**平均查找长度ASL**

ASL表示查找一个元素平均需要经过多少次比较，具有n个元素的列表的 ASL = P1C1 + P2C2 + ... + PnCn ，Pi为查找第i个元素的概率，Ci为查找到第i个元素需要经过多少次比较

**顺序查找**

顺序查找就是从第1个元素到第n个元素顺序查找，如果每元素被查找的概率都是相等的，那么其 ASL=(n+1)/2

时间上很多应用都遵循 8/2 规则，即 80% 的访问都集中在 20% 的记录上，所以我们可以改进我们的算法，如下

var list = ['a', 'b', 'c', 'd', 'e'];

我们访问c，我们将c向前移一步

var list = ['a', 'c', 'b', 'd', 'e'];

我们再访问c，我们将c向前移一般

var list = ['c', 'a', 'b', 'd', 'e'];

我们访问b，将b向前移一步

var list = ['c', 'b', 'a', 'd', 'e'];

这样，我们访问次数越多的，他就排在越靠前

**二分查找**

二分查找其数据的排序必须要有序，如下

我想要从数组中查到10

var list = [2, 5, 6, 7, 8, 10, 11, 14, 18, 20];

我记录最低下标0的值为2，记录最高下标9的值为20，从这2个下标中选取中间下标4，其值为8

10在8和20之间，所以我以刚刚所取得下标4为最低下标，最高下标9不变，取中间下标6，其值为11

10在6和11之间，所以最低下标4不变，最高下标取刚刚所取得下标6，取中间下标5，其值为10，比较完成

二分查找的 ASL约等于log(n+1) - 1

**哈希表**

如下数组，如何才能快速的找到gee字符串

var list = ['app', 'bff', 'cee', 'def', 'eff', null, 'gee'];

你可能会使用顺序查找一个个进行查找，但我有一种更快的方法，取gee的首个字符g，g在字母表排第7，所以我给出数组下标为6（不要问我为什么是6而不是7），查看list[6]是不是gee

我们把由关键字求出下标的函数称为哈希函数，根据哈希函数得出的表称为哈希表

**哈希函数**

下面列出几种哈希函数

1. 直接地址法

如下，我们为 1 3 4 6 7 关键字构建哈希函数，我们将关键字的值直接作为数组下标

function hash(key){

    return key;

}

这种方法的缺点也是显而易见，如果关键字是 1 2 10000，那么我要分配10000个数组空间去保存这3个关键字

1. 平方取中方

这是一种比较常见的哈希函数，我们将关键字进行平方，取中间的m位，m的值取决于我们分配的哈希表的大小

如我们哈希表的大小为10，那么我们m取值为1，因为0-9刚好10个数，如果我们的关键字为11，则11的平方为121，取中间的一位，即2，则11所在的数组下标为2

1. 除余法

目前最常用的哈希函数，我们选择一个适当的数p，用关键字除以p，取余数，将余数作为我们的下标

如p取13，如果关键字为16，则 16%13 等于3，则16的数组下标为3

p的取值根据哈希表的表长决定，取小于表长的最大素数，如表长为 8，16，32，64分别对应的p为7，13，31，61

**冲突处理方法**

如下，我们用平方取中法算关键字的下标，关键字12的下标为4，关键字2的下标也为4，那么怎么处理这样的冲突呢，如下2种方法处理

闭哈希法

开哈希法

**闭哈希法**

如下哈希表，我们使用除余法，取p为7

var list = [null, null, null, null, null, null, null, null]

现在插入关键字8，8%7等于1

var list = [null, 8, null, null, null, null, null, null]

现在插入关键字15，15%7等于1，但1已经存储8了，所以我们将15存到下一位

var list = [null, 8, 15, null, null, null, null, null]

现在插入22，22%7等于1，1已被使用，下一位2也被使用，所以存到3

var list = [null, 8, 15, 22, null, null, null, null]

这就是闭哈希法，出现冲突时，我们将关键字存储位置的下标+n（n可以等于1，2，...）进行存放

查找也是如此，如查找22，22%7等于1，list[1]等于8，查找下一位，list[2]等于15，查找下一位，list[3]等于22

如果我们删除关键字，我们需要判断后面的关键字是否需要移到前一位，如我们删除8，我们需要将15和22往前移一位

**开哈希法**

开哈希法使用链表取解决冲突，如下数组

var list = [null, null, null, null, null, null, null, null]

现在插入关键字8，8%7等于1

var list = [null, key8, null, null, null, null, null, null]

var key8 = { value: 8, next: null }

现在插入关键字15，15%7等于1

var list = [null, key8, null, null, null, null, null, null]

var key8 = { value: 8, next: key15 }

var key15 = { value:15, next: null }

现在插入关键字22，22%7等于1

var list = [null, key8, null, null, null, null, null, null]

var key8 = { value: 8, next: key15 }

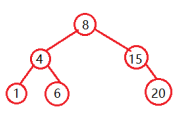
var key15 = { value:15, next: key22 }

var key22 = { value:22, next: null }

开哈希法的平均性能要优于闭哈希法

**二叉查找树**

二叉查找树左边的节点小于当前节点，右边的节点大于当前节点，如下

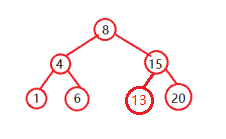


**查找**

如下，查找节点6，从根节点8开始，6 < 8，进入8左节点，6 > 4，进入4得右节点，6 = 6，查找到节点6

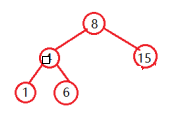
**插入**

如下，插入节点13，从根节点开始，13 > 8，进入右节点，13 < 15，进入左节点，但15没有左节点，将13插入到15得左节点

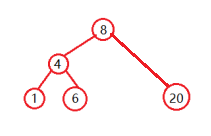


**删除**

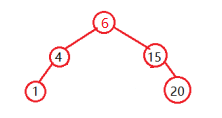
如果删除得节点无子节点，则直接删除，如删除20节点



如果删减节点有一个子节点，则子节点顶替当前节点，如删除节点15

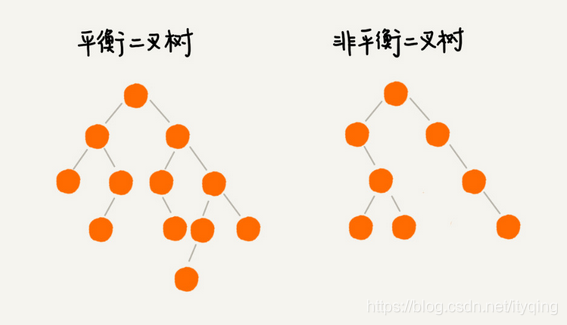


如果删除节点有2个子节点，则取当前节点左子树最右边的节点顶替当前节点，如果最右边的节点有左子树，则左子树顶替最右边节点的位置，如删除8，8左子树最右边的节点是6，则6替换掉8，如果6有左子树5，则5应该连接到4，但当前示例6没有左子树



**平衡二叉查找树（AVL树）**

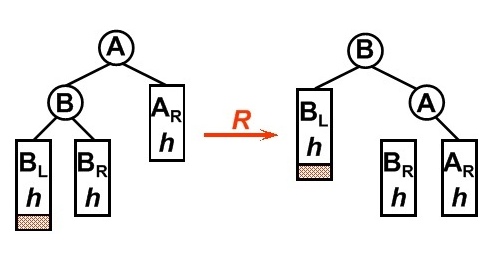
任意一节点的左右子树高度差步超过1，称为AVL树



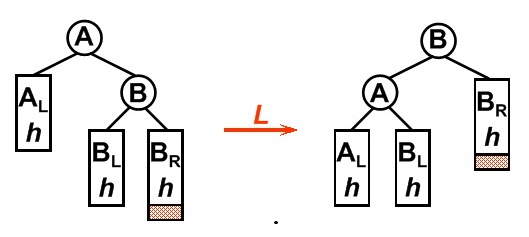
**平衡操作**

任何的插入和删除都可能会使二插入失去平衡，我们进行如下平衡操作调整二叉树

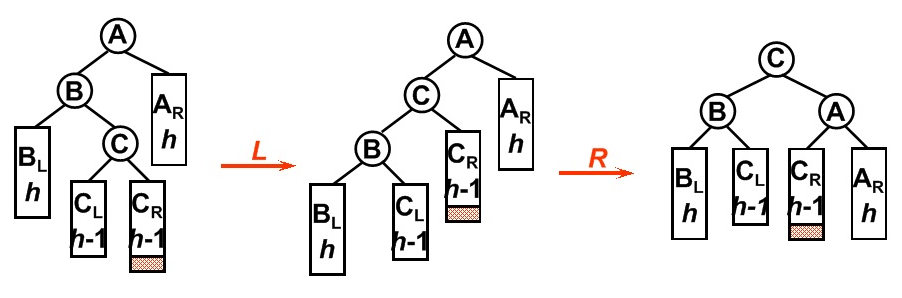
如下BL高度+1时得调整



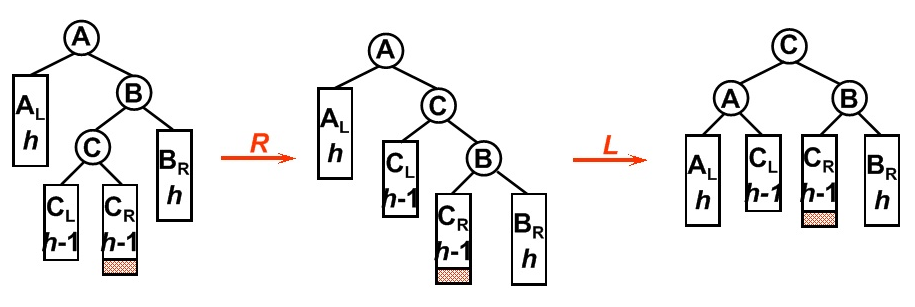
如下BR高度+1时的调整



如下CR高度+1时的调整



如下CR高度+1时的调整



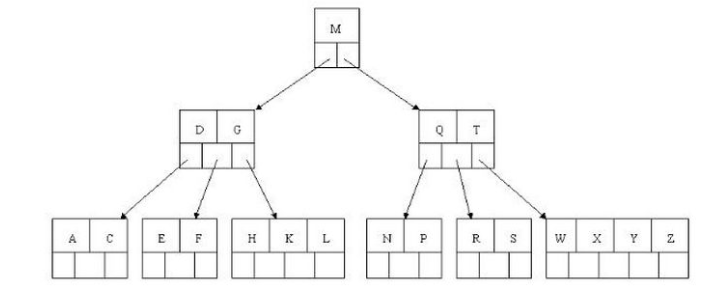
**B树**

参考文档：https://zhuanlan.zhihu.com/p/27700617

二叉树适用于内存数据的查找，如果数据在硬盘上，多次的数据读取会降低速度

**B树定义**

如下是一颗5阶B树



m阶B树定义如下

1. 如果根节点有子节点，则根节点至少有2个子节点，如上根节点（M）具有节点（DG）和节点（QT）
2. 除根节点外，每个非叶节点至少有m/2（向上取数，即5/2等于3）个子节点，如上5阶B树每个非叶节点必须至少有3个子节点
3. 每个节点至多有m个子节点，如上5阶B树每个节点至多有5棵子树
4. 除根节点外，每个节点至少有m/2 - 1（m/2向上取数）个元素，至多有m-1个元素，如上5阶B数每个节点至少有2个元素，至多有4个元素，如最左边的节点具有2元素A和C，最右边的节点具有4个元素W，X，Y，Z
5. 所有的叶节点都位于同一层上，如下所有叶节点位于第3层

**查找**

如上，要查找E，E < M，则往第0个索引，进入（D，G）节点，D < E < G，所以往第1个索引，进入节点（E，F），顺序查找节点（E，F），找到E

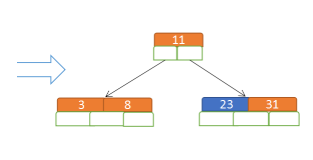
**插入**

定义一个5阶树（平衡5路查找树），现在插入如下数据 3、8、31、11、23、29、50、28

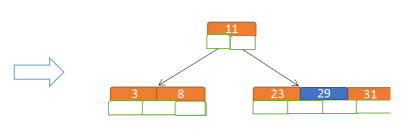
先插入 3、8、31、11



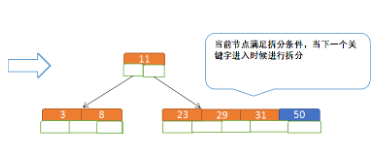
再插入23，插入23时元素数超过了5-1即4，这时候我们需要拆分，我们将23插入到节点种，然后取中间元素作为根，将节点分为3个



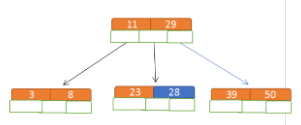
再插入29，29大于11，往右边插



再插入50



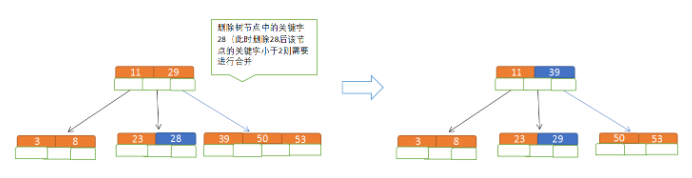
再插入28，插入28时右边的节点已经超过4个了，需要拆分，插入28，取中间元素，将节点拆分为左右2个节点，将中间元素移到父节点



如果我们一直这样插入下去，这根节点也会超过4个元素，这时候像上面一样我们将根节点拆分成3个节点，这样数的高度就变成了3层

**删除**

当元素数小于最小元素数时先从子节点取，子节点没有符合条件时就向向父节点取，而父节点则向要删除的节点的右兄弟节点取最左边一位元素，如下，删除元素28



如果要删除的节点的右兄弟节点的元素数是最小元素数，则父元素不用取了，直接合并这2个节点