**【计导非课系列】 第六节 算法 数据结构**

程序=算法+数据结构

众所周知，完美的程序是由优秀的算法和良好的数据结构组成的。什么是算法？什么是数据结构？这里，我们将一起探索算法和数据结构的奥秘，初步揭开它们的神秘面纱。

当然，算法和数据结构可以作为永远发展的大事业，绝非寥寥数语能够说清楚的。

**博文目录**

**算法**

* 什么是算法
* 流程图
* 基本算法

**数据结构**

* 基本数据结构 ADT 链表 栈 堆 树 图
* 高级程序设计语言
* 软件工程
* 算法案例（脑洞大开）

**before we start…**

人类是怎么样让计算机解决问题？怎样操作？  
首先，分析问题的本质。我要干些什么？知道我要干什么之后，我需要想想怎么去实现这些东西，我要有一个明确的构思，现在不一定需要太具体的细节，只需要大概。之后捏，程序猿就可以大显身手咯，编写程序把之前说好的算法给实现——具体是怎么做的，怎么写出来。之后就可以运行了，运行之后就可以得到结果了，当然极有可能经历过很多很多次的debug！

在这个过程中，怎么实现这件事情就叫做算法——一种逐步解决问题或完成任务的方法。

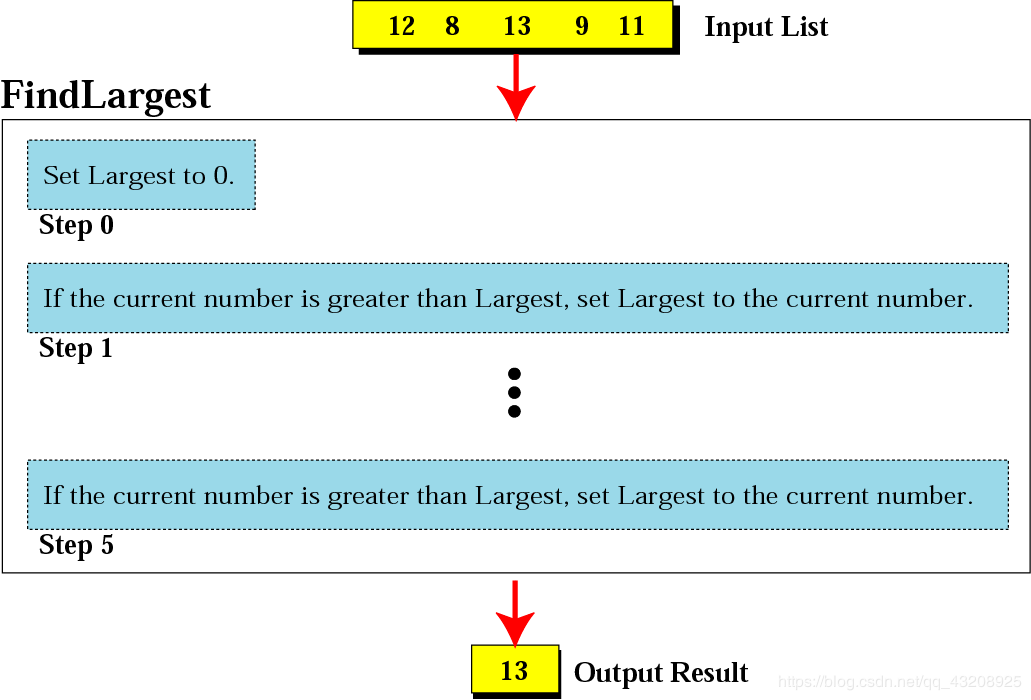
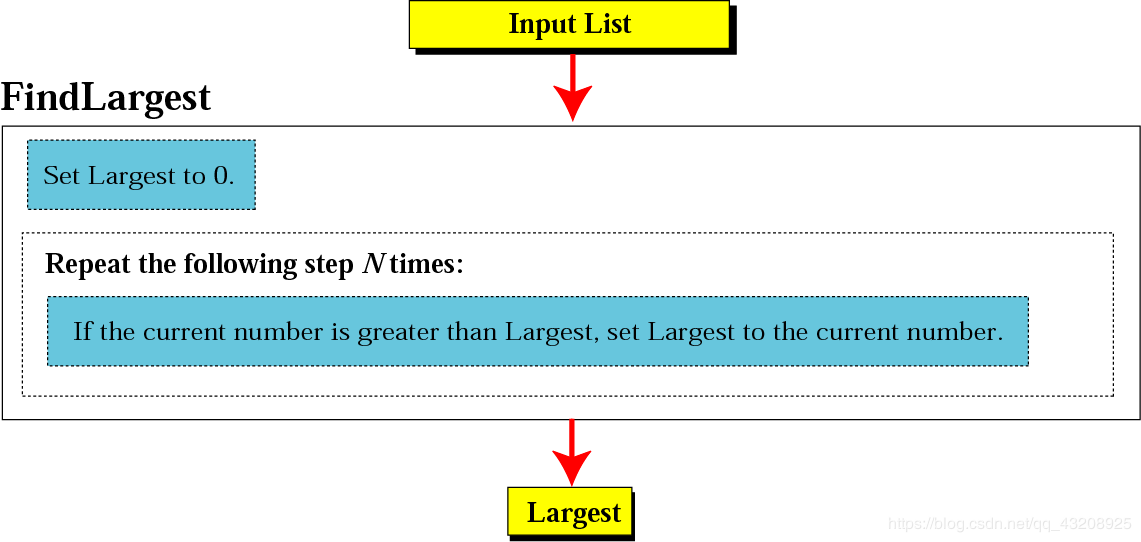
算法可以精化——每一步是怎么实现的；也可以泛化——我大概想怎么做，重复这一堆动作。

**算法**

**算法的定义**

算法是一组明确步骤的有序集合，它产生结果，并在有限的时间内终止。  
算法的5个特性：有穷性 确定性 输入 输出 可行性  
**另一种定义**：（了解一下）给定问题和设备，一个算法是用该设备可以理解的语言表示的。解决这个问题的一种方法是精确刻画的。算法具有以下属性：  
1.应用于一个具体的输入集合或将问题描述在有穷步动作之后得到结果  
2.有一个唯一的初始动作  
3.每一个动作有一个唯一的后继动作  
4.该序列终止时或者得到这个问题的解，或者因该问题不可解而获得不可解说明

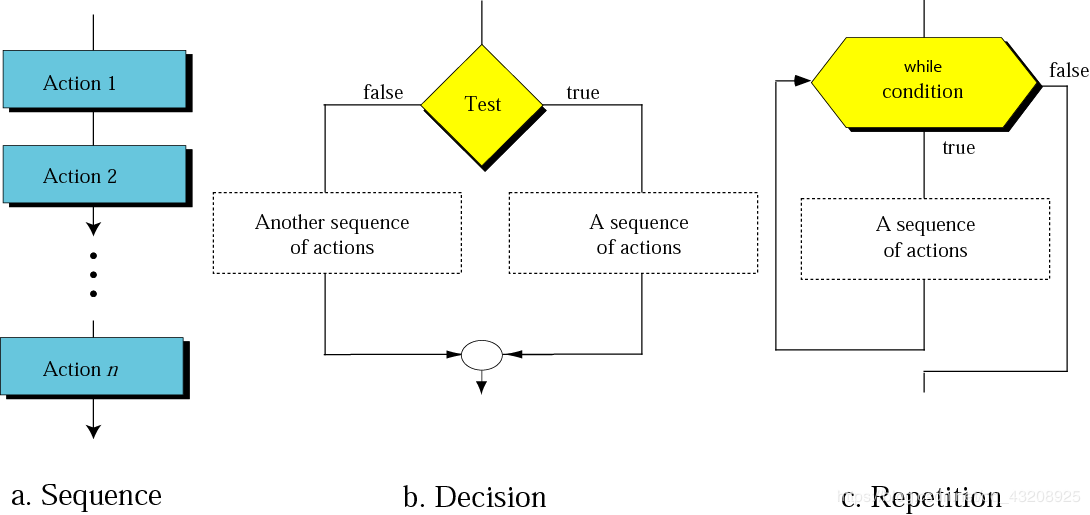
**算法的精化 泛化**

算法的精化：就是将每一步详细地写出来  
  
算法的泛化：大致描述一个程序具体需要怎么做，比如  
Repeat the following step N times:  
If the current number is greater than Largest, set it to the current number.  


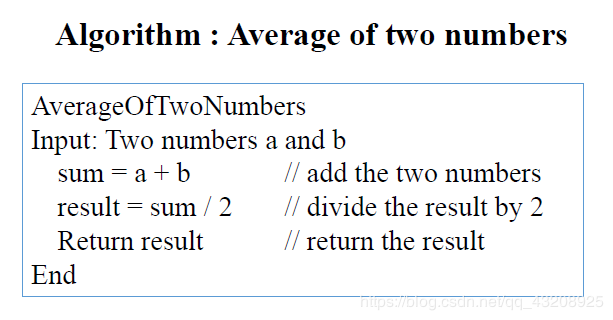
**算法的基本结构**

* 顺序 Sequence
* 选择 Decision
* 循环 Repetition

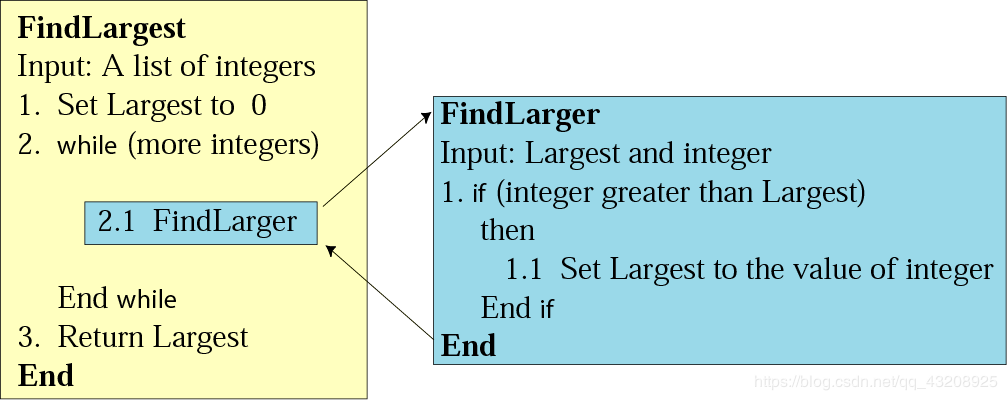
**流程图**

写程序之前画好流程图，这样可在写代码的时候让程序更加结构清晰。  


**伪代码：不管细节 只关心大体实现**

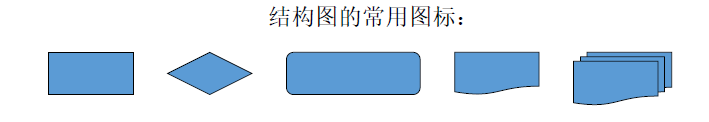
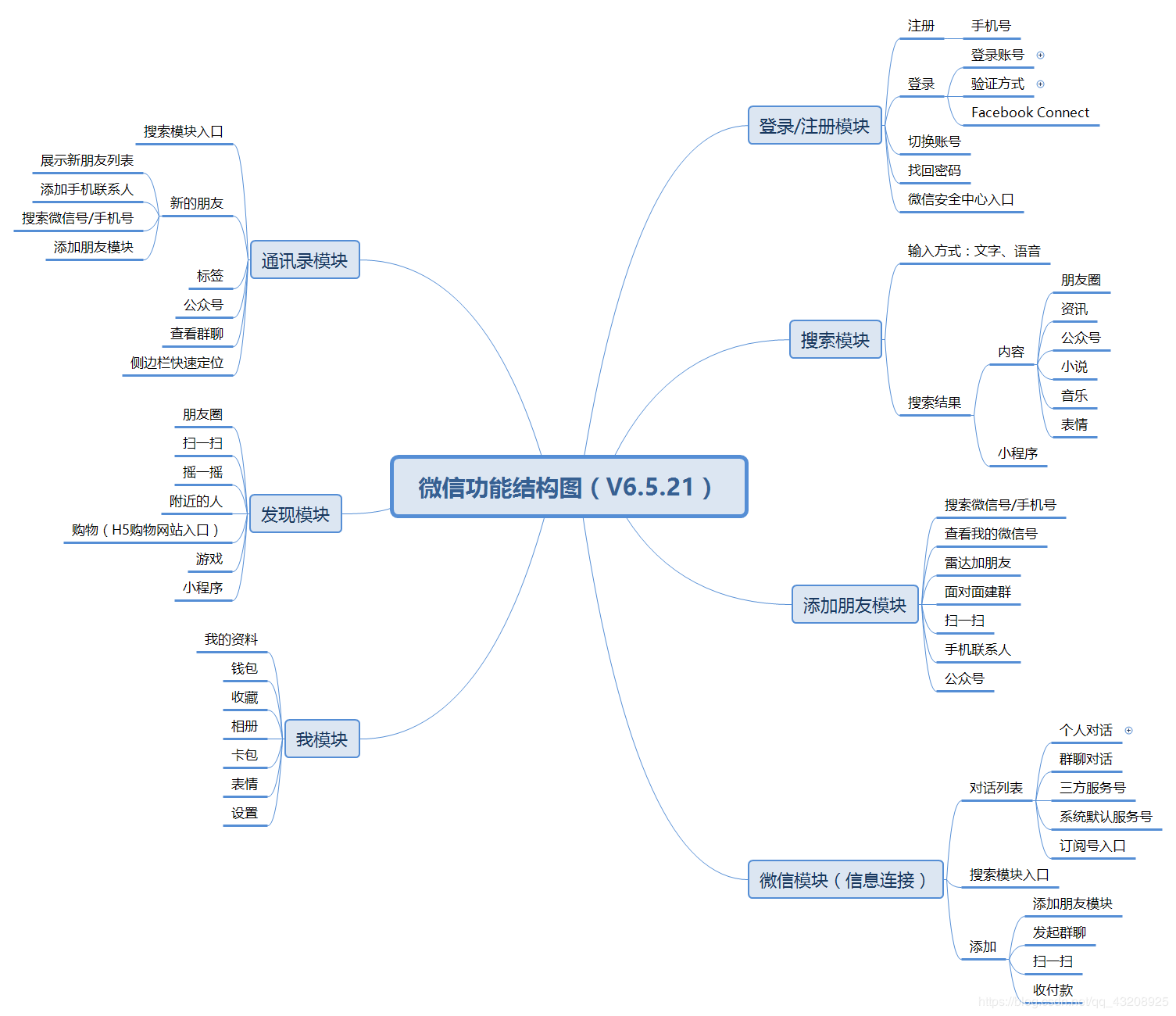
是在编写算法时，为了更好地表示算法本身，不在一些小的细节上纠缠，而采用类似于英语（或其他自然语言）表示算法的算法表示方法。  


**子算法（函数）：实现结构化编程**

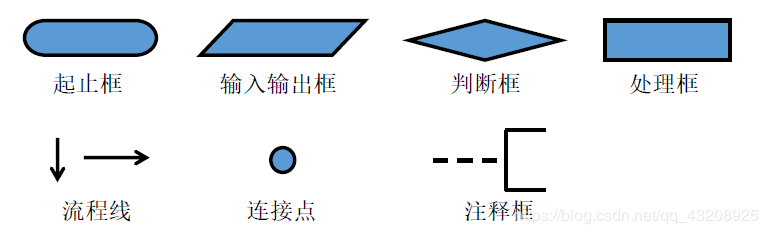
在结构化编程时使用模块化或分单元的办法来构成程序，这些被分成的小单元称之为子算法。  
每个子算法又可以划分为更小的子算法，这个过程持续到子算法变为最本质的（可被立即理解的）描述为止。  
**优点**：  
使程序更易理解  
可以在主算法不同的地方调用  
**其实，子算法就是我们平常所说的函数。**  
举个例子：  


**结构图和流程图**

**结构图：高层设计工具**

结构图能很好地表示程序设计者的逻辑思维的过程；把算法中各个模块之间的关系表示得更加清楚。  
结构图的常用图标：  
  
举例：结构图可以把大程序要实现的所有东西都表示出来，高屋建瓴  


**流程图：表示算法的工具**

用一些图框来表示各种操作  
用图形表示算法，直观形象，容易理解  


**基本算法**

基本算法包括求和、乘积、最大最小、排序、查找  
排序包括冒泡排序，选择排序，插入排序

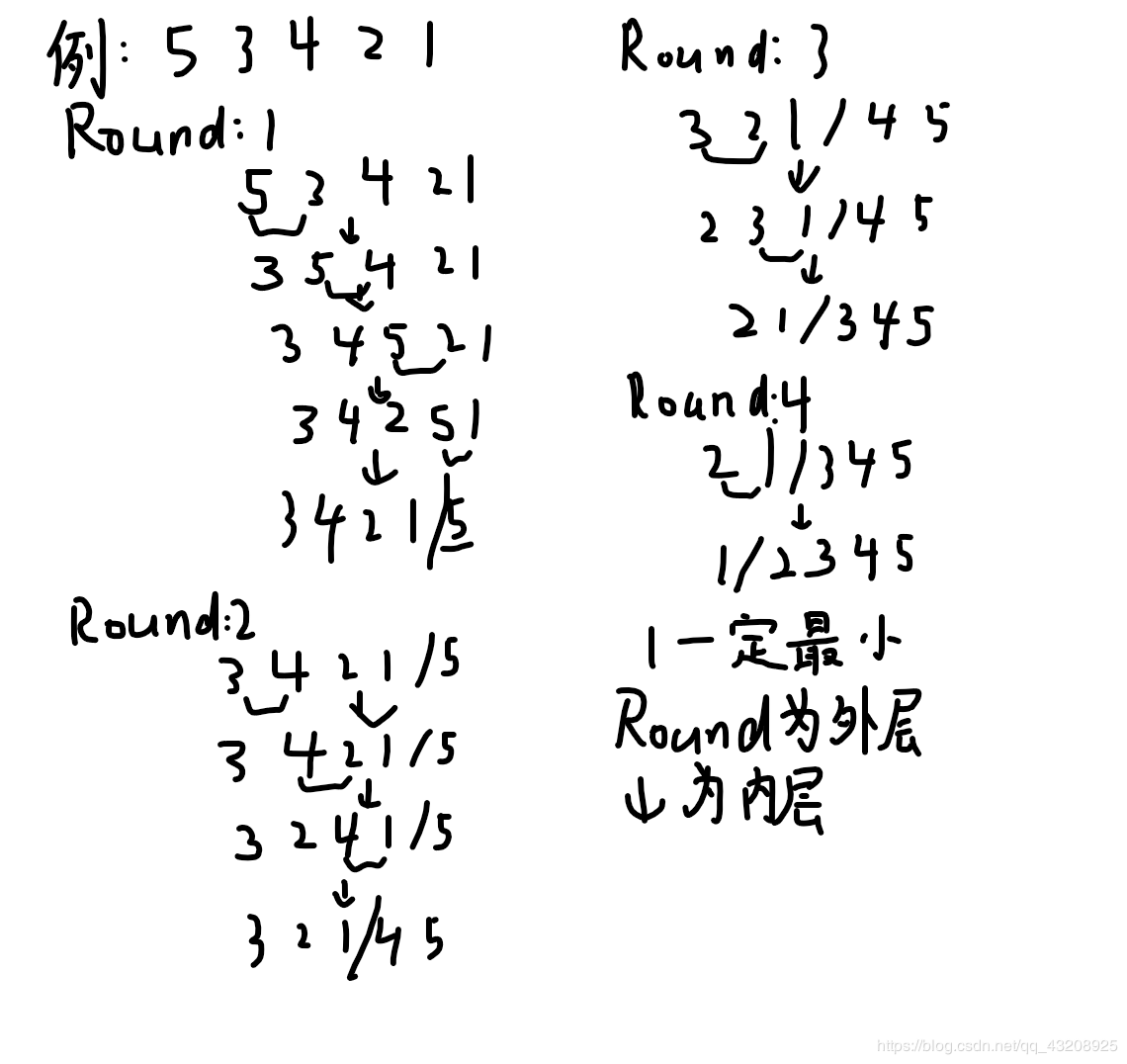
**求和、求积**

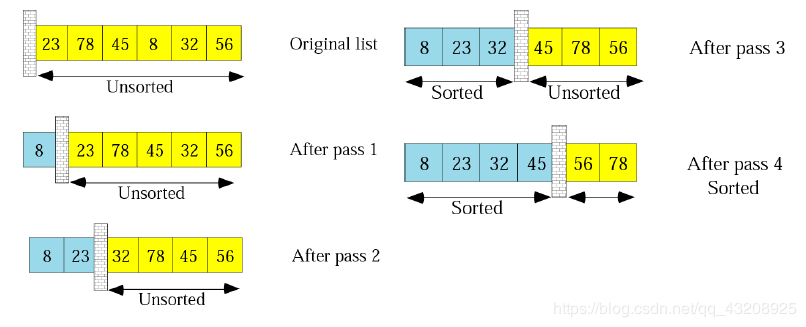
把初始结果设置为0或者1，然后设置一个变量记录执行了多少次。之后执行循环。

**排序**

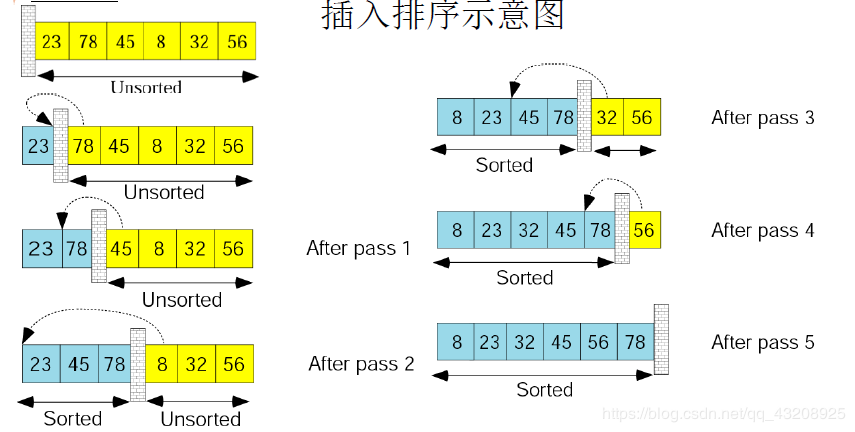
**冒泡排序：小的数据上升**

之前写过一篇关于冒泡排序的博客，传送门：  
<https://blog.csdn.net/qq_43208925/article/details/83684040>  
它重复走访要排序的数列，一次比较两个元素，顺序不对的话就把他们交换过来。直到某一次走访发现全部顺序都是对的。  
这样的直观感受是，大数像石头一样沉淀，小数像泡泡一样“浮出水面”，因此得名“冒泡排序”。  
实现原理：  
给定N个数，那么它最多需要执行N-1次（因为最后一个一定不需要再次排列）循环，每次循环是一轮。如果某一轮中已经出现了没有交换，则可以提前结束。因此需要一个flag变量，记录循环状态。  
在每一轮循环中，从第一个数开始，与下一个数进行比较。一直到上一次沉底的元素。上一次沉底的元素一定不需要排列，因为它一定会是最大的。

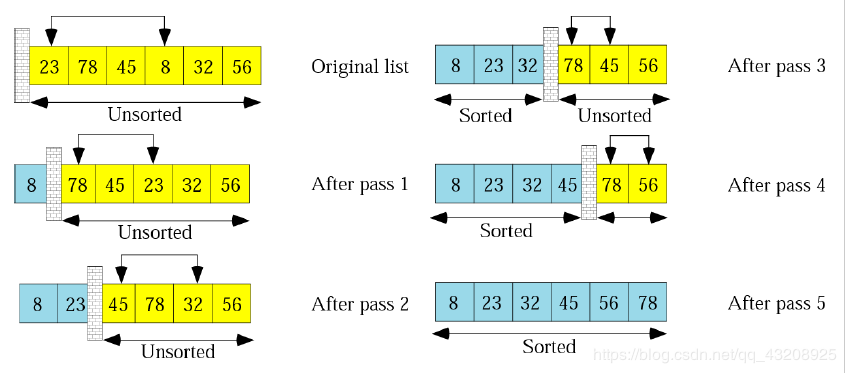
下面两个例子（第一个摘自之前的博客），第一个是让大数沉底，第二个是让小数上升。  




**插入排序：按顺序走访，把后面的数据放到前面已经排好序的阵营里面，一个一个插进去**

  
逐个排序

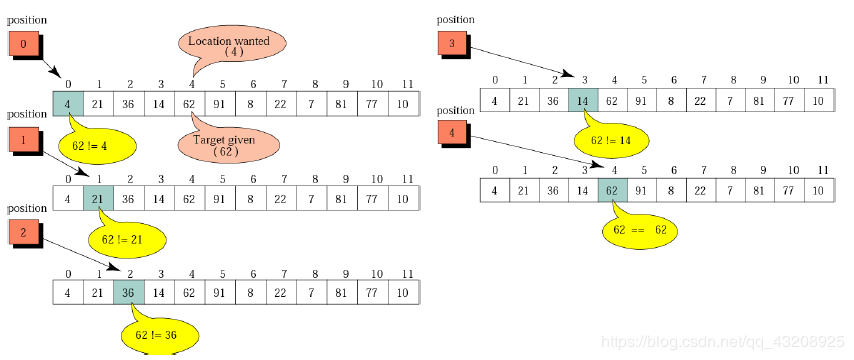
**选择排序：遍历，每次选出来一个最大/最小，放到排好序的阵营里面**

这和冒泡排序的区别：冒泡排序是逐个走访，看到谁不对就顺手换过来了；而选择排序是这一次循环里面我偏偏要找到最大/最小的那一个，把他拎出来。  


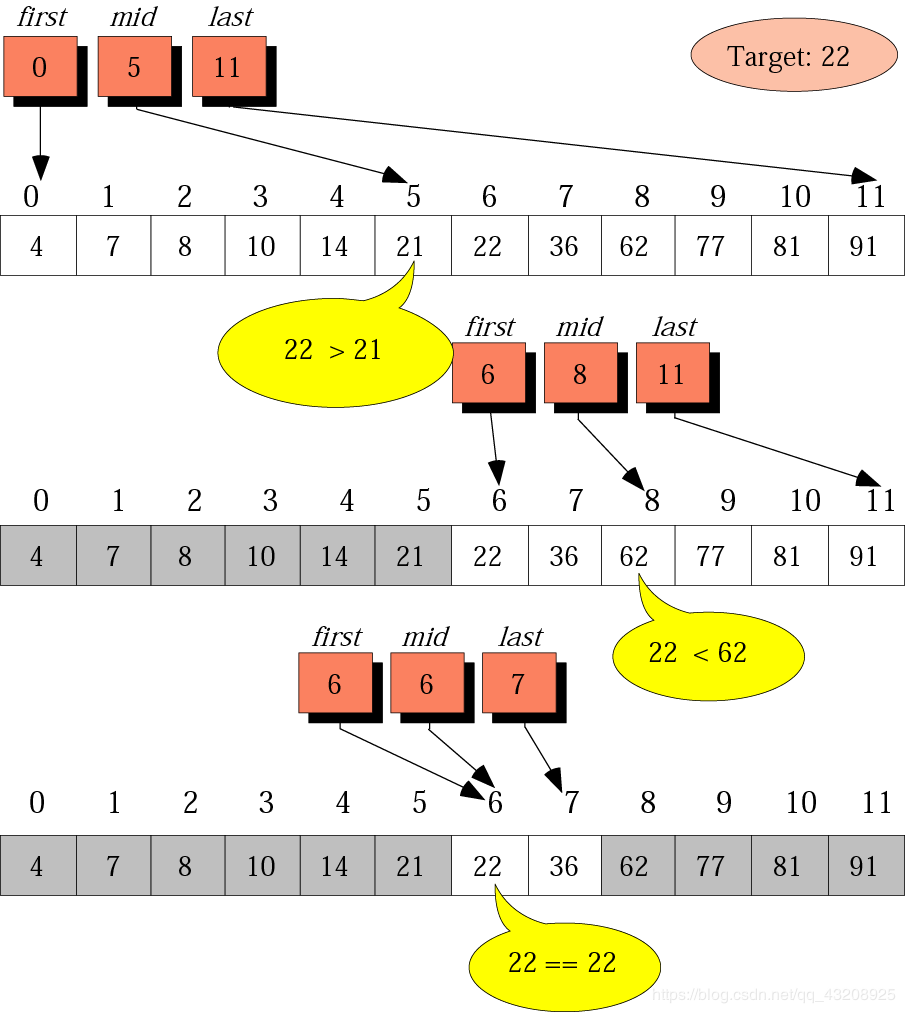
**查找**

在一些（有序的/无序的）数据元素中，通过一定的方法找出与给定关键字相同的数据元素的过程叫做查找。

**顺序查找**

从第一个/最后一个开始遍历，直到出现了我想要的。  


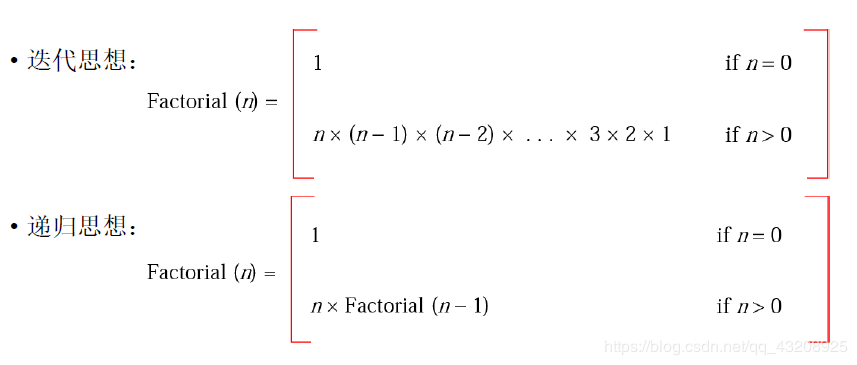
**折半查找（二分查找） Binary search**

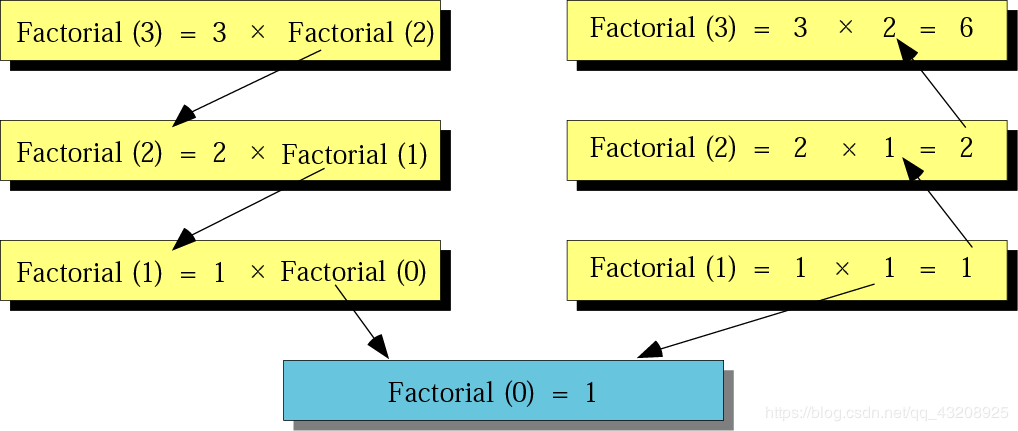
在有序数组中查找。  
  
需要变量三个：first,mid,last，给定了target。  
首先，target和mid去比较，如果mid大于target，那么first变成mid+1；  
如果mid小于target，那么last变成mid-1。

**递归、迭代算法**

迭代是重复反馈过程的活动，其目的通常是为了逼近所需目标或结果。每一次对过程的重复称为一次“迭代”，而每一次迭代得到的结果会作为下一次迭代的初始值。  
递归（Recursion）：算法中包含了对算法自身的调用的算法。  
分治算法（Divide and Conquer）：将一个难以直接解决的大问题，通过分析，分解为一些规模较小的相同问题，进而对各个小问题进行解决，最后达到整个问题的解决。

迭代，就是不断更新，不断更新。  
f2=f1  
f3=f2  
放到循环里，就可以构成迭代。

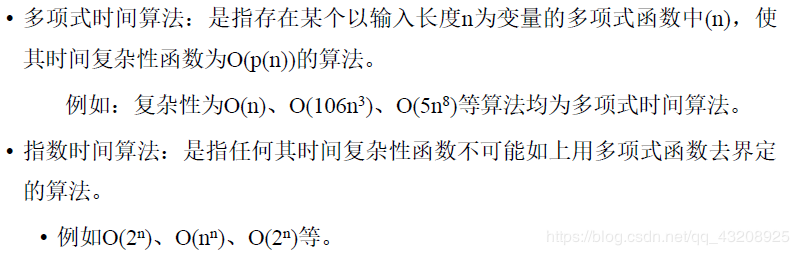
递归，就是函数自己调用自己，这个过程往往特别心酸。  




分治法是数据结构中用到的方法，把大问题分成很多小块。

**计算的复杂性：运用的初等运算的步数**

**时间复杂度**

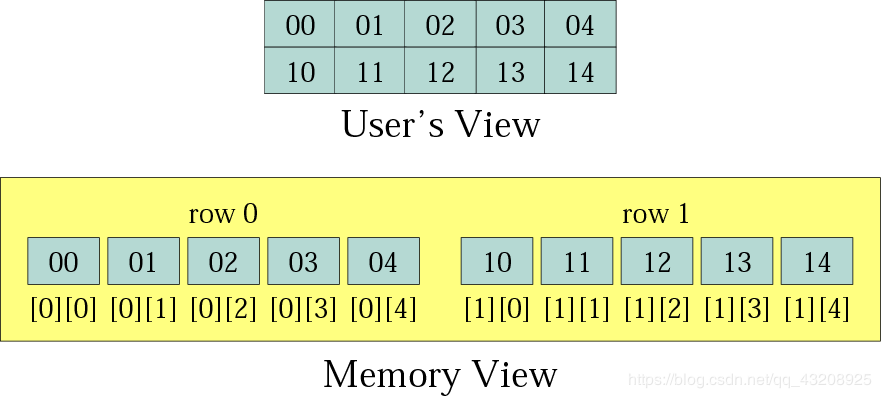
  
•O(1)称为常数级；  
•O(log n)称为对数级；  
•O(n)称为线性级；  
•O(nc)称为多项式级（C为常数）；  
•O(Cn)称为指数级（C为常数）；  
•O(n!)称为阶乘级

**空间复杂度**

临时占用了多少存储空间

**数据结构**

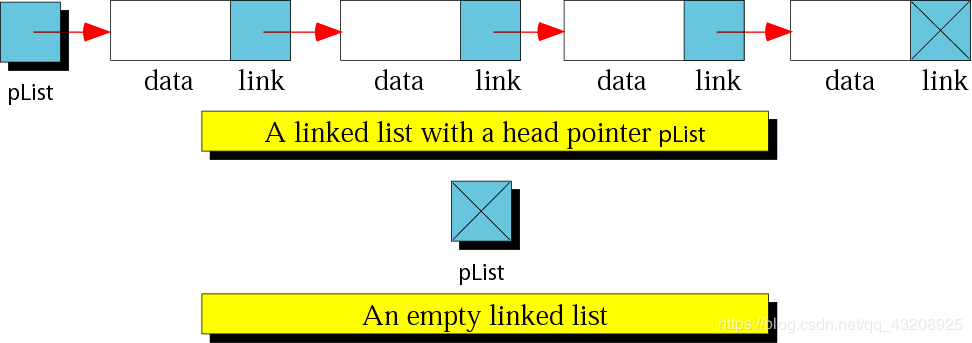
**数组 array**

什么是数组？  
首先，顾名思义，一组数字。数组放到了连续的一块儿上，而且还有顺序。这就是数组。  
数组可以实现批量处理数据，所以很有优势。  
数组分为一维和二维的，当然还有更多。对于二维数组，在内存中是怎么存储的呢？  
逐行存储。数组对于行来说，内存是连续的。  
**数组一旦确定，大小不能被改变！！**  
  
逐行存储。

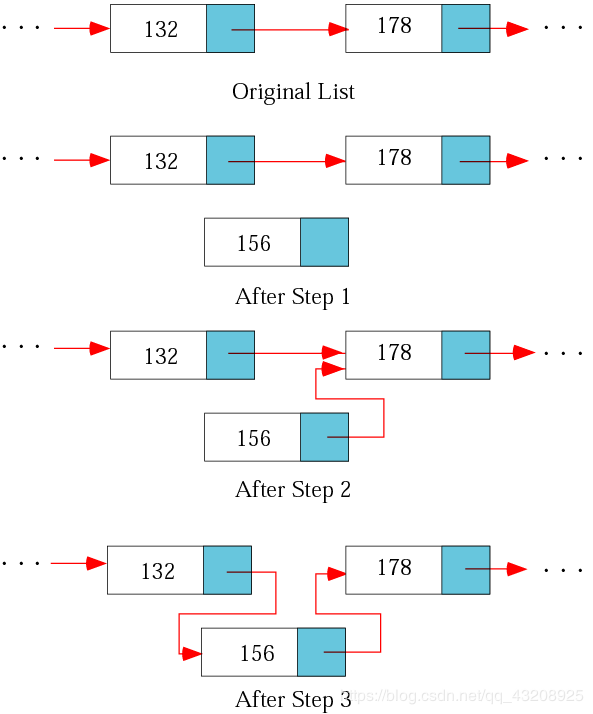
**链表 linked list**

什么是链表？顾名思义，不就是链接在一起形成的表嘛。  
既然我们有了数组存放数据，那么为什么会出现链表？看起来还更麻烦一些呢……  
数组有一个巨大的缺陷，那就是不能够改变大小。可是我一开始不知道要存储多少数据，怎么办？  
只能先让这个数组变得很大很大，大到能够满足我的需要。  
但是，我怎么知道有多大？还有，一开始开辟了很大的内存空间，一直等着给他赋值。不去赋值的话，那些空间不久白白浪费掉了？  
所以，我需要一个办法，能够改变长度，而且不造成资源的浪费。  
**链表**就是一种解决方案。

**链表的定义**

是一个有序的集合，其中每一个元素都包含下一个元素的地址，由数据和指针（地址）构成。  
  
图中每一个方块叫做一个节点——基本的单位。  
每个节点包括两个东西：一，我要存储的数据；二，下一个元素的地址，也就是下一个元素在哪儿。

**对链表的操作**

* 插入节点  
  为新节点分配内存、写入数据  
  使新节点指向后继节点，使前一个节点指向新节点。  
  
* 删除节点  
  定位要删除的节点  
  将前一个的指针指向被删除的节点的后继节点  
  之后可以释放掉被删除节点的内存空间  
  注意：删除第一个节点或仅有的单个节点时要特殊处理。比如删除最后一个的时候需要让前一个指向NULL，删除仅有的一个的时候，需要让Header直接指向NULL。
* 遍历链表  
  需要一个叫做Walker的东西，从Header出发，之后沿着指针走。  
  这种遍历只能从第一个开始，一直到最后一个。

**链表在内存中存储可以不连续**

因为有了指针，解决了定位的问题，这样对于内存可以更加了灵活地使用。

**抽象数据类型 ADT**

咱们用线性表，用树，用图多了，用的多了就会被抽象出来，这就是ADT。  
人们经过抽象出一类类具有一定固定模式的数据表示法，将其模型化，形成公共模板功能，供大家使用，使用者只需知道一个数据类型能做什么，而不必知道其怎样做。  
结构：数据生命、操作声明、封装数据和操作  
封装，就是把它们实现到我的程序中。

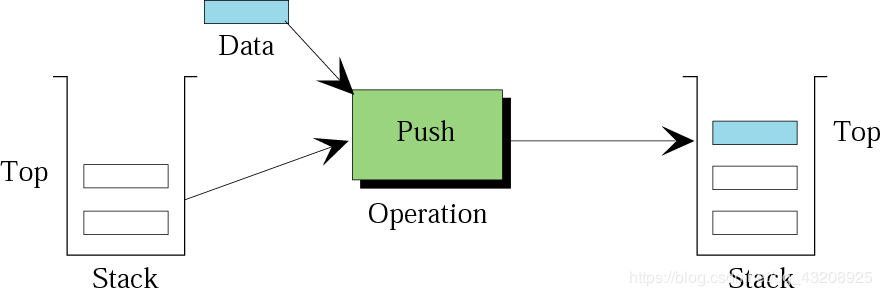
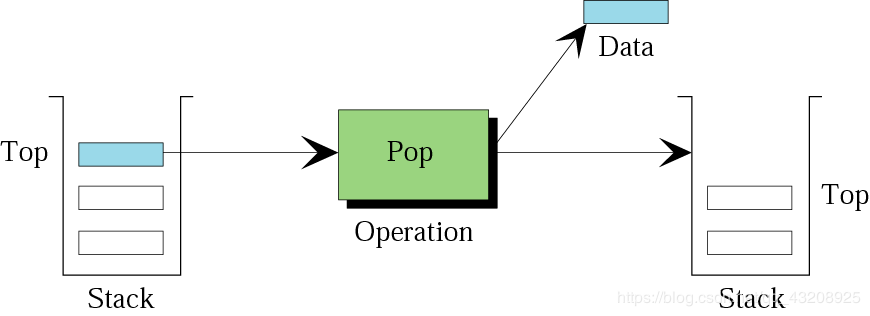
**常见：线性表 树 图**

**线性表**

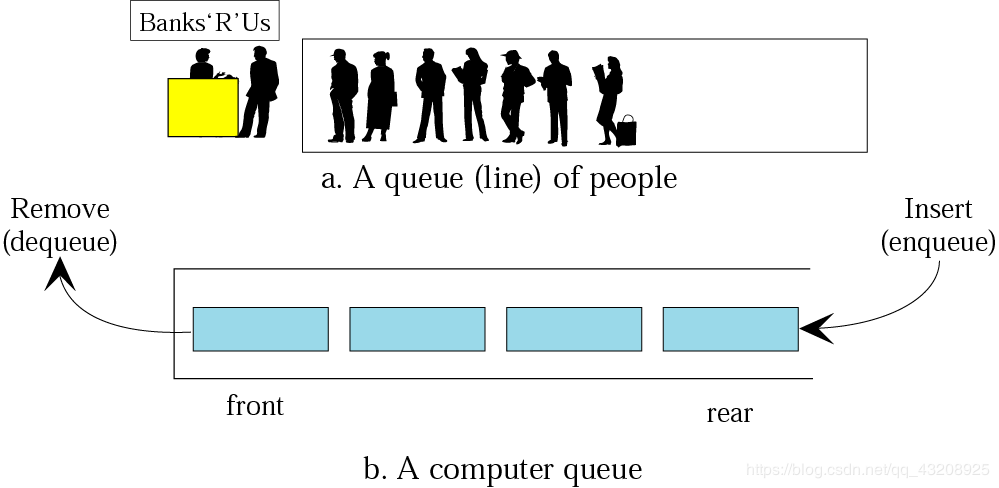
插入 删除 检索 遍历

**栈——干草堆**

限制线性表 只能在一端实现添加删除 LIFO 后入先出  
（怎么理解，很难？或许，可以这样理解——）  
栈的英文是stack，本义是干草堆。  
干草堆啥样？我先放进去的干草在最底下，我之后放上去的干草放在最上面。  
那么，如果我现在想要拿一些干草走，我最先拿到的是那一部分？最先放上去的还是最后放上去的？  
最后啊！  
这就是后入先出的字面解释。  
最后进去的，最先被拿出来。  
这样一说，是不是理解了很多呢？  
**栈的操作**

* Push 压栈\入栈  
  
* Pop 出栈
* 

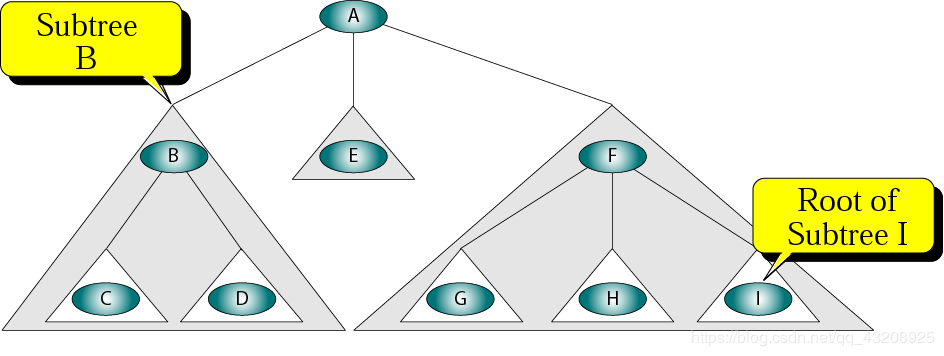
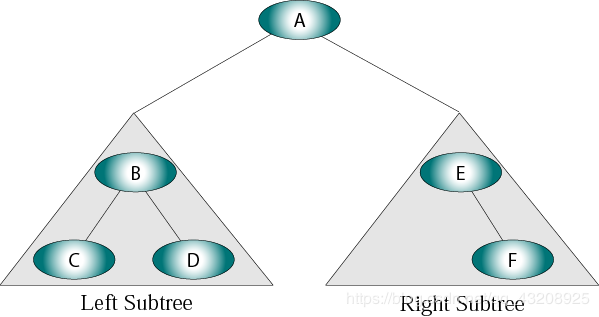
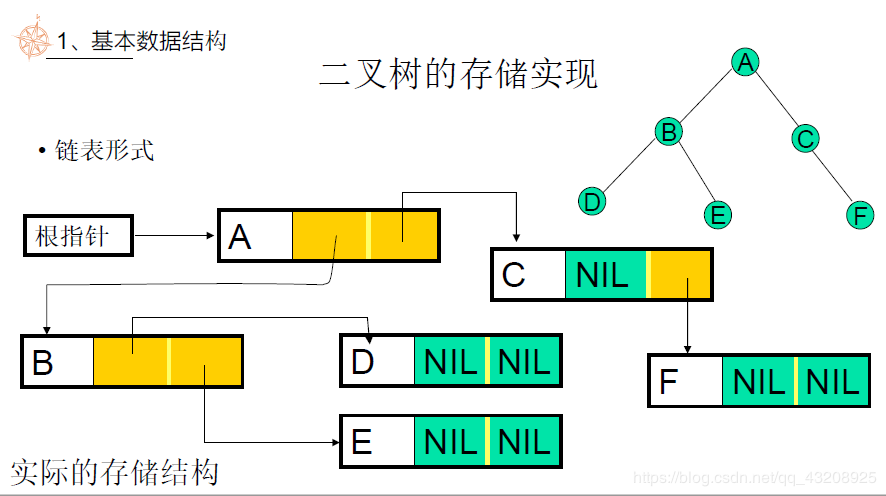
**队列——排队买饭**

两端都需要操作，但是只能一端进，另一端出。 FIFO  
  
**队列的操作**：

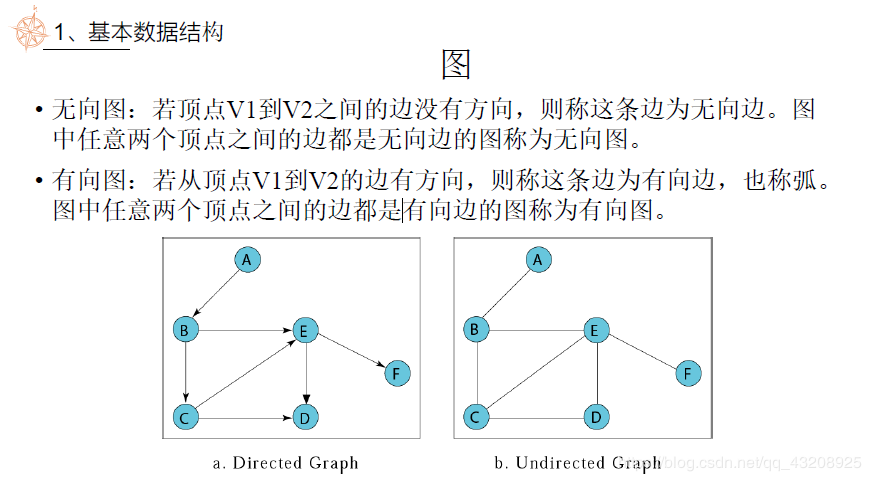
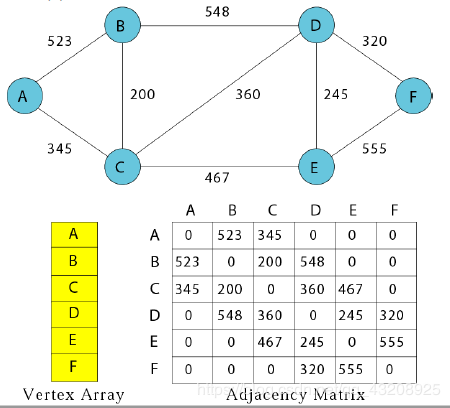
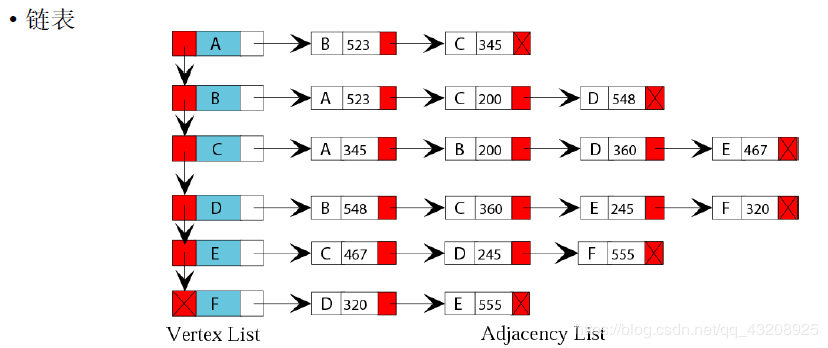
* 入列 Enqueue
* 出列Dequeue
* 空

**树——北林最离不开的东西**

**内容**：

* 节点：组成树的有限元素
* 分支：连接节点的有向线段
* 度：与节点连接的分支数目。指向节点：入度，离开节点：出度。
* 子树：根节点下面还有可以形成树的东西  
    
  **二叉树**：  
  每个节点最多左右两个子树。  
    
  **用链表实现二叉树**：  
  每个节点包括三个东西：存储的数据，左子树的地址，右子树的地址。  
  不完全存储会浪费内存空间。  
  

**图——关系不再单纯**

  
存储实现：邻接矩阵  
  


**高级程序设计语言**

**历史：机器语言——汇编语言——高级语言**

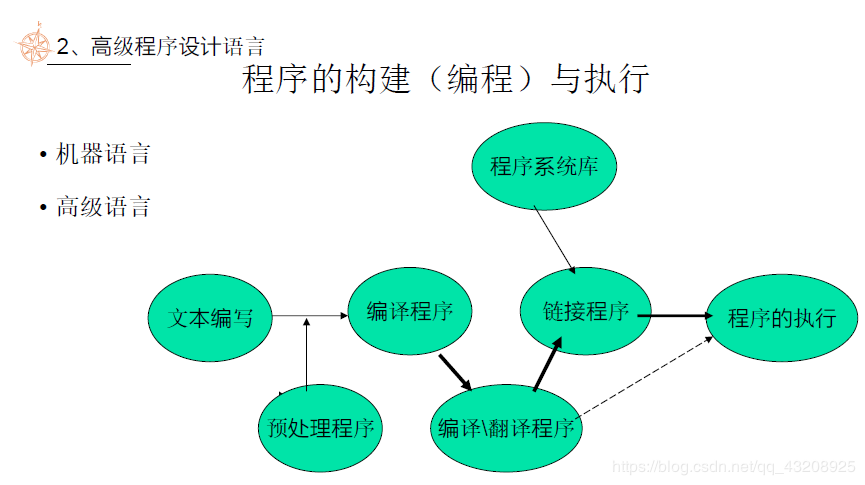
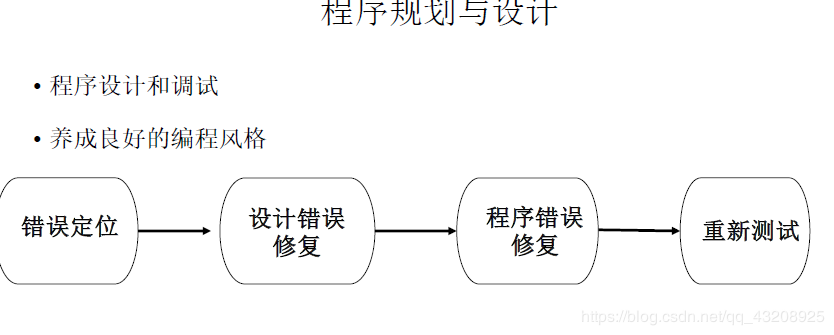
**程序规划与设计的五个步骤**

1.分析问题并制定概要设计方案。准确了解、用准确的语言描述问题。  
2.制定详细设计。（算法设计）精确的步骤，明确、详细、有限、在合理时间内完成，选定语言。  
3.用编程语言编写程序代码和文档。需要加上注释。  
4.测试程序。确保运行正确。前几步中不断重复。每部分都要测试。  
5.验证程序。大范围测试，根据用户需要，不断修改调整，直到用户满意。

**软件工程**

在大型的软件开发中，引入工程管理的一整套的管理方法，对软件开发过程进行规划、设计、监控和检测，以确保开发的过程、开发时间和软件质量都在人的控制管理之下，从而使软件开发的顺利进行。

**程序设计理论**

通过对程序设计的各种问题进行了系统研究，进行了规范总结。  
自顶向下逐步求精  
自底向上的程序设计方法  
  


**算法案例，拓展一下脑洞**

一个拿着7个金环组成的链子的旅行者需要在一个酒店里住7夜。每一夜的租金是金链中的一环。应该如何对链子进行最少次数的切割，使旅行者每天早上支付酒店的一环而不用提前支付旅费？  
最优解：切一次

