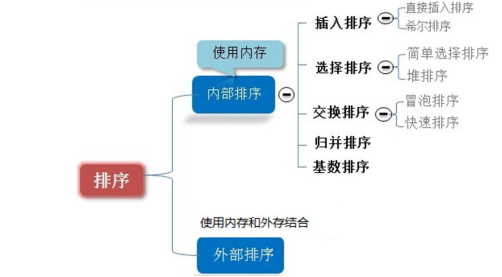
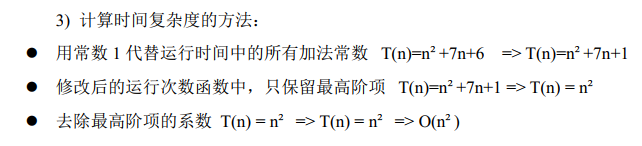
数据结构部分

1. **队列**
2. 队列是一个有序列表，可以用数组或是链表来实现。
3. 遵循先入先出的原则，需要头指针和尾指针来指引位置，尾部添加头部删除。
4. 种类：单队列和循环队列。单队列存在“假溢出”问题明明有位置却不能添加；循环队列避免了假溢出问题。
5. **链表**
6. 以节点的方式来存储是链式存储。
7. 每个节点包含data域，next域：指向下一个节点
8. 韩顺平笔记中有关于链表的面试题（倒数第k个节点、约瑟夫环问题）
9. 单循环链表、双循环链表
10. **栈**
11. 是一个先入后出的有序列表，只允许在栈顶进行插入和删除的操作
12. 入栈push,出栈pop
13. 栈的题目：综合计算器、逆波兰计算器
14. **递归**
15. 当程序执行到一个方法时，就会开辟一个独立的空间（栈），每个空间的数据（局部变量），是独立的。
16. 递归用到的：8皇后问题，汉诺塔，阶乘问题，迷宫问题，快排，归并排序，二分查找，分治算法
17. **排序算法**



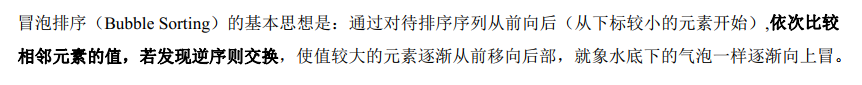
1. 时间复杂度

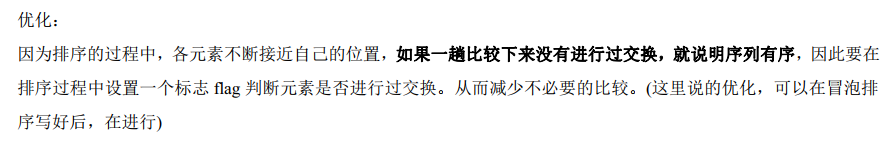


1. 常见的时间复杂度

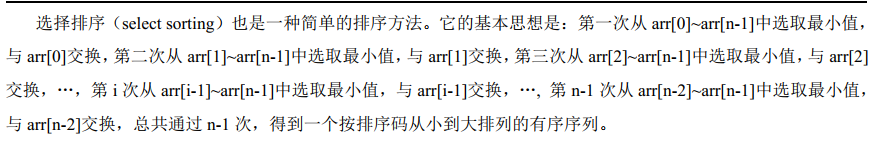
顺序结构为常数阶O（1），for循环为线性阶O（n），while循环为对数阶O（log2n），for循环嵌套着while循环为O（nlog2n）

1. 冒泡排序：

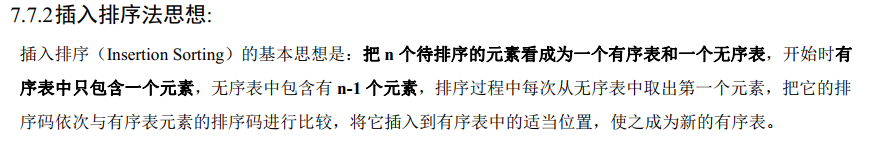




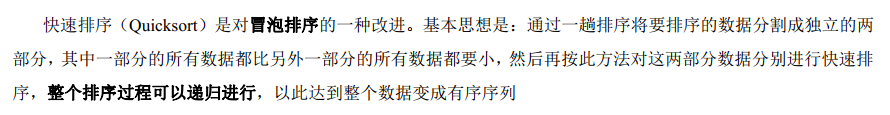
1. 选择排序：



1. 插入排序：



1. 希尔排序
2. 快速排序



1. 归并排序
2. 基数排序

**10.排序算法时间复杂度总结**：

⑴时间复杂度为O（n\*n）的排序算法：

* 冒泡排序（重要）
* 选择排序
* 插入排序
* 希尔排序

⑵时间复杂度为O（nlogn）的排序算法

* 快速排序（重要）
* 归并排序
* 堆排序

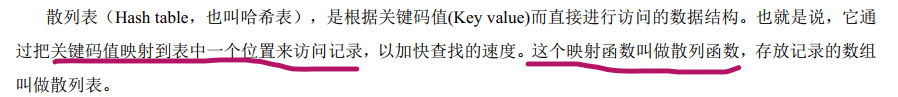
⑶时间复杂度为线性的排序算法

* 计数排序
* 桶排序
* 基数排序

1. **查找算法**
2. 顺序（线性）查找
3. 二分查找/折半查找
4. 插值查找
5. 斐波那契查找（黄金分割法）

斐波那契数列{1,1,2,3,5,8,13,21,34,55}，满足F[k]=F[k-1]+F[k-2]的性质

1. **哈希表**

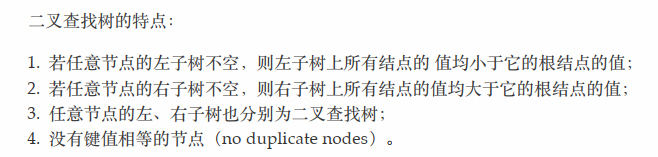


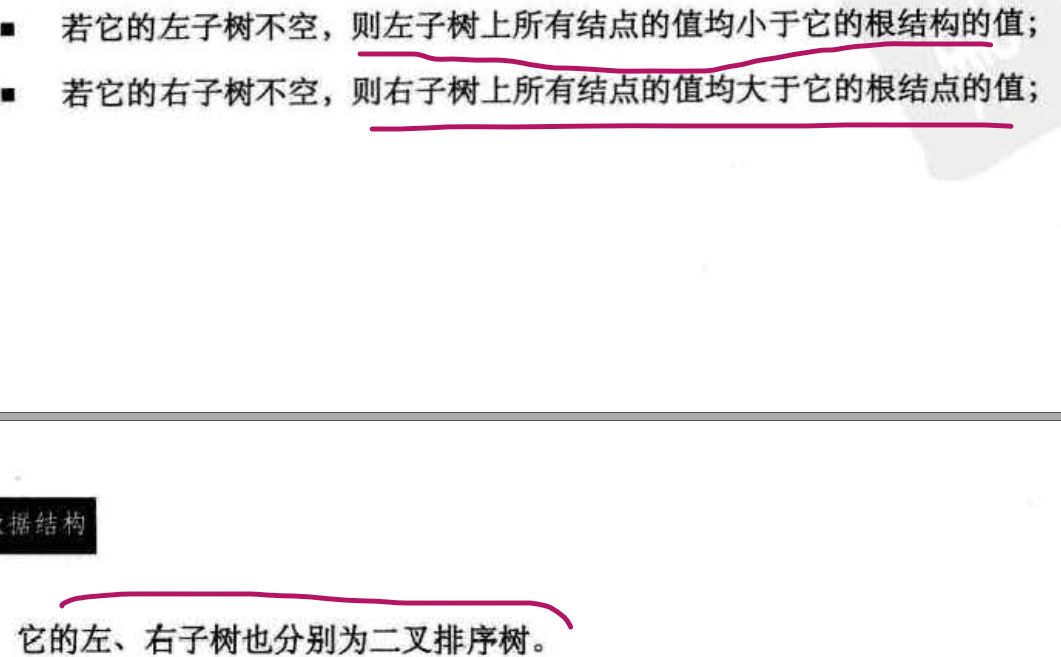
学习链接：https://blog.csdn.net/qq\_22238021/article/details/78258605

1. **集合类**
2. Set
3. List
4. Map

以上三个具体介绍见JavaGuide面试突击版里的详细介绍

1. **树**
2. **二叉查找树（BST）（二叉搜索树）**





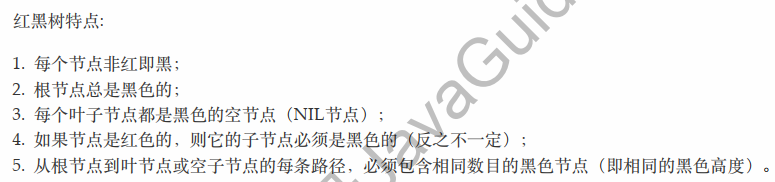
优势：查找所需的最大次数等同于二叉查找树的高度，在插入节点的时候也是通过一层层比较找到新节点适合插入的位置

缺点：多次插入新节点有可能导致不平衡，出现线性链状，导致查找困难。

由此红黑树应运而生，用来解决上述问题，红黑树是一种自平衡的二叉查找树

1. **平衡二叉树**
2. **红黑树**

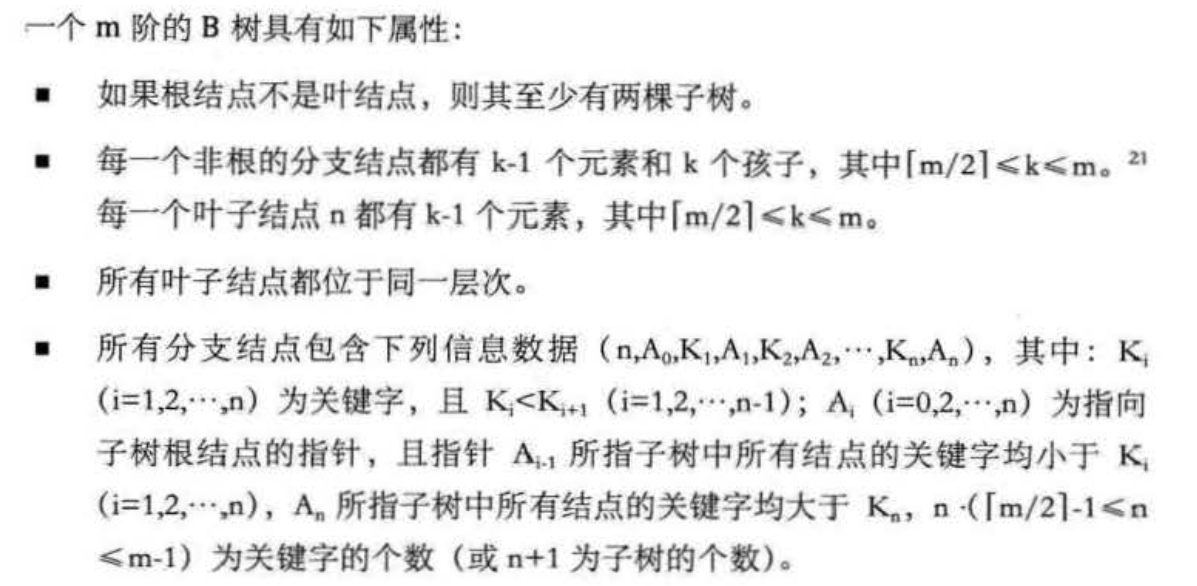
参考链接：https://juejin.im/post/5a27c6946fb9a04509096248



* 正是这些规则限制，保证了红黑树的自平衡，红黑树从根到叶子的最长路径不会超过最短路径的2倍
* 红黑树应用：TreeMap和TreeSet底层实现，JDK1.8中的HasnMap
* 红黑树通过变色和旋转来保持自身的平衡
* 红黑树里面的插入和删除的操作比较难理解，这时要注意记住一点：操作之前红黑树是平衡的，颜色是符合定义的。在操作的时候就需要向兄弟节点、父节点、侄子节点借调和互换颜色，要达到这个目的，就需要不断的进行旋转。所以红黑树的插入删除操作需要不停的旋转，一旦借调了别的节点，删除和插入的节点就会达到局部的平衡（局部符合红黑树的定义），但是被借调的节点就不会平衡了，这时就需要以被借调的节点为起点继续进行调整，直到整棵树都是平衡的。在整个修复的过程中，插入具体的分为3种情况，删除分为4种情况。

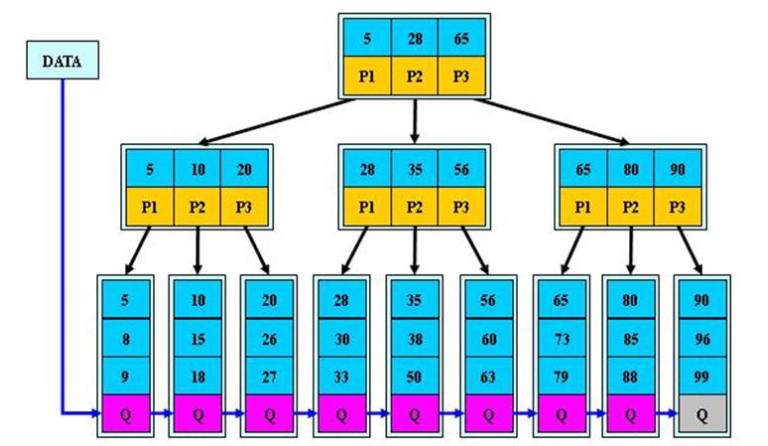
1. **B-,B+，B\*树**

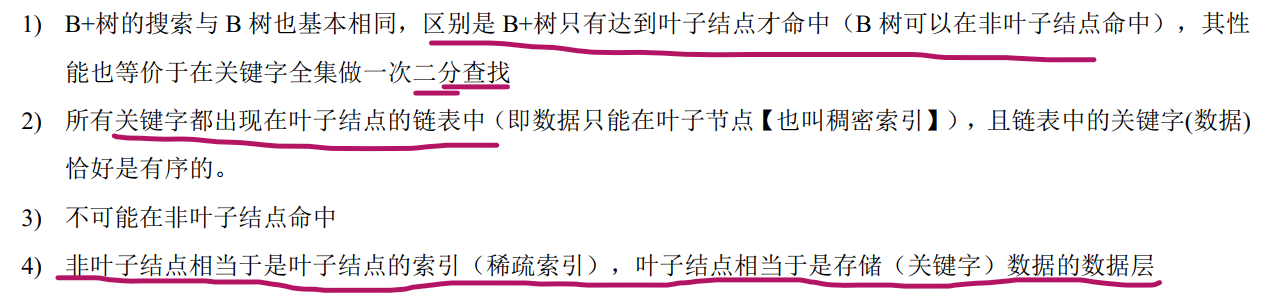
**⑴B树**是一种平衡的多路查找树，2-3树，2-3-4树都是B树的特例，结点最大的孩子数目称为B树的阶



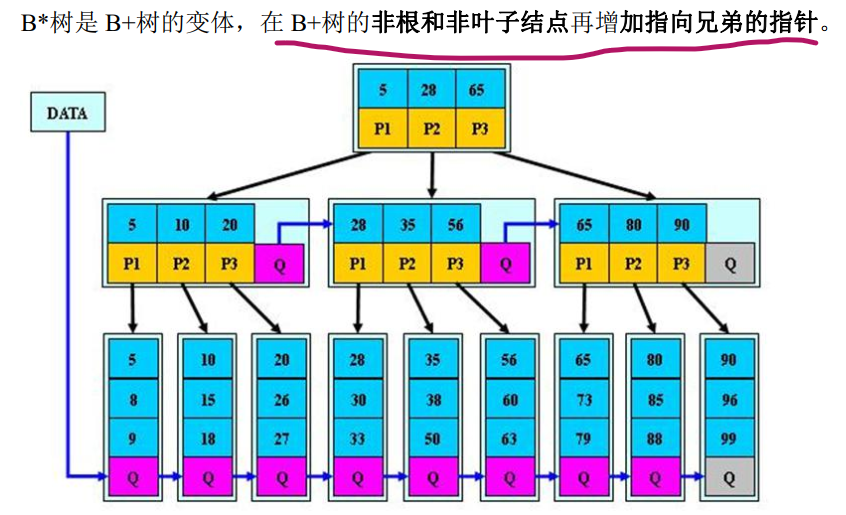


⑵**B+树**是B树的变体，也是一种多路搜索树，只在叶子结点做数据存储，所有关键字都在叶子结点，叶子结点包含了指向下一叶子节点的指针





**⑶B\*树**



1. **LSM树**
2. **二叉树的遍历**

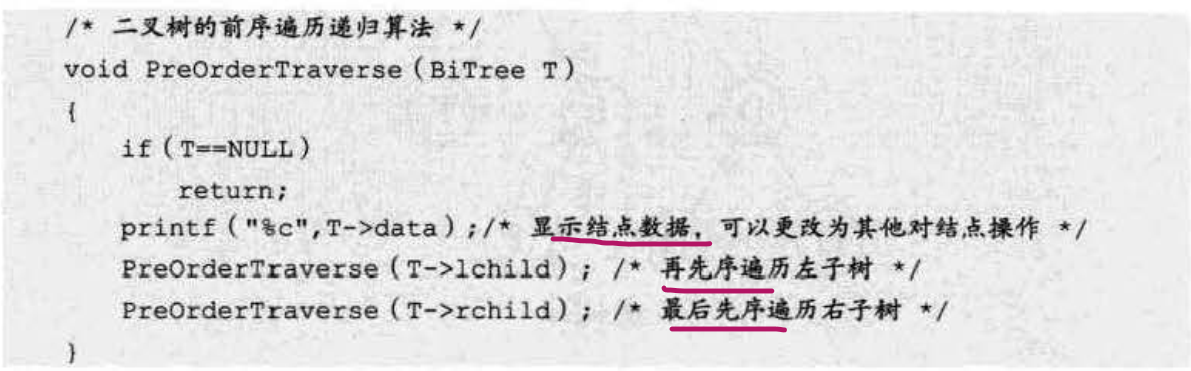
二叉树的遍历分为四种：前序遍历、中序遍历、后序遍历、层序遍历

树的遍历方式总体分为两类：深度优先搜索（DFS）、广度优先搜索（BFS）；

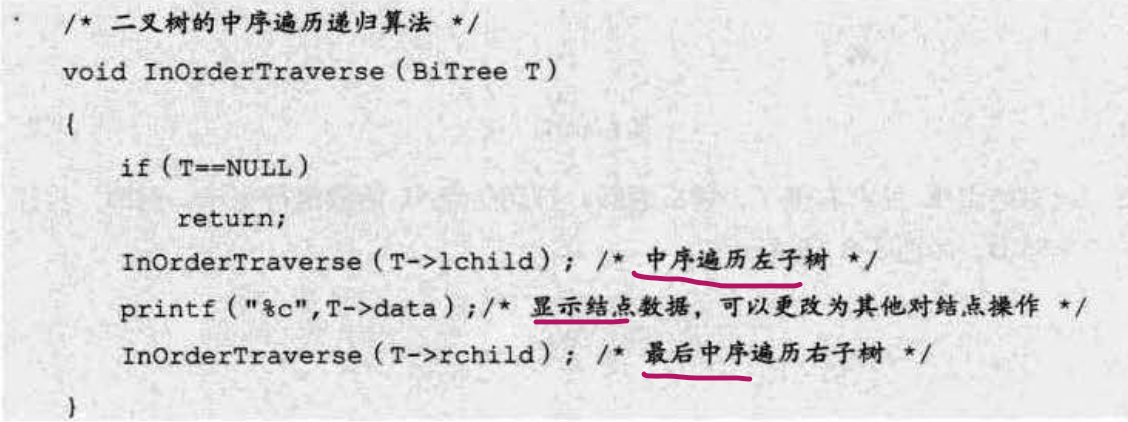
常见的 DFS ： 先序遍历、中序遍历、后序遍历；

常见的 BFS ： 层序遍历（即按层遍历）。

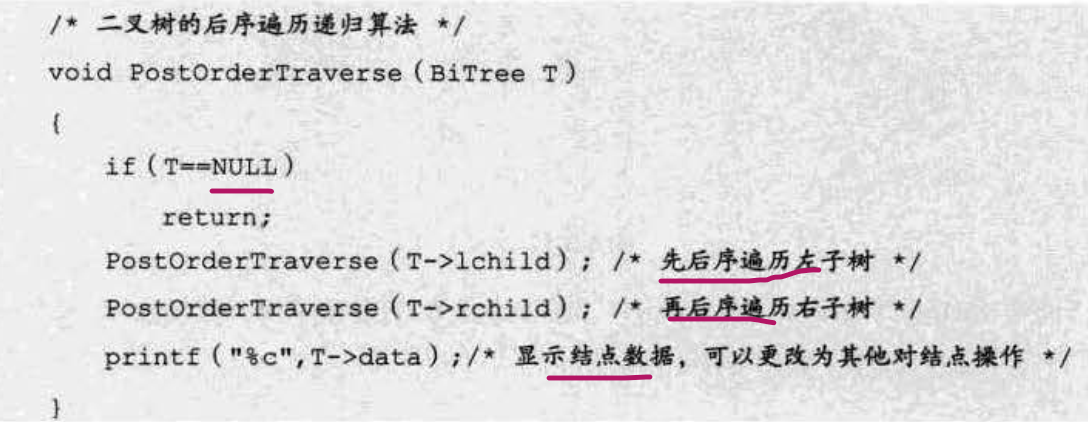
前序遍历



中序遍历



后序遍历



前三种遍历采用递归或栈实现，层序采用队列实现

层序遍历（求树的深度）

算法解析：

特例处理： 当 root​ 为空，直接返回 深度 000 。

初始化： 队列 queue （加入根节点 root ），计数器 res = 0。

循环遍历： 当 queue 为空时跳出。

初始化一个空列表 tmp ，用于临时存储下一层节点；

遍历队列： 遍历 queue 中的各节点 node ，并将其左子节点和右子节点加入 tmp；

更新队列： 执行 queue = tmp ，将下一层节点赋值给 queue；

统计层数： 执行 res += 1 ，代表层数加 111；

返回值： 返回 res 即可。

|  |
| --- |
| class Solution {  public int maxDepth(TreeNode root) {  if(root == null) return 0;  List<TreeNode> queue = new LinkedList<>() {{ add(root); }}, tmp;  int res = 0;  while(!queue.isEmpty()) {  tmp = new LinkedList<>();  for(TreeNode node : queue) {  if(node.left != null) tmp.add(node.left);  if(node.right != null) tmp.add(node.right);  }  queue = tmp;  res++;  }  return res;  }  }  作者：jyd  链接：https://leetcode-cn.com/problems/er-cha-shu-de-shen-du-lcof/solution/mian-shi-ti-55-i-er-cha-shu-de-shen-du-xian-xu-bia/  来源：力扣（LeetCode）  著作权归作者所有。商业转载请联系作者获得授权，非商业转载请注明出处。 |

**7.二叉树的存储结构（面试题考过）**

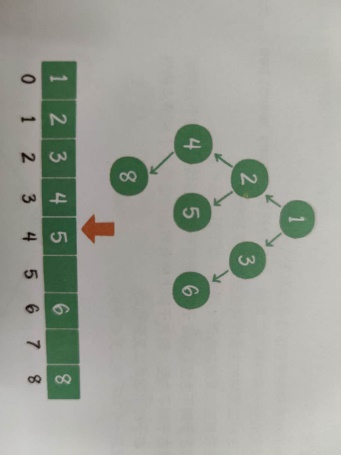
二叉树属于逻辑结构，它可以通过多种物理结构来表达

①**链式存储结构**

②**数组**

⑴链式存储结构是二叉树最直观的存储方式

每一节点包含：存储数据的data变量，指向左孩子的left指针，指向右孩子的right指针



使用数组存储时，会按照层级顺序把二叉树的节点放到数组中的对应位置上。如果某一个节点的左孩子或右孩子空缺，则数组的相应位置也空出来。（这样的设计方便在数组中定位二叉树的孩子节点和父节点）

①假设一个父节点的下标是parent，那么它的左孩子节点下标就是2\*parent+1；右孩子节点下标就是2\*parent+2.

②反过来，假设一个左孩子节点的下标是leftChild，那么它的父节点下标就是(leftChild-1)/2

**8.二叉堆**

二叉堆本质是一种完全二叉树

二叉堆实际存储在数组中

二叉堆分为最大堆和最小堆

最大堆任何一个父节点的值都大于或等于它的左、右孩子节点的值

最小堆任何一个父节点的值都小于或等于它的左、右孩子节点的值

堆的插入（从二叉树的叶子结点插入）操作是单一节点的“上浮”，堆的删除（删除堆顶）操作是单一节点的“下沉”，它们的时间复杂度都为O(logn)

**9.优先队列PriorityQueue()**

最大优先队列，无论入队顺序如何，都是当前最大的元素优先出队；

最小优先队列，无论入队顺序如何，都是当前最小的元素优先出队；

**十、BFS及DFS算法**

**十一、回溯算法**

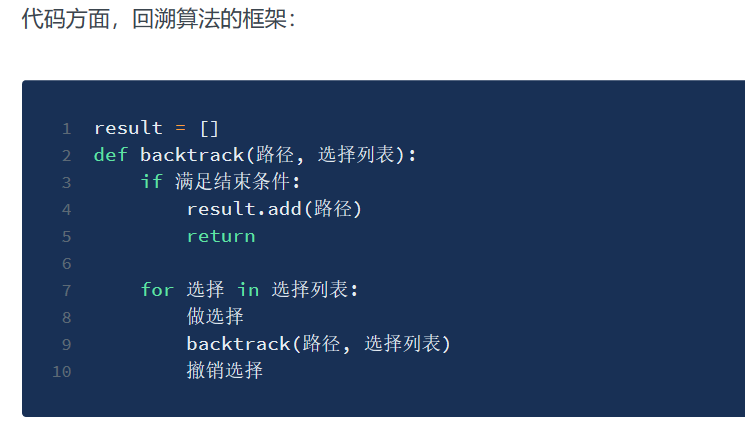
废话不多说，直接上回溯算法框架。解决一个回溯问题，实际上就是一个决策树的遍历过程。你只需要思考 3 个问题：

1、路径：也就是已经做出的选择。

2、选择列表：也就是你当前可以做的选择。

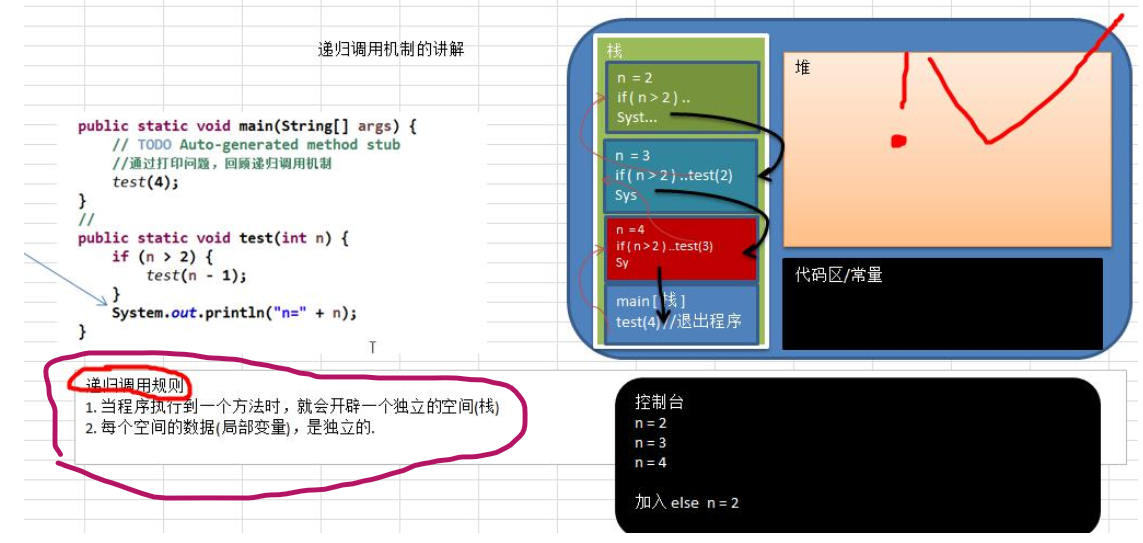
3、结束条件：也就是到达决策树底层，无法再做选择的条件。

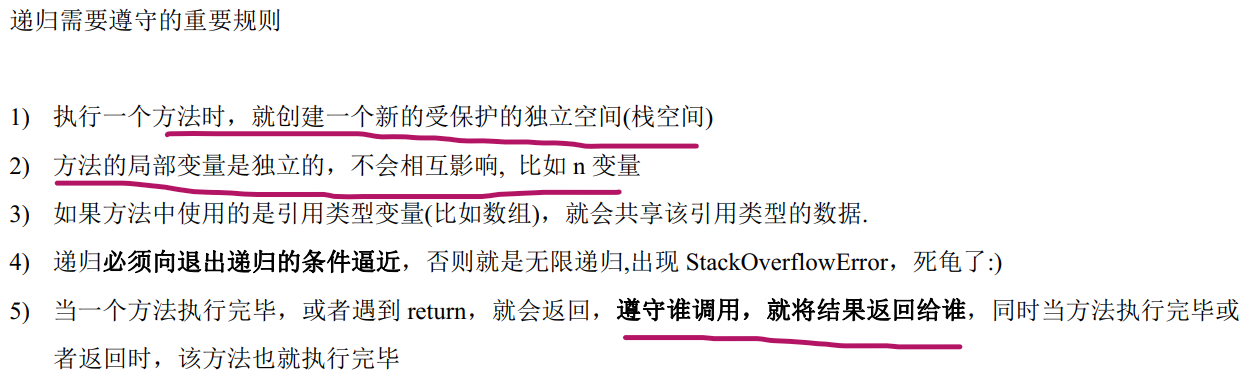
如果你不理解这三个词语的解释，没关系，我们后面会用「全排列」和「N 皇后问题」这两个经典的回溯算法问题来帮你理解这些词语是什么意思，现在你先留着印象。

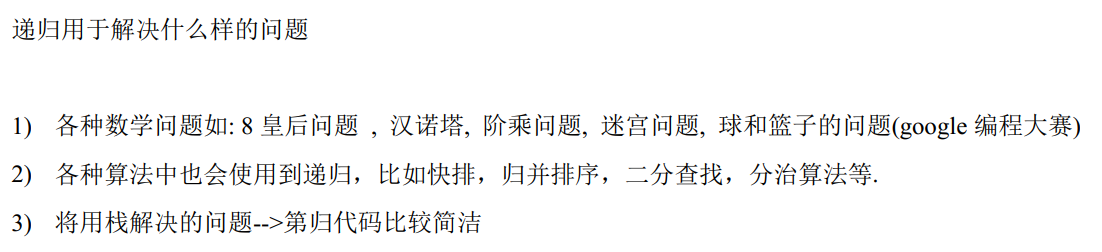
****

其核心就是 for 循环里面的递归，在递归调用之前「做选择」，在递归调用之后「撤销选择」，特别简单。

**十二、递归**







**十三、动态规划**

**⑴**动态规划问题的⼀般形式就是求最值。 动态规划其实是运筹学的⼀种最优化

⽅法， 只不过在计算机问题上应⽤⽐较多， ⽐如说让你求最⻓递增⼦序列

呀， 最⼩编辑距离呀等等

**⑵**既然是要求最值， 核⼼问题是什么呢？ 求解动态规划的核⼼问题是穷举

**⑶**⾸先， 动态规划的穷举有点特别， 因为这类问题存在「重叠⼦问题」 ， 如果

**⑷**暴⼒穷举的话效率会极其低下， 所以需要「备忘录」 或者「DP table」 来优

化穷举过程， 避免不必要的计算。⽽且， 动态规划问题⼀定会具备「最优⼦结构」 ， 才能通过⼦问题的最值得到原问题的最值。

**⑸**重叠⼦问题（备忘录）、 最优⼦结构（递归求解）⼦问题间必须互相独⽴、 状态转移⽅程（递归求解方程）就是动态规划三要素。

**⑹**时间复杂度分析：⼦问题总数x每个⼦问题的时间。列出树形结构，每一个节点为一个子问题。

**十四、KMP算法及java实现（字符串模式匹配算法）**

KMP算法，又称作“看猫片”算法（误），是一种改进的字符串模式匹配算法，可以在O(n+m)的时间复杂度以内完成字符串的匹配操作，其核心思想在于：当一趟匹配过程中出现字符不匹配时，不需要回溯主串的指针，而是利用已经得到的“部分匹配”，将模式串尽可能多地向右“滑动”一段距离，然后继续比较。

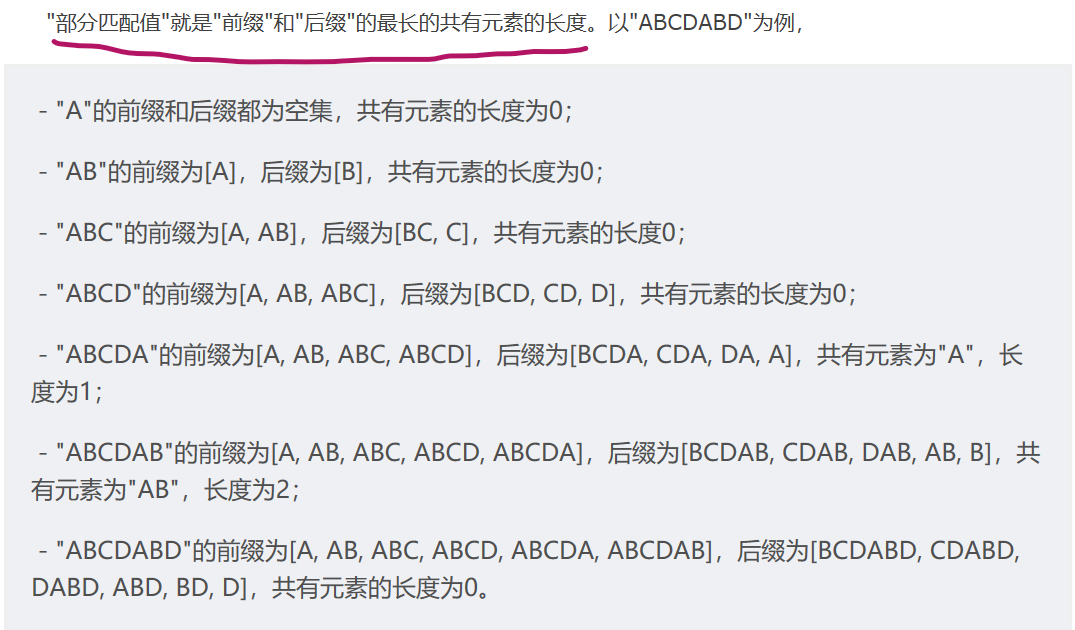
一个基本事实是，当空格与D不匹配时，你其实知道前面六个字符是"ABCDAB"。KMP 算法的想法是，设法利用这个已知信息，不要把"搜索位置"移回已经比较过的位置，继续把它向后移，这样就提高了效率。

怎么做到这一点呢？可以针对搜索词，算出一张《部分匹配表》

移动位数 = 已匹配的字符数 - 对应的部分匹配值



首先，要了解两个概念："前缀"和"后缀"。 "前缀"指除了最后一个字符以外，一个字符串的全部头部组合；"后缀"指除了第一个字符以外，一个字符串的全部尾部组合。

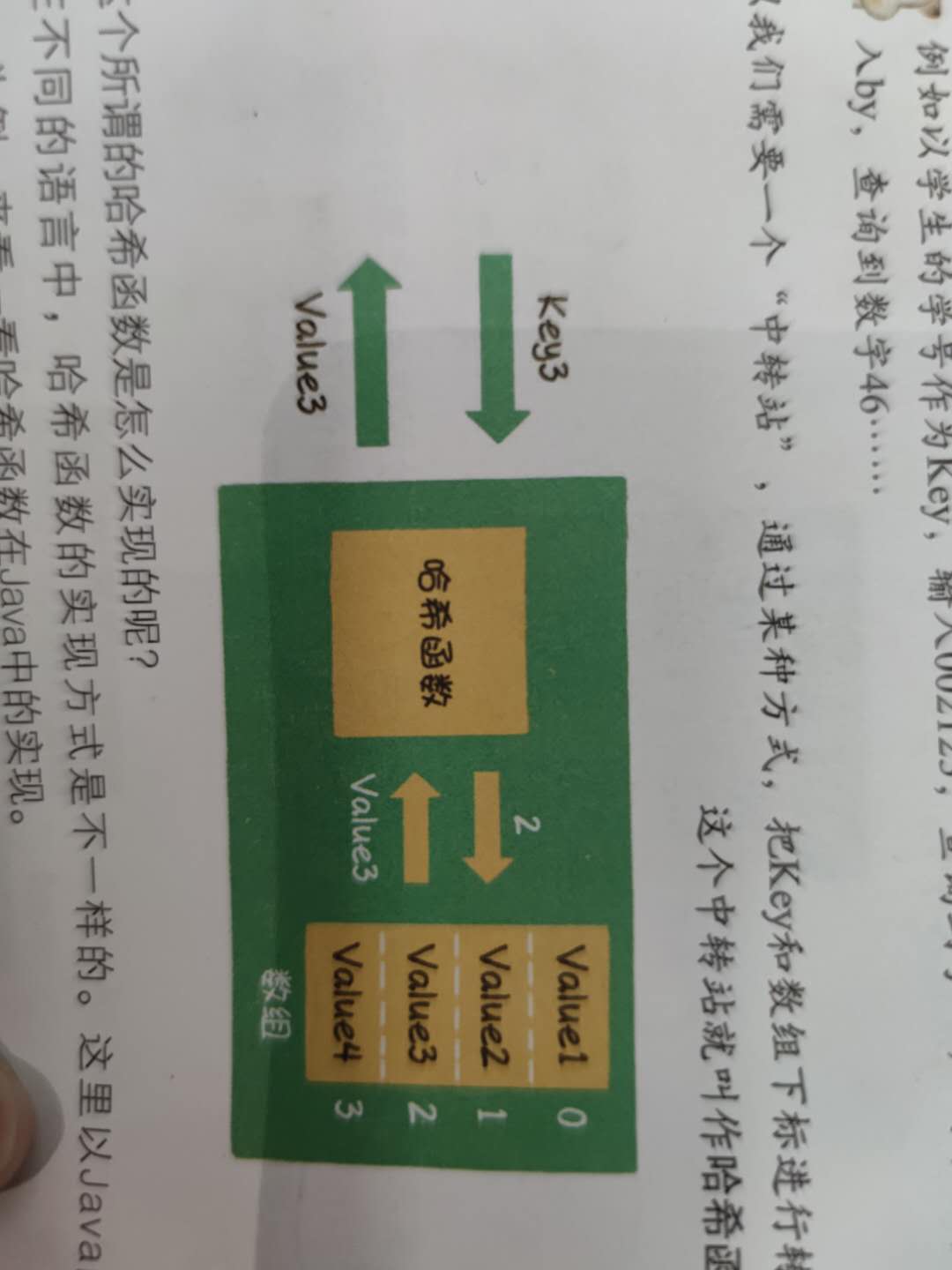


原文链接：<https://blog.csdn.net/cdnight/article/details/11935387>（讲得很好）

**十五、散列表**

1.散列表也叫作哈希表，这种数据结构提供了键（Key）和值（Value）的映射关系

2.散列表通过哈希函数实现Key和数组下标的转换，通过开放寻址法和链表法来解决哈希冲突



详解见漫画算法

**极客时间部分讲解总结**

**一、动态规划**

**1.动态规划和递归或者分支没有根本上的区别（关键看有无最优的子结构）**

**共性：找到重复的子问题**

**差异性：最优子结构、中途可以淘汰次优解**

**1.1最优子结构：问题能够分解成子问题来解决，子问题的最优解能递推到最终问题的最优解。这种子问题最优解称为最优子结构。**

**2.关键点：**

**①最优子结构：opt[n]=best\_of(opt[n-1],opt[n-2],…)**

**②储存中间状态：opt[i]**

**③递推公式（美其名曰：状态转移方程或dp方程）**

**Fib:opt[i]=opt[n-1]+opt[n-2]**

**3.动态规划讲究从上至下（从目标n到1）和从下至上（从1到n）两种方式推导递归**

**二维路径：opt[i,j]=opt[i+1][j]+opt[i][j+1](且判断a[i,j]是否为空地)**

**最难的是第二、三步，定义状态和状态转移方程**

**二、贪心算法**

**1.定义：它是一种在每一步选择中都采取在当前状态下最好或最优的选择，从而希望导致结果是全局最好或最优的算法**

**①贪心：当下做局部最优判断**

**②回溯：能够回退**

**③动态规划：最优判断+回退（保存之前结果）**

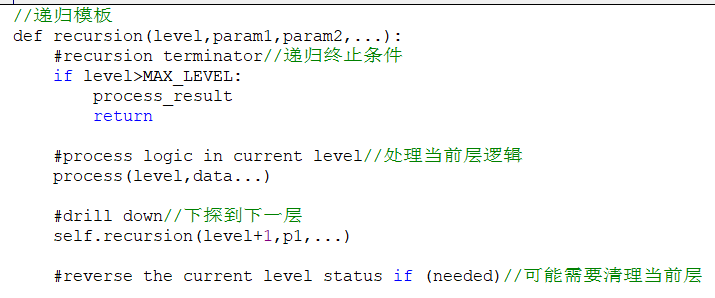
**三、递归**

**1.树的面试题一般解法都是递归：①节点的定义 ②重复性（自相似性）**

**2.递归-循环：通过函数体来进行的循环，它底层是用栈实现的，**

****

**3.递归的模板**

****

**4.思维要点：**

**①抵制人肉递归**

**②找最近重复性**

**③数学归纳法的思想（由f1,f2,推出fn）**

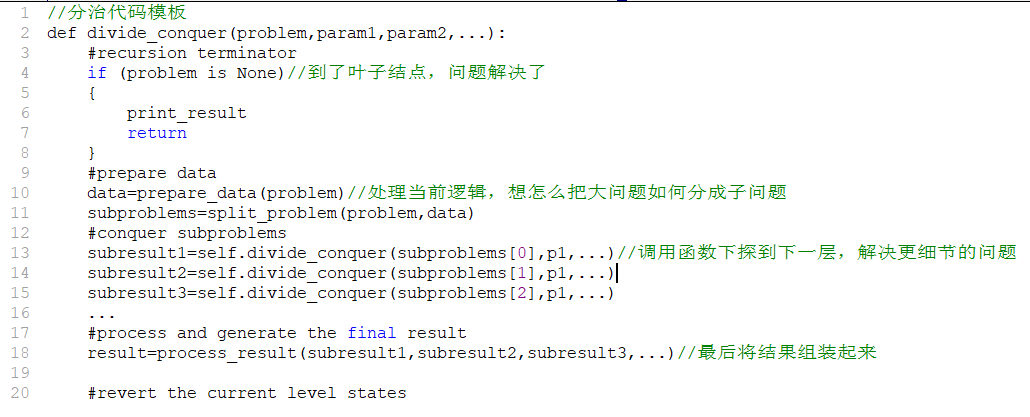
**5.递归思想（重要）**

**分治和回溯就是一种特殊的递归，较为复杂的递归即可**

**最后达到的效果就是说碰到一个题，你就去找它的重复性，重复性有最近的重复性或者是有所谓的最优的重复性，最优的重复性就是动态规划，最近的重复性就是根据重复性怎么构造以及怎么分解就有什么分治啊、回溯或者其他的各种办法，但本质上其实就是递归，就是找它的重复性**

**四、分治**

**1.模板**

****

**五、回溯**

**循环加递归=树枝的层层分叉**

**六、排序算法**

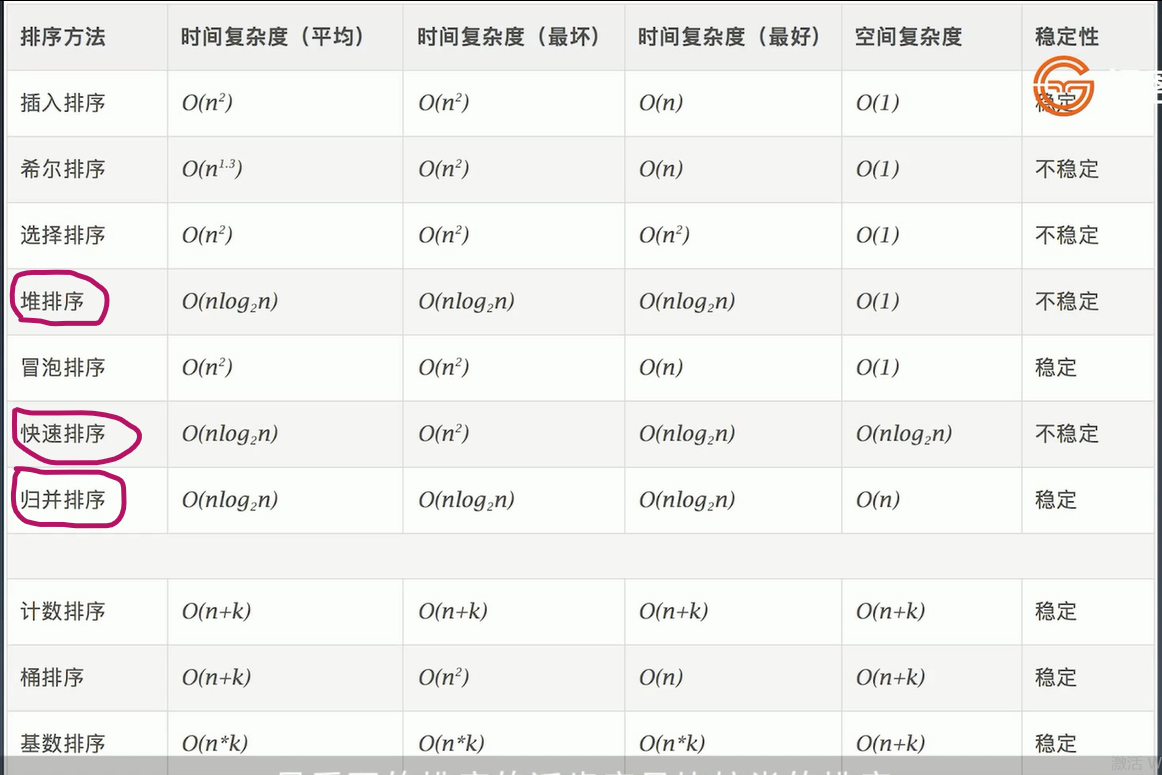
**1.比较类排序：**

**通过比较来决定元素间的相对次序，由于其时间复杂度不能突破O(nlogn)，因此也称为非线性时间比较类排序。**

**2.非比较类排序：**

**不通过比较来决定元素间的相对次序，它可以突破基于比较排序的时间下界，以限行时间运行，因此也称为线性时间非比较类排序**

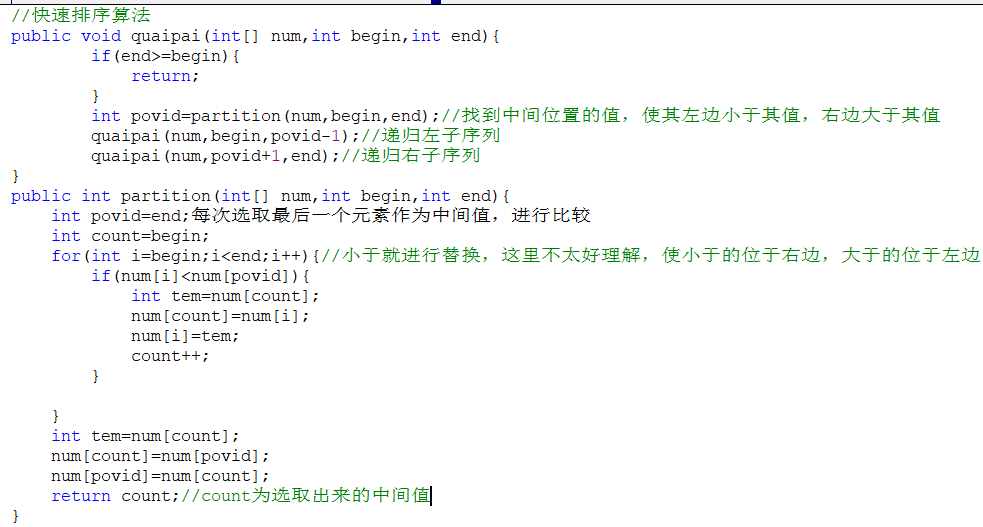


****

**3.高级排序-O(nlogn)**

**3.1快速排序**

**思想：数组取标杆pivot，将小元素放pivot左边，大元素放右边，然后依次对右边和左边的子数组继续快排，以达到整个序列有序。**

****

**3.2归并排序（分治）**

**1.步骤；**

**①把长度为n的输入序列分成两个长度为n/2的子序列；**

**②对着两个子序列分别采用归并排序；**

**③将两个排序好的子序列合并成一个最终的排序序列；**

**3.3归并和快排具有相似性，但步骤顺序相反**

**归并：先排序左右子数组，然后合并两个有序子数组**

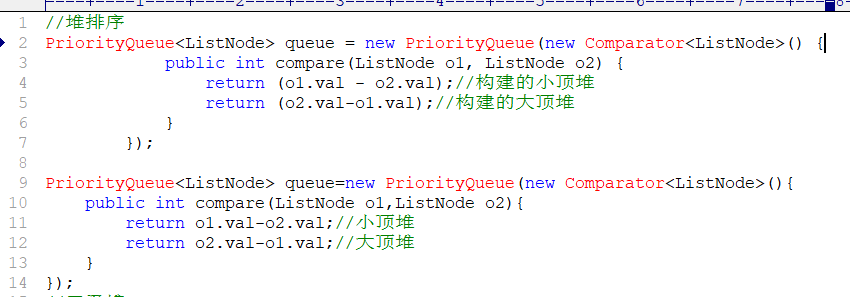
**快排：先调配处左右子数组，然后对于左右子数组进行排序**

****

**3.4堆排序-对插入O(logN),取最大/小值O(1)**

**①数组元素依次建立小顶堆**

**②依次取堆顶元素，并删除**

****

**3.5计数排序（HashMap）**

**计数排序要求输入的数据必须是有确定范围的整数。将输入的数据值转化为键存储在额外开辟的数组空间中；然后依次把技术大于1的填充回原数组**

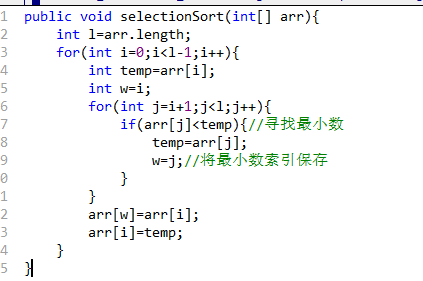
**4.排序算法思想总结**

**⑴初级排序O（n\*n）**

**①选择排序**

**思想：每次找最小值，然后放到待排序数组的起始位置**

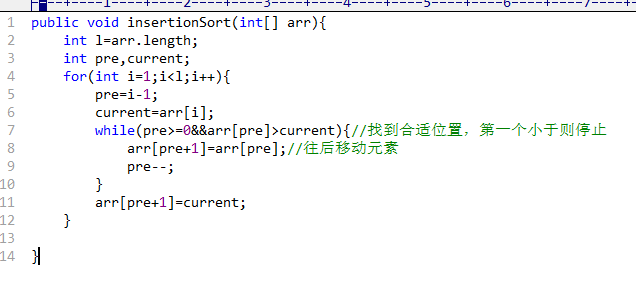
**选择排序(Selection-sort)是一种简单直观的排序算法。它的工作原理：首先在未排序序列中找到最小（大）元素，存放到排序序列的起始位置，然后，再从剩余未排序元素中继续寻找最小（大）元素，然后放到已排序序列的末尾。以此类推，直到所有元素均排序完毕。**

****

**②插入排序**

**思想：从前到后逐步构建有序序列；对于未排序数据，在已排序序列中从后向前扫描，找到相应位置并插入。**

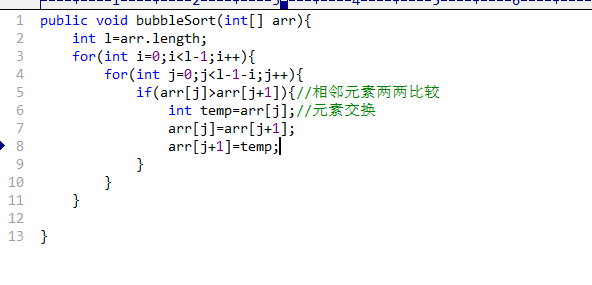
**它的工作原理是通过构建有序序列，对于未排序数据，在已排序序列中从后向前扫描，找到相应位置并插入。**

****

**③冒泡排序**

**嵌套循环，每次查看相邻的元素如果逆序，则交换。**

**冒泡排序是一种简单的排序算法。它重复地走访过要排序的数列，一次比较两个元素，如果它们的顺序错误就把它们交换过来。走访数列的工作是重复地进行直到没有再需要交换，也就是说该数列已经排序完成。这个算法的名字由来是因为越小的元素会经由交换慢慢“浮”到数列的顶端。**

****

**⑵高级排序O(N\*LogN)**

**④快速排序**

**思想：数组取标杆pivot，将小元素放pivot左边，大元素放右边，然后依次对右边和左边的子数组继续快排，以达到整个序列有序。**

**⑤归并排序**

**①把长度为n的输入序列分成两个长度为n/2的子序列；**

**②对着两个子序列分别采用归并排序；**

**③将两个排序好的子序列合并成一个最终的排序序列；**

**⑥堆排序**

**七、字符串算法**

**（1）高级字符串算法与动态规划**

**思想就是：两个字符串弄成二维数组，然后考虑状态转移方程**

**Dp[i][j]与dp[i-1][j-1]\dp[i-1][j]\dp[i][j-1]的关系**

**八、栈与队列**

**1.栈是限定仅在表尾进行插入和删除操作的线性表**

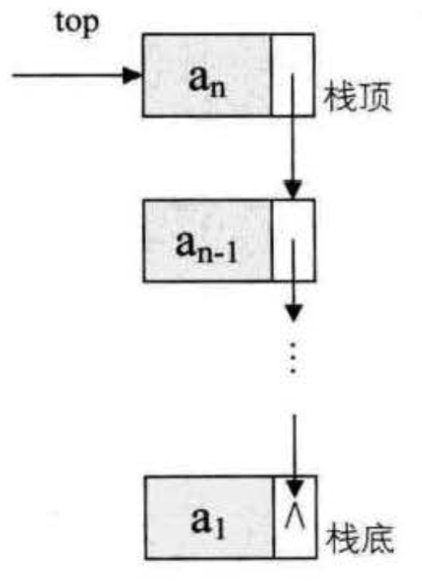
**2.队列是只允许在一端进行插入操作，而在另一端进行删除操作的线性表**

**（1）栈的顺序存储结构**

**数组的0作为栈底，另一端指针作为栈顶进行插入删除**

**（2）栈的链式存储结构**

**由于单链表有头指针，而栈顶也是必须的，所以把栈顶放在单链表的头部**

****