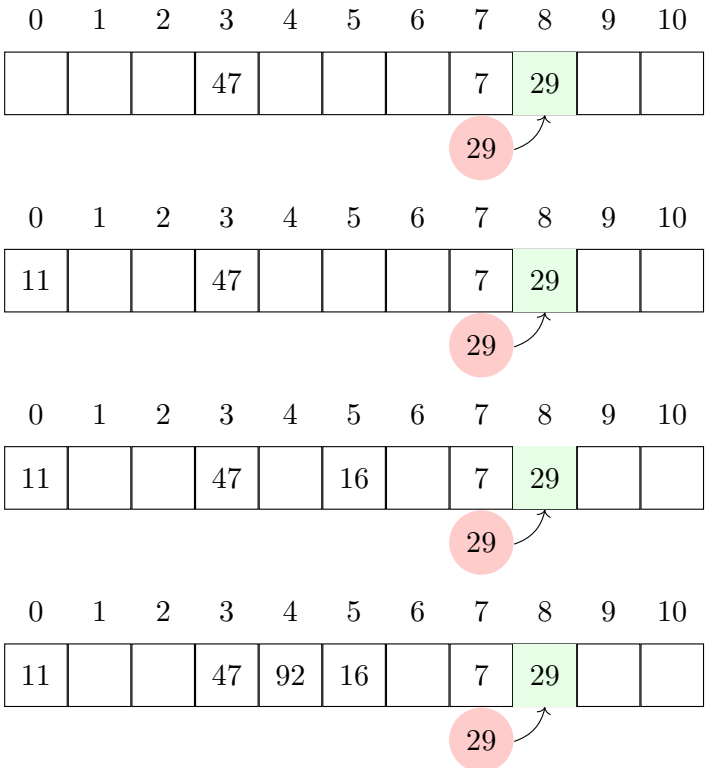


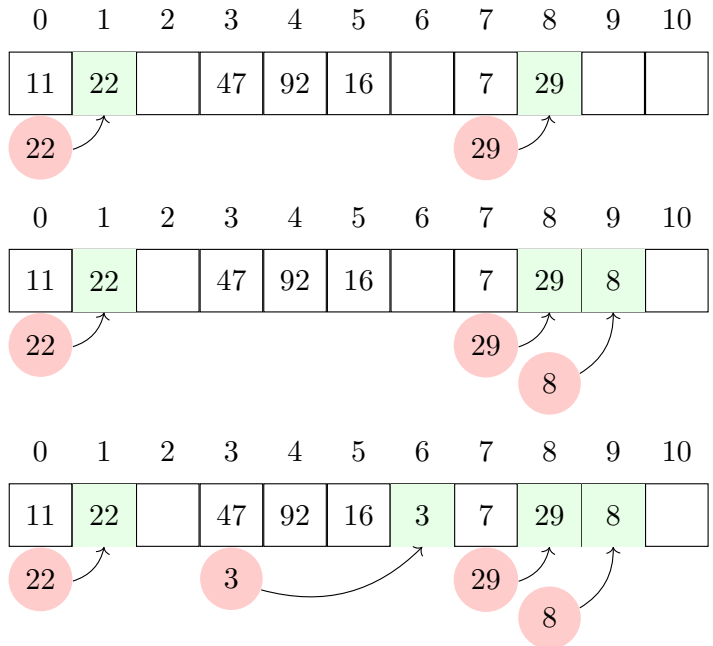
1. 开放定址法–示例: $\{47, 7, 29, 11, 16, 92, 22, 8, 3\}$





1. 开放定址法–示例: $\{47, 7, 29, 11, 16, 92, 22, 8, 3\}$





开放定址法

另一种冲突时空间选择的策略：

每次检查位置空间的步长以平方倍增加。也就是说，如果位置 s 被占用，则首先检查 $s+1^2$ 处，然后检查 $s-1^2$，$s+2^2$，$s-2^2$，$s+3^2$，$\cdots$ 依此类推，而不是象线性挖掘那样从 $s+1, s+2, \cdots$ 线性增长。

2. 再哈希法

如果 $\mathrm{Hash(key)} = \mathrm{a}$ 时产生地址冲突, 就再用 $\mathrm{ReHash(key)} = \mathrm{b}$ 确定移动的步长因子, 寻找空的哈希地址。

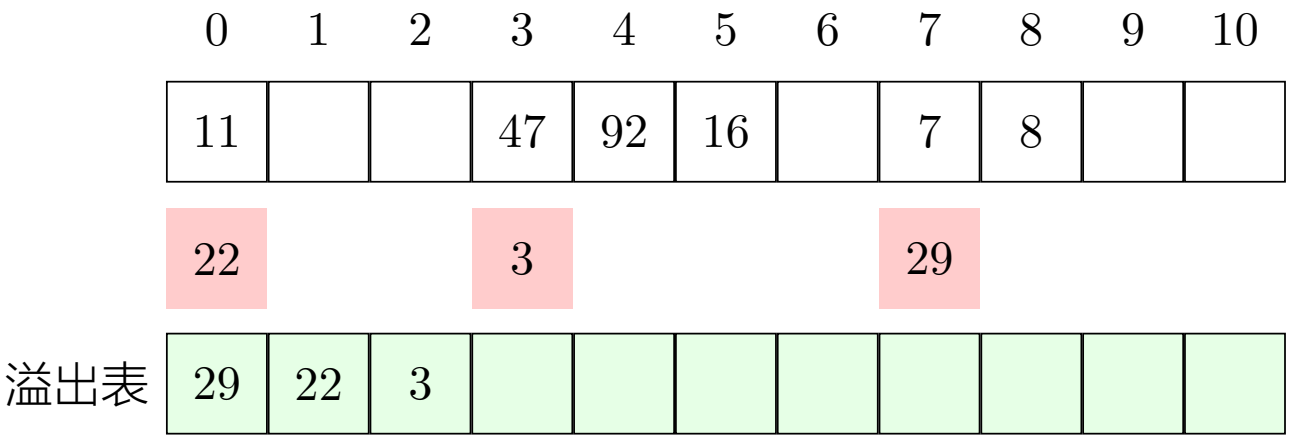
3. 拉链法

- 将所有散列地址相同的记录, 即所有同义词的记录存储在一个单链表中 (称为同义词子表), 在散列表中存储的是所有同义词子表的头指针。
- 例如, 关键码集合为 $\{47, 7, 29, 11, 16, 92, 22, 8, 3\}$, 按 $\mathrm{Hash(key)} = \mathrm{keyMOD11}$ 和拉链法得到 hash 表如下:

0	1	2	3	4	5	6	7	8	9	10
11			47	92	16		7	8		
22			3				29			

4. 建立公共溢出区

- 另分配一个溢出表。只要产生地址冲突, 都将存入溢出表。
- 例如, 关键码集合为 $\{47, 7, 29, 11, 16, 92, 22, 8, 3\}$



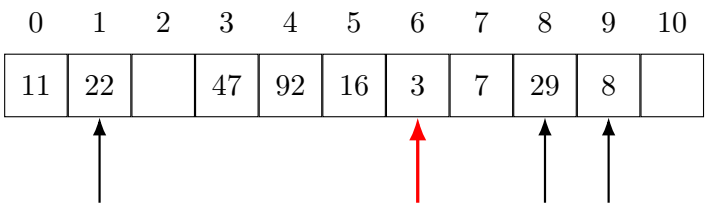
哈希表的查找分析

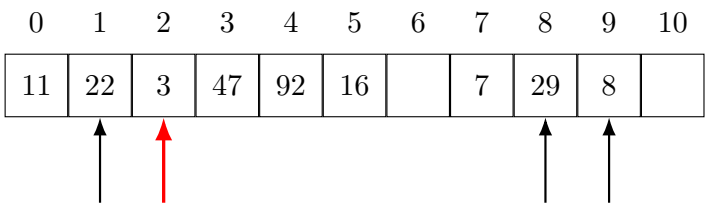
- 虽然 hash 表中关键字与记录存储位置之间建立了直接映像, 但由于冲突的存在使得 hash 表的查找仍然是一个给定值和关键字比较的过程。因此, 仍需以平均查找长度衡量 hash 表查找效率。
- 查找效率取决于产生冲突的多少, 产生的冲突少, 需要进行关键字比较次数就少, 查找效率就高, 反之则效率较低。
- 影响产生冲突多少的因素也就是影响查找效率的因素。

哈希表的查找分析 (cont.)

影响产生冲突多少的因素 (即影响查找效率的因素):

- 1. 哈希函数是否均匀
  - 一般认为所选的哈希函数是“均匀的”，忽略此类影响
- 2. 处理冲突的方法
  - 线性探测法 $\mathrm{ASL} = (5\times1 + 3\times2 + 1\times4)/9 = 5/3$
  - 二次探测法 $\mathrm{ASL} = (5\times1 + 3\times2 + 1\times2)/9 = 13/9$





- 3. 哈希表的装填因子

哈希表的装填因子

$$\alpha = \frac{\text{填入表中的元素个数}}{\text{哈希表的长度}}$$

- α 是哈希表装满程度的标志因子。α 越大, 产生冲突的可能性就越大; 反之则冲突的可能性越小。哈希表的平均查找长度是装填因子 α 的函数

处理冲突的方法	查找成功时的平均长度	查找失败时的平均长度
线性探测法	$S_{nl} \approx \frac{1}{2}(1+\frac{1}{1-\alpha})$	$U_{nl} \approx \frac{1}{2}(1+\frac{1}{(1-\alpha)^2})$
二次探测法与双哈希法	$S_{nr} \approx -\frac{1}{\alpha}\ln(1-\alpha)$	$U_{nr} \approx \frac{1}{1-\alpha}$
拉链法	$S_{nc} \approx 1+\frac{\alpha}{2}$	$U_{nc} \approx \alpha + e^{-\alpha}$

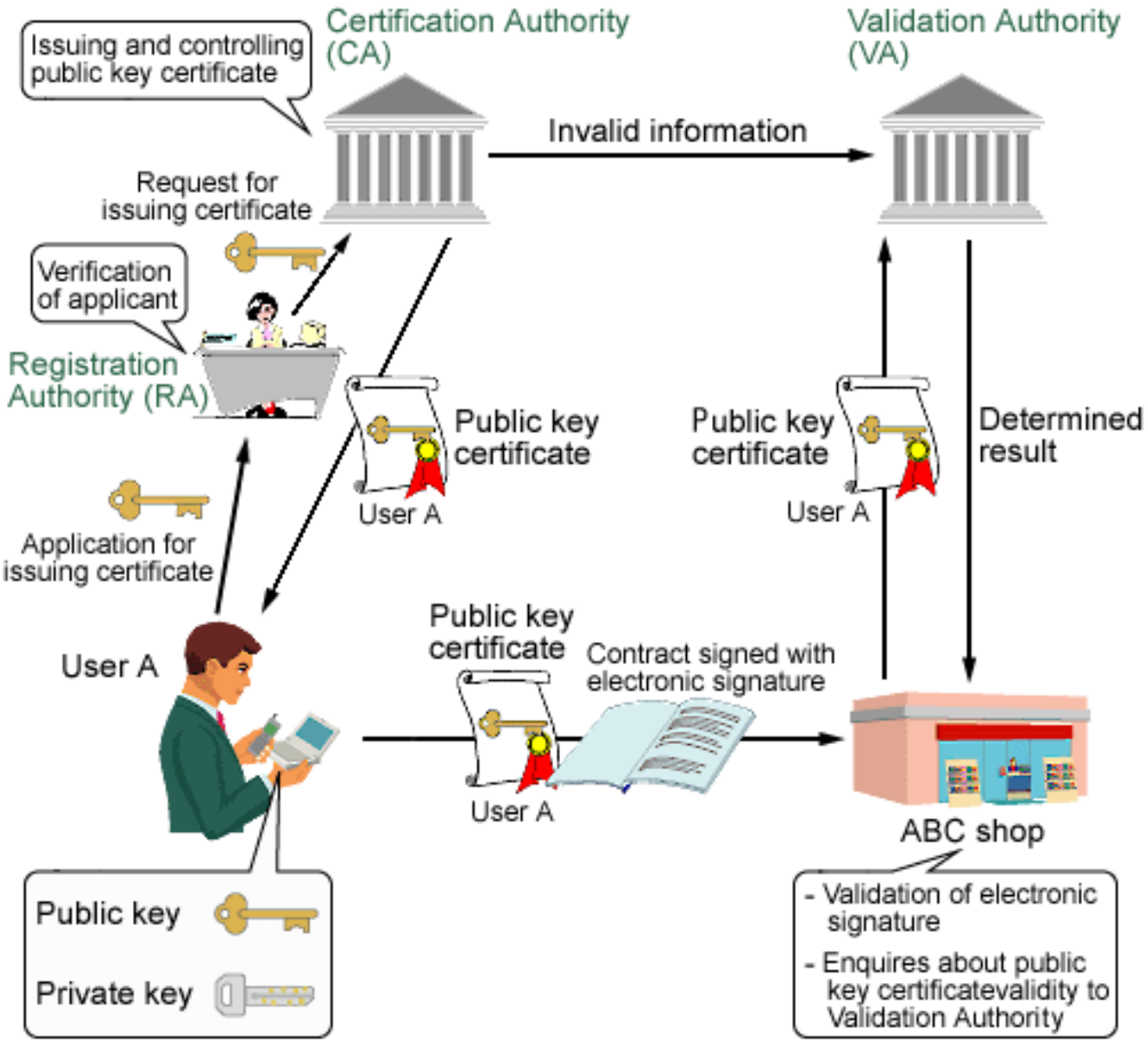
哈希算法与数字安全

- 哈希密码是对口令进行一次性的加密处理而形成的杂乱字符串。这个加密的过程被认为是不可逆的, 也就是说, 人们认为从哈希串中是不能还原出原口令的。
- 哈希算法应用于数字安全的几乎所有方面。Hash 函数的种类很多,MD5 和 SHA-1 算法是目前广泛应用于金融、证券等电子商务领域的关键技术。后者在 1994 年便为美国政府采纳, 是目前美国政府广泛应用的计算机密码系统。
- 指纹是人们身份惟一和安全的标志。在网络安全协议中, 使用 Hash 函数能够产生理论上独一无二的电子文件的" 指纹", 形成" 数字手印"。按照理想安全要求, 原始信息即使只改变一位, 其产生的指纹也会截然不同。即使调用全球的计算机, 也难以找到两个相同的数字手印, 因此能够保证数字签名无法被伪造。

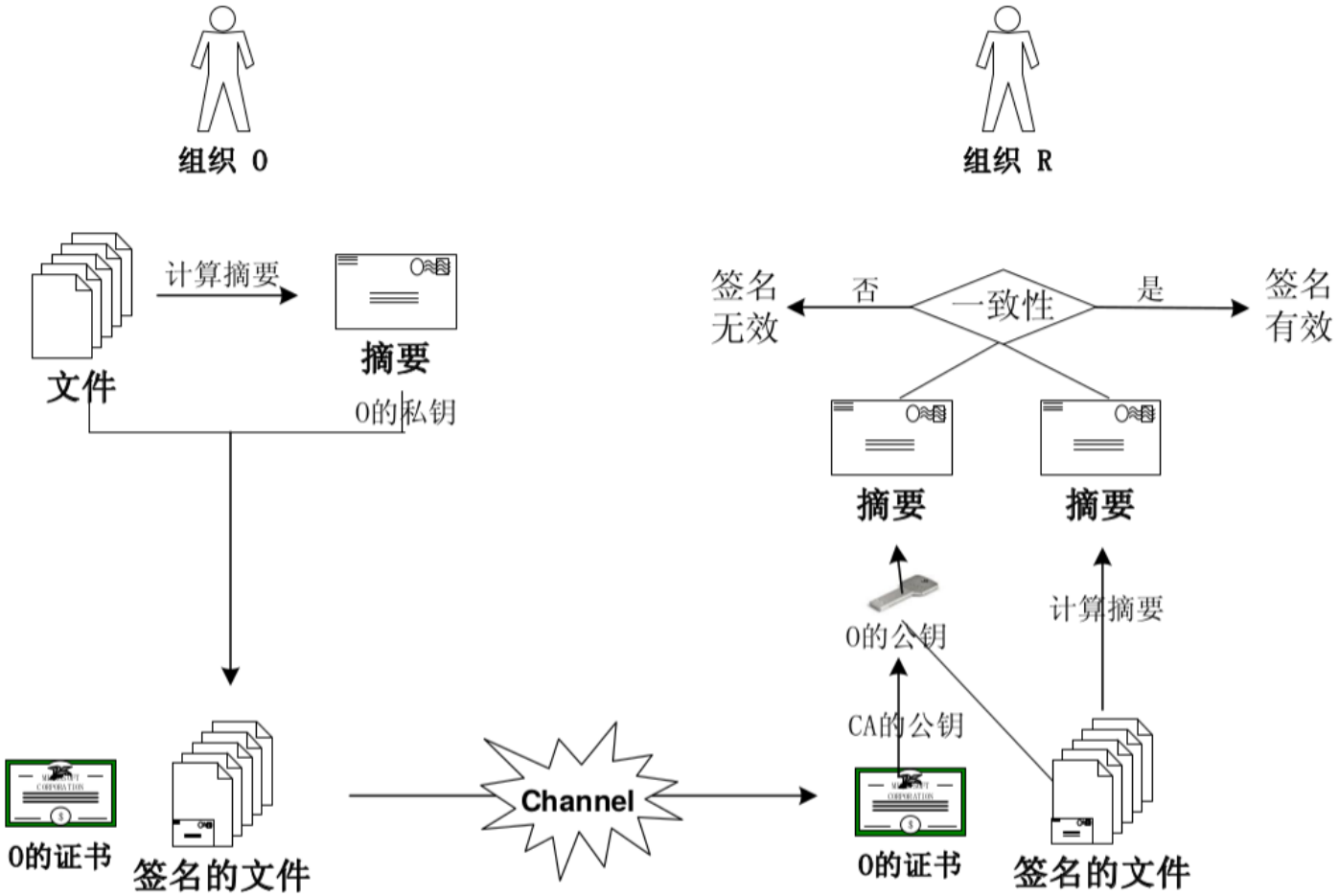
Message Digest Algorithm- MD5

- MD5 是计算机安全领域广泛使用的一种散列函数, 用以提供消息的完整性保护。在 90 年代初由 MIT 和 RSA Data Security Inc 开发, 经 MD2、MD3 和 MD4 发展而来。它的作用是让大容量信息在用数字签名软件签署私人密钥前被变换成一定长的大整数。MD5 广泛用于各种软件的密码认证和钥匙识别上, 通俗的讲就是人们讲的序列号。
- 有学者考虑过在散列中暴力搜寻冲突的函数, 他们猜测一个被设计专门用来搜索 MD5 冲突的机器 (在 1994 年的制造成本大约是 1 百万美元), 可以平均每 24 天就找到一个冲突。但尚不足以成为 MD5 的在实际应用中的问题。并且, 由于 MD5 算法的使用是 free 的, 所以在一般的情况下, MD5 怎么都应该算得上是很安全的了。

PKI


Issuing and controlling public key certificate
Certification Authority (CA)
Validation Authority (VA)
Invalid information
Request for issuing certificate
Verification of applicant
Registration Authority (RA)
Public key certificate
User A
Public key certificate
User A
Determined result
Application for issuing certificate
User A
Public key certificate
Contract signed with electronic signature
User A
ABC shop
Public key
Private key
- Validation of electronic signature
- Enquires about public key certificatevalidity to Validation Authority

基于公钥体系 PKI 的数字签名



- 散列算法的用途不是对明文加密，让别人看不懂，而是通过对信息摘要的比对，防止对原文的篡改。

MD5 的安全性

- 2004 年 8 月, 在美国加州圣芭芭拉召开的国际密码大会上, 王小云首次宣布了她及她的研究小组近年来的研究成果——对 MD5、HAVAL-128、MD4 和 RIPEMD 等著名密码算法的破译结果。
  - 研究发现, 不同的数据能够产生相同的 Hash 值
  - 找到了的强无碰撞
- 强无碰撞是指能找到相同的摘要信息, 但无法篡改和伪造出有意义的明文。
- 弱无碰撞是对给定的消息进行运算得出相同的摘要信息, 也就是说你可以控制明文的内容
- 理论上讲, 电子签名可以伪造