JAVA中常用的数据结构和算法

1. 栈，先进后出，操作只有push和pop两种。
2. 队列，先进先出，只允许在front和rear操作。受限制的特性表。
3. 链表：
   1. 单链表
   2. 双链表
   3. 循环单链表
4. 散列表：

散列函数构造法。

* 1. 直接定址法：取线性函数
  2. 数字分析法
  3. 平方取值法：取关键字平方后的中间几位
  4. 折叠法：将关键字分割，按位相加
  5. 除留余数法：取mod值
  6. 随机数法：取关键字的随机函数

1. 排序二叉树，左小右大
   1. 删除时要考虑找哪个作为新的节点
2. AVL树
   1. AVL树又是平衡二叉搜索树：这棵树上的所有节点的左右子树高度差必须在1以内。就是带了平衡功能的二叉搜索树，强平衡树。
   2. 插入破化平衡后的调整工作：
      1. LL（向左子树的左孩子插入）：这时需要右旋操作，以左子树为轴，把原根节点放到左子树右边，左子树根节点作为新的根节点并且把原左子树的右孩子放到现在右子树的左孩子。
      2. RR：左旋操作，和上面类似。
      3. LR：向左子树的右孩子插入：以左子树的右孩子为轴，先左旋，再右旋。
      4. RL：同理先右旋，再左旋。

这里需要注意的是前两种情况和后两种情况的旋转区别在于轴不一样。

1. 删除操作：

删除方法为deletenodefromtree：

递归调用左右子树的该方法，并在调用后做平衡性检验。如果左小于右，先右旋再左旋。如果左大于右，先左旋再右旋。

调到找到该节点，如果该节点为叶子，直接删，有左或右子树，就删了再加上。都有，就找到左子树最大值，或者右子树最小值，删除。

然后递归返回，进行平衡性处理。

1. 红黑树
   1. 红黑树是一个二叉树，还满足五个特性：

每个结点是红色或者黑色的。

根结点是黑色的。

每个空结点（NULL/NIL）是黑色的。（这里将空结点作为一个特殊的结点对待，设定他们必须是黑色的。）

如果一个结点是红色的，则它的左右子结点都必须是黑色的。（但黑色结点的子结点可以是黑色的。）

对任意一个结点来说，从它到空结点的所有路径必须包含相同数目的黑色结点。

1. 好处在于：弱平衡，相比于AVL树调整少，适用于插入，删除较多的场景。
2. 添加时的调整：

被插入节点必须是红色。不然违反要求五。

* + 1. 被插入的节点是根节点，直接涂黑。
    2. 被插入的节点的父亲是黑节点，什么也不做
    3. 被插入节点的父亲时红色，这代表了他一定有祖父节点（黑），也有叔叔节点。
       1. 父节点为红，叔叔节点为红：
          1. 父，叔，祖父变色。递归对祖父节点操作
       2. 父节点为红，叔叔节点为黑，当前节点是右孩子
          1. 父为新当前，右旋，原祖父变红，新当前变黑
       3. 父节点为红，叔叔节点为黑，当前节点是左孩子
          1. 先把第三种调整成第二种，再操作。

1. 删除时的调整：

一切都可以归结为删除只有一个子节点且为黑色的情况

1. B树
   1. B-Tree特性：

树中每个结点至多有 m 个孩子；

2. 除根结点和叶子结点外，其它每个结点至少有有 ceil(m / 2)个孩子；

3. 若根结点不是叶子结点，则至少有 2 个孩子（特殊情况：没有孩子的根结点，即根结点为叶子

结点，整棵树只有一个根节点）；

4. 所有叶子结点都出现在同一层，叶子结点不包含任何关键字信息(可以看做是外部结点或查询

失败的结点，实际上这些结点不存在，指向这些结点的指针都为 null)；

5. 每个非终端结点中包含有 n 个关键字信息： (n，P0，K1，P1，K2，P2，......，Kn，Pn)。其中：

1. 堆：
   1. 用数组实现的完全二叉树，一层没填满前不允许填下一层
   2. 有上浮和下沉操作