目录

[第一部分 基础知识 3](#_Toc523776888)

[第1章 算法在计算中的作用 3](#_Toc523776889)

[1.1 算法 3](#_Toc523776890)

[1.2 作为一种技术的算法 3](#_Toc523776891)

[第2章 算法基础 3](#_Toc523776892)

[2.1 插入排序 3](#_Toc523776893)

[2.2 分析算法 3](#_Toc523776894)

[2.3 设计算法 3](#_Toc523776895)

[第3章 函数的增长 4](#_Toc523776896)

[3.1 渐近记号 4](#_Toc523776897)

[3.2 标准记号与常用函数 4](#_Toc523776898)

[第4章 分治策略 4](#_Toc523776899)

[4.1 最大子数组问题 4](#_Toc523776900)

[4.2 矩阵乘法的Strassen算法 4](#_Toc523776901)

[4.3 用代入法求解递归式 4](#_Toc523776902)

[4.4 用递归树方法求解递归式 4](#_Toc523776903)

[4.5 用主方法求解递归式 5](#_Toc523776904)

[4.6\* 证明主定理 5](#_Toc523776905)

[第5章 概率分析和随机算法 5](#_Toc523776906)

[5.1 雇用问题 5](#_Toc523776907)

[5.2 指示器随机变量 5](#_Toc523776908)

[5.3 随机算法 5](#_Toc523776909)

[5.4\* 概率分析和指示器随机变量的进一步使用 5](#_Toc523776910)

[第二部分 排序和顺序统计量 5](#_Toc523776911)

[第6章 堆排序（heapsort） 5](#_Toc523776912)

[6.1 堆 5](#_Toc523776913)

[6.2 维护堆的性质 5](#_Toc523776914)

[6.3 建堆 5](#_Toc523776915)

[6.4 堆排序算法 5](#_Toc523776916)

[6.5 优先队列（priority queue） 5](#_Toc523776917)

[第7章 快速排序 5](#_Toc523776918)

[7.1 快速排序的描述 5](#_Toc523776919)

[7.2 快速排序的性能 6](#_Toc523776920)

[7.3 快速排序的随机化版本 6](#_Toc523776921)

[7.4 快速排序分析 6](#_Toc523776922)

[第8章 线性时间排序 6](#_Toc523776923)

[8.1 排序算法的下界 6](#_Toc523776924)

[8.2 计数排序 6](#_Toc523776925)

[8.3 基数排序（radix sort） 6](#_Toc523776926)

[8.4 桶排序（bucket sort） 6](#_Toc523776927)

[第9章 中位数和顺序统计量 6](#_Toc523776928)

[9.1 最小值和最大值 6](#_Toc523776929)

[9.2 期望为线性时间的选择算法 6](#_Toc523776930)

[9.3 最坏情况为线性时间的选择算法 6](#_Toc523776931)

[第三部分 数据结构 6](#_Toc523776932)

[第10章 基本数据结构 6](#_Toc523776933)

[10.1 栈和队列 6](#_Toc523776934)

[10.2 链表 6](#_Toc523776935)

[10.3 指针和对象的实现 7](#_Toc523776936)

[10.4 有根树的表示 7](#_Toc523776937)

[第11章 散列表 7](#_Toc523776938)

[11.1 直接寻址表（direct-address table） 7](#_Toc523776939)

[11.2 散列表 7](#_Toc523776940)

[11.3 散列函数 7](#_Toc523776941)

[11.4 开放寻址法（open addressing） 7](#_Toc523776942)

[11.5 \* 完全散列 7](#_Toc523776943)

[第12章 二叉搜索树 8](#_Toc523776944)

[12.1 什么是二叉搜索树 8](#_Toc523776945)

[12.2 查询二叉搜索树 8](#_Toc523776946)

[12.3 插入和删除 8](#_Toc523776947)

[12.4 随机构建二叉搜索树（randomly built binary search tree） 8](#_Toc523776948)

[第13章 红黑树 8](#_Toc523776949)

[13.1 红黑树的性质 8](#_Toc523776950)

[13.2 旋转 8](#_Toc523776951)

[13.3 插入 8](#_Toc523776952)

[13.4 删除 8](#_Toc523776953)

[第14章 数据结构的扩张 8](#_Toc523776954)

[14.1 动态顺序统计 8](#_Toc523776955)

[14.2 如何扩张数据结构 8](#_Toc523776956)

[14.3 区间树（interval tree） 8](#_Toc523776957)

[第四部分 高级技术和分析技术 9](#_Toc523776958)

[第15章 动态规划（dynamic programming） 9](#_Toc523776959)

[15.1 钢条切割 9](#_Toc523776960)

[15.2 矩阵链乘法 9](#_Toc523776961)

[15.3 动态规划原理 9](#_Toc523776962)

[15.4 最长公共子序列（longest-common-subsequence problem） 10](#_Toc523776963)

[15.5 最优二叉搜索树（optimal binary search tree） 10](#_Toc523776964)

[第16章 贪心算法（greedy algorithm） 10](#_Toc523776965)

[16.1 活动选择问题 10](#_Toc523776966)

[16.2 贪心算法原理 10](#_Toc523776967)

[16.3 赫夫曼编码 11](#_Toc523776968)

[16.4\* 拟阵和贪心算法 11](#_Toc523776969)

[16.5\* 用拟阵求解任务调度问题 11](#_Toc523776970)

[第17章 摊还分析（amortized analysis） 11](#_Toc523776971)

[17.1 聚合分析aggregate analysis 11](#_Toc523776972)

[17.2 核算法accounting method 11](#_Toc523776973)

[17.3 势能法potential method 11](#_Toc523776974)

[17.4 动态表 11](#_Toc523776975)

[第五部分 高级数据结构 11](#_Toc523776976)

[第18章 B树 11](#_Toc523776977)

[第21章 用于不相交集合的数据结构（disjoint-set data structure） 11](#_Toc523776978)

[21.1 不相交集合的操作 11](#_Toc523776979)

[21.2 不相交集合的链表表示 12](#_Toc523776980)

[21.3 不相交集合森林 12](#_Toc523776981)

算法策略总结与特征比较：

·分治法（递归法🡪迭代）   
特点：存在不同层次互不相交的子问题，子问题合并得上一级结果，最终合并为原问题。一般算法结构分两部分：子问题的递归和同级子问题的合并（或类似操作）

·增量法   
特点：只有一个子问题，子问题由小至大，一点点增加规模至原问题

·动态规划（自顶向下递归🡪自底向上迭代）【最优化】  
特点：存在不同层次的子问题，但子问题之间在更低层面有所重叠。将低层问题结果记录在表格或类似数据结构中，便于高层问题重复利用。如果使用纯粹的分治法，会导致不必要的对子问题的重复计算。

·贪心算法【最优化】  
特点：简单直接，通过对每个子选择都追求局部最优来实现最快的最优化问题解答。当然不是所有问题都适合。

# 第一部分 基础知识

## 第1章 算法在计算中的作用

### 1.1 算法

### 1.2 作为一种技术的算法

## 第2章 算法基础

### 2.1 插入排序

**循环不变式**用以证明算法正确性，其性质如下，须证明这些性质才能证明其为循环不变式，进而印证算法正确性。p10/24  
·初始化  
·保持  
·终止

伪代码中的一些常见约定：p11/25

### 2.2 分析算法

大O的引入

### 2.3 设计算法

2.3.1 分治法

思想：将原问题分解为几个规模较小但类似于原问题的子问题，递归求解这些子问题，然后合并求解。或将原问题分解为几个规模较小但类似原问题的子问题以及一些与原问题不同的子问题。

2.3.3 分析分治算法

用递归树分析并求解递归式，递归式示例p20/34，递归树示例p21/35

## 第3章 函数的增长

### 3.1 渐近记号

Θ渐近紧确界asymptotically tight bound

Ο渐近上界

Ω渐近下界

ο非渐近紧确上界

ω非渐近紧确下界

### 3.2 标准记号与常用函数

包含一些基本的数学知识：p31/45

函数单调性

向下取整与向上取整函数性质

模运算定义

多项式定义、指数定义与性质，对数定义与性质，阶乘的定义与性质

多重函数定义，多重对数函数以及lg\*n的定义

斐波那契数的定义与性质

## 第4章 分治策略

求解递归式的三种方法——求解递归算法时间复杂度：  
1.代入法  
2.递归树法  
3.主方法

求解递归式时，通常忽略向下或向上取整以及边界条件等，因为多数情况下这些不会影响分析结果至多一个常数级。但具体情况仍需具体分析。

### 4.1 最大子数组问题

使用分治法O(nlgn)

使用增量法O(n)

### 4.2 矩阵乘法的Strassen算法

矩阵乘法的定义式算法需要Ω(n^3)时间，但Strassen算法将n×n矩阵相乘时耗降为Θ(n^lg7)或约为O(n^2.81)，算法详见p45/59

### 4.3 用代入法求解递归式

通过代数方法，数学归纳法，以及猜测解的形式证明递归式的结果——时间复杂度。p48/62

注意：使用代入法的归纳证明时，若发现证明结果相差一个相对猜测结果而言的低阶项，可以在原猜测结果上减去一个低阶项再证。实际上，更紧的界不一定难证。

### 4.4 用递归树方法求解递归式

递归树法可以用以为代入法生成基础猜测，若递归树的计算足够精确也可以自己求解递归式的结果。然而很多时候精确求解递归树很难，生成一个大致猜测后用代入法证明更简单，因猜测结果在输入规模趋向极限时不见得一定错误。

### 4.5 用主方法求解递归式

对一种泛型递归式的定理性求解：

主定理：即主方法依赖的定理，包含三种情况以及三种解。三种情况并非覆盖全部情况，要先严格证明问题符合主定理适用的三种情况之一，p53/67

### 4.6\* 证明主定理

暂不必要，未看

## 第5章 概率分析和随机算法

### 5.1 雇用问题

随机算法：如果一个算法的行为不仅由输入决定，而且也由随机数生成器决定，则这个算法是随机的

平均运行时间：当概率分布是在算法的输入上时，对算法时间的估计

期望运行时间：当算法内部存在随机选择时，对算法时间的估计

### 5.2 指示器随机变量

指示器随机变量是一个分析方法或概率分析工具，该变量在事件发生时为1，反之为0，为概率与期望之间的转换提供了一个便利方法。p67/81

### 5.3 随机算法

随机算法即在算法流程中引入随机机制（随机数生成器），随机算法的意义在于将不确定概率分布的输入转换成已知概率分布的结果（期望）。这可以用来稳定算法的性能，即可以确保算法结果一定为一个几乎可以确定的期望值，这在很多实际应用中非常需要。但显然不是所有的情形下都适合使用随机算法，需要依据不同的问题与需求，尤其是不能随意改变、打乱输入的情况下。

随机排列一个数组的两种算法p70/84

### 5.4\* 概率分析和指示器随机变量的进一步使用

暂不必要，未看

# 第二部分 排序和顺序统计量

## 第6章 堆排序（heapsort）

### 6.1 堆（二叉堆binary heap）

见数据结构知识笔记

### 6.2 维护堆的性质

见数据结构笔记中维护大顶堆或最大堆结构的算法

### 6.3 建堆

反复调用维护堆算法，从底至上遍历堆的非叶节点即可，时间复杂度为O(n)

### 6.4 堆排序算法

先将输入序列建成最大堆，然后反复调用维护堆算法，同时将堆顶移至未排序部分的尾部

### 6.5 优先队列（priority queue）

包含INSERT插入，MAXIMUM返回最大值，EXTRACT-MAX去掉最大值，INCREASE-KEY提高关键字（优先级）p90/104

## 第7章 快速排序

### 7.1 快速排序的描述

略

### 7.2 快速排序的性能

定性分析，略

### 7.3 快速排序的随机化版本

略

### 7.4 快速排序分析

快速排序性能的严谨分析

暂不必要，未看

## 第8章 线性时间排序

### 8.1 排序算法的下界

比较排序的算法下界为nlgn

### 8.2 计数排序

一种类似散列表形式的排序方法，稳定算法

### 8.3 基数排序（radix sort）

是一个排序方式，具体的排序实现仍然取决于它嵌套什么排序算法。基数排序方式的特点就是从待排序数据的低位（或低级）开始向高位，逐位（级）对所有数据进行排序。适合待排序数据并非数值的一般情况。

### 8.4 桶排序（bucket sort）

先使用散列技术将数据存放于不同的桶中（大类排序），桶内进行插入排序，桶外直接合并即可。

## 第9章 中位数和顺序统计量

顺序统计量order statistic：第i个顺序统计量是集合中第i小的元素

中位数median：第(n+1)/2个顺序统计量，n为输入规模。实际上中位数不止一个，对于n为偶数的情况存在上中位数和下中位数两种。

### 9.1 最小值和最大值

最大值和最小值通过比较法分别需要n-1次比较。同时找到最大与最小值则只需3n/2次比较，即每两个数据比较后将较大者与当前最大值比较，较小者与当前最小值比较，则每两个数据进行3次比较。

### 9.2 期望为线性时间的选择算法

基于快速排序算法中的快速划分算法（随机化版本），然后将pivot和所求的顺序号i进行比较，决定下一次递归深入划分后的哪一个区域继续查找。

### 9.3 最坏情况为线性时间的选择算法

同样基于快速划分，然而先将输入数据划分为固定大小的小组，对每小组排序后去中位数，对所有小组的中位数再取中位数，最后以该中位数作为理想的pivot来划分输入数据，余下的部分和递归方式与上一个算法类似。

# 第三部分 数据结构

## 第10章 基本数据结构

### 10.1 栈和队列

利用数组实现栈和队列的方法

### 10.2 链表

双向链表doubly linked list：元素包含两个指针，分别指向前驱和后继

单链接singly linked：只包含指向后继的指针

循环链表circular list：首尾相接的链表

哨兵sentinel节点：充当链表首尾的边界节点，虽然可以简化代码，但会增加额外空间消耗

### 10.3 指针和对象的实现

·对不支持指针和对象数据类型的编程语言，如何使用数组等基本数据结构模拟。

①对象的多数组并列表示法

适用于同构对象

②对象的单数组串联表示法

适用于同构或异构对象

·对象的手动分配与释放，以多数组表示法为例p135/149

### 10.4 有根树的表示

二叉树表示法

分支无限制的有根树：左孩子右兄弟表示法left-child，right-sibling representation

其他表示法：并没有绝对正确的表示法，根据具体需求选取不同的数据结构实现方式

## 第11章 散列表

### 11.1 直接寻址表（direct-address table）

最理想情况下的散列表，关键字全域不是很大时，使用数组元素一一对应每个关键字

### 11.2 散列表

链接法chaining排解冲突：使用独立链表来存放冲突元素

### 11.3 散列函数

11.3.1 除法散列法

将关键字除以一个数的余数作为散列值。该数最好为一个不太接近2的整数幂的素数

11.3.2 乘法散列法

不是简单的乘以一个数作为散列值，而是利用关键字和一个常数的乘积结果的小数部分的一部分来作为散列值。p148/162

11.3.3\* 全域散列法universal hashing

全域散列法不使用单一固定的散列函数，而是用一组函数，每次随机选择一个作为散列函数。

设计一个全域散列函数类：数论知识，未看

### 11.4 开放寻址法（open addressing）

开放寻址法是一种不用散列表外存储结构（即数组外）来进行冲突解决的实现方式，或者说是针对独立链法的另一种方式。然而根据其特点，在必须删除关键字的应用中，开放寻址并不是一个好的实现方式，因为删除后的槽并非空槽，只是在插入时等效，而在查找时这些实际意义上的空槽会浪费查找操作的性能。

①线性探查

试探偏移量为线性递增

②二次探查quadratic probing

试探偏移量以二次函数递增

③双重散列double hashing

试探偏移量以关键字的某个函数方式递增，该函数可以看成是另一个散列函数

开放寻址散列分析p154/168

### 11.5 \* 完全散列

数论知识，未看p170

## 第12章 二叉搜索树

### 12.1 什么是二叉搜索树

二叉搜索树性质

### 12.2 查询二叉搜索树

查找p163/177

最大关键字元素和最小关键字元素p163/177

后继和前驱p164/178

### 12.3 插入和删除

略，插入简单，删除较复杂

### 12.4 随机构建二叉搜索树（randomly built binary search tree）

定义：按随机次序插入关键字到一棵空树形成的树，即关键字的所有排列等可能出现作为插入操作的输入，然而这不同于由这些关键字组成的所有可能的二叉搜索树等可能出现。

随机构建二叉搜索树的高度期望为O(lgn)

## 第13章 红黑树

### 13.1 红黑树的性质

平衡二叉搜索树的一种

### 13.2 旋转

对红黑树结构改变的一个子操作

### 13.3 插入

可参考数据结构C++笔记，暂不必要，未看

### 13.4 删除

可参考数据结构C++笔记，暂不必要，未看

## 第14章 数据结构的扩张

### 14.1 动态顺序统计

顺序统计树（order-statistic tree）：在每个结点上增加size属性的红黑树，size为以该结点为根的子树包含的所有内结点树。

有了动态顺序统计，当然还要更改相应的插入和删除算法用以加入更新size属性的部分，就可以快速地依据二叉搜索树的性质递归寻找顺序统计量的位置。

### 14.2 如何扩张数据结构

①增加附加信息或属性

②修改已有的基本操作以维护附加信息或属性

③增加该信息或属性支持的新操作

### 14.3 区间树（interval tree）

区间三分律interval trichotomy：任意两个区间必满足三条性质之一  
①两个区间重叠——包括两个区间彼此包含、两个区间前后重叠每个情景两种共四种情况

②两个区间一左一右

③两个区间一右一左

区间树是一个基于红黑树扩展的数据结构。

# 第四部分 高级技术和分析技术

## 第15章 动态规划（dynamic programming）

•动态规划的英文含义不是指编程，而是一种动态表格记录法，这也表明了它的特点。

•动态规划方法通常用来求解最优化问题（optimization problem）的一个最优解（an optimal solution）

•通常按照如下4个步骤来设计一个动态规划算法：

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

•动态规划方法是付出额外的内存空间来节省计算时间，是典型的时空权衡（time-memory trade-off）的例子，一般来说动态规划方法的总运行时间是多项式阶的

•动态规划有两种等价的实现方法

①带备忘的自顶向下法（top-down with memoization）：带有备忘的递归调用，避免重复计算；仍按照自然地递归形式编写。

②自底向上法（bottom-up method）：不需要递归调用，但是一般需要恰当定义子问题，使得任何子问题的求解都只依赖于更小的子问题的解。因为没有递归调用，虽然两种方法具有相同的渐近运行时间，但是自底向上法的时间复杂度具有更小的常系数

### 15.1 钢条切割

•钢条切割问题：给定一段长度为n英寸的钢条和一个价格表pi(i=1,2,…,n),求切割钢条方案，使得销售收益rn最大。

### 15.2 矩阵链乘法

•完全括号化（fully parenthesized）：它要么是单一矩阵，或者是两个完全括号化的矩阵乘积链的乘积，且已经外加括号。

•矩阵链乘法问题（matrix-chain multiplication problem）：给定n个矩阵的链，<A1,A2,…,An>，矩阵Ai的规模为pi-1×pi(1≤i≤n)，求完全括号化方案，使得计算矩阵乘积所需标量乘法次数最少。

### 15.3 动态规划原理

•适合应用动态规划方法求解的最优化问题应具备两个要素：

①最优子结构

最优子结构（optimal substructure）：问题的最优解由相关子问题的最优解组合而成，而这些子问题可以独立求解

子问题无关（independent）：同一个问题的多个子问题相互独立，彼此不影响，都可以自由求解最优值

无权最短路径问题满足最优子结构，无权最长路径问题不满足。

②子问题重叠

重叠子问题（overlapping subproblems）：如果递归算法反复求解相同的子问题，这样可以使用动态规划的表格法，如果每次递归产生的是全新的子问题，则应使用标准的分治法。

·自顶向下带备忘的算法不一定就比较差，尤其是在不必完全求解子问题空间时，此方法只会求解必要的子问题；而自底向上的方法一般会全面求解子问题。

·优化空间：对于动态规划的表格法也有学问，根据实际问题的情况使用不同的方法存储子问题的结果，可以降低空间开销。一方面，使用不同的数据结构，数组、散列表都可以完成这项任务，后者有时效率更高。另一方面，不一定要将所有子问题的结果全程保存，有时解决了更高层的子问题后低层的子问题结果将不再有用。

### 15.4 最长公共子序列（longest-common-subsequence problem）

两个序列中最长的公共子序列查找问题是一个动态规划问题。  
·确定子问题形式，一般可分为一维和二维（形容子问题规模的参数的数量）  
·建立最优解的递归式，通过递归式可以证明问题是否具有最优子结构和子问题重叠性质。  
·根据递归式可以设计出自底向上的迭代算法。注意递归式中的每个含有维度的量都应该有一个表来记录其值以避免重复计算，亦即除了用于重构最优解的表和记录最优解值的主表外还可能需要其他的表用以记录递归式中的量。

### 15.5 最优二叉搜索树（optimal binary search tree）

给定每个关键字（结点）的搜索概率分布的前提下，构造一棵期望搜索代价最小的二叉搜索树。搜索代价即为搜索路径访问的结点数。

递归式的形式为：从i到j的关键字子序列构成的子树的最优搜索代价为将该子序列划分为两个更小的子序列的搜索代价之和，以及两个子子序列成为根节点子树时额外产生的代价。要在所有的划分中选择最小代价的划分。

## 第16章 贪心算法（greedy algorithm）

### 16.1 活动选择问题

贪心算法一般也可以使用动态规划问题解决，可以视为一种特殊的动态规划问题类型。

### 16.2 贪心算法原理

贪心算法能够解决的问题必须具备的性质：

①贪心选择性质（greedy-choice property）

虽然贪心和动态规划有可求解同一类型问题的能力，但动态规划一般是自底向上的（无论采用递归还是迭代，其实质的计算过程），而贪心算则一般是自顶向下的，逐渐通过无需事先求解子问题即可做出的局部最优选择来逐渐缩小问题规模。

②最优子结构

虽然贪心算法的最优子结构和动态规划的概念一样，但是不同之处在于动态规划是由最优子问题组合而成最优解，而贪心算法是最优解由下一级最优子问题和贪心选择组成。可以理解为每一次贪心选择将原问题降低一个规模，但仍然是相似的问题。

贪心选择vs动态规划：  
道理很显然，可用于贪心选择的问题显然是可用于动态规划的问题的子集。不能进行贪心选择的问题一般需要针对该选择进行数次横向比较才能确定最优或非最优，此时必须依靠动态规划。反之，可以证明贪心选择正确的算法自然也就是正确的可以简化动态规划作出选择的算法。  
一般对于贪心选择的证明过程大致围绕在贪心选择的结果并不会产生任何其他可能存在的对最优目标不利的因素，如果有可能的不利因素那么必须通过比较来确定最终结果；反之，可以很容易证明贪心选择的结果一定是最优解的一个子集，即每次贪心选择都独立地只为最优解贡献正向影响。  
另外，对于可用于贪心选择的问题也不一定只有一种贪心选择标准，而贪心选择有时提供的也只是最优解的一个子集。

举例：0-1背包问题（0-1knapsack problem）只能用动态规划，分数背包问题（fraction knapsack problem）可以用贪心，贪心选择为每次选取性价比最大的物品装入背包。贪心选择对于分数背包问题无负面影响，因为背包一定会满，但0-1背包问题可能存在空间利用率不足的问题，因此（这一种）贪心选择无效了。

### 16.3 赫夫曼编码

略

### 16.4\* 拟阵和贪心算法

暂不必要，未看

### 16.5\* 用拟阵求解任务调度问题

暂不必要，未看

## 第17章 摊还分析（amortized analysis）

定义：求数据结构的一个操作序列中所执行的所有操作的平均时间。注意平均时间确切定义为摊还时间，每个操作的摊还时间可相等可不等，依分析方法而异。单独的操作的实际代价必然是已知的。

摊还分析有几种具体的实现方法。如下：

### 17.1 聚合分析aggregate analysis

原理：首先证明对所有n，一个n个操作的操作序列最坏情况下花费的总时间为T(n)，则最坏情况下，每个操作的平均代价或摊还代价为T(n)/n。即为每个操作不论类型，均赋予一致的摊还代价。

### 17.2 核算法accounting method

原理：首先为每个操作赋予摊还代价，是为预估过程，每个操作的摊还代价可以不相等。之后针对所有可能的操作序列，保证至每步操作都不使总摊还代价小于总实际代价，是为核算过程。可以理解为先预测后验证的分析方法。

核算法使用了会计学的概念，其中每个操作的摊还代价与实际代价的差值，若为正称为信用，若为负称为负债。

### 17.3 势能法potential method

原理：道理上与核算法类似，只是将“资产”的概念解释为“势能”。对每个操作的摊还代价不使用预估，而是选取一个势函数计算得来。势函数是从被考察数据结构到一个数值（势能）的映射。势函数的选取必须保证每一步操作后的数据结构的势都不小于初始的势，才能保证总摊还代价始终不小于总实际代价。

### 17.4 动态表

动态表是一种广义的数据结构，也就是具体而言它是列表、向量、栈或堆都不重要，它的特点在于可以根据其内容多少自己分配更多或更少空间，即扩张与收缩。

17.4.1 表扩张

参加数据结构C++中向量的空间扩张部分知识，此处多为利用摊还分析分析扩张操作的性能，暂未看

17.4.2 表扩张和收缩

# 第五部分 高级数据结构

其他高级数据结构：

动态树

伸展树

持久数据结构

聚合树

指数搜索树

动态图数据结构

## 第18章 B树

参考数据结构C++笔记，暂未看

## 第19章 斐波那契堆

可合并堆mergeable heap是支持5种操作的一种数据结构，其中每个元素有一个关键字：  
MEAP-HEAP()建堆  
INSERT(H, x)插入  
MINIMUM(H)返回最小值（针对最小堆）  
EXTRACT-MIN(H)返回并删除最小值  
UNION(H1,H2)合并二堆

二项堆（binomial heap）是一种类似于二叉堆（binary heap）的堆结构。与二叉堆相比，其优势是可以快速合并两个堆，因此它属于可合并堆（mergeable heap）抽象数据类型的一种。二项堆是指满足以下性质的二项树的集合：  
①每棵二项树都满足最小堆性质，即结点关键字大于等于其父结点的值  
②不能有两棵或以上的二项树有相同度数（包括度数为0）。换句话说，具有度数k的二项树有0个或1个。

二项树递归定义如下：  
①度数为0的二项树只包含一个结点  
②度数为k的二项树有一个根结点，根结点下有k个子女，每个子女分别是度数分别为k-1,k-2,...,2,1,0的二项树的根

## 第21章 用于不相交集合的数据结构（disjoint-set data structure）

### 21.1 不相交集合的操作

不相交集合数据结构本身是一组集合S1,S2,S3…的集合S。其成员子集合彼此不相交。

代表representative：用一个数据成员代表其所在的集合成员，代表的选择依具体实现方法而定，但选择必须稳定。

功能接口：

MAKE-SET(x) 建立一个以x为数据成员的集合成员，x为唯一成员和代表

UINON(x,y) 将x，y所在的集合成员合并为一个集合成员，代表的选择自定义

FIND-SET(x) 返回x所在的集合成员，即其代表的指针

### 21.2 不相交集合的链表表示

可以使用链表作为核心结构构建不相交集合中的每个集合成员：p326/340

加权合并启发式策略weighted-union heuristic：对于链表表示的不相交集合的合并union操作，在集合中维护一个表长度数据，每次将较短的集合合并至较长的集合，可提高效率。

### 21.3 不相交集合森林

不相交集合森林disjoint-set forest即使用有根树表示集合成员组成的不相交集合的一种实现：p328/342

按秩合并union by rank启发式策略：在每个节点中维护一个rank变量，用于优化UNION操作p329/343

路径压缩path compression启发式策略：在每个FIND-SET操作中，顺便将查找路径上的所有节点都直接至于根节点之下，用于优化整个数据结构p329/343