## 伪代码

1. 缩进表示块结构。

2. while、for、repeat-until等循环结构以及if-else等条件结构与C中那些结构有类似的解释。

3. 符号//表示行注释

4. 形如i=j=e的多重表达式将表达式e的值赋给变量i和j；它应被处理成等价于赋值j=e后跟着赋值i=j。

5. 变量是局部于给定过程的。若无显示说明，不使用全局变量。

6. 数组元素通过“数组名[下标]”的形式来访问。记号..表示数组中值的一个范围。

7. 复合数据通常被组织成对象，对象又由属性组成。

8. 参数按值传递。

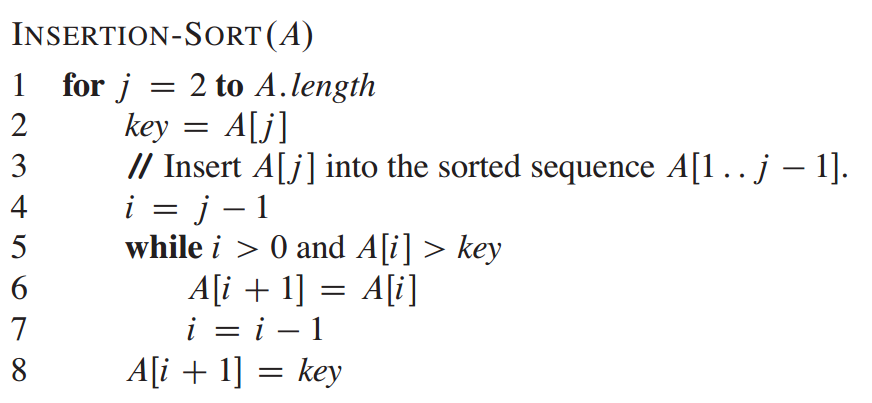
9. return语句立即将控制返回到调用过程的调用点。

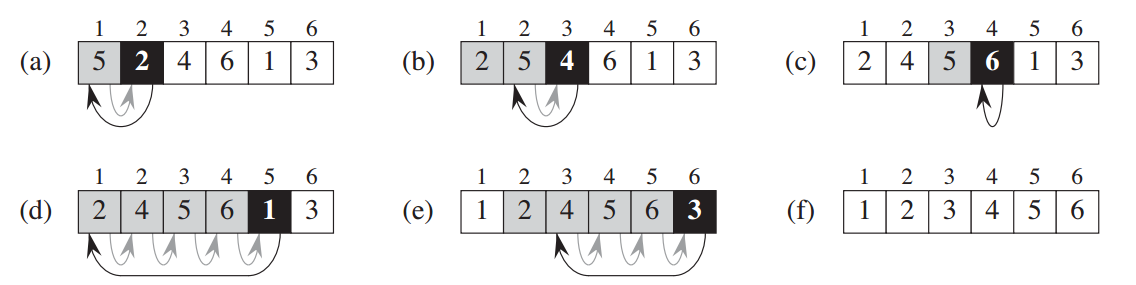
10. 布尔运算符and和or都是短路的。（类似C语言）

11. 关键词error表示因为已被调用的过程情况不对而出现了一个错误。

## CHAPTER 2 算法基础

#### 2.1 插入排序





**循环不变式：**

初始化：循环第一次迭代之前，它为真。

保持：如果循环的某次迭代之前为真，那么下次迭代之前它仍为真。

终止：在循环终止时，不变式为我们提供一个有用的性质，该性质有助于证明算法是正确的。

例：对于上述插入排序算法：

**初始化：**在第一次循环迭代前(j=2)时，子数组A[1..j-1]仅由单个元素A[1]组成，实际上就是A[1]原来的元素。而且该子数组是排序好的（当然很平凡）。故第一次迭代前，循环不变式成立。

**保持：**非形式化地，for循环体的第4~7行将A[j-1]、A[j-2]、A[j-3]等向右移动一个位置，直到找到A[j]的适当位置，第8行将A[j]的值插入该位置。这时子数组A[1..j]由原来在A[1..j]的元素组成，但已按序排列。那么对for循环的下一次迭代增加j时将保持循环不变式。

**终止：**导致for循环终止的条件是j>A.length=n。因为每次循环迭代j增加1，那么必有j=n+1。在循环不变式的表述中将j用n+1代替，有：子数组A[1..n]由原来在A[1..n]中的元素组成，但已按序排列。注意到，子数组A[1..n]即为整个数组，我们推断出整个数组已排序。因此算法正确。

#### 2.2 分析算法

假定一种通用的单处理器计算模型——随机访问机(random-access machine)：

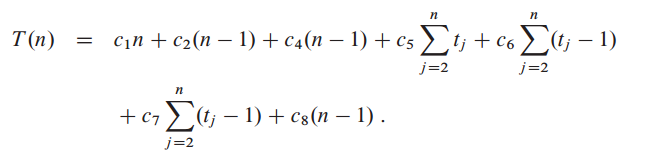
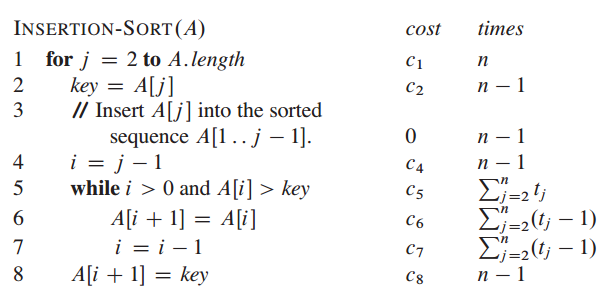
1. RAM模型包含真实计算机中常见的指令：算术指令、数据移动指令、控制指令。每条这样的指令所需时间都为常量。RAM中的数据类型有整型数和浮点实数型。

2. 在RAM模型中，没有对cache和虚拟内存进行建模。

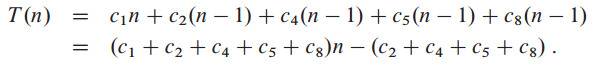
3. 内存无上限，字长有限制。

以2.1中插入排序算法为例：

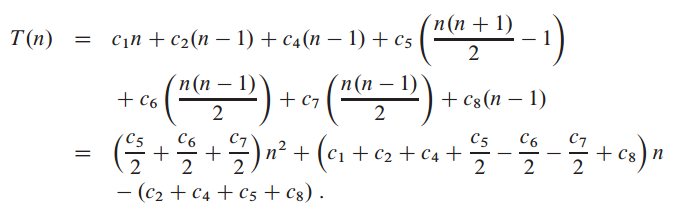
（假设tj为对值j第五行执行while循环测试的次数）



最佳形况下运行时间为(tj=1)：



最坏情况下运行时间为(tj=j)：



本书中，往往只求最坏情况运行时间；某些特定情况下，求算法的平均情况运行时间（如随机化算法）

#### 2.3 设计算法

**分治法：**

**分解**原问题为若干子问题，这些子问题是原问题的规模较小的实例。

**解决**这些子问题，递归地求解各子问题。若子问题规模足够小，则直接求解。

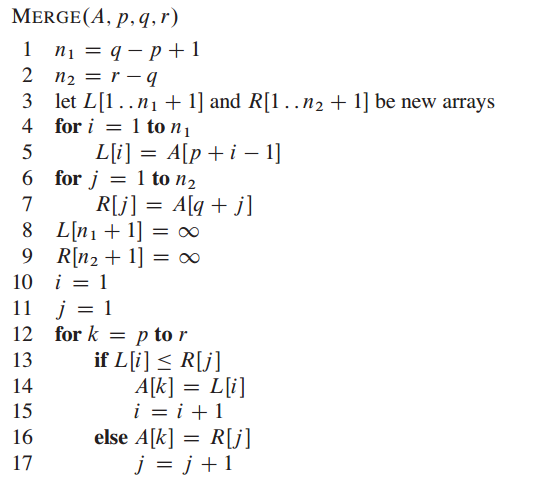
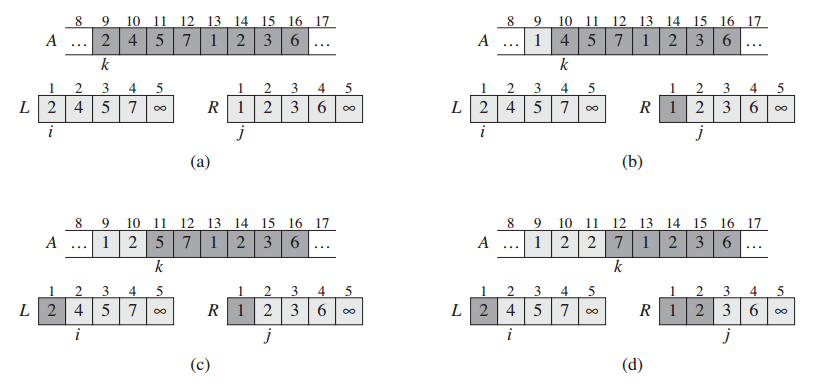
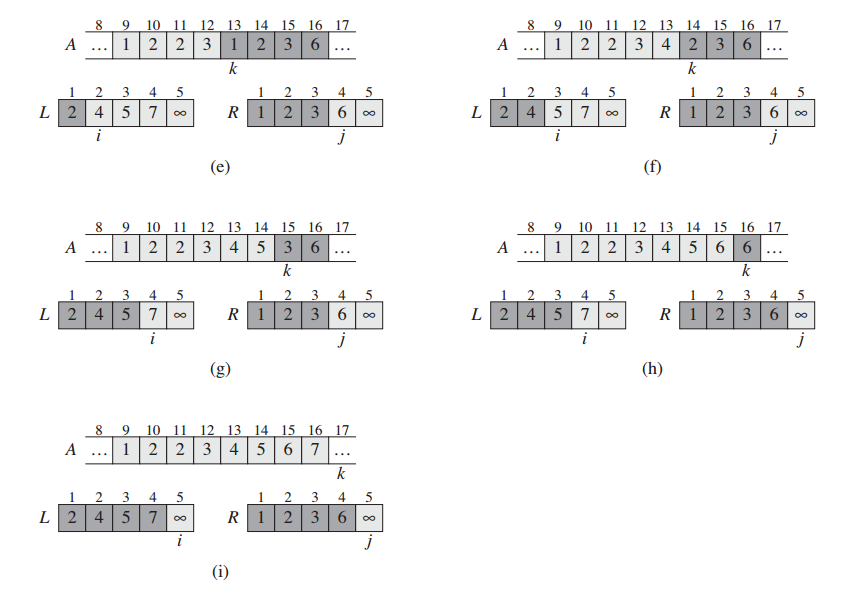
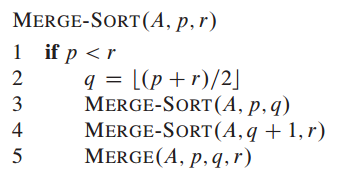
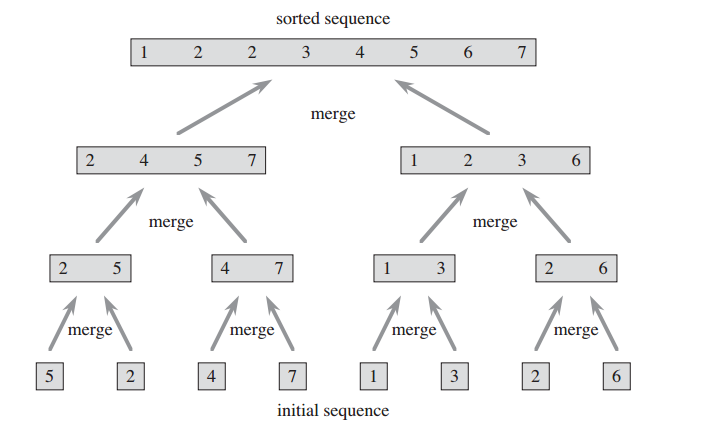
**合并**这些子问题的解成原问题的解。

仍以排序问题为例：

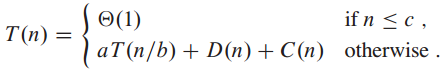
**分解**：分解待排序的n个元素的序列成各具n/2个元素的两个子序列。

**解决**：使用归并排序递归地排序两个子序列。

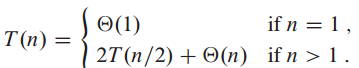
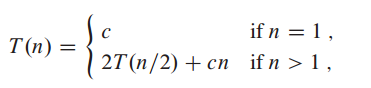
**合并**：合并两个已排序的子序列以产生已排序的答案



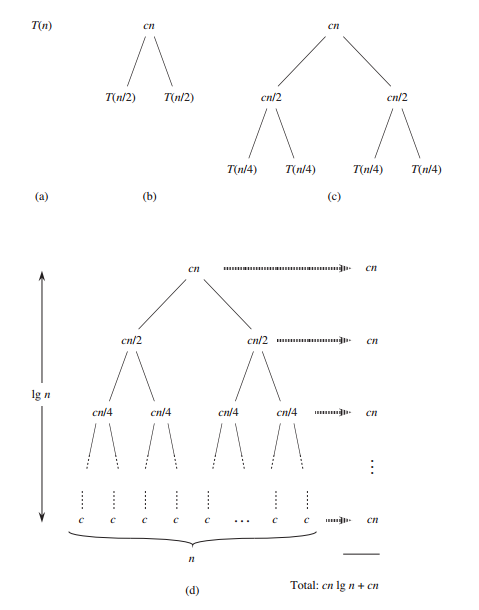
当一个算法包含对其自身的递归调用时，往往可以用递归方程或递归式来描述其运行时间。分治算法运行时间的递归式来自基本模式的三个步骤。假设规模为n的问题的运行时间为T(n)。若原问题足够小，如对某常量c，n≤c，则直接求解，运行时间为ϴ(1)。假设把原问题分解为a个子问题，每个子问题的规模是原问题的1/b。分解问题成子问题需要时间D(n)，合并子问题的解成原问题的解所需时间为C(n)，故：



对于merge-sort算法，a=b=2，D(n)=ϴ(1)，C(n)=ϴ(n)，故：

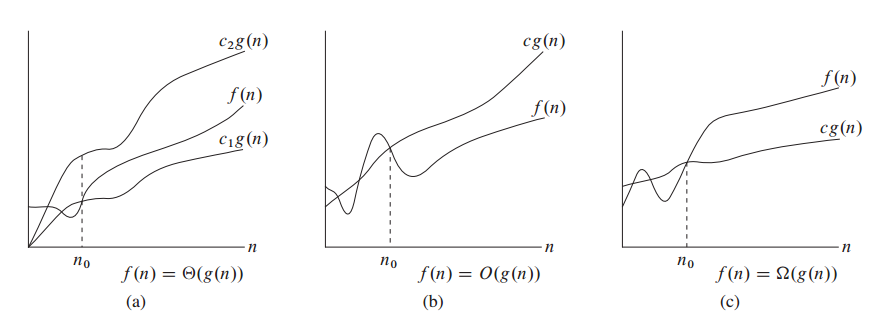
 

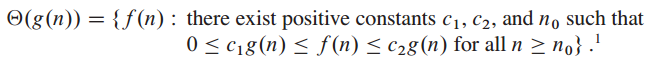
使用递归树表示：

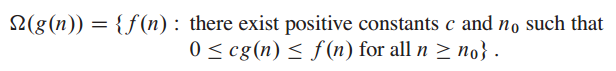


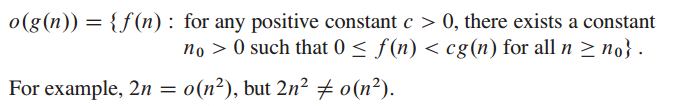
## CHAPTER 3 函数的增长

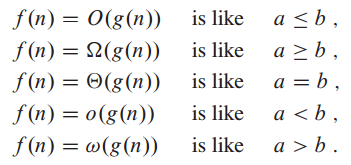
#### 3.1 渐近记号











## CHAPTER 4 分治策略

三种求解递归式的方法：

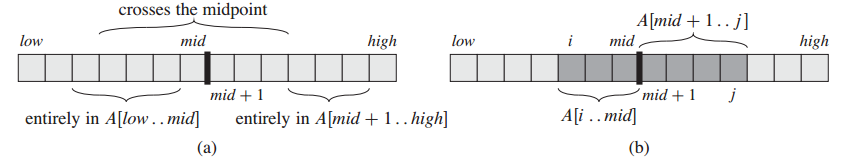
· 代入法 猜测一个界，然后用数学归纳法证明

· 递归树法 将递归式转换为一棵树，其结点表示不同层次的递归调用产生的代价。然后采用边界和技术来求解递归式。

· 主方法 可求解形如T(n)=aT(n/b)+f(n)公式的递归式的界，其中a≥1, b>1, f(n)是给定函数。

#### 4.1 最大子数组问题

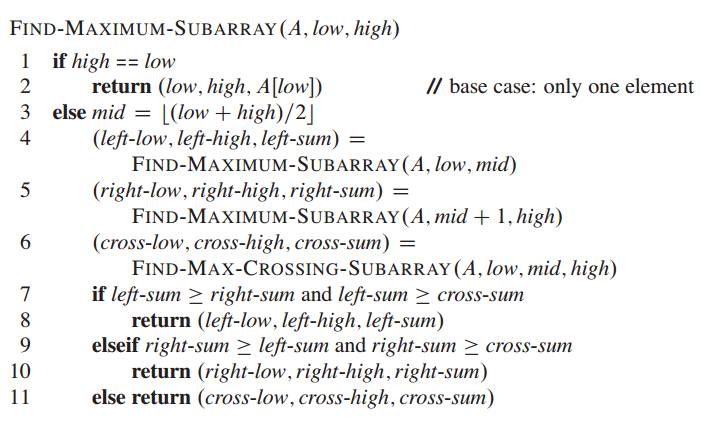
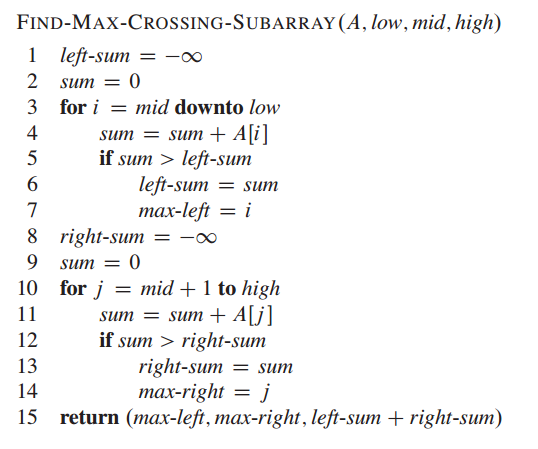
采用分治法求解，如下图所示，A[low..high]的任何连续子数组A[i..j]所处的位置必然是以下三种情况之一：

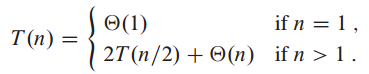


· 完全位于子数组A[low..mid]中；

· 完全位于子数组A[mid..high]中；

· 跨越了中点



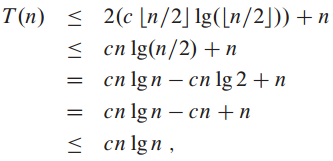
时间复杂度为：

#### 4.3 代入法求解递归式

1. 猜测解的形式

2. 用数学归纳法求出解中的常数，并证明解是正确的。

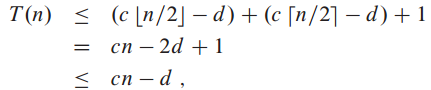
例1. ：猜测，T(n)≤cnlg(n)



例2. 

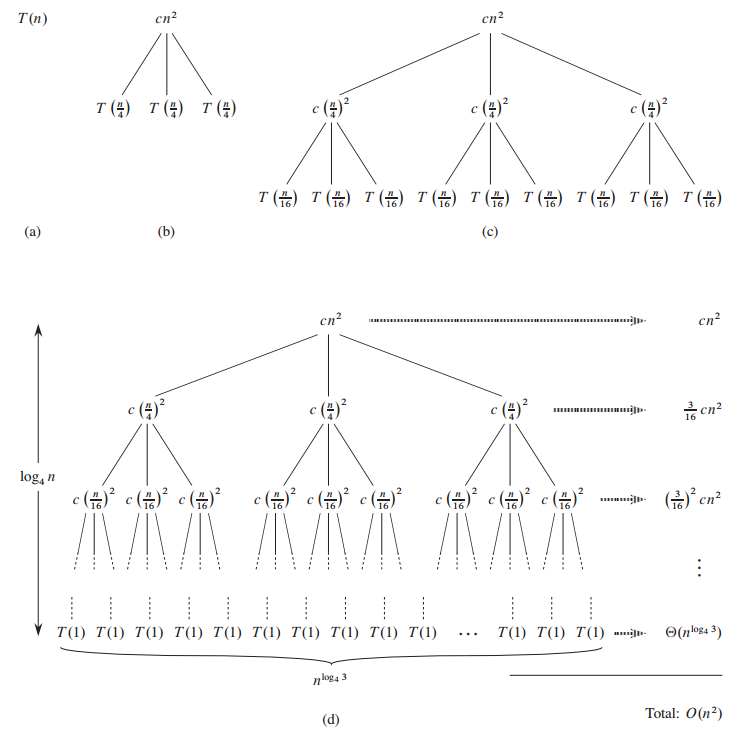
由于T(n+34)=2T(n/2+34)+n+34，令g(n)=T(n+34)，则只需求解g(n)=2g(n/2)+n+34

例3.  猜测T(n) = O(n), T(n)≤cn-d

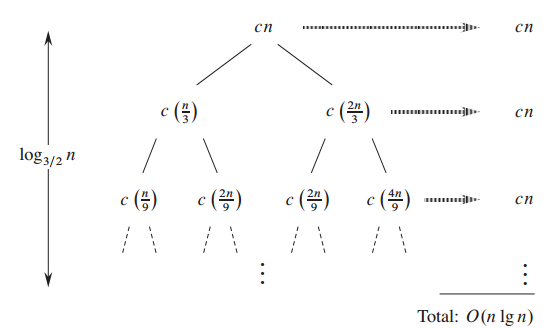


#### 4.4 用递归树方法求解递归式

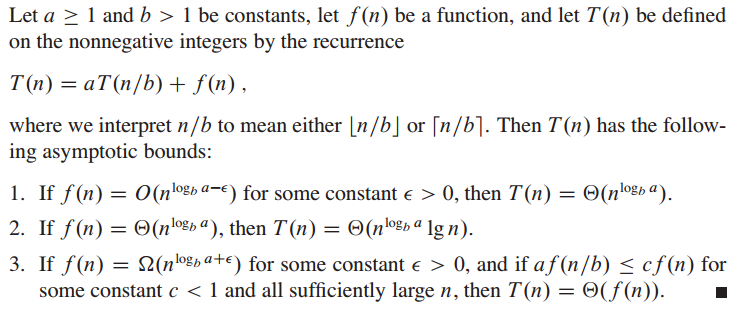
例1. 



例2. 



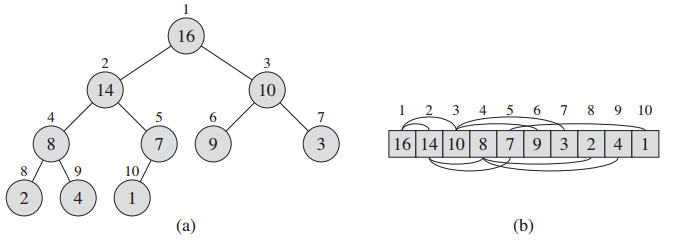
#### 4.5 用主方法求解递归式



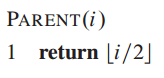
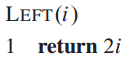
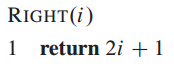
## CHAPTER 6 堆排序

二叉堆是一个数组，它可以被看成一个近似的完全二叉树，除底层外，该树是完全充满的，而且是从左向右填充。

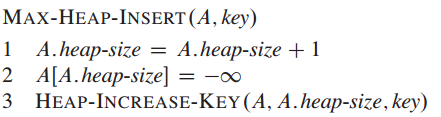
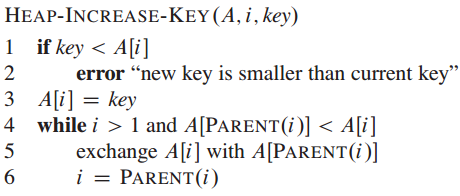
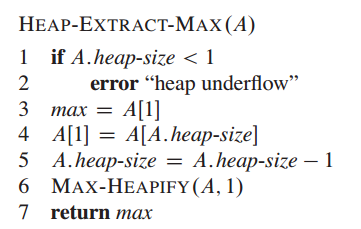
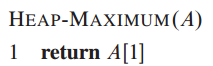
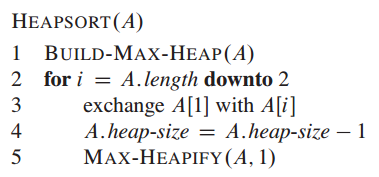
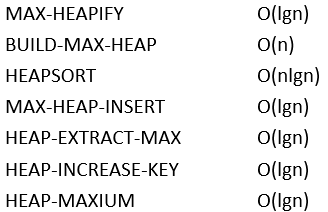
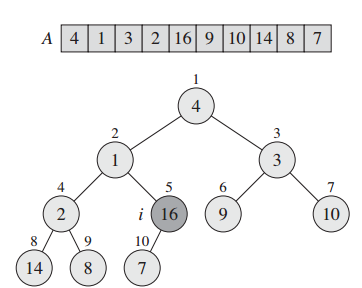
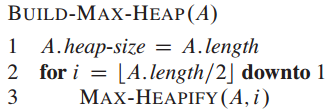
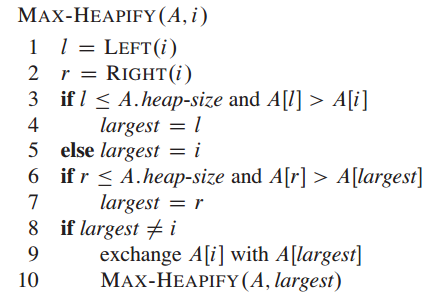
下图为一个最大堆（A[parent(i)]≥A[i]）



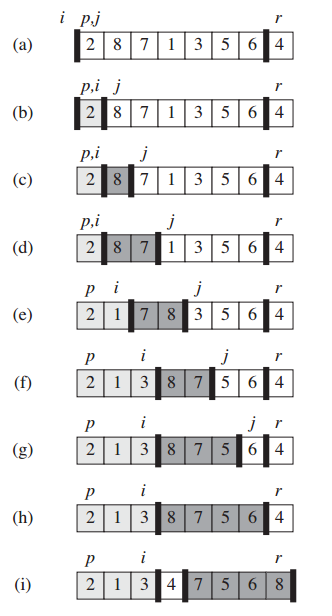
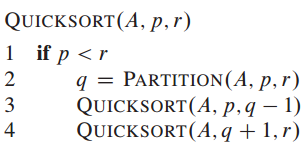
最小堆中，A[parent(i)]≤A[i]

对堆操作的基本过程：

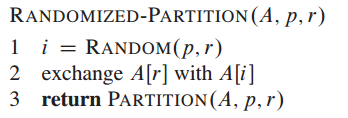
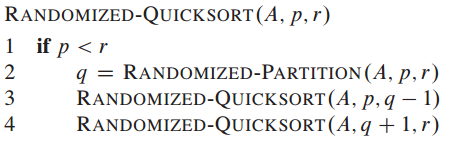


## CHAPTER 7 快速排序



最坏情况运行时间: ϴ(n2)

随机化版本: **期望运行时间：O(nlgn)**

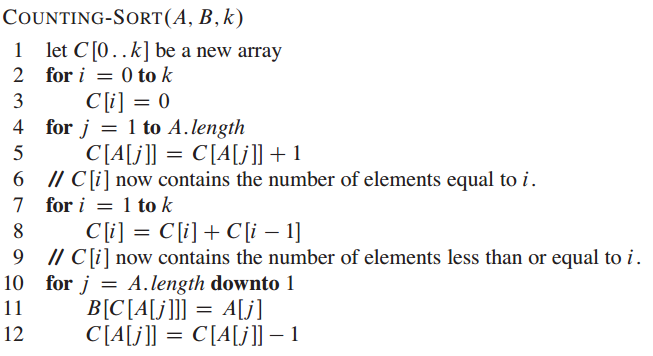
## CHAPTER 8 线性时间排序

**定理8.1** 在最坏情况下，任何比较排序算法都需要做Ω(nlgn)次比较

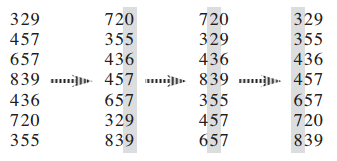
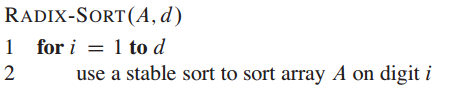
**推论8.2** 堆排序和归并排序都是渐近最优的比较排序算法

#### 8.1 计数排序

假设n个输入元素中每一个都是在0到k区间内的一个整数，其中k为某个整数。当k=O(n)时，排序的运行时间为ϴ(n)。



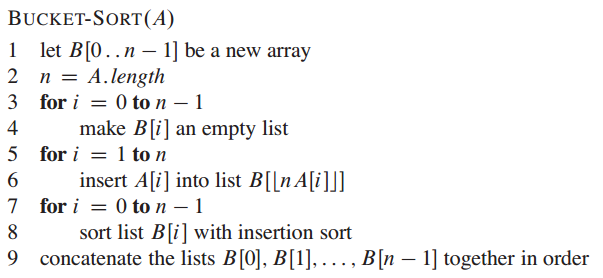
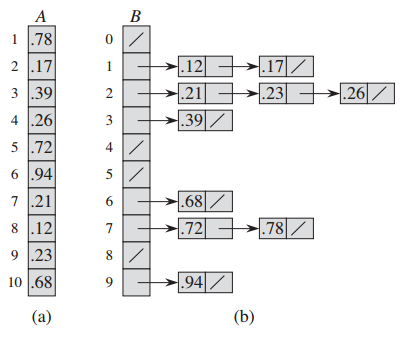
#### 8.2 基数排序



引理8.3 给定n个d位数，其中每一位数位都有k个可能取值。如果RADIX-SORT使用的稳定排序方法耗时ϴ(n+k)，那么它就可以在ϴ(d(n+k))时间内将这些数排好序。

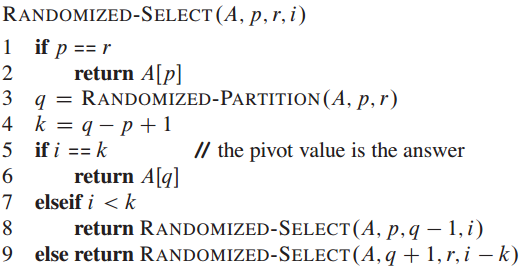
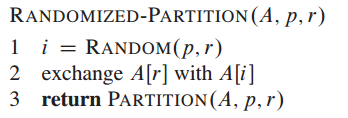
8.4 桶排序

假设输入数据服从[0,1)上的均匀分布，平均情况下它的代价为O(n)。



## CHAPTER 9 中位数和顺序统计量

#### 9.2 期望为线性时间的选择算法



#### 9.3 最坏时间为线性时间的选择算法

算法SELECT可以确定一个有n>1个不同元素的输入数组中第i小的元素。

1. 将输入数组的n个元素划分成组，每组5个元素，且至多只有一组由剩下的n mod 5个元素组成。

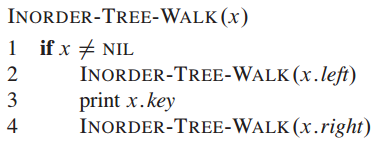
2. 寻找这组中每一组的中位数：首先对每组元素进行插入排序，然后确定每组有序元素的中位数。

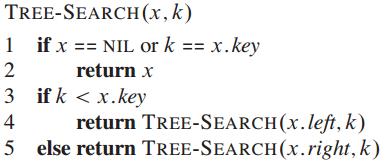
3. 对第2步找出的个中位数，递归调用SELECT以找出其中位数x（如果有偶数个中位数，为了方便，约定x是较小的中位数）。

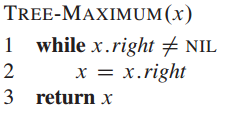
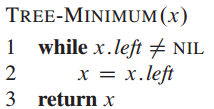
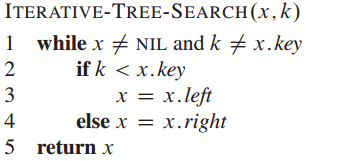
4. 利用修改过的PARTITION版本，按中位数的中位数x对输入数组进行划分。让k比划分的低区中的元素数目多1，因此x是第k小的元素，且有n-k个元素在划分的高区。

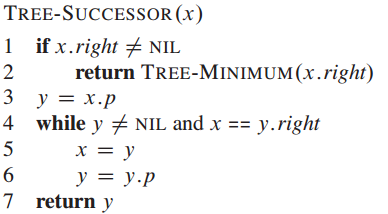
5. 如果i=k，则返回x；如果i<k，则在低区递归调用SELECT来找出第i小的元素。如果i>k，则在高区递归查找第i-k小的元素。

## CHAPTER 12 二叉搜索树

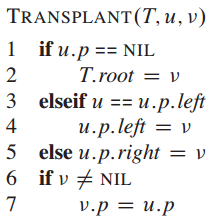
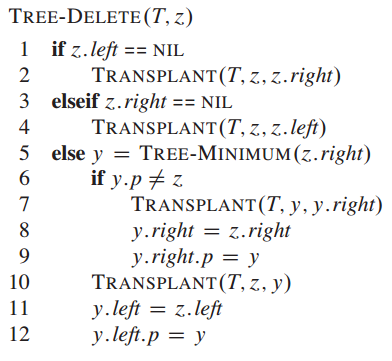
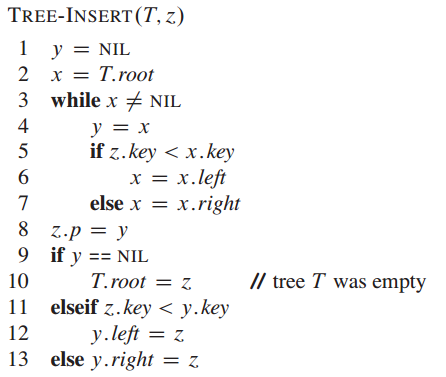
 时间复杂度为ϴ(n) （n为树的结点数目）

 时间复杂度为O(h) （h为树的高度）



 以上四个算法时间复杂度均为O(h)

插入和删除算法：时间复杂度均为O(h)



定理12.4 一棵有n个关键字的随机构建二叉搜索树的期望高度为O(lgn)

## CHAPTER 13 红黑树

红黑树是平衡搜索树的一种，可以保证在最坏情况下基本动态集合操作的时间复杂度为O(lgn)

黑红性质：

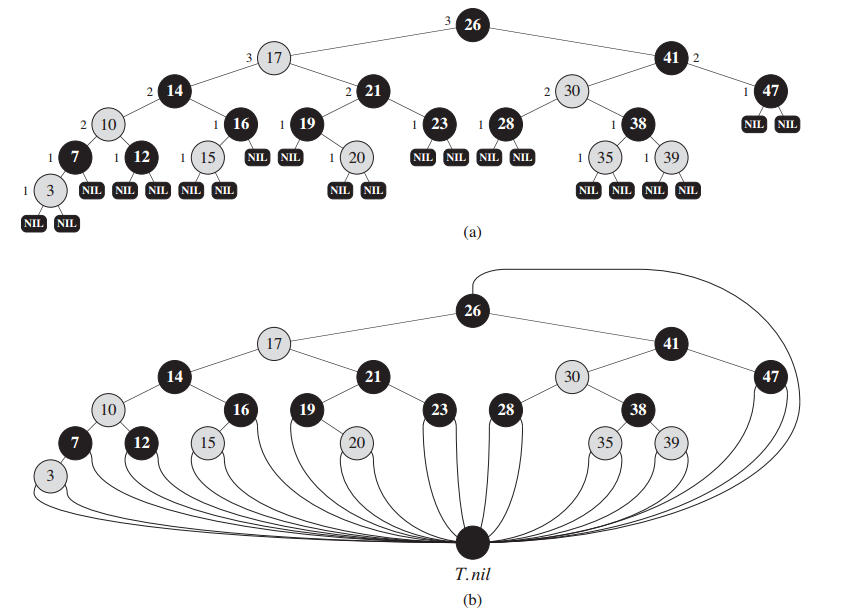
1. 每个结点或是红色的，或是黑色的。

2. 根结点是黑色的。

3. 每个叶节点(NIL)是黑色的。

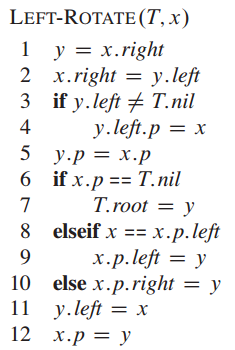
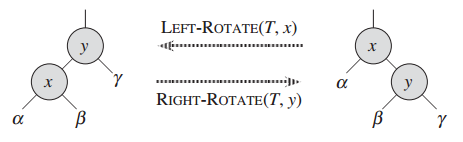
4. 如果一个结点是红色的，则它的两个子结点都是黑色的

5. 对每个结点，从该结点到其所有后代叶结点的简单路径上，均包含相同数目的黑色结点。

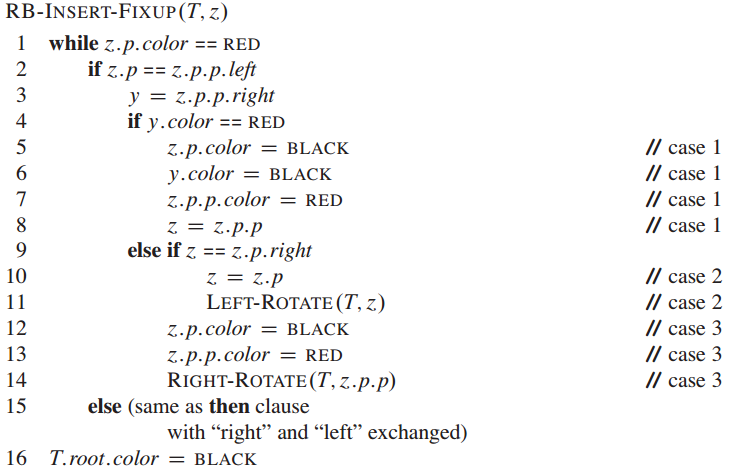
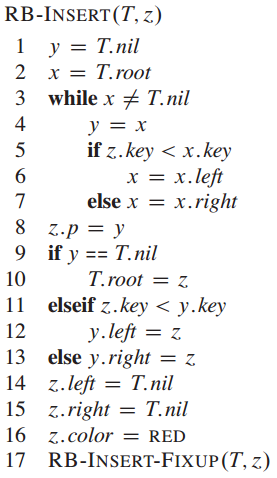


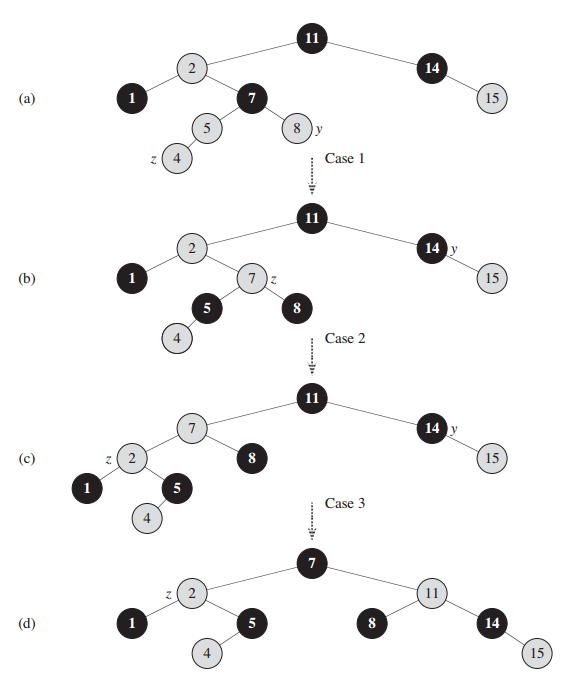
引理13.1 一棵有n个内部结点的红黑树的高度至多为2lg(n+1)

左旋/右旋 时间复杂度均为O(1)

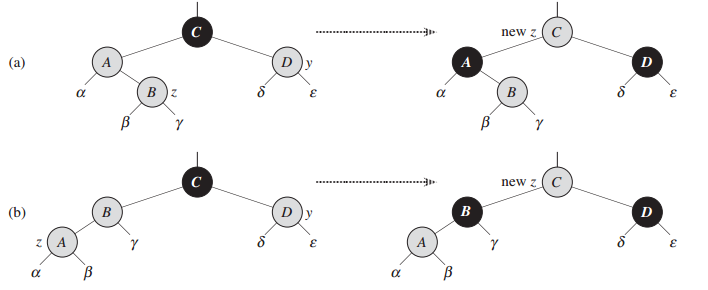


插入算法：时间复杂度为O(lgn)



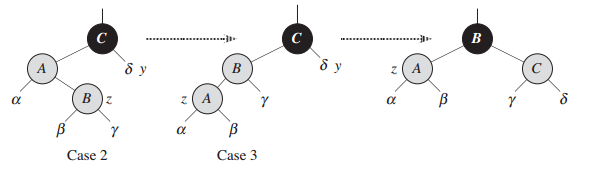


情况1：z的叔结点y是红色的

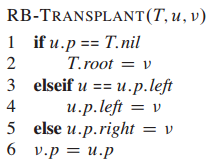
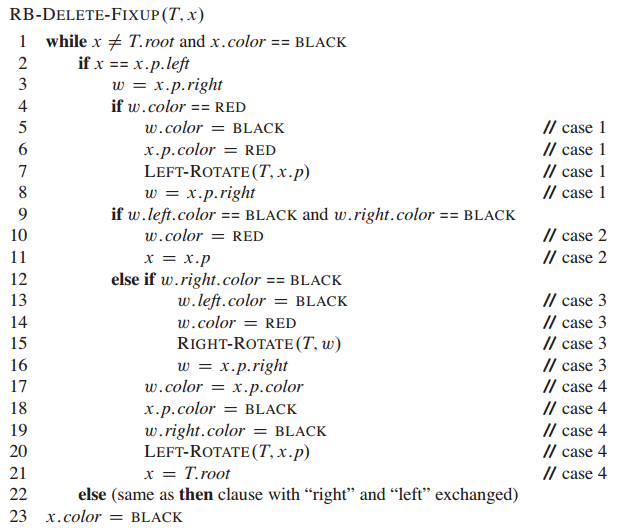
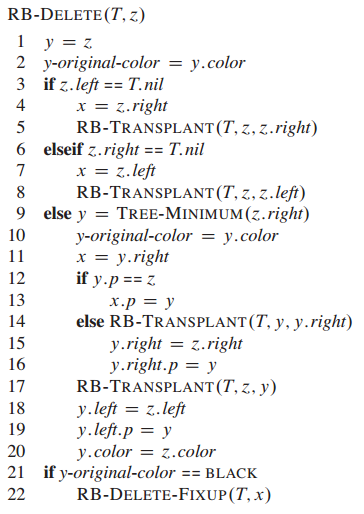
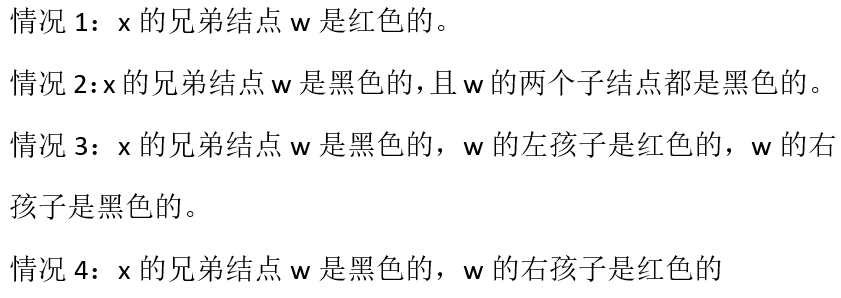


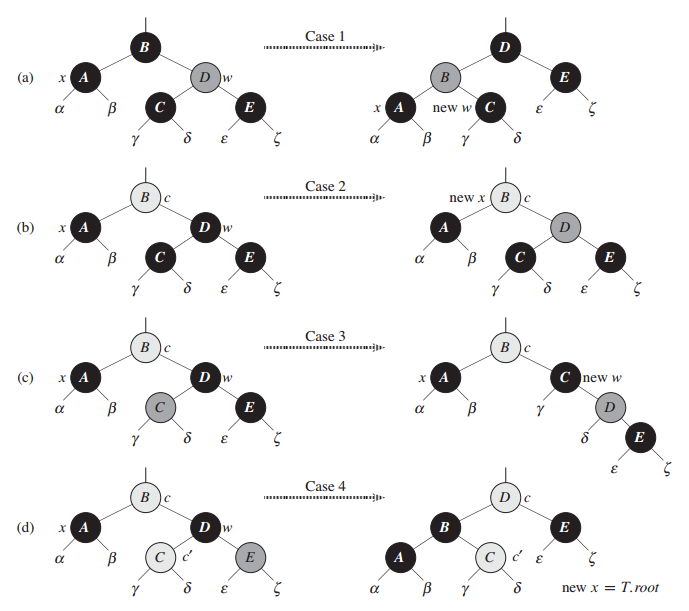
情况2：z的叔结点y是黑色的且z是一个右孩子

情况3：z的叔结点y是黑色的且z是一个左孩子



删除操作：时间复杂度为O(lgn)

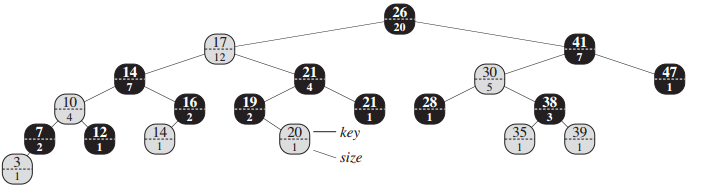
 

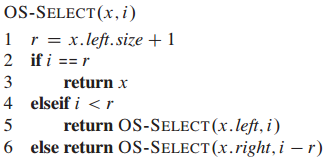
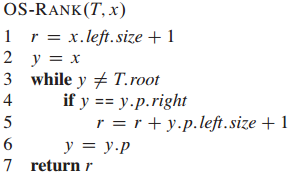


## CHAPTER 14 数据结构的扩张

#### 14.1 动态顺序统计

顺序统计树： x.size = x.left.size + x.right.size + 1



 时间复杂度均为O(lgn)

#### 14.2 如何扩张数据结构

1. 选择一种基础数据结构。

2. 确定基础数据结构中要维护的附加信息

3. 检验基础数据结构上的基本修改操作能否维护附加信息。

4. 设计一些新操作。

**定理14.1（红黑树的扩张）** 设f是n个结点的红黑树T扩张的属性，且假设对任一结点x，f的值仅依赖于结点x、x.left和x.right的信息，还可能包括x.left.f和x.right.f。那么，可以在插入和删除操作期间对T的所有结点的f值进行维护，且不会影响这两个操作的O(lgn)渐近时间性能。

#### 14.3 区间树

区间树是一种对动态集合进行维护的树，其中每个元素x都包含一个区间x.int。区间树支持下列操作：

INTERVAL-INSERT(T, x)：将包含区间属性int的元素x插入到区间树T中。

INTERVAL-DELETE(T, x)：从区间树T中删除元素x。

INTERVAL-SEARCH(T, x)：返回一个指向区间树T中元素x的指针，使x.int与i重叠；若此元素不存在，则返回T.nil。

**步骤1：基础数据结构**

选择一棵红黑树，其每个结点x包含一个区间属性x.int，且x的关键字为区间的低端点x.int.low。因此，该数据结构按中序遍历列出的就是按低端点的次序排列的各区间。

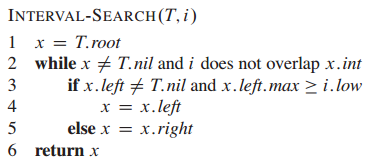
**步骤2：附加信息**

每个结点x除了自身区间信息外，还包含一个值x.max，它是以x为根的子树中所有区间的端点的最大值。

**步骤3：对信息的维护**

x.max = max (x.int.high, x.left.max, x.right.max)

**步骤4：设计新的操作**



## CHAPTER 15 动态规划

1. 刻画一个最优解的结构特征

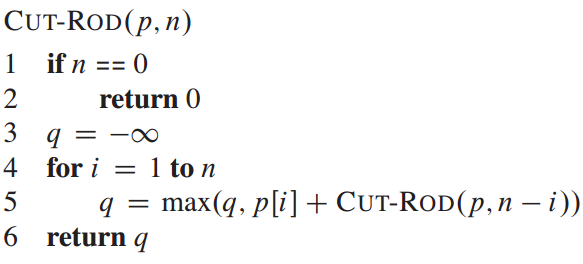
2. 递归地定义最优解的值

3. 计算最优解的值，通常采用自底向上的方法

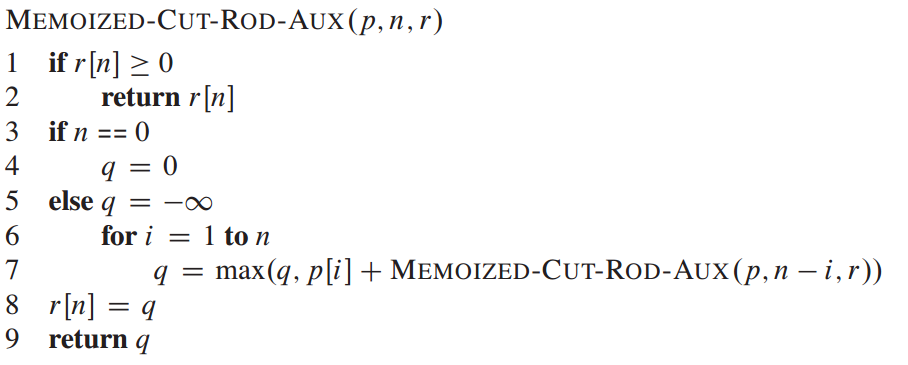
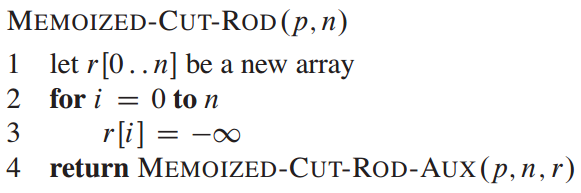
4. 利用计算出的信息构造一个最优解

#### 15.1 钢条切割

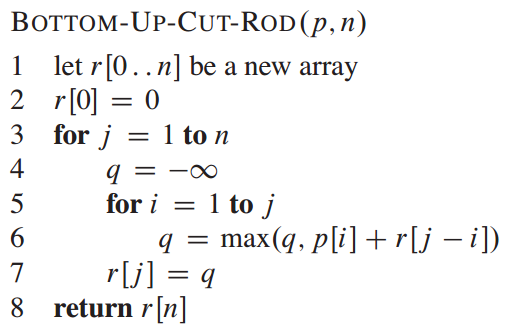
自顶向下递归实现：

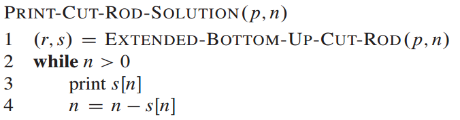
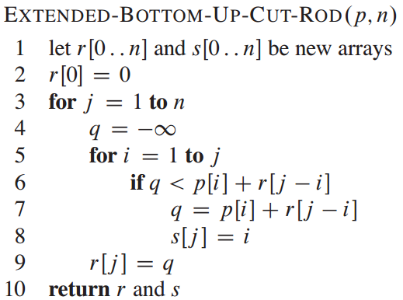


带备忘的自顶向下方法：

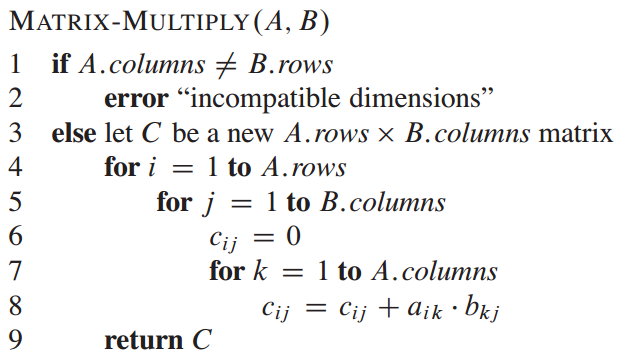
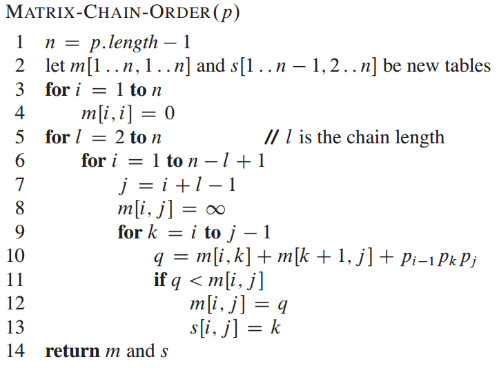


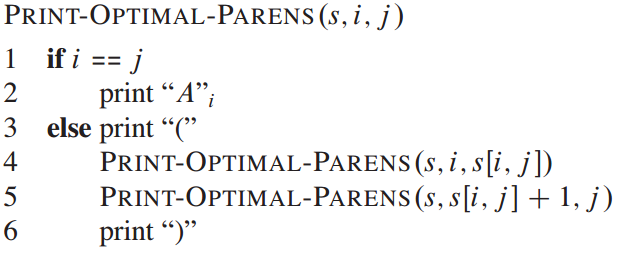
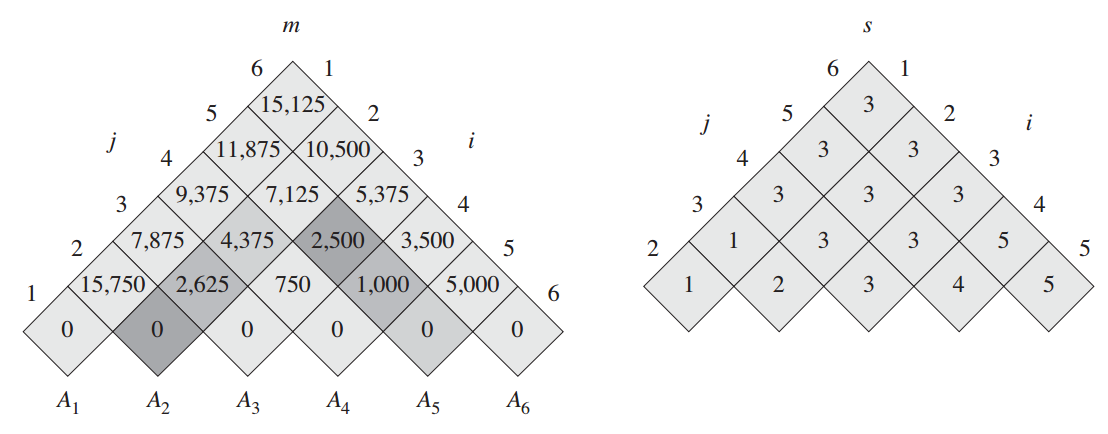
自底向上方法： 扩展版本：（保存最优解对应的第一段钢条的切割长度sj）





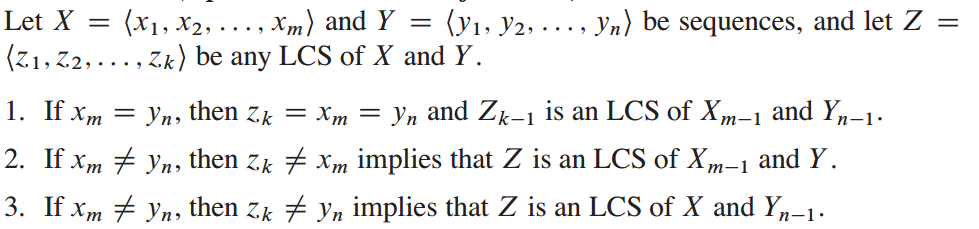
#### 15.2 矩阵链乘法



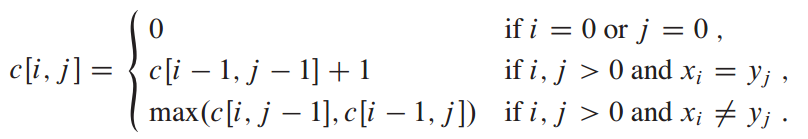


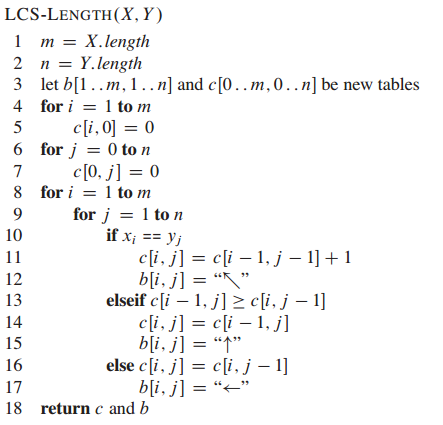
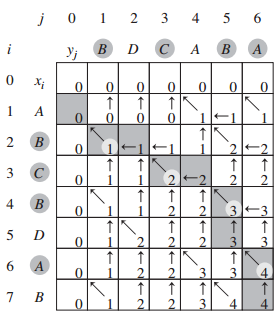
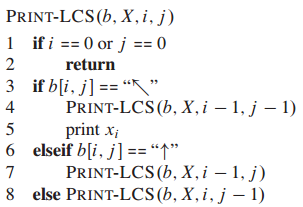
15.4 最长公共子序列LCS

定理15.1 （LCS的最优子结构）



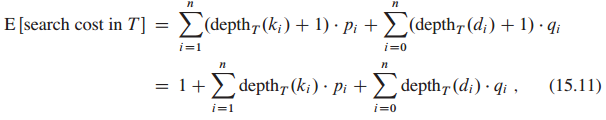
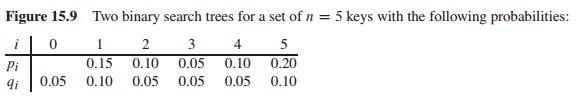
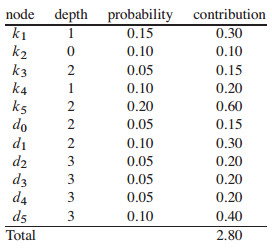
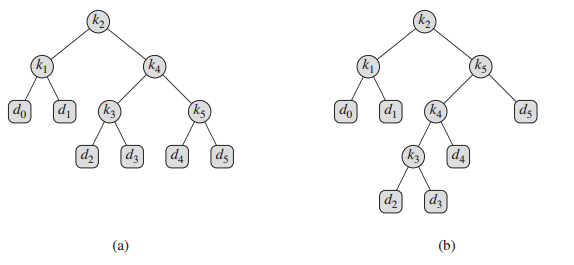
用c[i, j]表示Xi和Yj的LCS的长度：





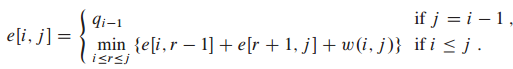
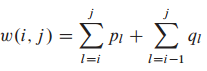
15.5 最优二叉搜索树

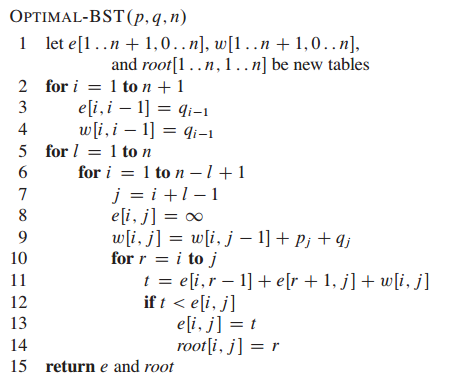
对每个关键字ki，都有一个概率pi表示其搜索频率；对每个伪关键字di，也都有一个概率qi表示对应的搜索频率。



最优二叉搜索树：期望搜索代价最小的二叉搜索树

e[i, j]为在包含关键字ki,…,kj的最优二叉搜索树中进行一次搜索的期望代价：

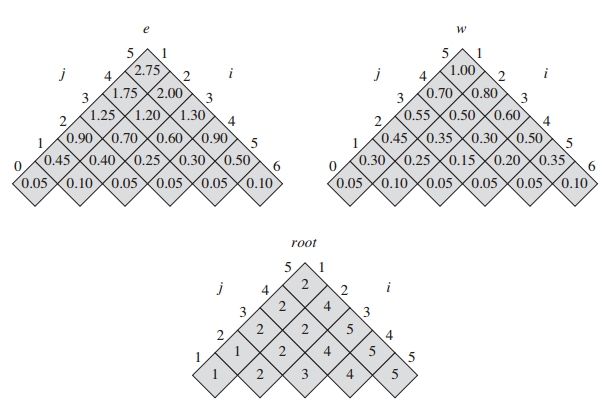
 



每次选择的r即为根结点；若选择i，它的左子树只有di-1；若选择j，它的右子树只有dj

故j=i-1时，e[i, j]=qi-1

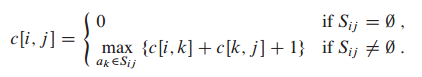
由于计算左右子树的BST时，深度都比实际-1，且根结点r的代价为pr，故需加上w(i, j)



## CHAPTER 16 贪心算法

#### 16.1 活动选择问题

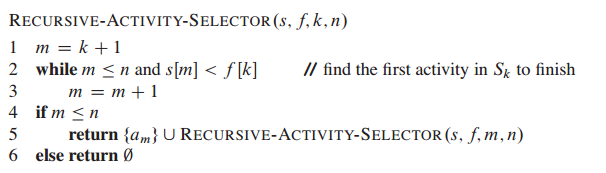
另Sij表示在ai结束之后开始，且在aj开始之前结束的活动的集合。设Aij为Sij的一个最大的互相兼容的活动子集。c[i, j]表示Sij最优解的大小。

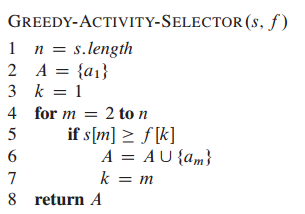


定理16.1 考虑任何非空子问题Sk，另am是Sk中结束时间最早的活动，则am在Sk的某个最大兼容活动子集中。

求解源问题调用RECURSIVE-ACTIVITY-SELECTOR(s,f,0,n)

s: start time f: finish time 假定输入的n个活动已经按结束时间的单调递增顺序排列好。





#### 16.2 贪心算法原理

贪心选择性质：可以通过做出局部最优（贪心）来选择来构造全局最优。

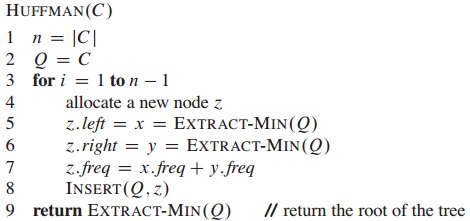
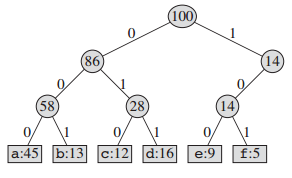
0-1背包问题：

n个商品，第i个商品价值vi美元，重wi磅，背包最多能容纳W磅重的商品。

分数背包问题：对每个商品，可以拿走一部分而不是只能作出0-1选择

贪心策略可以求解分数背包问题，不能求解0-1背包问题

#### 16.3 赫夫曼编码



## CHAPTER 17 摊还分析

1. 聚合分析

确定一个n个操作的序列的总代价的上界T(n)，因而每个操作的平均代价为T(n)/n。所有操作具有相同的摊还代价。

2. 核算法

当存在不止一种操作时，每种操作的摊还代价可能是不同的。核算法将序列中某些较早的操作的“余额”作为“预付信用”存储起来，与数据结构中的特定对象相关联。在操作序列随后的部分，储存的信用即可用来为那些缴费少于实际代价的操作支付差额。

3. 势能法

分析每个操作的摊还代价，类似核算法。将信用作为数据结构的“势能”储存起来，与hesuanfa 不同，它将势能作为一个整体储存，而不是将信用与数据结构中的单个对象关联分开储存

#### 17.1 聚合分析

例1. 栈操作

PUSH(S,x) 实际运行时间：O(1)

POP(S) O(1)

MULTIPOP(S,k) O(min(s, k))

分析整个序列的n个操作：在一个空栈上执行n个PUSH、POP和MULTIPOP的操作序列，代价至多是O(n)，一个操作的平均时间为O(n)/n=O(1)。

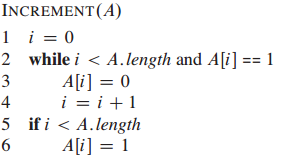
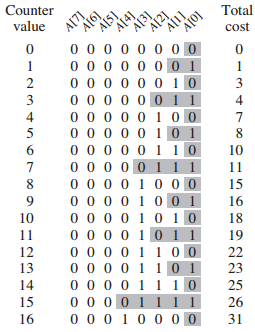
例2. 二进制计数器递增

每次INCREMENT操作的代价与翻转的二进制位的数目呈线性关系

根据下图可以看出，总代价始终不超过INCREMENT操作总次数的2倍。

对于n个INCREMENT操作组成的序列，最坏情况下代价为O(n)，摊还代价为O(1)。

A[0]每次INCREMENT都要翻转一次，A[1]每2次INCREMENT翻转一次，……，A[k]每2k次INCREMENT翻转一次。总代价为(1+1/2+…+2-k)n≤2n

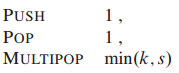
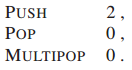
 

#### 17.2 核算法

用表示第i个操作的真实代价，用表示第i个操作的摊还代价，则：

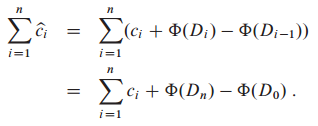
。数据结构中存储的信用恰好等于总摊还代价与实际代价的差值

例1. 操作的实际代价： 指定如下摊还代价：

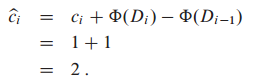
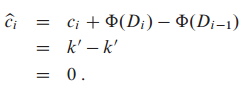
例2. 每次INCREMENT都只将一位数置1，其他的翻转都是复位，设摊还代价为2，则每次INCREMENT中，都取1作为置1的实际代价，剩下的1存放在该位，作为日后将它复位时的代价（信用）。

#### 17.3 势能法

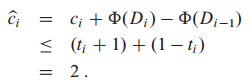
例1. 势函数定义为栈中的对象数量

PUSH MULTIPOP

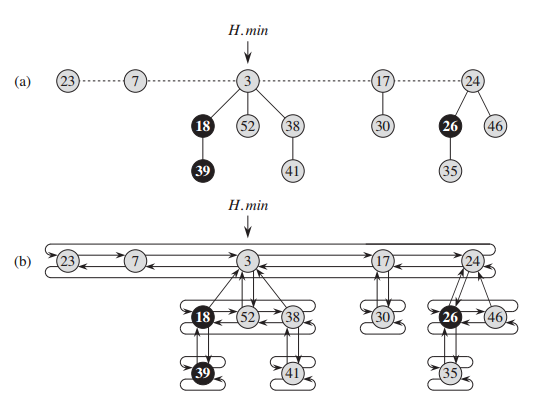
例2. 势函数bi定义为第i次操作后计数器中1的个数

假设第i个INCREMENT操作将ti个位复位，则其实际代价至多为ti+1。若bi=0，则所有k位都在第i次操作中复位了，因此bi-1=ti=k；否则，bi=bi-1-ti+1。无论哪种情况，bi≤bi-1-ti+1



## CHAPTER 19 斐波那契堆

一个斐波那契堆是一系列具有最小堆序的有根树的集合。



每个结点x包含一个指向它父结点的指针x.p和一个指向它的某一个孩子的指针x.child。x的所有孩子被连接成一个环形的双向链表，称为x的孩子链表。孩子链表中的每个孩子y均有指针y.left和y.right，分别指向y的左兄弟和右兄弟。如果y是仅有的一个孩子，则y.left=y.right=y。孩子链表中各兄弟出现的次序是任意的。

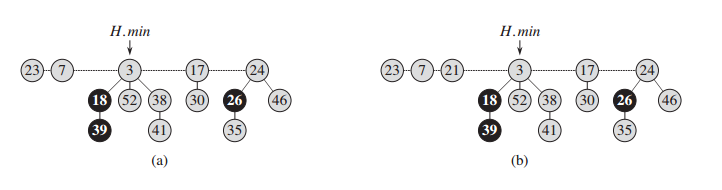
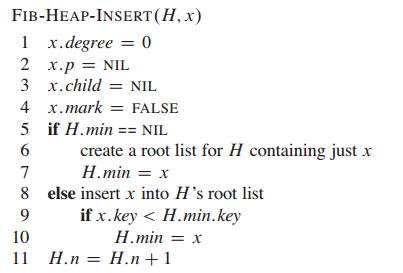
每个结点另有两个属性。把结点x的孩子链表中的孩子数目存储在x.degree。布尔值属性x.mark指示结点x自上一次成为另一个结点的孩子后，是否失去过孩子。新产生的结点是未标记的，并且当结点x成为另一个结点的孩子时，它便成为未标记结点。直到DECREAE-KEY操作，才把所有的mark属性值设为FALSE。

通过指针H.min来访问一个给定的斐波那契堆，该指针指向具有最小关键字的树的根结点，这个结点为斐波那契堆的最小结点。如果不止一个根结点具有最小关键字，那么这些根结点中任何一个都有可能成为最小结点。如果一个斐波那契堆H的空的，则H.min=NIL。

在斐波那契堆中，所有树的根都用其left和right指针链成一个环形的双链表。H.n表示H中当前的结点数目。D(n)为任何结点的最大度数的上界。

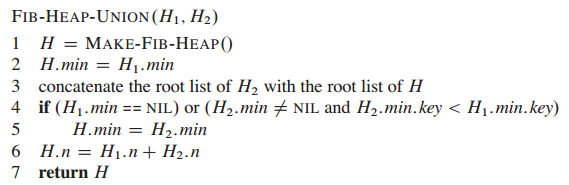
势函数：ф(H) = t(H) + 2m(H) t(H)根链表中树的数目 m(H)已标记的结点数目

插入结点： 摊还代价=实际代价O(1)

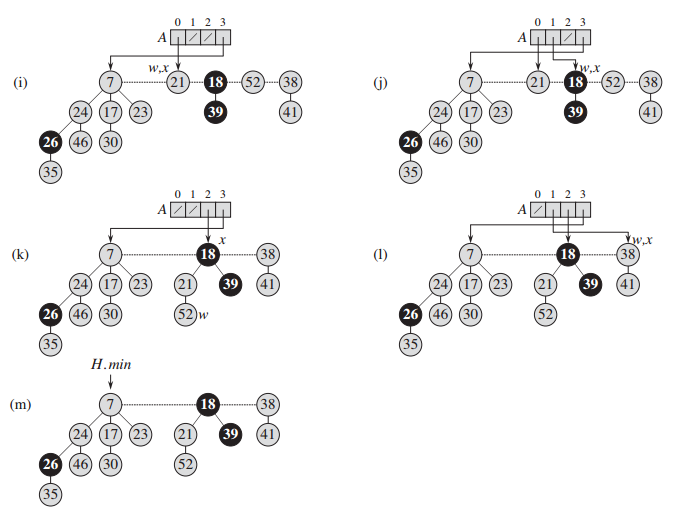
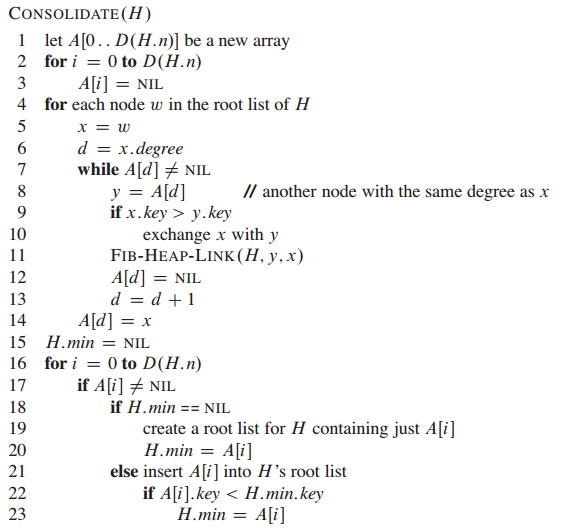
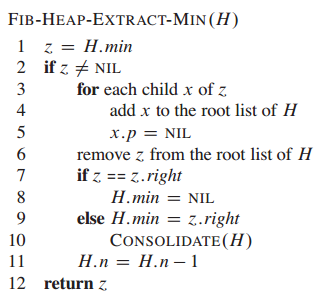


寻找最小结点：返回H.min，摊还代价=实际代价=O(1)

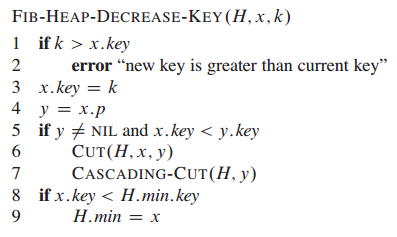
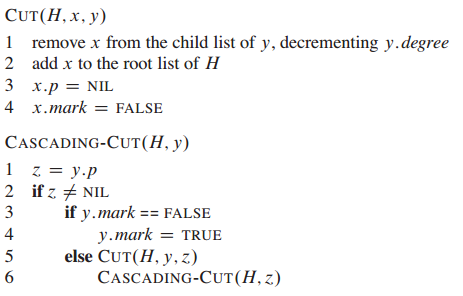
两个斐波那契堆的合并：摊还代价=实际代价=O(1)



抽取最小结点：摊还代价=O(lgn)

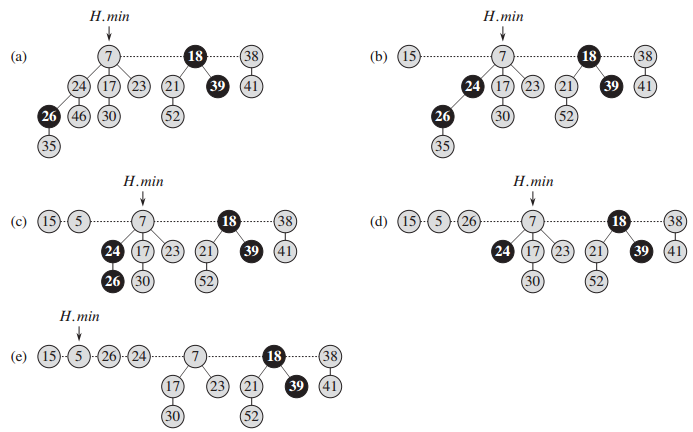


关键字减值：实际代价=O(c)，摊还代价=O(1)

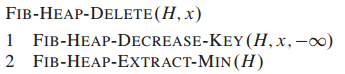
 

下图：(b)将关键字为46的结点的关键字减小到15.

(c)将关键字为35的结点关键字减小为5



删除一个结点：摊还时间=O(D(n))



## CHAPTER 22 基本的图算法

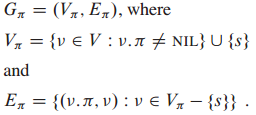
**图的表示**

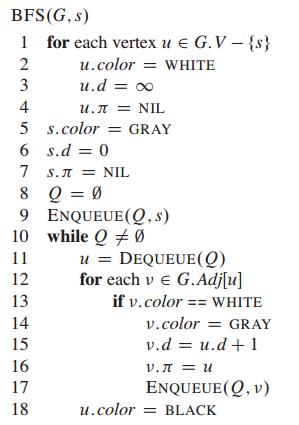
1. 邻接表

2. 邻接矩阵

**广度优先搜索：**可以找到从定源结点s到所有可以到达的结点之间的距离

广度优先搜索可以正确计算出s到可达结点的最短路径距离

前驱子图（广度优先树）：

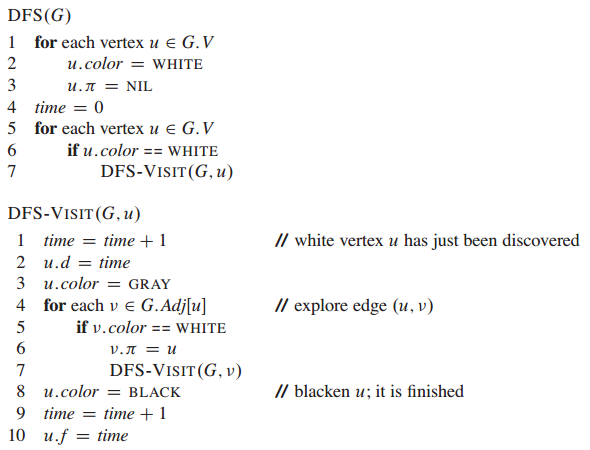


**深度优先搜索：**

1. 其生成的前驱子图形成一个由多棵树所构成的森林

2. 结点的发现时间u.d和完成时间u.f具有括号化结构。

前驱子图（由多棵深度优先树构成的深度优先森林）：



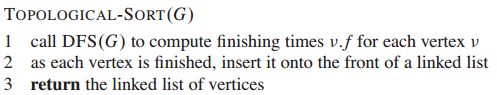
树边：深度优先森林中的边

后向边(u,v)：结点v是结点u在深度优先树上的祖先结点

前向边(u,v)：结点v是结点u在深度优先树上的后代结点

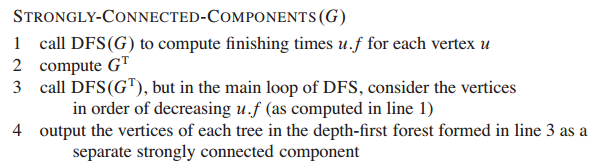
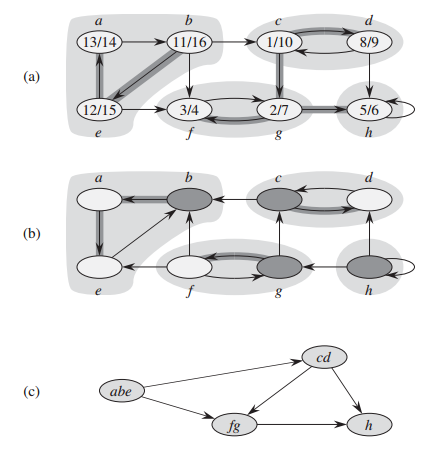
交叉边：图中所有不符合以上定义的边

#### 22.4 拓扑排序



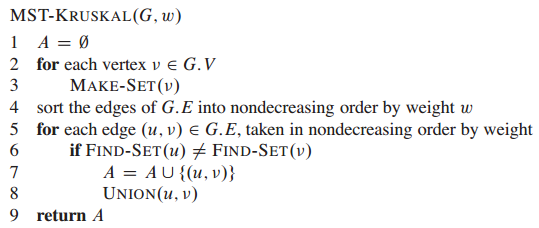
生成有向无环图的拓扑排序

#### 22.5 强连通分量

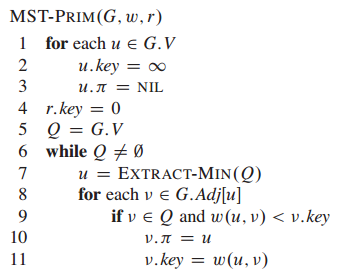


## CHAPTER 23 最小生成树

Kruskal算法：

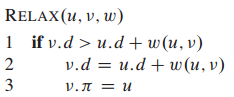


Prim算法：



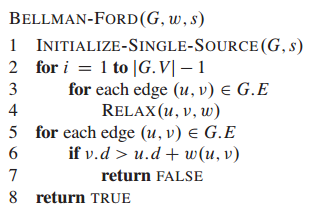
## CHAPTER 24 单源最短路径

初始化： 松弛技术：

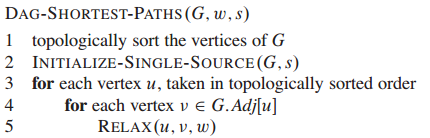


#### 24.1 Bellman-Ford算法

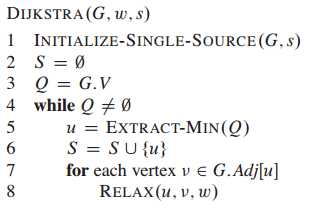
解决一般情况下的单源最短路径问题，在这里，边的权重可以为负。Bellman-Ford算法返回一个布尔值，以表明是否有权重为负的回路。



#### 24.2 有向无环图中的单源最短路径问题



#### 24.3 Dijkstra算法



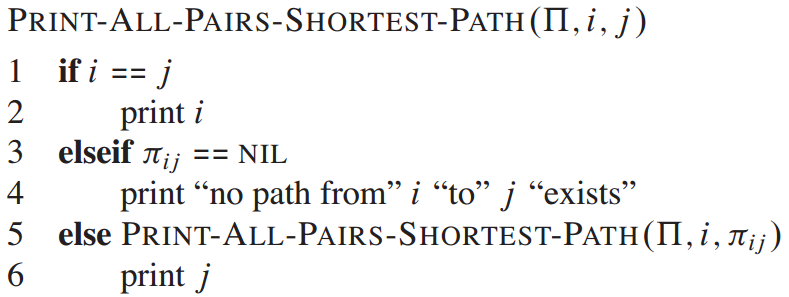
## CHAPTER 25 所有结点对的最短路径

允许存在负权重的边，但假定图中不包括权重为负的环路。

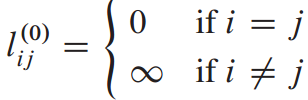
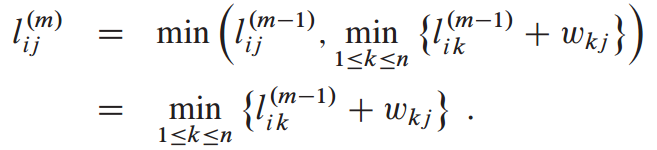
前驱矩阵

当i=j或ij之间不存在路径时，为NIL，否则为从结点i到结点j的某条最短路径上结点j的前驱结点。矩阵的第i行所诱导的子图应当是一棵根结点为i的最短路径树。

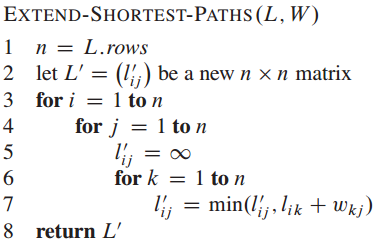
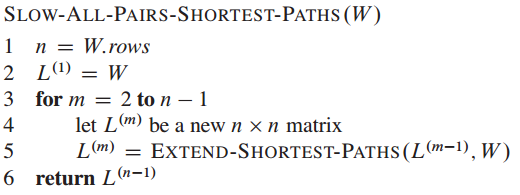
  



设为从结点i到结点j的至多包含m条边的任意路径中的最小权重。

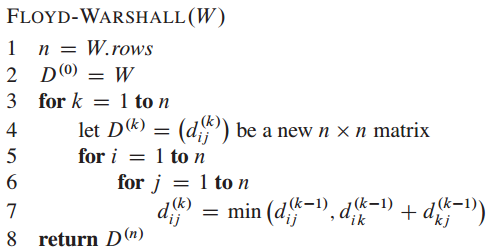
自底向上计算最短路径权重：

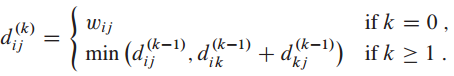
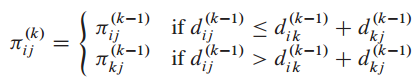


#### 25.2 Floyd-Warshall算法

自底向上计算最短路径权重

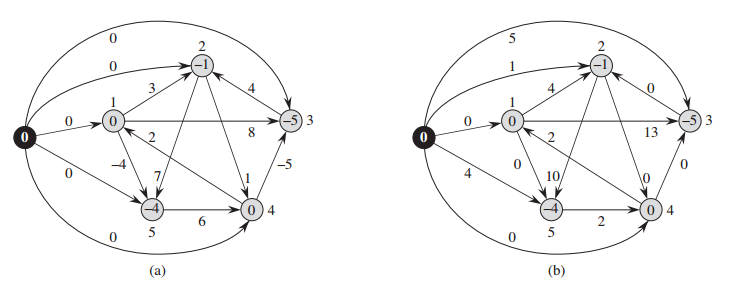
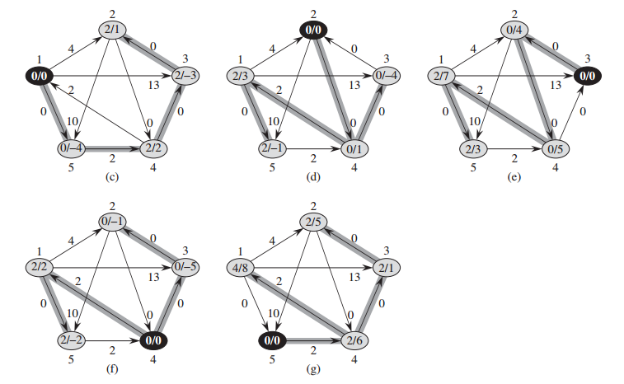
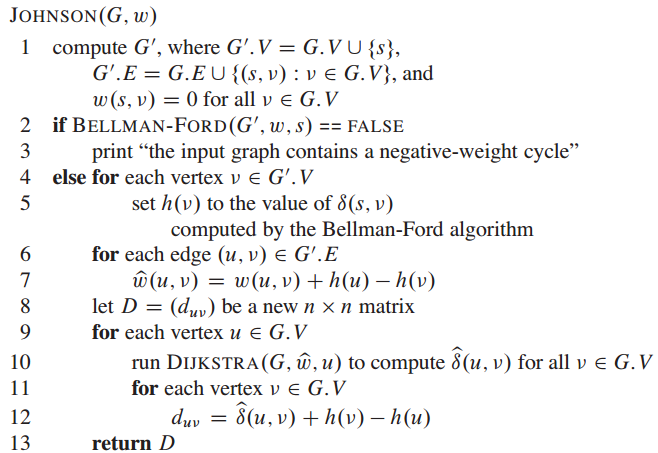
设为从结点i到结点j的所有中间结点全部取自集合{1, 2, … , k}的一条最短路径的权重。





#### 25.3 用于稀疏图的Johnson算法

重新赋予权重来维持最短路径

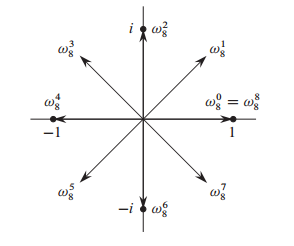


## CHAPTER 30 多项式与快速傅里叶变换

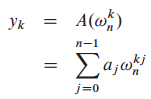
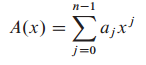
多项式的表示：

1. 系数表达

2. 点值表达



**DFT**

 y=（y0, y1, …, yn-1）是a=(a0, a1, …, an-1)的离散傅里叶变换（DFT），y=DFTn(a)

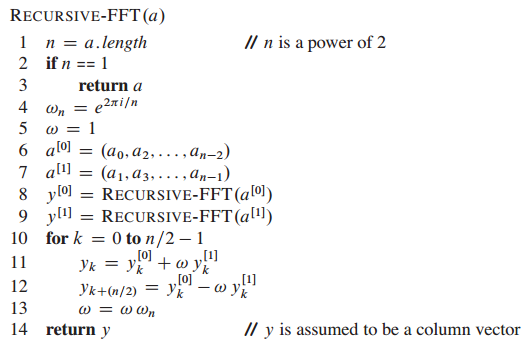
**FFT**

假设n是2的整数幂

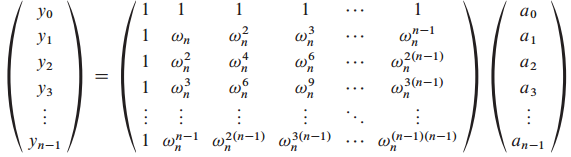


1. 求次数界为n/2的多项式

2. 根据式综合上述结果

运行时间为ϴ(nlgn)

在单位复数根出插值：



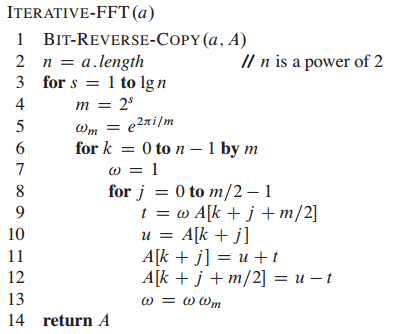
定理30.7 对j, k=0, 1, …, n-1，的(j, k)处元素为

根据逆矩阵，可以推导出：

定理30.8 （卷积定理） 对任意两个长度为n的向量a和b，其中n是2的幂，

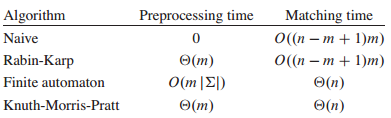
，其中向量a和b用0填充，使其长度达到2n，·为点乘。

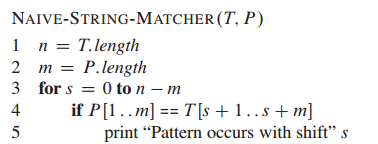
FFT的一种迭代实现：

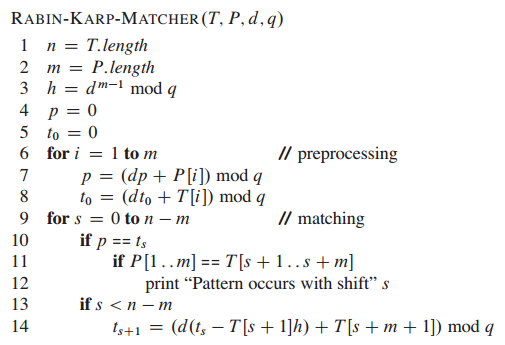


## CHAPTER 32 字符串匹配

 w是x的前缀  w是x的后缀







**有限自动机的构造：**

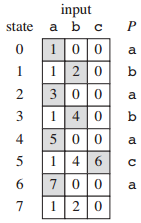
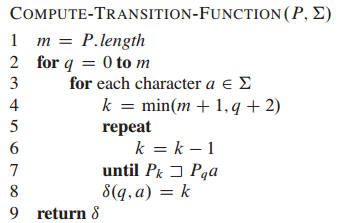
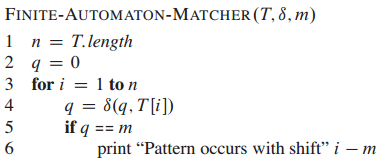




给定模式P[1..m]，其相应的字符串匹配自动机定义如下：

·状态集合Q为{0, 1, …, m}开始状态q0是0状态，并且只有状态m是唯一被接受的状态。

·对任意的状态q和字符a，转移函数



**KMP算法**

