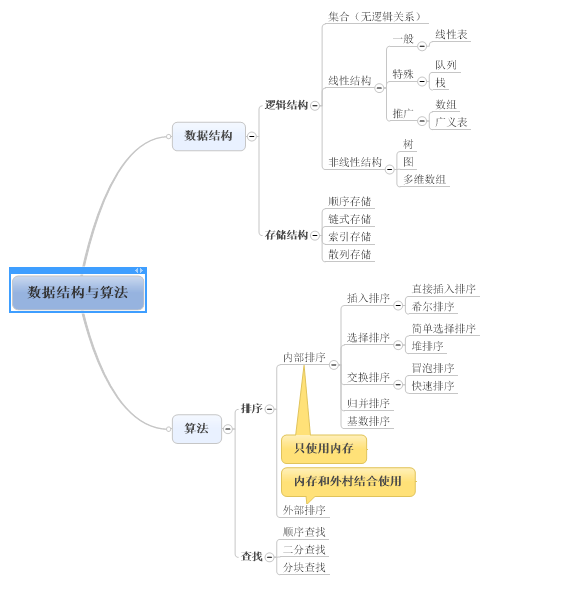
**线性表篇**

数据结构与算法是程序设计的两大基础，大型的IT企业面试时也会出数据结构和算法的题目，它可以说明你是否有良好的逻辑思维，如果你具备良好的逻辑思维，即使技术存在某些缺陷，面试公司也会认为你很有培养价值，至少在一段时间之后，技术可以很快得到提高。同时，它也是软考的重点，我们需要对这部分的内容进行一下总结。

       这篇博文主要是通过图来解决抽象问题。不多说了，我们先看一下数据结构和算法的整体内容。



       数据结构和算法的内容并不多，我们先从数据结构开始，**数据结构总结**完之后，再学习算法，我们要小火慢炖，一口一口吃掉它。

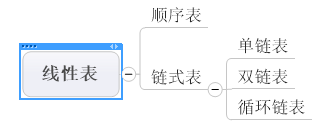
1、线性表

       概念：

             数据元素的排列方式是线性。

      分类：

             分类规则是根据上图中元素的存储结构来划分的。



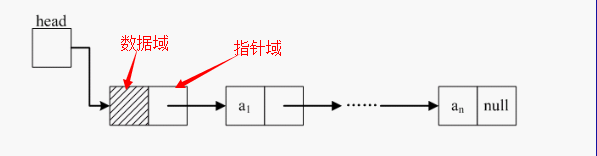
      （1）顺序表

          基本思想：元素的存储空间是连续的。在内存中是以顺序存储，内存划分的区域是连续的。存储结构如下图：

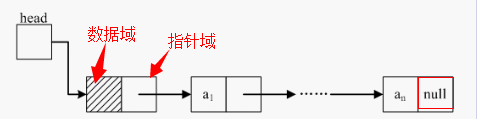


      （2）链表

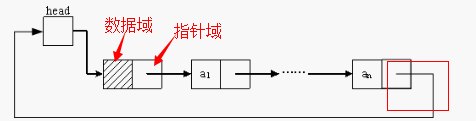
          基本思想：元素的存储空间是离散的，单独的（物理），它们可以通过在逻辑上指针的联系使得它成为了整体的链表。存储结构如下图：



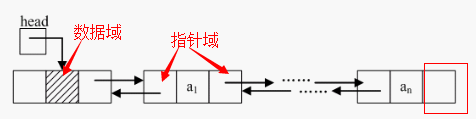
**1.单链表**



**2.循环链表**



**3.双链表（双向循环表）**

****

           三者的区别（从上面三个图我们可以总结出来）：

       1、它们都有数据域（data(p)）和指针域(next(p))，但是从图中可以看出双链表有两个指针域，一个指向它的前节点，一个指向它的后节点。

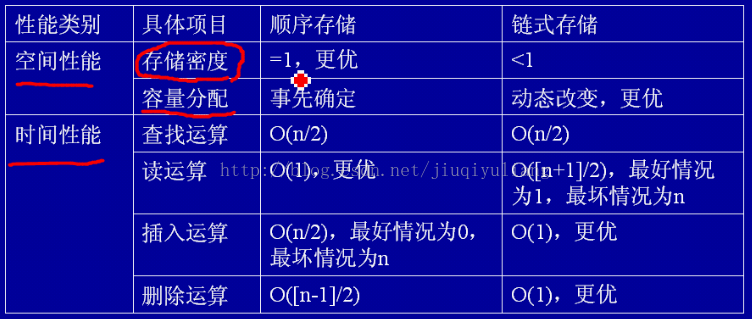
       2、单链表最后一个节点的指针域为空，没有后继节点；循环链表和双链表最后一个节点的指针域指向头节点，下一个结点为头节点，构成循环；

          3、单链表和循环链表只可向一个方向遍历；双链表和循环链表，首节点和尾节点被连接在一起，可视为“无头无尾”；双链表可以向两个方向移动，灵活度更大。

      线性表操作：

         理解了顺序表和链表的基本思想之后，线性表的操作是简单，并且网上有很多讲解插入和删除结点的博客，在这里我就不过多的介绍了。

        顺序表和链表的对比：



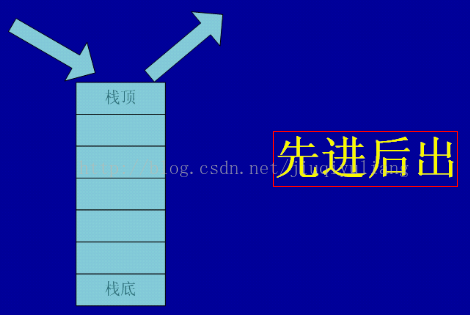
          栈和队列是特殊的线性表，既然特殊就有不同点。

2、栈

      基本思想：后进先出（先进后出）即栈中元素被处理时，按后进先出的顺序进行，栈又叫后进先出表（LIFO）。

      举例：

      日常生活中有很多栈的例子。例如，放在书桌上的一摞书，只能从书顶上拿走一本书，书也只能放在顶上。如下图所示：

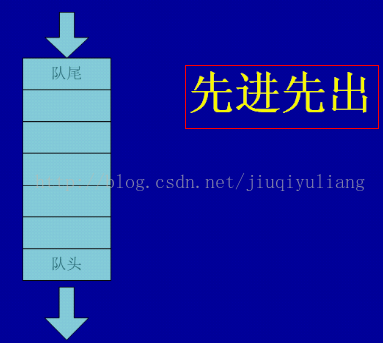


3、队列

     基本思想：先进先出即先被接收的元素将先被处理，又叫先进先出表（FIFO）。如下图所示：

     举例：

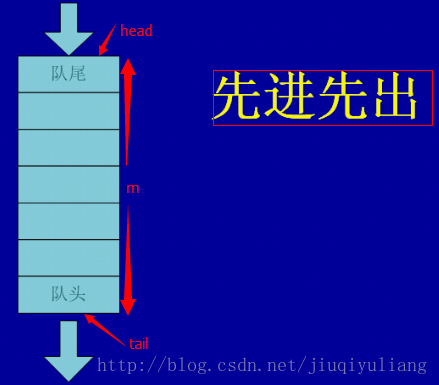
     队列的例子，生活中更多。比如：买车票排队，排头最先买到车票，新来的排的队尾；进车站时，安检行李，先进去的最先出来，后进去的后出来。



分类：

1.顺序队列

          如下图所示：



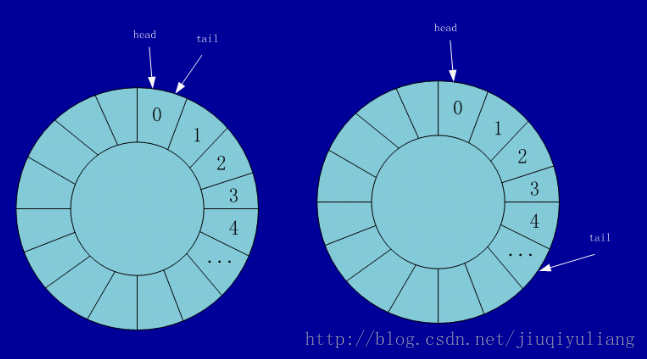
      顺序队列的操作，要判断队满和队空的标志，从图中我们可以总结得到：

      1.队空：head = tail

      2.队满：tail = m

2.循环队列

        如下图所示：



      循环队列的操作，要判断队空和队满的情况，从图中我们可以总结得到：

         1.队空：head = tail

      2.队满：tail + 1 = head(在队列中会留一个空着的空间，所以要加1)

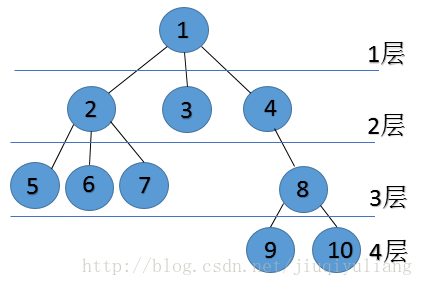
总结

           线性表真的很简单，我反正是会了，你会了吗？

**树与二叉树篇**

树的基本概念：

           树的概念是学习树的关键所在，掌握了树的基本概念，学会树与二叉树，so easy。我通过一棵树来了解树的基本概念，如下图



1、结点的度

      结点的度是子结点的个数。例如：结点1有三个字结点2，3，4，所以结点1的度为3。

2、树的度

      树的度等于所有结点度中度最高的值。例如：上图中结点度最高为3，所以树的度为3。

3、叶子结点

      叶子结点是度为0的结点即没有子结点的结点。例如：上图中3，5，6，7，9，10。

4、分支结点

      分支结点是除了叶子结点，树中的其他所有结点。例如：上面树的分支结点为1，2，4，8。

5、内部结点

      内部结点是除了根结点以及叶子结点或在分支结点的基础之上在去掉根结点。例如：上面树的内部结点为2，4，8。

6、父结点、子结点、兄弟结点

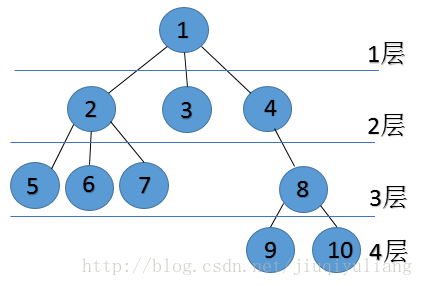
     父节点、子结点和兄弟结点是相对而言的。例如：结点1是结点2，3，4的父节点，结点2，3，4也是结点1的子结点，结点2，3，4又是兄弟结点。

7、层次

     图中我们已经表出来了，根为第一层，根的孩子为第二层，依此类推，若某结点在第i层，则其孩子结点在第i+1层。

树的遍历

 树的遍历特别简单，我们还是以上面的树为例：



1、前序遍历

      基本思想：前序遍历就是先访问根结点，再访问叶子结点。

      图中树的前序遍历为：1，2，5，6，7，3，4，8，9，10。

2、后序遍历

基本思想：本后序遍历就是先访问子结点，再访问根结点。

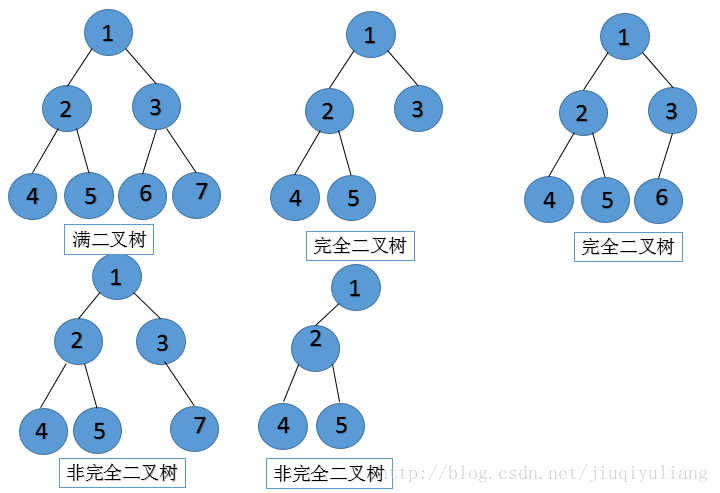
      图中树的后序遍历为：5，6，7，2，3，9，10，8，4，1。

3、层次遍历

     基本思想：从第一层开始，依此遍历每层，直到结束。

     图中树的层次遍历为：1，2，3，4，5，6，7，8，9，10。

二叉树的一些相关概念和特性



(1)**完全二叉树**——若设二叉树的高度为h，除第 h 层外，其它各层 (1～h-1) 的结点数都达到最大个数，第h层有叶子结点，并且叶子结点都是从左到右依次排布，这就是完全二叉树。

(2)**满二叉树**——除了叶结点外每一个结点都有左右子叶且叶子结点都处在最底层的二叉树，深度为k且有2k -1个节点。它是完全二叉树的一种。

(3)**平衡二叉树**——平衡二叉树又被称为AVL树（区别于AVL算法），它是一棵二叉排序树，且具有以下性质：它是一棵空树或它的左右两个子树的高度差的绝对值不超过1，并且左右两个子树都是一棵平衡二叉树。

**一般二叉树性质：**

(1) 在非空二叉树中，第i层的结点总数不超过2i-1 (i>=1)；

(2) 深度为h的二叉树最多有2h -1个结点(h>=1)，最少有h个结点；

(3) 对于任意一棵二叉树，如果其叶结点数为N0，而度数为2的结点总数为N2，则N0=N2+1；**完全二叉树性质:**

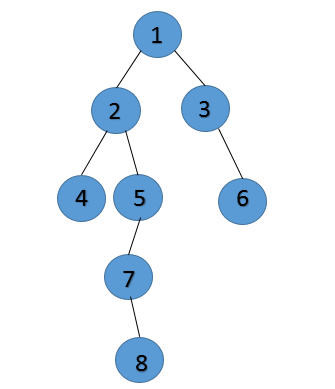
1. 具有n个节点的完全二叉树的高度k为[log2n]+1
2. 对于具有n个节点的完全二叉树,如果按照从上(根节点)到下(叶节点)和从左到右的顺序对二叉树中的所有节点从0开始到n-1进行编号,则对于任意的下标为k的节点，有：

* 如果k=0,则它是根节点，它没有父节点；如果k>0,则它的父节点的下标为[(i-1)/2];
* 如果2k+1 <= n-1,则下标为k的节点的左子结点的下标为2k+1;否则,下标为k的节点没有左子结点.
* 如果2k+2 <= n-1,则下标为k的节点的右子节点的下标为2k+2;否则,下标为k的节点没有右子节点

**满二叉树性质:**

      在满二叉树中，叶节点的个数比分支节点的个数多1

二叉树遍历



1、前序遍历（与树的前序遍历一样）

      基本思想：先访问根结点，再先序遍历左子树，最后再先序遍历右子树即根—左—右。

      图中前序遍历结果是：1，2，4，5，7，8，3，6。

2、中序遍历

      基本思想：先中序遍历左子树，然后再访问根结点，最后再中序遍历右子树即左—根—右。

      图中中序遍历结果是：4，2，7，8，5，1，3，6。

3、后序遍历

      基本思想：先后序遍历左子树，然后再后序遍历右子树，最后再访问根结点即左—右—根。

      图中后序遍历结果是：4，8，7，5，2，6，3，1。

4、层次遍历（与树的层次遍历一样）

      基本思想：从第一层开始，依此遍历每层，直到结束。

      图中层次遍历结果是：1，2，3，4，5，6，7，8。

树与二叉树区别

1、树可以有多个子结点，二叉树最多只能两个结点。

2、树中的子结点是无序的，二叉树是分左子结点和右子结点。

3、二叉树不是特殊树，而是独立的数据结构。

总结

          这篇博文都是树的基本内容，这些基本内容可以帮助你更加深刻的理解树的其他内容，只要你能努力，世界充满爱。

**图篇**

  图跟树一样，也是非线性结构，咋看起来有点复杂，其实它很简单。树具有层次关系，上层元素可以与下一个多个元素连接，但是只能和上层的一个元素连接。在图结构中，节点间的连接是任意的，任何一个元素都可以与其他元素连接。

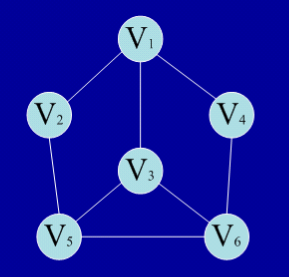
       图相对而言很简单，我们只介绍的图的遍历和最小生成树，现在我们开始。

遍历

1.概念

      从图中某一个顶点出发，访问图中的每一个结点，并要求只能访问一次，不能重复访问。

2.方法



**（1）广度优先遍历**

       基本思想：从图中某顶点v出发，在访问了v之后依次访问v的各个未曾访问过的邻接点，然后分别从这些邻接点出发依次访问它们的邻接点，并使得“先被访问的顶点的邻接点先于后被访问的顶点的邻接点被访问，直至图中所有已被访问的顶点的邻接点都被访问到。如果此时图中尚有顶点未被访问，则需要另选一个未曾被访问过的顶点作为新的起始点，重复上述过程，直至图中所有顶点都被访问到为止。

换句话说，广度优先搜索遍历图的过程是以v为起点，由近至远，依次访问和v有路径相通且路径长度为1,2...的顶点。

  广度优先遍历：V1，V2，V3，V4，V5，V6或V1，V4，V3，V2，V6，V5

**（2）深度优先遍历**

       基本思想：假设初始状态是图中所有顶点均未被访问，则从某个顶点v出发，首先访问该顶点，然后依次从它的各个未被访问的邻接点出发深度优先搜索遍历图，直至图中所有和v有路径相通的顶点都被访问到。 若此时尚有其他顶点未被访问到，则另选一个未被访问的顶点作起始点，重复上述过程，直至图中所有顶点都被访问到为止。

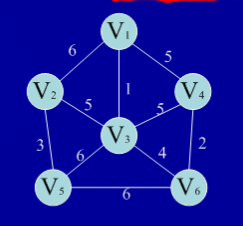
显然，深度优先搜索是一个递归的过程。

深度优先遍历：V1，V2，V5，V3，V6，V4或V1，V4，V6，V3，V5，V2

总结，图的广度优先遍历和深度优先遍历的结果并不唯一。

参考文章：<http://www.cnblogs.com/skywang12345/p/3711483.html>

最小生成树

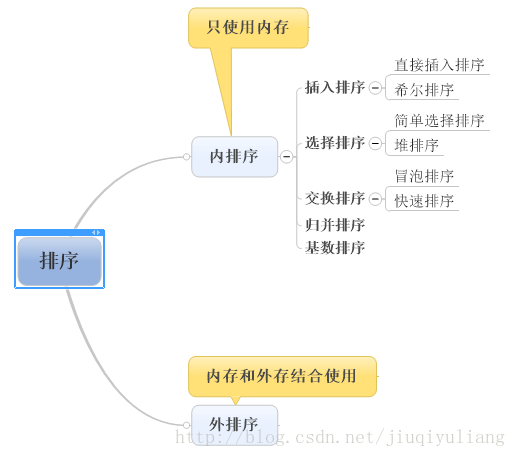


（1）普里姆（Prim）算法  
       基本思想：选一个顶点开始，查找与顶点相邻且代价（边值）最小的边的另一个顶点，直到最后。  
       例如：V1作为顶点，V1->V3->V6->V4，V3->V2->V5，连接图中所有的结点即可。  
（2）克鲁斯卡尔（Kruskal）算法  
       基本思想：选择图中最小的边，直到所有结点都连通。  
       例如：第一小边：V1->V3，第二小边：V4->V6，第三小边：V2-V5，第四小边：V3->V6，第五小边：V3->V2，此时所有的结点都连到了一起。  
（3）算法对比  
       普里姆算法更加注重的是结点，点与点之间距离最短的优先；克鲁斯卡尔算法更加注重的是边，将边排序，最小边排在前面，最大边排在后面。

八大排序方法

  排序是编程的基础，在程序中会经常使用，好的排序方法可以帮助你提高程序运行的效率，所以学好排序，打好基础，对于程序的优化会手到擒来。无论你的技术多么强，如果没有基础也强不到哪去。

       不多说了，我们直接进入今天的主题。



    总有人说排序很难，但是我总觉得很简单，我们只需要了解基本思想就好了，通过简单的例子来加深理解。

1、直接插入排序

（1）简介：直接插入排序，从字面意思可以看出，直接插入数据完成排序。

（2）基本思想：在插入第i个数时，假设前i-1数已经排好序了，只需要将第i个数插入到i-1中，使得这i个数也是顺序的。

（3）例如：

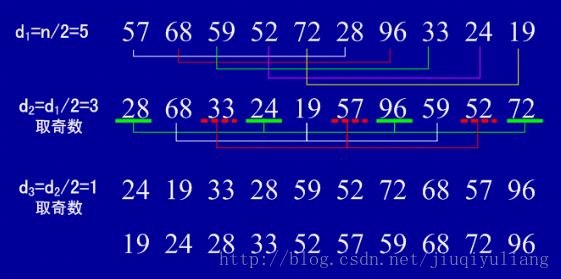


2、希尔排序（Shell排序）

（1）简介： 希尔排序又称为缩小增量排序，是对直接插入排序方法的改进。

（2）基本思想：将整个序列分成多个子序列，然后分别进行直接插入排序，直到整个序列中的所有数基本有序时，再对整体进行一次直接插入排序。

（3）例如：



3、简单选择排序

（1）简介：简单选择排序也叫直接选择排序，其实说白了跟直接插入排序的道理特别简单，效率低。

（2）基本思想：首先在 n个数中选择一个最小的数，并将它从中删除，作为新的一组数的第一个；再在剩下的数中选择最小的数，将它从中删除，作为新的一组数的第二个。。。。如此反复，直到排序完成，最后得到一组从小到大排序的数。

（3）例如：



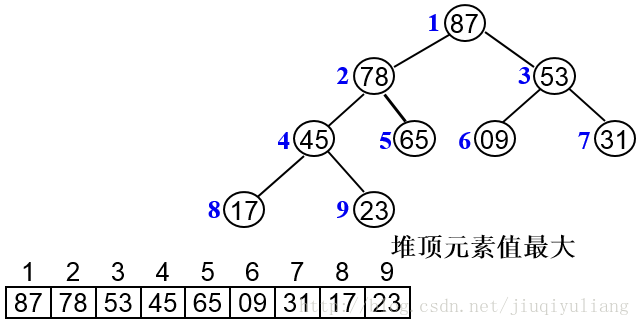
4、堆排序

（1）简介：堆排序是一个相当有用的排序技术，特别适用于对大量的记录进行排序。同时，堆排序也是对简单选择排序的改进。

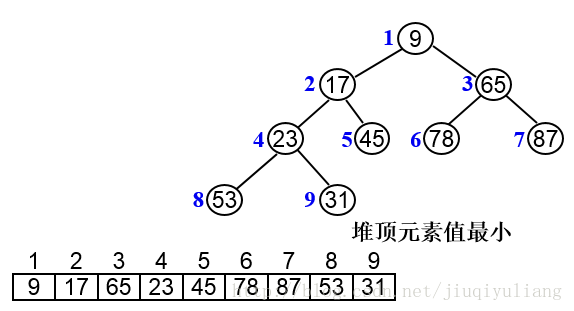
堆的定义：n个元素的序列｛K1，K2，...，Kn｝当满足下列关系时，称为堆：Ki≤K2i且Ki≤K2i+1或者Ki≥K2i且Ki≥K2i+1。注意：堆树必须是一颗完全二叉树。

（2）基本思想：利用堆积树这种数据结构所设计的一种排序，可以利用数组的特点快速的定位指定索引的元素。

最大堆：父结点大于或等于儿子结点



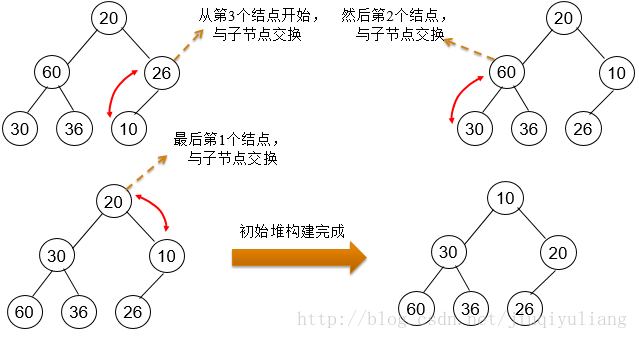
最小堆：父结点小于或等于儿子结点



（3）例如：

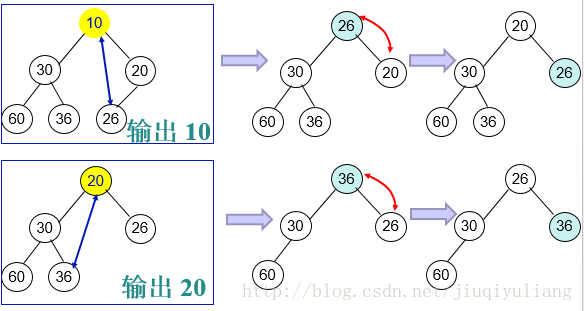
将序列｛20,60,26,30,36,10｝调整为递增序列。

1、首先将数据建立完全二叉树，填充规则是按层次遍历将数据一一填入，最后构建最小堆；

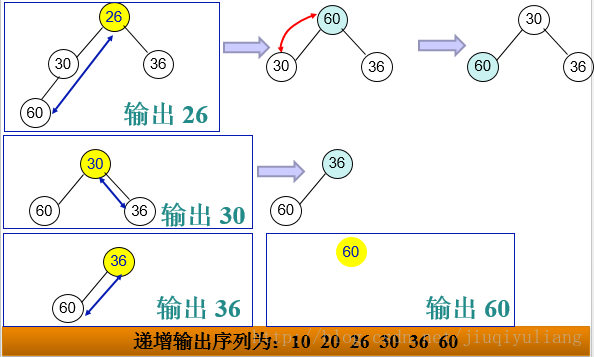


2、提取堆顶并调整删除队顶后的元素为新堆；

3、重复第2步，直到堆空；



4、每次提取的堆顶依次排序即为递增序列。

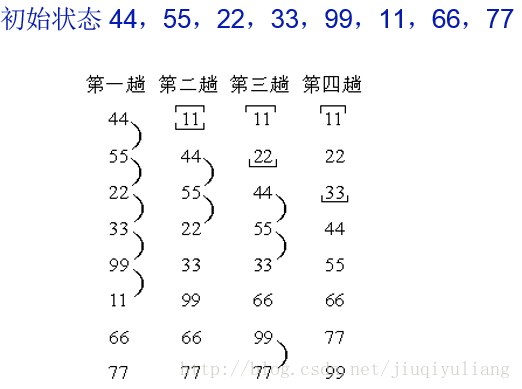


5、冒泡排序

（1）简介：冒泡排序，就跟水里的物体一样，小的往上浮，大的往下沉。

（2）基本思想：将数组垂直排列，取出最后一个元素逐个向上交换，得到大数（小数），继续步骤一的操作，直到排序完成。

（3）例如：



6、快速排序

（1）简介：快速排序是目前内部排序中速度最快的一种排序算法。

（2）基本思想：选取一个数据（通常是数组的第一个数）作为关键数据，然后将所有比它小的数都放在它前面，所有比它大的数都放在它后面，这个过程称为一趟快速排序，再从分开的部分选取基准数，进行分组划分，重复执行，直到完成。

（3）例如：

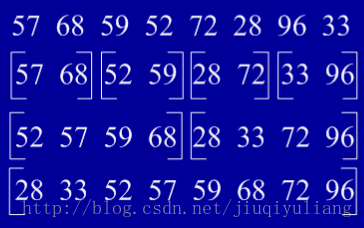


7、归并排序

（1）简介：归并排序又称为二路合并操作，使用合并操作完成排序的算法。

（2）基本思想：将两个或两个以上的有序表合并成一个新的有序表，最后将所有的有序表合成一个整体有序表。

（3）例如：

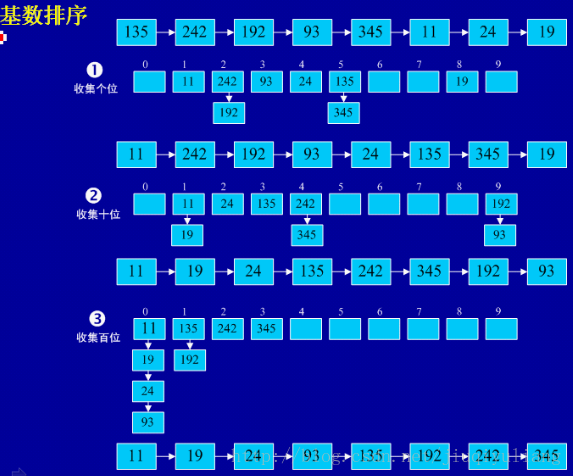


8、基数排序

（1）简介：前面介绍的排序方法都是对元素进行的，基数排序是对元组进行的。

（2）基本思想：从低位到高位依次对待排序的数进行分配和收集，经过d趟分配和收集，就可以得到一个有序序列。

（3）例如：



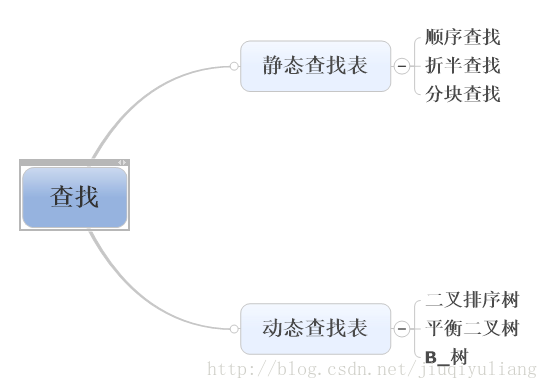
排序算法对比



总结

         人处理图像的能力是最强的，相信你通过这一张张的图，你已经学会了这八种排序。

**查找**



**静态查找**

若查找目的是为了查询某个特定的数据是否在表中或检索某个特定数据的各种属性，则此类查找表为静态查找表。

1、顺序查找

基本原理：从表一端开始逐个和关键字进行比较，若找到一个记录和给定值相等，则查找成功，反之失败。再简单点就是，一个一个的比大小，看看是否相等。

例子：

http://img.blog.csdn.net/20140515143248625?watermark/2/text/aHR0cDovL2Jsb2cuY3Nkbi5uZXQvaml1cWl5dWxpYW5n/font/5a6L5L2T/fontsize/400/fill/I0JBQkFCMA==/dissolve/70/gravity/SouthEast

顺序查找更适合于顺序存储结构和链式存储结构的查找表。顺序查找需要一个个的去比较，效率很低。

2、折半查找（二分查找）

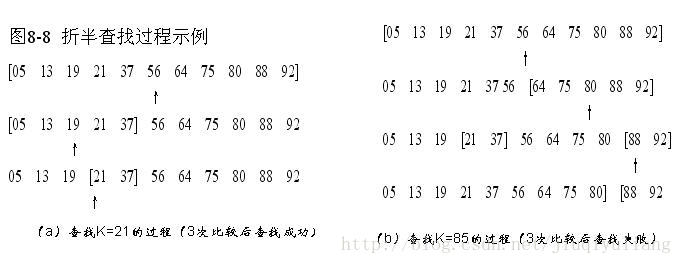
基本原理：1.把序列分成左中右三部分，左部分小于中间值，右部分大于中间值；

                  2.把给定值与中间值比较，确定下次查找是在左部分还是右部分；

                  3.继续上面两步操作，直到成功或失败。

注意：折半查找需要注意给定的序列必须是一个有序序列。

例子：

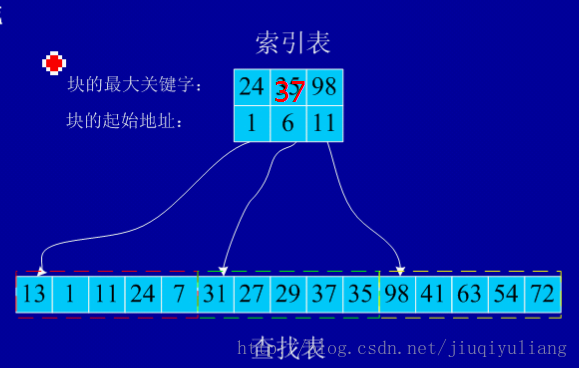


3、分块查找

基本原理：顺序查找和二分法查找的折中，先分块，在块中顺序查找。

注意：分成的各块内部数据可能无序；各块之间有序（第二个块中的元素都比第一个块中元素都大）；建立了索引表，索引表按关键字有序。

例子：



静态查找表方法的性能分析



        对于动态查找的插入和删除不是特别好讲，我们就不在这里讲了，只是简单的介绍一下什么是二叉排序树和平衡二叉树，B\_树只做了解。

**动态查找**

若再查找的过程中同时插入查找表中不存在的数据，或从查找表中删除已存在的某个数据，则称此类查找表为动态查找表。

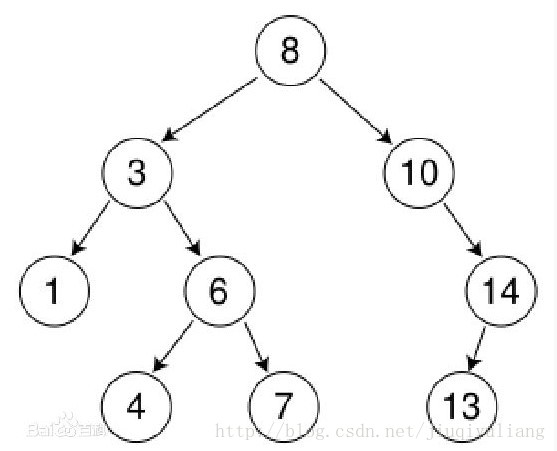
1、二叉排序树

定义：1.若它的左子树非空，则左子树上所有的结点的值均小于根结点的值；

         2.若它的右子树非空，则右子树上所有的结点的值均大于根结点的值；

         3.左右子树本身就是两棵二叉排序树。

例子：



       定义看上去不是特别好理解，其实特别简单，我们再以例子简单的说一下。左子树的所有节点：3，1，6，4，7，都小于父节点8，右子树所有节点：10，14，13，都大于父节点。什么时候都是父节点大于左孩子，小于右孩子例如：8>3，8<10；3>1，3<6。

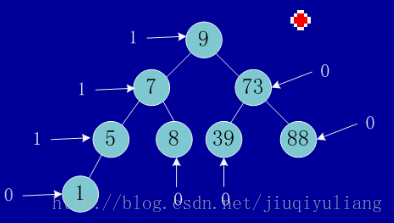
2、平衡二叉树

定义：1.它或者是一棵空树

         2.或者树中任一结点的左右子树深度相差不超过1。

注意：从定义我们可得到：想要一颗树平衡，有三种情况，节点的平衡度要么为了0，要么为1，要么为-1。（平衡度：节点左子树的高度减去其右子树的高度。）

例子：



上面图在每个节点上标出了平衡度，所有的节点的平衡度的绝对值都小于等于0或1，所以它是一棵平衡二叉树。