1. 引言

1. **算法的定义**

算法是一个有穷的指令集，这些指令为解决某一特定任务规定了一个运算序列。

**特点：**

**有穷性** 算法应在执行有穷步后结束；

**有效性** 每一条运算都是可行的；

**确定性** 每步定义都是确切无歧义的；

**输入** 有0个或多个输入；

**输出** 有一个或多个输出(处理结果)；

2．**算法的描述**

一般分为三层：

（1）基于模型的抽象层（自然语言，抽象伪代码）

（2）基于数据的逻辑结构层（伪代码，API）

（3）基于具体的数据存储层 （伪代码，语言）

3. **算法的时空复杂度（判断算法好坏的条件）**

通常有以下几个标准，但这不是绝对的。

**正确性 工作量（时间复杂度） 占用空间量（空间复杂度） 简单性和清晰性 最优性**

**4. 算法分析的目的**：

* 对算法作出一个客观的评价；
* 比较两个算法，选择算法；
* 改进算法；

5. **算法设计的基本步骤**

（1）**分析问题**：理解问题的本质、约束条件、内在规律、规模、输入、输出、结果表达形式、重要例外等

（2）**建立模型**：根据问题分析结果，建立合理的模型，模型一般有：数学模型（方程式、数学公式）、有限状态机、Petri网、图、树

（3）**设计算法**：采用逐步求精法，由粗到细、由高层到低层、由抽象到具体。

对算法进行时间与空间复杂性分析。

（4）**实现算法**：设计抽象数据结构：必要的话设计抽象数据结构；

设计和实现数据的逻辑结构和存储结构：设计具体的合适的逻辑结构和存储结构；

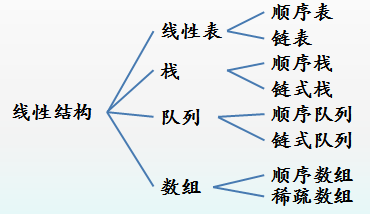
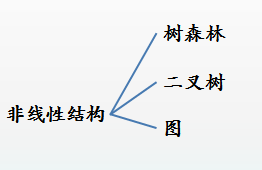
实现算法：基于具体的数据存储结构，实现算法；

（5）**测 试**：设计测试例（测试数据），对算法的实现（程序）进行测试，发现其中的错误，进行修正。反复进行。

6. **抽象数据类型与基本数据结构**

抽象数据类型： (1) 数学模型 (2) 定义在该模型上的一组操作

基本数据类型：

第二章：排序

**1 概述：**

**排序：**将一组数据对象按某一域值顺次排列起来。

**数据表：**它是待排序数据对象的有限集合**。**

**排序码：**通常数据对象有多个属性域，即多个数据成员组成，其中有一个属性域可用来区分对象，作为排序依据。该域即为排序码。任一域都可做排序码。

**主关键码:** 如果在数据表中各个对象的关键码互不相同，这种关键码即主关键码。按照主关键码进行排序，排序的结果是唯一的。

**次关键码:** 数据表中有些对象的关键码可能相同，这种关键码称为次关键码。按照次关键码进行排序，排序的结果可能不唯一

**排序算法的稳定性:（排序前后关键码相同的数的相对位置不变）**

如果在对象序列中有两个对象*r*[*i*]和*r*[*j*]，它们的关键码 *k*[*i*] *== k*[*j*]，且在排序之前，对象*r*[*i*]排在*r*[*j*]前面。如果在排序之后，对象*r*[*i*]仍在对象*r*[*j*]的前面，则称这个排序方法是稳定的，否则称这个排序方法是不稳定的。

**内排序与外排序:**

内排序是指在排序期间数据对象全部存放在内存的排序；

外排序是指在排序期间全部对象个数太多，不能同时存放在内存，必须根据排序过程的要求，不断在内、外存之间移动的排序。

**排序的时间复杂度:**

排序的时间开销是衡量算法好坏的最重要的标志。排序的时间开销可用算法执行中的数据比较次数与数据移动次数来衡量。各节给出算法运行时间代价的大略估算一般都按平均情况进行估算。对于那些受对象关键码序列初始排列及对象个数影响较大的，需要按最好情况和最坏情况进行估算。

**空间复杂度:** 评价算法好坏的另一标准。

**静态排序**: **（物理地址改变，一般用数组存储）** 排序的过程是对数据对象本身进行物理地重排，经过比较和判断，将对象移到合适的位置。这时，数据对象一般都存放在一个顺序的表中。

**动态排序**: **（物理地址不变，一般通过指针操作）**  给每个对象增加一个链接指针，在排序的过程中不移动对象或传送数据，仅通过修改链接指针来改变对象之间的逻辑顺序，从而达到排序的目的。

**2 简单排序算法**

**2.1直接插入排序**

**算法思想：**

待排序序列为r[1],r[2], ….. , r[n];

1. 把r[1]看作是排好序的序列；

2. 将r[2], ….. , r[n]按如下方法依次插入排好序的子序列中：

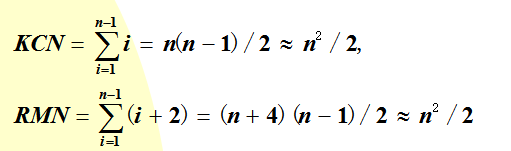
当插入r[i]时， r[1], ….. , r[i-1]已排好序，在其中找一合适位置j，将r[i]插入。



**算法分析：**

最好情况：排序前对象已经按关键码大小从小到大有序，每趟只需与前面的有序对象序列的最后一个对象的关键码比较 1 次，移动 2 次对象，总的关键码比较次数为 *n*-1，对象移动次数为 2(*n*-1)。

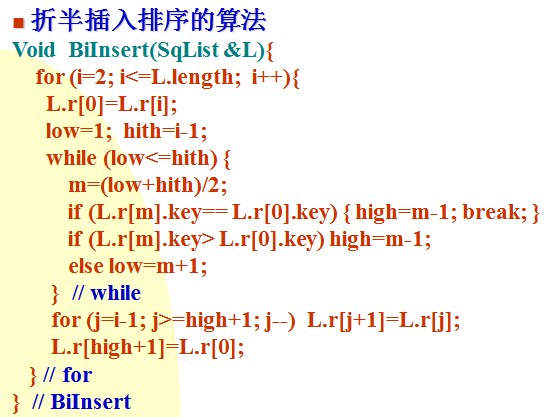
最坏情况**：**最坏情况下，第 *i* 趟时第 *i* 个对象必须与前面 *i* 个对象都做关键码比较，并且每做 1 次比较就要做 1 次数据移动。则总的关键码比较次数*KCN*和对象移动次数*RMN*分别为



**2.2折半插入排序（？？？）**

**算法思想**

1. 当插入第*i* (*i* ≥ 1) 个对象时，前面的r[1], …, r[*i*-1]已经排好序。
2. 利用折半搜索法寻找 r[*i*] 的插入位置j;
3. 将r[*i*]插入到位置j处， r[j],…, r[*i*-1]向后顺移。



**算法分析**

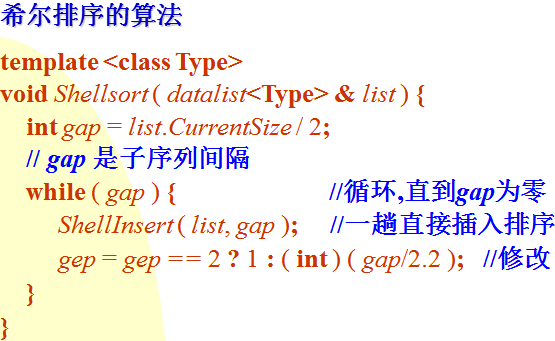
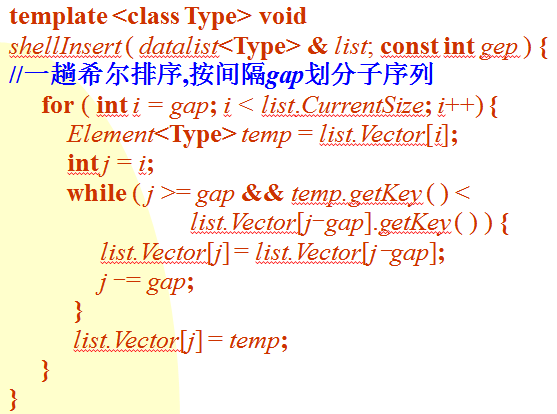
1. 当 *n* 较大时，总关键码比较次数比直接插入排序的最坏情况要好得多，但比其最好情况要差。
2. 在对象的初始排列已经按关键码排好序或接近有序时，直接插入排序比对分插入排序执行的关键码比较次数要少。对分插入排序的对象移动次数与直接插入排序相同，依赖于对象的初始排列。
3. 折半插入排序是一个稳定的排序方法

**3 希尔排序与快速排序**

* 1. **希尔排序（缩小增量排序）**

**基本思想：**

设待排序对象序列有 *n* 个对象，首先取一个整数 *gap* < *n* 作为间隔，将全部对象分为 *gap* 个子序列，所有距离为 *gap* 的对象放在同一个子序列中，在每一个子序列中分别施行直接插入排序。然后缩小间隔 *gap*，例如取 *gap* = ⎡*gap*/2⎤，重复上述的子序列划分和排序工作。直到最后取 *gap ==* 1，将所有对象放在同一个序列中排序为止。

**3.2交换排序**

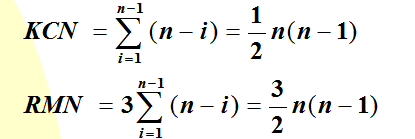
**3.2.1 冒泡排序**

**算法思想：**设待排序对象序列中的对象个数为 *n*。最多作 *n*-1 趟，*i* = 1, 2, …, *n*-2。在第 *i* 趟中顺次两两比较*v*[*n*-*j*-1].*Key*和*v*[*n*-*j*].*Key*，*j* = *n*-1, *n*-2, …, *i*。如果发生逆序，则交换*v*[*n* -*j*-1]和*v*[*n*-*j*]。

**算法分析**

在对象的初始排列已经按关键码从小到大排好序时，此算法只执行一趟起泡，做 *n*-1 次关键码比较，不移动对象。这是最好的情形。

最坏的情形是算法执行了*n*-1趟起泡，第 *i* 趟 (1≤ *i*< *n*) 做了 *n- i* 次关键码比较，执行了*n-i* 次对象交换。这样在最坏情形下总的关键码比较次数*KCN*和对象移动次数*RMN*为：



**3.2.2选择排序（？？？）**

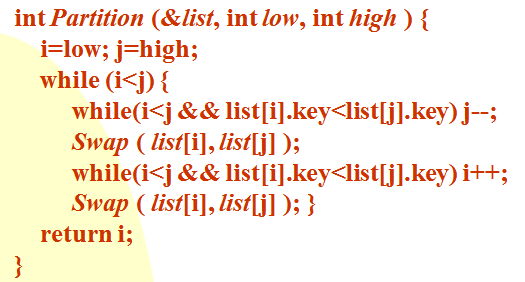
**算法思想：**快速排序方法的基本思想是任取待排序对象序列中的某个对象 (例如取第一个对象) 作为基准，按照该对象的关键码大小，将整个对象序列划分为左右两个子序列：

左侧子序列中所有对象的关键码都小于或等于基准对象的关键码

右侧子序列中所有对象的关键码都大于基准对象的关键码

基准对象则排在这两个子序列中间(这也是该对象最终应安放的位置)。

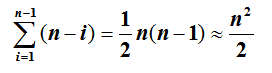
然后分别对这两个子序列重复施行上述方法，直到所有的对象都排在相应位置上为止。

**算法分析**：快速排序是递归的，需要有一个栈存放每层递归调用时的指针和参数。

最大递归调用层次数与递归树的深度一致，理想情况为 ⎡log2(*n*+1)⎤ 。因此，要求存储开销为 o(log2*n*)。

在最坏的情况，即待排序对象序列已经按其关键码从小到大排好序的情况下，其递归树成为单支树，每次划分只得到一个比上一次少一个对象的子序列。这样，必须经过 *n*-1 趟才能把所有对象定位，而且第 *i* 趟需要经过 *n*-*i* 次关键码比较才能找到第 *i* 个对象的安放位置，总的关键码比较次数将达到



**快排改进方法**：

**1.选取合理的分割点：**如，取每个待排序对象序列的第一个对象、最后一个对象和位置接近正中的3个对象，取其关键码居中者作为基准对象。

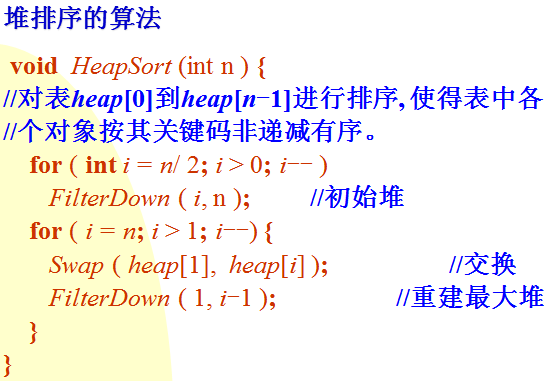
**2.使用栈而不使用递归：**把子序列的上下界存入栈中，

**3.改变子序列的排序次序：**先对长度较小的子序列进行排序，然后再对子序列较大的子序列进行排序。

**4 归并排序与堆排序**

**4.1堆排序：（？？？）**

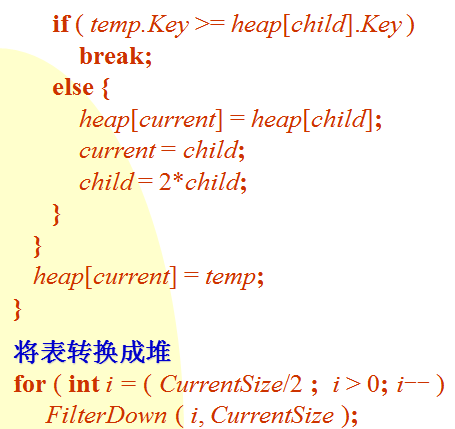
**算法思想：**第一步，根据初始输入数据，利用堆的调整算法 *FilterDown*( ) 形成初始堆，第二步，通过一系列的对象交换和重新调整堆进行排序。

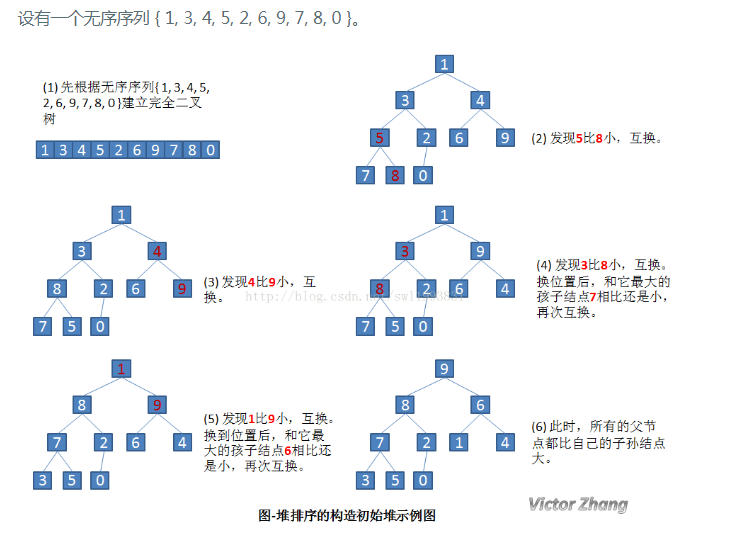


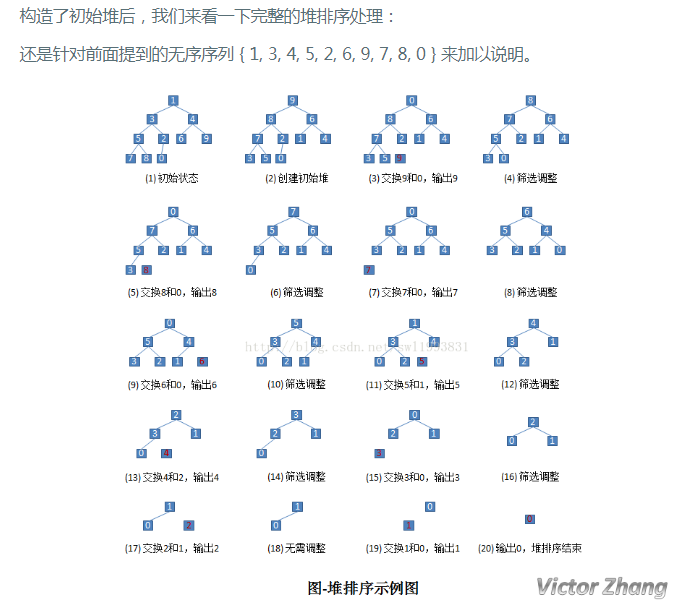
**算法分析：堆排序的时间复杂性为O(*n*log2*n*)**

**该算法的空间复杂性为O(1)**

**堆排序是一个不稳定的排序方法。**





**4.2归并排序**

**算法思想：** 归并，是将两个或两个以上的有序表合并成一个新的有序表。



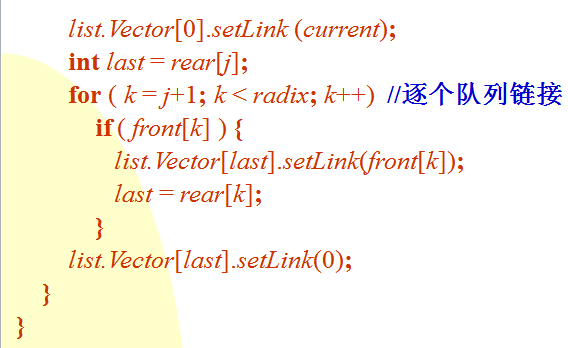
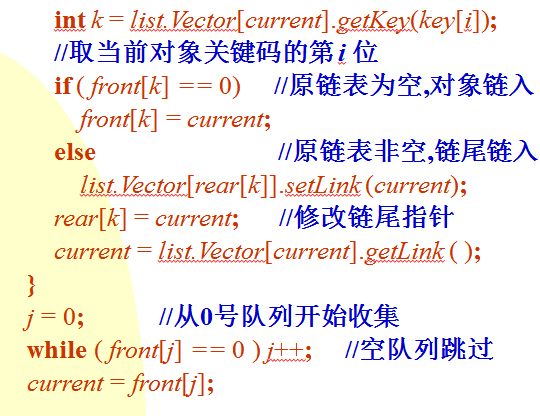
**算法分析：**算法总的时间复杂度为O(*n*log2*n*)。

**4.3 基数排序（非比较类排序）**

**实现多关键码排序有两种常用的方法**

**最高位优先MSD**

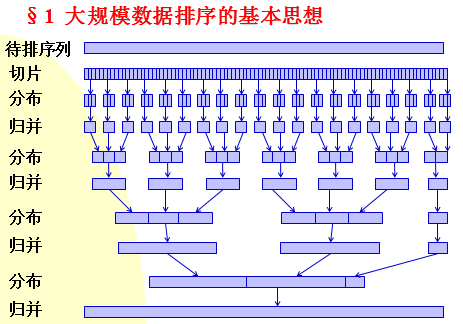
**最低位优先LSD**



**5 排序算法的分析与比较**



**6 大规模数据的排序**



**切片：**待排序列是一个或若干个大型文件

将待排序列分割成n个片段（Segment）,每个片段内存可容。（S1,S2，…, Sn）

将片段分配给各节点，进行排序，形成有序的片段，成为初始归并段。

**分布：**每个节点分配k个初始归并段；

各节点并行进行k路归并

**归并**：归并段储存在本地外存中；

内存中设k+1个缓冲区：

每个归并段对一个缓冲区，1个输出缓冲区

归并算法将k个归并段归并为一个归并段

**排序的时间复杂度**

有三种操作时间：

Tst :数据元素从一个节点传送到另一个节点的平均时间；（最主要关注）

TIO：本地内、外存间传输数据数据元素的平均时间；

Tc:归并算法中元素比较的时间；

在做2路归并排序时，第一趟从 *m* 个初始归并段得到 ⎡*m*/2⎤ 个归并段，以后各趟将从 *l* (*l* >1) 个归并段得到 ⎡*l*/2⎤ 个归并段。总归并趟数等于归并树的高度 ⎡log2*m*⎤；

每趟归并，每个数据元素需要传输2次，1次输入，1次输出；

根据 2 路归并树, 估计 2 路归并排序数据元素传输的总次数是：***tES*=2n ⎡log2m⎤**

**如果是k-路归并，则归并趟数是**⎡**logk*m***⎤**；**

**显然，k越大，归并趟数越少， *tES*越小；**

**但是 ，一个节点的内存是有限的，缓冲区就有限，k的大小要根据内存大小来确定。不能无限制大；**

**K路平衡归并**

**做 *k* 路平衡归并时，如果有 *m* 个初始归并段，则相应的归并树有 ⎡log*km*⎤ +1 层，需要归并⎡log*km*⎤ 趟。**

**（k路归并多看一下）**

**最佳归并树**

**定义**：归并树是描述归并过程的 *m* 叉树。因为每一次做 *m* 路归并都需要有 *m* 个归并段参加，因此，归并树是只有度为0和度为 *m* 的结点的正则 *m* 叉树。

**归并树的带权路径长度 *WPL* 即为归并过程中的总读对象数。因而，在归并过程中总的读写对象次数为 2\**WPL* = 754。**

第三章 查找

**集合**：集合是成员(对象或元素)的一个群集。

**并查集：**建立等价类的另一种解决方案是先把每一个对象看作是一个单元素集合，然后按一定顺序将属于同一等价类的元素所在的集合合并。在此过程中将反复地使用一个查找运算，确定一个元素在哪一个集合中。能够完成这种功能的集合就是并查集。

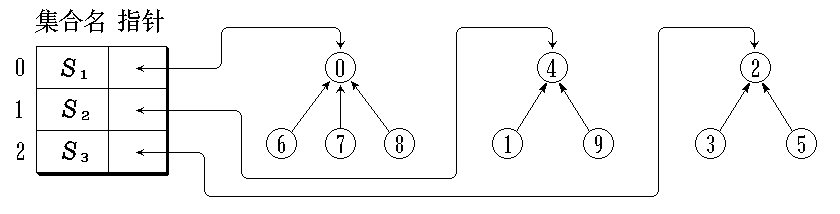
**支持操作**：▪ *Union* (*Root*1, *Root*2) //并操作；

▪ *Find* (*x*) //查找操作；

▪ *UFSets* (*s*) //构造函数

**涉及数据类型**：**集合名类型**和**集合元素的类型**。

对于并查集来说，每个集合用一棵树表示。



**利用并查集来解决等价问题的步骤如下**

* 利用*UFSets*操作, 建立*UFSets*型集合this, 集合中每一个元素初始化为0，各自形成一个单元素子集合, *i =*1, 2, …, *n*。*n*是集合中元素个数。
* 重复以下步骤, 直到所有等价对读入并处理完为止。
  + 读入一个等价对[*i*][*j*];
  + 用*Find*(*i*), *Find*(*j*)查找 *i*、*j* 所属子集合的名 字*x*和*y*;
  + 若*x* ≠ *y*. 用 *Union*(*x*,*y*) 或 *Union*(*y,x*) 将它们合并, 前者的根在 *x*；后者的根在 *y*。

**查找概念**：所谓查找，就是在数据集合中寻找满足某种条件的数据对象。

**查找结果：查找成功：**即找到满足条件的数据对象。这时，作为结果，可报告该对象在结构中的位置，还可进一步给出该对象中的具体信息。

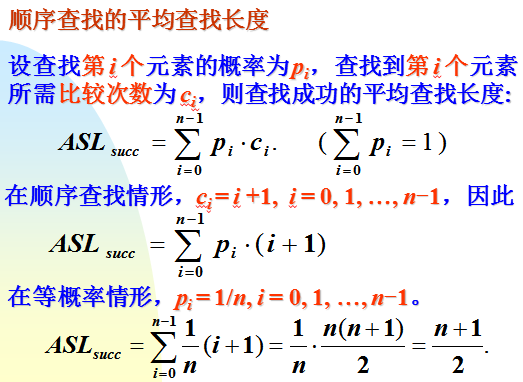
**查找不成功：**或查找失败。作为结果，也应报告一些信息，如失败标志、失败位置等。

**查找的环境：静态查找：**查找结构在执行插入和删除等操作的前后不发生改变。⎯ 静态查找表

**动态查找：**保持较高的查找效率，查找结构在执行插入和删除等操作的前后将自动进行调整，结构可能发生变化。⎯ 动态查找表

**顺序查找：**主要用于在线性结构中进行查找。从表的先端开始用关键码和给定值比较，若找到返回查找成功，若没找到返回查找失败。

**顺序查找的平均查找长度（ASL）**



**折半查找：**

1 设*n*个对象存放在一个有序顺序表L中，并按其关键码从小到大排好了序。查找范围为l=0, r=n-1;

2 求区间中间位置*mid=（l+r）/2*

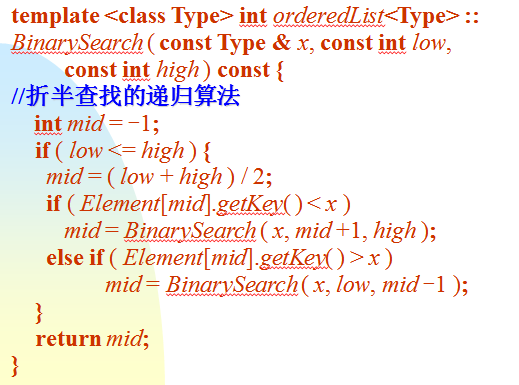
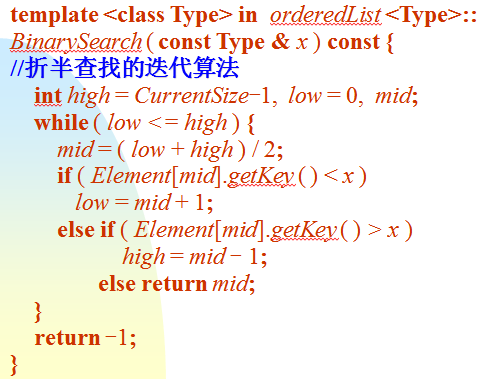
3 比较：

L[*mid*].*Key* = *x*，查找成功，结束；

*L*[*mid*].*Key*> *x*，r=mid-1;

L[*mid*].*Key*< *x*，l=mid+1;

4 若l<=r 转2，否则查找失败；

**折半查找的判定树：**

1．有2n+1个结点， 高度不超过完全二叉树。最多的比较次数不超过二叉树的高度

2. 最坏情况下的查找长度是

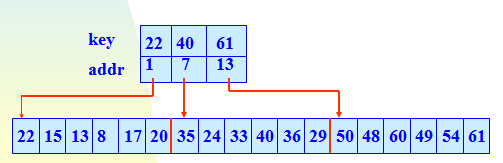
3. 查找成功的平均查找长度： 设查找任一记录的概率都相等，即：pi=1/n 在第k层上最多有2k-1 个结点，在第k层上的结点需比较k次。



**索引顺序表的查找(分块查找):**

**索引项： （key,addr）, key 是关键字，addr是地址。**

**索引表：有索引项构成的顺序表。**



**分块查找的数据结构：**

D={d1,d2,….,dn}

1.将数据集分为s个组 B1,B2, …, Bs ；

2.组间有序：当i<j时，Bi中的任一元素小于Bj中的任一元素；

3.为每个Bi组设一索引项(keyi, addri)；

4.s个索引项构成索引表；

5.组内可有序可无序;

**查找:**

1.首先在索引表中查找，确定在哪个块中。（可以是折半查找，也可以是顺序查找。）

2.在块中查找。

算法分析：

设在索引表中和块内都是顺序查找。

索引表中查找：ALSindex=(s+1)/2

块中查找： ALSlock=(n/s+1)/2

所以： ALS= (s+1)/2 + (n/s+1)/2

显然它与s 的取值有关。

**二叉查找树**

**定义**：

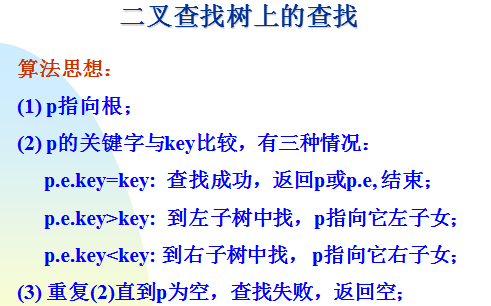
二叉查找树或者是一棵空树，或者是具有下列性质的二叉树：

▪ 每个结点都有一个作为查找依据的关键码(key)，所有结点的关键码互不相同。

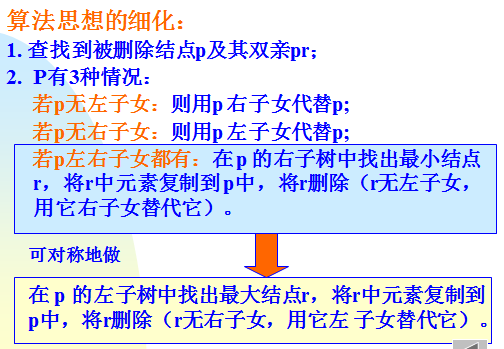
▪ 左子树(如果存在)上所有结点的关键码都小于根结点的关键码。

▪ 右子树(如果存在)上所有结点的关键码都大于根结点的关键码。

▪ 左子树和右子树也是二叉查找树。



**二叉查找树的删除算法**



**二叉查找树分析：**

**最坏情况下：**

**n个元素可构成的二叉查找树有**：

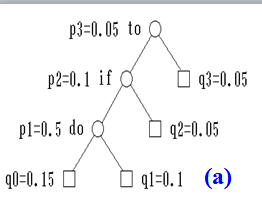
**P(n)是对具有n个结点的二叉排序树的平均查找长度；**

**最优二叉查找树：**

扩充二叉查找树总的平均查找长度为： *ASL* = *ASLsucc* + *ASLunsucc*

1. **相等查找概率的情形（？？？）**

**若设树中所有内、外部结点的查找概率都相等： p[*i*] = q[*j*] = 1/7，1** ≤ ***i*** ≤ **3, 0** ≤ ***j*** ≤ **3**



**图(a)： *ASLsucc* = 1/7\*3+1/7\*2+1/7\*1 = 6/7, *ASLunsucc* = 1/7\*3\*2+1/7\*2+1/7\*1 = 9/7。**

**总平均查找长度 *ASL* = 6/7 + 9/7 = 15/7。**

**AVL树的定义**

一棵*AVL*树或者是空树，或者是具有下列性质的二叉查找树：它的左子树和右子树都是*AVL*树，且左子树和右子树的高度之差的绝对值不超过1。

*AVL*树：如果它有 *n* 个结点，其高度可保持在O(log2*n*)，平均查找长度也可保持在O(log2*n*)

**哈希表**



**构造哈希函数**

**原则**

* 值在值域中分布均匀（减少冲突）
* 计算简单
* 值域落在表地址范围内；

**常用方法：**

1 直接定址法： h(key)=a\* key + b（a,b为常数）

2 数字分析法：对关键字集要有充分的了解。

表地址为： s…t, (t-s+1)是r位；取分布较均匀的r位，组成addrH(key)=addr\*t/10r + s

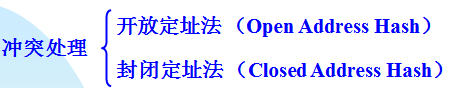
3 平方取中法：H(key)=取(key)平方的中间r位；r取决于表长；

4 折叠法与位移法

5 除留余数法： H(key)=key MOD p p<=m , m 是表长；

p的选择是很重要的：通常取p为小于等于m的质数；

6 随机数法：选择一个随机函数random()，H(key)=random(key); 通常key的长度不等时易采用此方法。



**封闭定址法：**

**Hash表链表数组，地址空间是H[0….n-1];Hash表单元H[i]是一个链表，存放若干数据元素，他们是同义词;**

**它的基本思想是： 将同义词防在key的hash 码对应的链表中；**

**Class ElemNode {**

**KeyType key**

**ElemNode next;**

**public:**

**ElemNode(KeyType K) { key=K; next=NULL;}**

**}**

**Class HashTable {**

**ine size;**

**ElemNode H[];**

**Public:**

**HashTable(int s) {size=s; H=new ElemNode[s]; }**

**int hash(KeyType K) {…}**

**int search(KeyType K) {..}**

**boolean insert(KetType K) {…}**

**boolean delete (KetType K) {…}**

**void reCreate(int size) {…}**

**} // HashTable**

**查找：对给定的key，**

**计算hash码 i=H(key)；**

**在链表H[i]中顺序查找key；**

**查找算法分析：**

计算hash码，很少的工作量，是常数 a；在链表H[i]中顺序查找，（Li+1）/2, Li是该链表长度；

所以 

**开放定址法**

**Hash表是数组，地址空间是H[0….n-1];**

**Hash表单元H[i]只存放一个数据元素;**

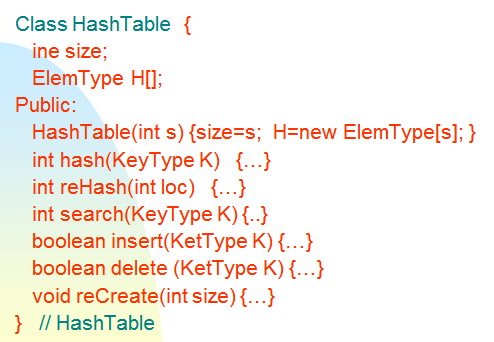
**它的基本思想是：**

**如果一个元素的Hash地址对应的Hash单元已被另一个元素占有（冲突），我们需定义一个候选地址序列，每当发生冲突时，选择下一个地址去试探。这个序列称为探测序列**

**Key的探测序列：d0, d1, ……, dm-1, m是表长；**

**d0=H(key)**

**d1, ……, dm-1,需要我们定义， 计算d1, ……, dm-1,的过程称为rehash(再Hash);**



**再哈希的方法：**

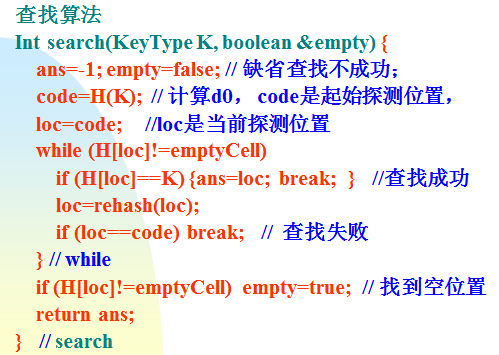
1. **线性探测再散列（最简单）**
2. **随机探测再散列**

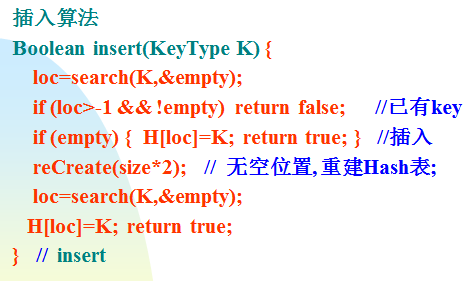
**di=(di-1+p) mod m p是一质数；**

**di=(di-1+hashk(key)+1) mod m**

**di=(di-1+(-1)i i) mod m**

**哈希表算法：**





**删除解决方案**

**解决办法:**

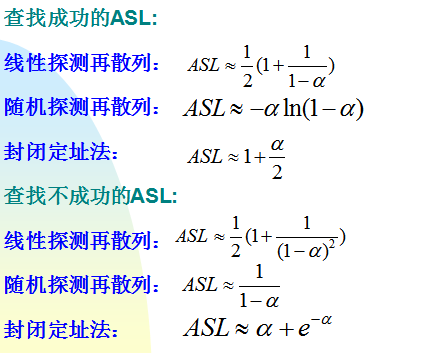
**删除一个元素后,在其位置上做一删除标记 delete；查找时遇到delete，认为是冲突；插入时遇到delete，认为是空，可以插入；**

**算法分析:**

**负载因子（Load factor）α=n/m**

**其中n是表中元素个数， m是表长；**

**显然， α越大，冲突的可能性越大，从而ASL越大。**



**B-树和B+树（明白概念即可）**

**B-树：**B-树是一种**多路平衡查找树**，它在文件系统中做**文件索引**，很有用。

**定义：**

m阶B-树：

或是空树，或是满足以下特性的m叉树：

（1）树中每个结点最多有m个子树；

（2）若根不是叶，至少有2个子树；

（3）除根外的非叶结点至少有[m/2]个子树；

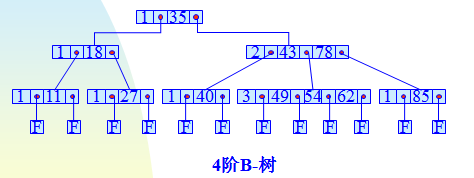
（4）所有非叶结点包含以下信息

(n, A0, K1,A1, K2, A2, …, Kn, An)

其中Ki是关键字，且 Ki < Ki+1, Ai, 是第i子树，它的所以结点均大于Ki, 小于Ki+1



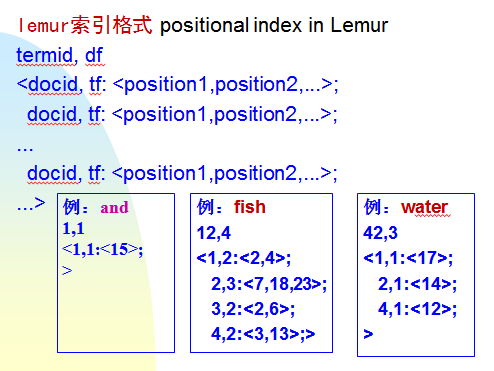
(5)所有叶都在同一层。且无信息。



**倒排索引及其压缩：**

所有现代搜索引擎都是基于倒排索引的，是最有效、最灵活的索引结构。

**倒排索引结构：**



And： 1,1 文档1中出现，出现频率1次

<1,1:<15>> 文档1中出现，出现频率1次，位置15

fish <1,2:<2,4>; 2,3:<7,18,23>;3,2:<2,6>;4,2:<3,13>;>

文档1,2,3,4中出现，频率分别为2,3,2,2，出现位置为文档1中2,4位置，文档2中7,8,23位置，文档3中2,6位置，文档4中3,13位置。

**索引构建：**

方法：

简单构建：内存可容纳整个postings

分块构建：内存容纳不了整个postings,分块构建若干个倒排表，然后合并他们。

内存单遍扫描构建索引法：Parse（Collection）形成token-stream（termid,docid）

分布式构建等方法：

1- 将collection 分为n个切片，大小一般为16M 或64M；

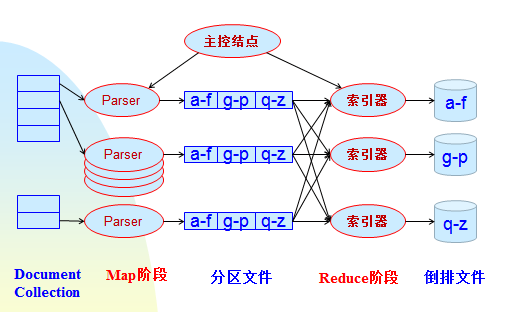
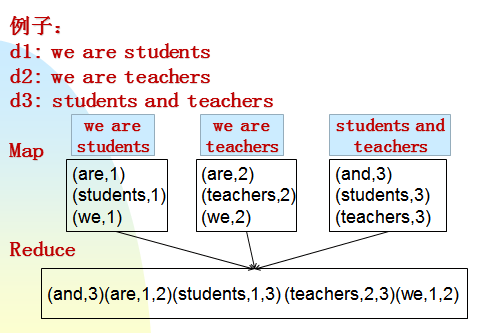
2- Map：将分不到各结点上，每个节点将数据片映射为 （term，docID）序列；并按term排序；

3- Reduce：将各结点上具有相同term 的项进行归约，集中存储。

主控结点：负责分配任务，负载均衡，宕机平移等

数据节点：存储数据

运算结点：执行程序，处理数据

第四章 递归（重点为递归的三要素）

**分治算法**

其核心思想是将大的问题分解成若干小的子问题进行处理，从而使复杂的问题变得简单。表现在算法中就是递归算法。在前面我们已设计了许多递归算法，如：Hanoi塔，快速排序，归并排序，二叉树遍历，dfs，等等。

**分治算法的适用情况**

**分治法所能解决的问题一般具有以下几个特征：**

1) 该问题的规模缩小到一定的程度就可以容易地解决

2) 该问题可以分解为若干个规模较小的相同问题，即该问题具有最优子结构性质。（前提）

3) 利用该问题分解出的子问题的解可以合并为该问题的解；（关键）

4) 该问题所分解出的各个子问题是相互独立的，即子问题之间不包含公共的子问题。（效率）

**分治算法通用模板**

Function DC(x)

if x<=最小问题规模 then return adhoc(x)

splite x into small sub\_problems x1..xk

for i=1 to k do yi=DC(xi)

combined y1..yk into y that is solution of x

return y

**递归算法的设计**

一般来说，如果一个复杂的问题能够被分解成若干个同类型的子问题，那么这个问题就可以用递归算法实现。在**设计递归算法**时，应该遵循下面的**规则**：

（1）定义一个最小规模的问题，并给出其解；

（2）把复杂的问题划分为同类型的若干规模较小的子问题，并分别解决子问题；

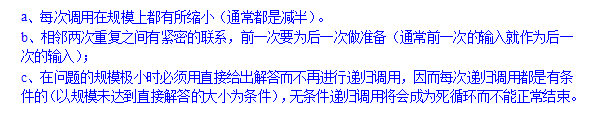
（3）把各子问题的解组合起来，即可得到原问题的解。

**递归算法的三要素**

1、明确递归终止条件；

2、给出递归终止时的处理办法；

3、提取重复的逻辑，缩小问题规模。



第五章 贪心算法

**贪婪算法的一般特性**

（1）优化问题，有一个候选对象的集合，如硬币，边，路径，顶点等。

（2）随着算法的进行，形成2个集合，一个是已经被选中的对象集合，另一个是被抛弃的对象集合

（3）有一个函数，检查候选对象集是否是问题的解，不一定是最优的。

（4）还有一个函数（solution function），检查候选对象是否可加入到当前解的对象集合中，不一定是最优的。

（5）选择函数（selection function），指出哪个剩余的候选对象最有可能构成问题的解。

（6）目标函数(object function)，给出解的值。如硬币个数，路径长度，顶点个数等。

**贪心算法模板**

**function greedy(C:set):set**

**{C是候选对象集合}**

**S=Φ {在集合S中构造解}**

**while C<>Φ and not solution(S) do**

**x=select(C)**

**C=C\{x}**

**if feasible(S∪{x}) then S=S∪{x}**

**if solution(S)**

**then return S**

**else**

**return “No solution”**

**最小生成树**

**Kruskal算法**

算法思想：

（1）将所有边按权由小到大排列 ；

（2）MST置空；

（3）依次取最小边（u,v）：

\* 若（u,v）加入MST不构成回路则将（u,v）加入MST;

\* 若MST中的边达到n-1条，则结束；

**Kruskal算法正确性证明：**

证明：施归纳与G的边数

归纳基础：当G只有一条边，显然G就是最小生成树；

归纳假设：设T中有s<n-1条边时，T是有希望成为最优解的。

当 往T中再加一条最小边（u,v）时，且进入后不会使T构成回路。原T中有若干联通分量，加入（u,v）后，减少一个。所以加入后T仍然是有希望成为最有解的。

所以算法结束后，T中有n-1条边，构成了生产树。且是最优的。

**Prim算法**

算法思想： G=(V,E)

1. T为空， B为任意顶点
2. 从B和V-B之间的边中选一最小边(u,v), u属于B,v属于V-B。
3. 将(u,v)加入T,将v加入B
4. 重复(2)-(3)n-1次

**Dijkstra算法中边上有负权值的情况：**

带权有向图*D*的某几条边或所有边的长度可能为负值。利用Dijkstra算法，不一定能得到正确的结果。

**定理:从1个源点开始，Dijkstra算法可以找到图中其他所有顶点的最短路径。**

证明：数学归纳证明：

(a)如果一个顶点i满足i<>1，且i在S中，则D[i]给出了从源到i的最短路径长度。

(b)一个顶点i不在S中，则D[i]给出了从源到i的最短特殊路径长度。（i的前驱在S中）

归纳基础：初始S 中只有源点1,所以(a)明显成立。对于不属于S 的顶点i，D[i]是L[1,i],(b)也成立。

归纳假设：假设在往S添加v之前，条件(a)(b)都成立。

条件(a)的归纳： 将v加入S，即 D[v]最小。且所经过的顶点都在S中。 否则，假设至少经过了x，D[v]<D[x], 条件(b)的归纳： 将v加入后，对任何一个不属于S的w,

**第六章 动态规划算法**

**适用情况**

动态规划的思想是对贪心算法和分治法的一种折衷，它所解决的问题往往不具有可爱的贪心实质，但是各个子问题又不是完全零散的，这时候我们用一定的空间来换取时间，就可以提高解题的效率。

**一个动态规划算法，通常可以按以下几个步骤进行**：

（1）找出最优子性质、结构特征；

（2）递归定义最优值；

（3）自底向上计算最优值，记录最优解信息；

（4）构造最优解；

**Floyd 算法**

初始 A[i,j]=cost[i,j]

考虑：只经过顶点1

只经过顶点1，2

只经过顶点1，2，3 。。。。。

只经过 1..k 时：

最短路径 A[i,j]=min{A[i,j], A[i,k]+A[k,j]}

（最长公共子序列）

**第七章 搜索图**

**状态（Status）**:在某一时刻的状态(属性值集合)。例：将A,B放在该网格中，有13种态。

**状态空间（Status Space）**:所有状态的集合。

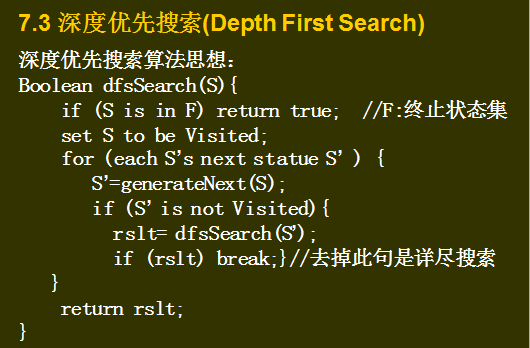
**状态变迁（产生式）：**有一个状态变为另一个状态的规则。

**开始状态（Initial Status）：**问题的初始状态。

**终止状态(End Status)：**问题的目标状态。

**状态可达树：**从初始状态开始，按状态变迁产生下层状态，构成的树，叶结点是终止状态或祖先出现过的状态。

**深度优先搜索**：**（深度优先算法是回溯法）**

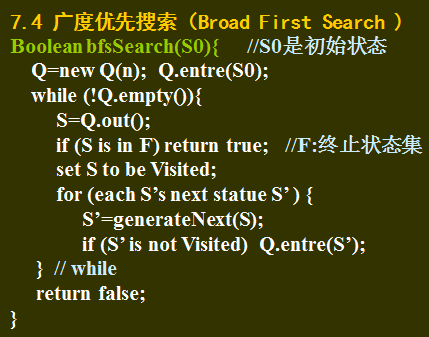


**生成状态的基本方法：**

算法是一边搜索一边创建后继状态，而不是先创建状态可达树，然后再搜索。开始时只有一个初始状态。

* 扩展结点:一个正在产生子女的结点称为扩展结点
* 活结点:一个自身已生成但其子女还没有全部生成的节点称做活结点
* 死结点:一个所有子女已经产生的结点称做死结点

**深度优先的问题状态生成法**：如果对一个扩展结点R，一旦产生了它的一个子女C，就把C当做新的扩展结点。在完成对子树C（以C为根的子树）的穷尽搜索之后，将R重新变成扩展结点，继续生成R的下一个儿子（如果存在）



**宽度优先的问题状态生成法：**一个扩展结点变成死结点之前，它一直是扩展结点,产生了它的所有子结点。

**回溯法的基本思想**

(1)针对所给问题，定义问题的解空间；

(2)确定易于搜索的解空间结构；

(3)以深度优先方式搜索解空间，在搜索过程中可用剪枝函数避免无效搜索。

**常用剪枝函数：**

用约束函数在扩展结点处剪去不满足约束的子树；

用限界函数剪去得不到最优解的子树

**详尽搜索：**将状态可达树遍历；

**非详尽搜索：**只要到达终止状态就结束搜索。这种搜索不完备，得到的解不一定是最优的。

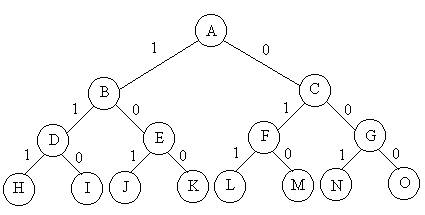
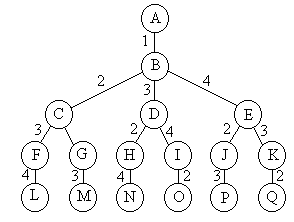
**盲目搜索：**在一个状态下，对下一个状态的选择是盲目的。以上算法都是盲目搜索。

**启发式搜索：**在一个状态下，根据已知的一些信息选择下一个状态。

**限界搜索：**如果弧上有权（默认为1），在搜索中从S0到当前状态S的路径长度为length(S). 给定一个len，搜索到S，其路径长度大于len，就不再向前搜索，退回上一状态。

**子集树与排列树**

**子集树：1表示该元素属于子集，0表示不属于子集。  
排列树：有根到叶是一个排列。**

子集树 排列树

**分支限界法基本思想**

分支限界法常以**广度优先**或以**最小耗费（最大效益）优先**的方式搜索问题的解空间树。

在分支限界法中，每一个活结点只有**一次**机会成为扩展结点。活结点一旦成为扩展结点，就一次性产生其所有儿子结点。在这些儿子结点中，导致不可行解或导致非最优解的儿子结点被舍弃，其余儿子结点被加入活结点表中。

此后，从活结点表中取下一结点成为当前扩展结点，并重复上述结点扩展过程。这个过程一直持续到找到所需的解或活结点表为空时为止。

**常见的两种分支限界法**

**（1）队列式(FIFO)分支限界法**

按照队列先进先出（FIFO）原则选取下一个节点为扩展节点。

**（2）优先队列式分支限界法**

按照优先队列中规定的优先级选取优先级最高的节点成为当前扩展节点。

**A算法与A\*算法**

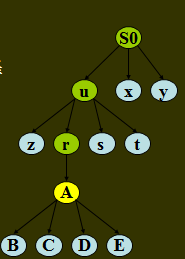
**A算法**

A 是当前状态，下一个状态选择蓝色状态中f(x)最小的。它既不是深度优先，也不是广度优先。

我们把蓝色的状态和绿色的状态记录下来。

OPEN表：存放蓝色状态（活结点），已经产生而未访问的状态。该表按f(s)排序（不减）

CLOSE表：存放绿色状态（死结点），已经访问过的状态。



**算法思想**

产生初始状态S0，入OPEN表，计算f(S0)；

若OPEN表空，搜索失败，结束；

取出OPEN表的第1个状态S，放入CLOSE表；

若S是终止状态，得到解，结束；

若S不可扩展，转2；

产生S的所有后继状态，计算他们的f(x), 并入OPEN表，转2。

**A\*算法**是A算法的改进**（？？？？？）**

每个状态都有一个评估函数，

f(x)=g(x)+h\*(x)

g(x):是有根状态到达当前状态x的以用代价；到达x后，该值是已知的。

h\*(x):是从当前状态x到终止状态要花费代价的实际值；是未知的。在算法中只能估计一个值h(x).

h(x):是从当前状态x到终止状态要花费的代价的估计值；

要求： h(x) <= h\*(x)

**第八章 概率算法（大致了解概念即可）**

概率算法的特点：

* 对同一实例多次执行，结果和用时可能不同；这一点有时是很有用的。
* 概率算法允许失败，但概率要小，如除0错误。
* 对一实例多次执行概率算法，可能获得最好解（类似选举）。

**概率算法有两类：（Monte Carlo算法、Las Vegas算法）**

1. 给出概率性答案，只有精确度区分。有时可能会给出错误答案。数值概率算法，Monte Carlo算法

2. 在搜索过程中，选择下一步是随机的，一次引导算法快速找到最优解。答案通常是正确的，而且能够验证答案是正确的。发现答案是错误的，返回错误信号。这种说法叫做Las Vegas算法。

**蒙特卡罗算法（Mont Carlo algorithms）**

蒙特卡罗算法可以高效地高概率地得到问题的正确解，但有时会给出错误的解，概率很低，并且不知道解是错误的。

**拉斯维加斯算法（Las Vegas algorithms）**

Las Vegas算法进行概率性选择，帮助和指导我们更快地得到正确的解。不会给出错误答案，而给出一个提示。

**Las Vegas 算法有2种：**

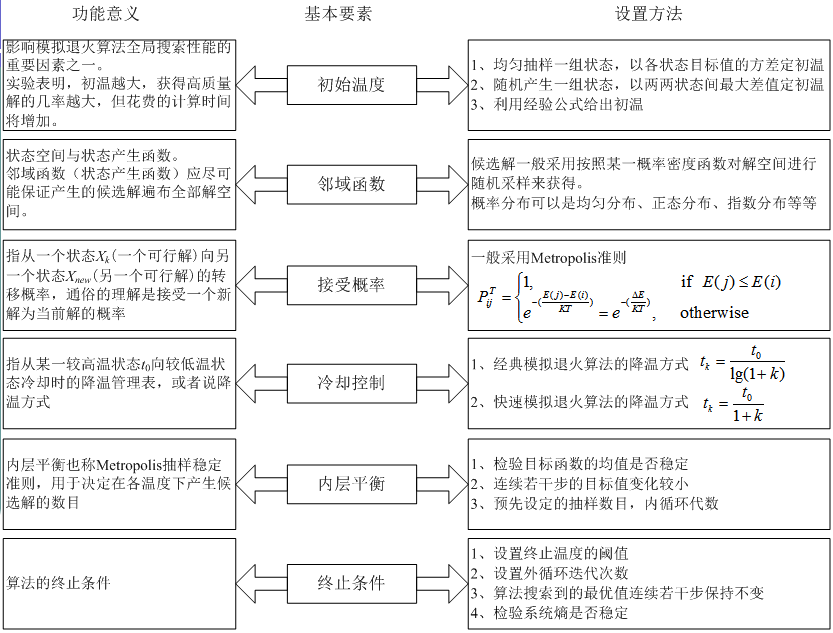
（1）总能给出正确答案，只是选择不合适时，花的时间长。

适合于：有确定的算法，但是平均情况比最坏情况要快得多，差别很大。与实例本身有很大的关系。例如快速排序算法。

打破差异，搞平均，劫富济贫，罗宾汉效应（ Robin hood effect），降低了最好情况的效率，提高了最坏情况。

(2) 选择不恰当，可能导致算法陷入绝境，它会报告找不到解。

**模拟退火算法**



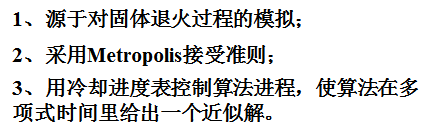
**算法目的：**

解决NP复杂性问题提供有效近似算法；

克服优化过程陷入局部极小；

克服初值依赖性。

**算法基础：**



**第9 章 遗传算法**

1. **定义：**遗传[算法](http://baike.baidu.com/view/7420.htm)是一类借鉴生物界的进化规律（适者生存，优胜劣汰遗传机制）演化而来的[随机](http://baike.baidu.com/view/269420.htm)化搜索方法**。**
2. **特点:**

* 直接对结构对象进行操作，不存在求导和[函数](http://baike.baidu.com/view/15061.htm)连续性的限定；
* 具有内在的隐并行性和更好的全局寻优能力；
* 采用[概率](http://baike.baidu.com/view/45320.htm)化的寻优方法，能自动获取和[指导](http://baike.baidu.com/view/426323.htm)优化的搜索[空间](http://baike.baidu.com/view/31260.htm)，自适应地调整搜索方向，不需要确定的规则。

1. **应用：**[**组合**](http://baike.baidu.com/view/67312.htm)**优化、机器学习、信号处理、**[**自适应控制**](http://baike.baidu.com/view/262307.htm)**和人工生命等**[**领域**](http://baike.baidu.com/view/257682.htm)
2. **遗传算法的基本运算过程：**

a)**初始化**：设置进化代数计数器t=0，设置最大进化代数T，随机生成M个个体作为初始群体P(0)。

b)**个体评价**：计算群体P(t)中各个个体的适应度。

c)**选择运算**: 将选择算子作用于群体。选择的目的是把优化的个体直接遗传到下一代或通过配对交叉产生新的个体再遗传到下一代。选择操作是建立在群体中个体的适应度评估基础上的。

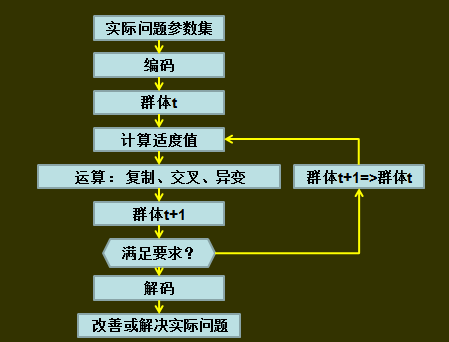
d)**交叉运算**：将交叉算子作用于群体。所谓交叉是指把两个父代个体的部分结构加以替换重组而生成新个体的操作。遗传算法中起核心作用的就是交叉算子。

e)**变异运算**：将变异算子作用于群体。即是对群体中的个体串的某些基因座上的基因值作变动。

　　 群体P(t)经过选择、交叉、变异运算之后得到下一代群体P(t 1)。

f)**终止条件判断**:若t=T ,则以进化过程中所得到的具有最大适应度个体作为最优解输出，终止计算。

5. **遗传算法流程**



1. **算法中的生物学术语**

**染色体：**染色体又可以叫做基因型个体(individuals),一定数量的个体组成了群体(population),群体中个体的数量叫做群体大小。

**基 因：**基因是串中的[元素](http://baike.baidu.com/view/19993.htm)，用于表示个体的特征。例如有一个串S=1011，则其中的1，0，1，1这4个元素分别称为基因。它们的值称为等位基因(Alleles)。

**基因地点：**基因地点在算法中表示一个基因在串中的位置称为基因位置(Gene Position)，有时也简称基因位。基因位置由串的左向右计算，例如在串 S=1101 中，0的基因位置是3。

**特征值：**在用串表示[整数](http://baike.baidu.com/view/71484.htm)时，基因的特征值与二进制数的权一致；例如在串 S=1011 中，基因位置3中的1，它的基因特征值为2；基因位置1中的1，它的基因特征值为8。

**适应度：**各个个体对环境的适应程度叫做适应度(fitness)。为了体现染色体的适应能力，引入了对问题中的每一个染色体都能进行度量的函数，叫适应度函数。 这个函数是计算个体在群体中被使用的概率。

**运算过程：**遗传操作是模拟生物基因遗传的做法。在遗传算法中，通过编码组成初始群体后，遗传操作的任务就是对群体的个体按照它们对环境适应度(适应度评估)施加一定的操作，从而实现优胜劣汰的进化过程。从优化搜索的角度而言，遗传操作可使问题的解，一代又一代地优化，并逼进最优解。

**遗传操作三个遗传算子（选择、交叉、变异）：**

**选择：**

**定义：**从群体中选择优胜的个体，淘汰劣质个体的操作叫选择。

**目的：**把优化的个体(或解)直接遗传到下一代或通过配对交叉产生新的个体再遗传到下一代。

**常用的选择算子**：适应度[比例](http://baike.baidu.com/view/339062.htm)方法、随机遍历抽样法、局部选择法、**轮盘赌注选择法**

**交叉（主要算子）**：

**定义：**把两个父代个体的部分结构加以替换重组而生成新个体的操作。

**目的**：通过交叉，遗传算法的搜索能力得以飞跃提高**。**

**常用的交叉算子：**实值重组（离散重组、中间重组、[线性](http://baike.baidu.com/view/300474.htm)重组、扩展线性重组）；

二进制交叉（单点交叉、多点交叉、均匀交叉、洗牌交叉）

**变异（辅助算子）：**

**定义:** 对群体中的个体串的某些基因座上的基因值作变动。

**目的**：使遗传算法具有局部的随机搜索能力。

使遗传算法可维持群体多样性，以防止出现未成熟收敛现象。

**常用的变异算子：**实值变异、二进制变异

**终止条件**：当最优个体的适应度达到给定的阈值，或者最优个体的适应度和群体适应度不再上升时，或者迭代次数达到预设的代数时，算法终止。预设的代数一般设置为100-500代。

1. **编码**

**评估编码策略常采用以下3个**[**规范**](http://baike.baidu.com/view/113045.htm)：

**a)完备性:**问题空间中的所有点(候选解)都能作为GA空间中的点(染色体)表现。

**b)健全性:** GA空间中的染色体能对应所有问题空间中的候选解。

**c)非冗余性:**染色体和候选解一一对应。

**常用编码技术**：[二进制](http://baike.baidu.com/view/18536.htm)编码，[浮点数](http://baike.baidu.com/view/339796.htm)编码，[字符](http://baike.baidu.com/view/263416.htm)编码，变成编码等

1. **适应度函数（直接影响算法性能的设计）**

**适应度函数的设计主要满足以下条件：**

a）单值、连续、非负、最大化

　　 b) 合理、一致性

　　 c）计算量小

　　 d）通用性强。

**9．遗传算法的不足之处：**

（1）编码不规范及编码表示的不准确性；

（2）单一的遗传算法编码不能全面地将优化问题的约束表达出来，考虑约束的一个方法就是对不可行解采用阀值，这样计算时间必然增加；

（3）遗传算法的效率通常比传统的优化方法低；

（4）遗传算法容易过早收敛；

（5）遗传算法对精度、可行度和时间复杂度等方面还没有有效的定量分析方法；