查找技术 2

Tan Yiqing

2025 年 12 月 8 日

1 树表查找技术 (续)

1.1 最小平衡二叉树的分类

根据插入结点的位置不同，最小平衡二叉树的失衡类型可分为四种：

1. LL 型：在某结点的左子树的左子树上插入结点，导致该结点失衡。
2. RR 型：在某结点的右子树的右子树上插入结点，导致该结点失衡。
3. LR 型：在某结点的左子树的右子树上插入结点，导致该结点失衡。
4. RL 型：在某结点的右子树的左子树上插入结点，导致该结点失衡。

1.2 旋转调整方法

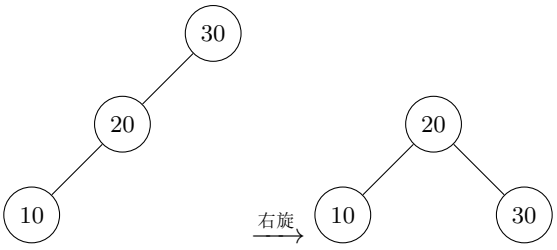
为了恢复最小平衡二叉树的平衡性，需要对失衡的子树进行旋转调整。
先介绍一些调整方法：

1. 右旋操作：假设以某结点为根
   (a) 将其左子结点-> 新的根
   (b) 该结点-> 新根的右子结点
   (c) 原左子结点的右子结点-> 该结点的左子结点
2. 左旋操作：假设以某结点为根
   (a) 将其右子结点-> 新的根
   (b) 该结点-> 新根的左子结点
   (c) 原右子结点的左子结点-> 该结点的右子结点

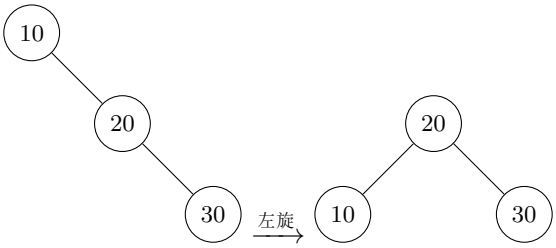
所以对于每一种失衡类型，旋转调整的方法如下：

1. LL 型失衡：对失衡结点进行右旋操作。
2. RR 型失衡：对失衡结点进行左旋操作。
3. LR 型失衡：先对失衡结点的**左子结点**进行左旋操作，然后对**失衡结点**进行右旋操作。
4. RL 型失衡：先对失衡结点的**右子结点**进行右旋操作，然后对**失衡结点**进行左旋操作。

1. LL 型失衡（右旋）
插入顺序：$30, 20, 10$，插入 10 后 30 失衡。

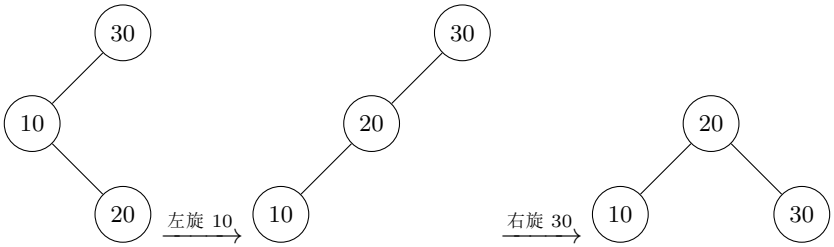


2. RR 型失衡（左旋）
插入顺序：$10, 20, 30$，插入 30 后 10 失衡。



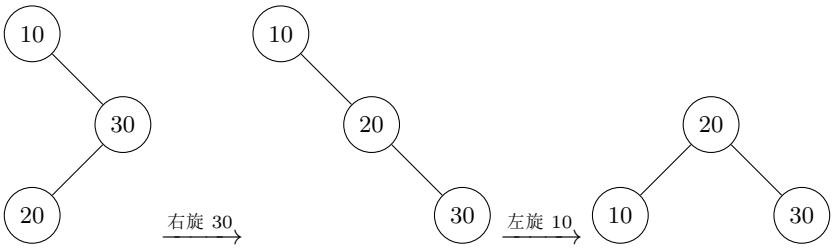
3. LR 型失衡（先左旋再右旋）

插入顺序：$30, 10, 20$，插入 20 后 30 失衡。



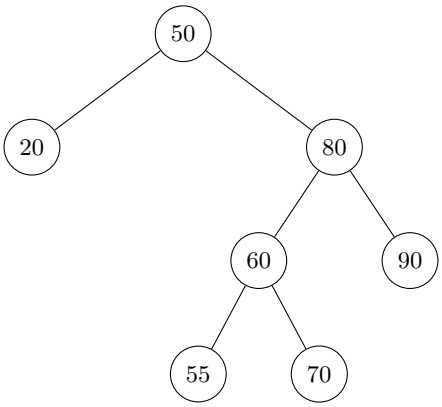
4. RL 型失衡（先右旋再左旋）

插入顺序：$10, 30, 20$，插入 20 后 10 失衡。



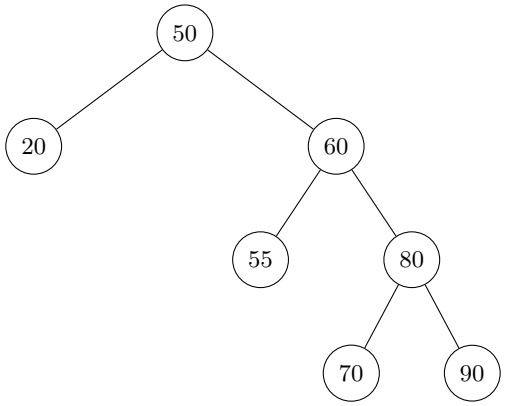
5. 复杂 RL 型失衡示例（包含子树移动）

初始树包含结点：$50, 20, 80, 60, 90, 55$。插入结点 70 后，根结点 50 失衡。失衡路径为 $50 \to 80 \to 60$，属于 RL 型。

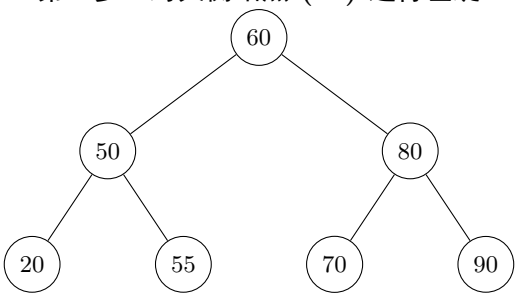


插入 70，50 失衡 (RL 型)

第一步：对失衡结点的右子结点 (80) 进行右旋



第二步：对失衡结点 (50) 进行左旋



1.3 构造平衡二叉树

1.3.1 基本思想

构造平衡二叉树（AVL 树）时，每插入一个结点，需保证整棵树的平衡性。其基本思想如下：

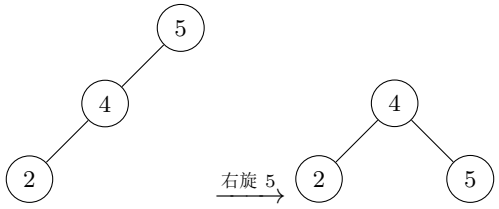
1. 插入新结点后，从插入点向上回溯，依次更新各祖先结点的平衡因子（BF）。若所有结点的平衡因子绝对值都不超过 1，则树仍然平衡，无需调整。
2. 若某个祖先结点的平衡因子绝对值超过 1，说明树已失衡。此时，找到距离插入点最近的、平衡因子绝对值大于 1 的结点（即“最小不平衡子树”的根结点）。
3. 根据新插入结点与该根结点之间的关系，判断属于哪种失衡类型（LL、RR、LR、RL），并进行相应的旋转调整（单旋或双旋），使该子树恢复平衡，从而整棵树重新成为平衡二叉树。

1.3.2 构造示例

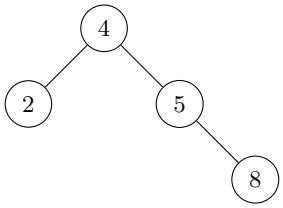
设有关键码序列 $\{5, 4, 2, 8, 6, 9\}$，构造平衡二叉树的过程如下：

1. 插入 5, 4, 2 (LL 型失衡)

插入 2 后，结点 5 失衡 (LL 型)，需对 5 进行右旋。

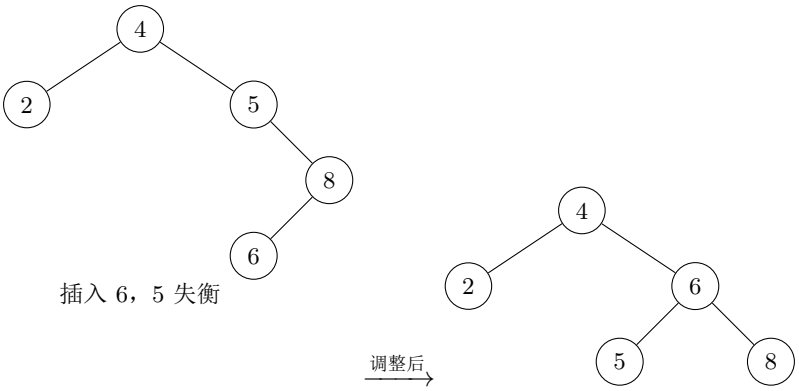


2. 插入 8 (平衡)



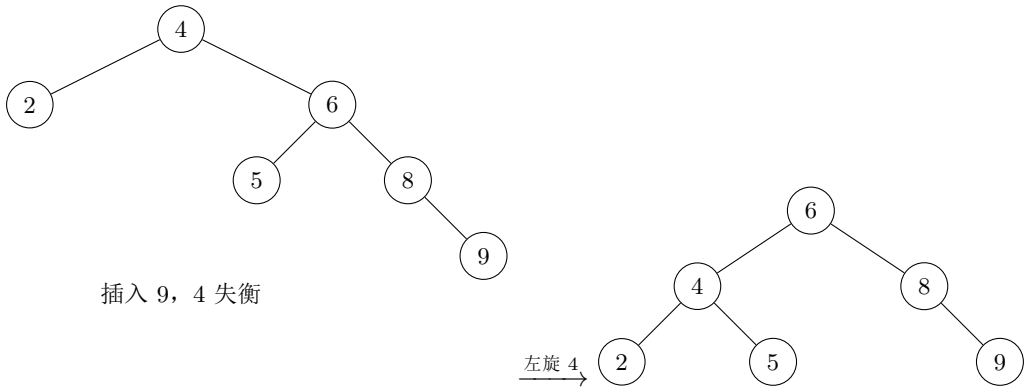
3. 插入 6 (RL 型失衡)

插入 6 后，结点 5 失衡 (RL 型：5 右子树的左子树插入)。需先对 8 右旋，再对 5 左旋。



4. 插入 9 (RR 型失衡)

插入 9 后，根结点 4 失衡 (RR 型：4 右子树的右子树插入)。需对 4 进行左旋。

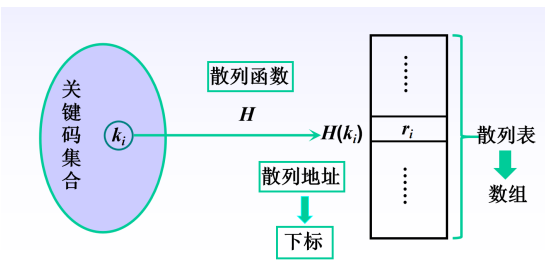


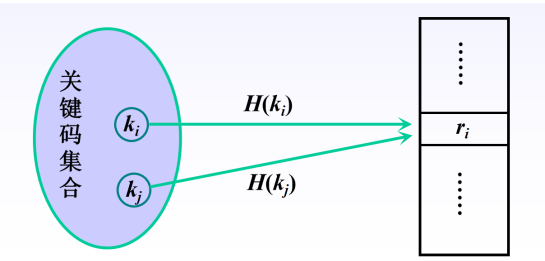
2 散列表的查找技术

基本思想：不用比较，在存储位置和关键码之间建立一个确定的对应关系，通过关键码直接确定存储位置

2.1 散列与其相关定义

1. 散列的基本思想：在记录的存储地址和它的关键码之间建立一个确定的对应关系。不经过比较，一次读取就能得到所查元素。
2. 散列表：采用散列技术将记录存储在一块连续的存储空间中，这块连续的存储空间称为散列表。
3. 散列函数：将关键码映射为散列表中适当存储位置的函数。
4. 散列地址：由散列函数所得的存储地址。
5. 冲突：对于两个不同关键码 k_i k_j，有 $H(k_i)$ $H(k_j)$，即两个不同的记录需要存放在同一个存储位置,k_i 和 k_j 相对于 H 称做同义词。





2.2 散列技术的特点

1. 散列既是一种查找技术，也是一种存储技术。
   (a) 但只是通过记录的关键码定位该记录，没有完整地表达记录之间的逻辑关系。
   (b) 所以，散列主要是面向查找的存储结构。
2. 缺点：
   (a) 散列技术一般不适用于允许多个记录有同样关键码的情况。在散列表中，不可能找到最大或最小关键码的记录。
   (b) 散列方法也不适用于范围查找。在散列表中，不可能找到在某一范围内的记录。
3. 优点：速度快，适合回答“如果有的话，哪个记录的关键码等于待查值。”
4. 散列技术的关键问题：
   (a) 如何设计一个简单、均匀、存储利用率高的散列函数。
   (b) 如何采取合适的处理冲突方法来解决冲突。

2.3 散列函数的设计

设计原则：计算简单，函数值 (即散列地址) 分布均匀。

2.3.1 直接定址法

散列函数是关键码的线性函数，即：

$$H(key) = a * key + b$$

其中 a 和 b 为常数，key 为关键码。

举例： 关键码集合为 10, 30, 50, 70, 80, 90，选取的散列函数为 $H(key) = key/10$，则散列表为：

$H(key)=1$		$H(key)=3$		$H(key)=5$		$H(key)=7$	$H(key)=8$	$H(key)=9$
10		30		50		70	80	90

使用情况 事先知道关键码，关键码集合不是很大且连续性较好。

2.3.2 除留余数法

散列函数为：

$$H(key) = key \mod p$$

p 一般取为素数 (或不包含小于 20 质因子的合数)，且接近散列表长度 m，这样可以产生较少的**冲突**。

举例： 自己根据上面例子想象一下。

使用情况： 最简单、最常用的构造散列函数的方法，不要求事先知道关键码的分布。

2.3.3 数学分析法

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散列地址。

举例： 关键码为 8 位十进制数，可选取其第 2、4、6 位组成散列地址 (最好没有重复数字)。

使用情况： 能预先估计出全部关键码的每一位上各种数字出现的频度，不同的关键码集合需要重新分析。

2.3.4 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散列地址（平方后截取）。

举例： 散列地址为 2 位，则关键码 1234 的散列地址为：

$$H(1234) = 1234^2 = 1522756 \Rightarrow H(1234) = 27$$

使用情况： 关键码的位数比较均匀分布时事先不知道关键码的分布且关键码的位数不是很大。

2.3.5 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。

举例： 关键码为 12345678，分割成 1234 和 5678 两部分，求和得到 6912，取后两位作为散列地址，即 12。

使用情况： 关键码位数很多，事先不知道关键码的分布。

2.4 冲突的处理

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。主要方法有线性探测法、二次探测法和随机探测法。

2.4.1 线性探测法

当发生冲突时，从冲突位置的下一个位置起，依次寻找空的散列地址。
对于键值 key，设 $H(key) = d$，闭散列表的长度为 m，则发生冲突时，寻找下一个散列地址的公式为：

$$H_i = (H(key) + d_i) \mod m, \quad d_i = 1, 2, \ldots, m-1$$

用开放定址法处理冲突得到的散列表叫**闭散列表**。

举例： 关键码集合为 $\{47, 7, 29, 11, 16, 92, 22, 8, 3\}$，散列表表长为 11，散列函数为 $H(key) = key \mod 11$，用线性探测法处理冲突，则散列表为：

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

注意：线性探测法容易产生**堆积**，即：在处理冲突的过程中出现的非同义词之间对同一个散列地址争夺的现象。

伪代码：

1. 计算散列地址 j;
2. 若 ht[j] 等于 k，则查找成功，返回记录在散列表中的下标；否则执行第 3 步；
3. 若 ht[j] 为空或整个散列表探测一遍，则查找失败，转 4；否则，j 指向下一单元，转 2；
4. 若整个散列表探测一遍，则表满，抛出溢出异常；否则，将待查值插入；

2.4.2 二次探测法

当发生冲突时，寻找下一个散列地址的公式为：

$$H_i = (H(key) + d_i) \bmod m \left(d_i = 1^2, -1^2, 2^2, -2^2, \ldots, q^2, -q^2 \quad 且 \quad q \le \frac{m}{2}\right)$$

举例： 关键码集合为 $\{47, 7, 29, 11, 16, 92, 22, 8, 3\}$，散列表表长为 11，散列函数为 $H(key) = key \bmod 11$，用线性探测法处理冲突，则散列表为：

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

2.4.3 随机探测法

当发生冲突时，下一个散列地址的位移量是一个随机数列，即寻找下一个散列地址的公式为：

$$H_i = (H(key) + d_i) \bmod m$$

其中 d_i 是一个随机数列，$i = 1, 2, \ldots, m-1$。随机的本质是伪随机，这里计算机产生随机是通过线性同余法：

$$\begin{cases} a_0 = d \\ a_{i+1} = (b * a_i + c) \bmod m \end{cases}$$

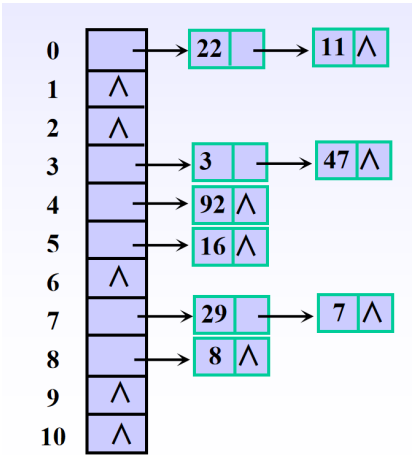
其中，d 称为随机种子。当 b、c 和 m 的值确定后，给定一个随机种子，产生确定的随机数序列。

2.4.4 拉链法

将所有散列地址相同的记录，即所有同义词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。用拉链法处理冲突构造的散列表叫做**开散列表**。开散列表不会出现堆积现象。

设 n 个记录存储在长度为 m 的散列表中，则同义词子表的平均长度为 n/m。

举例： 关键码集合为 $\{47, 7, 29, 11, 16, 92, 22, 8, 3\}$，散列表表长为 11，散列函数为 $H(key) = key \bmod 11$，用拉链法，则散列表为：



伪代码：

1. 计算散列地址 j;
2. 在第 j 个同义词子表中顺序查找；
3. 若查找成功，则返回结点的地址；否则，将待查记录插在第 j 个同义词子表的表头。

2.5 性能分析

由于冲突的存在，产生冲突后的查找仍然是给定值与关键码进行比较的过程。在查找过程中，关键码的比较次数取决于产生冲突的概率。影响冲突产生的因素有：

1. 散列函数是否均匀；
2. 处理冲突的方法；
3. 散列表的装载因子。

记：

$$\alpha = \frac{n}{m}$$

为散列表的**装载因子**，其中 n 为散列表中填入的记录数，m 为散列表的长度。散列表的平均查找长度是装填因子 α 的函数，散列表在空间换时间。

平均查找长度 / 处理冲突的方法	查找成功时	查找不成功时
线性探测法	$\frac{1}{2}(1+\frac{1}{1-\alpha})$	$\frac{1}{2}(1+\frac{1}{(1-\alpha)^2})$
二次探测法	$-\frac{1}{\alpha}\ln(1+\alpha)$	$\frac{1}{1-\alpha}$
拉链法	$1+\frac{\alpha}{2}$	$\alpha+e^{-\alpha}$

2.6 开散列表与闭散列表的对比

对比项	开散列表（拉链法）	闭散列表（开放定址法）
冲突处理方式	同义词用链表存储在同一槽	冲突时在表内寻找下一个空槽
估计容量	不需要（链表可扩展）	需要（表满即不能插入）
查找效率	效率高，受链表长度影响，装载因子大时效率下降较慢	效率低，装载因子大时查找效率急剧下降
删除操作	简单，直接删除链表结点	复杂，需特殊标记或重排
结构开销	有，需要额外存储指针	无
适用场景	适合经常插入、删除的场景	适合查找多、删除少的场景
是否易产生堆积	不产生堆积	易产生堆积