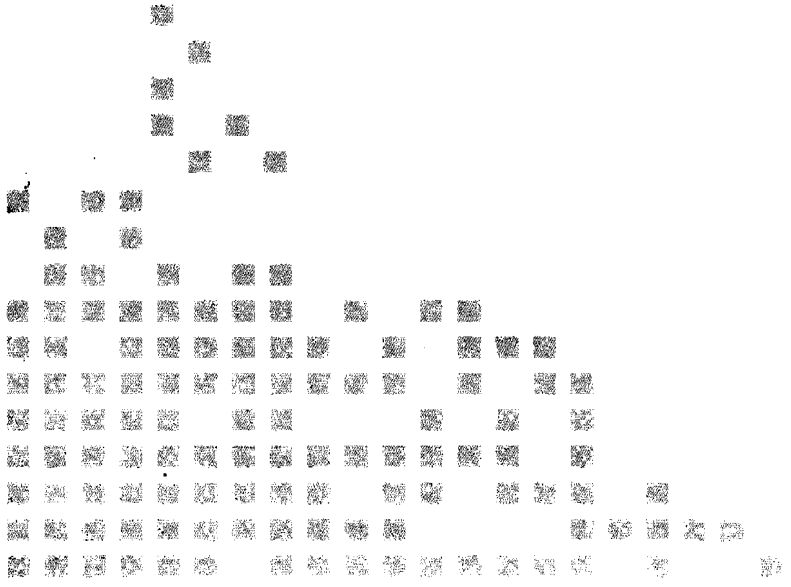
朱明方 吴及 编著

数据结构与算法习题解答和实习指导

清华大学出版社

朱明方 吴及 编著

数据结构与算法
习题解答和实习指导



清华大学出版社
北京

内 容 简 介

本书是主教材《数据结构与算法》的配套辅导用书，其内容包括主教材中各章习题的解答和上机实习指导两部分。

在习题解答部分中，对主教材中的各章习题作了详细解答，有意识地突出了对重要概念和知识点的解释，对可以用多种方法和思路解决的问题，同时给出几种不同的求解方法或算法。

上机实习指导部分中，包括上机实验指导和上机大作业两部分。其中，实验部分配合课程内容给出了12个实验；而上机大作业部分要求在规定时间内独立上机完成6个与实际问题比较贴近、有一定综合性的作业题。为兼顾读者希望有足够的独立思考空间和解决难以下手的实际情况，在实验和上机大作业题中，除了明确目的、要求以外，还给出了求解的思路或实现算法的提示。

通过这些书面和上机的练习，可以加深读者对课程中的重要概念和知识点的理解，掌握重要知识的应用、锻炼独立分析问题和解决问题的能力，从而达到更好的学习效果。

本书可作为普通高等院校数据结构课程的辅助教材，也可供自学者参考。

图书在版编目(CIP)数据

数据结构与算法习题解答和实习指导/朱明方，吴及编著．—北京：清华大学出版社，2011.4

ISBN 978-7-302-23367-1

Ⅰ．①数…　Ⅱ．①朱…　②吴…　Ⅲ．①数据结构—高等学校—教学参考资料　②算法分析—高等学校—教学参考资料　Ⅳ．①TP311.12

中国版本图书馆CIP数据核字（2010）第152997号

责任编辑：王一玲　刘佩伟
责任校对：李建庄
责任印制：何　芊
出版发行：清华大学出版社　　**地　　址**：北京清华大学学研大厦A座
http://www.tup.com.cn　　**邮　　编**：100084
社　总　机：010-62770175　　**邮　购**：010-62786544
投稿与读者服务：010-62795954，jsjjc@tup.tsinghua.edu.cn
质　量　反　馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
印 装 者：北京国马印刷厂
经　　销：全国新华书店
开　　本：185×260　**印　张**：9.25　**字　数**：231千字
版　　次：2011年4月第1版　　**印　　次**：2011年4月第1次印刷
印　　数：1～3000
定　　价：18.00元

产品编号：036334-01

前　言

实践训练是学习数据结构与算法设计课程极其重要的环节。本书是配合主教材《数据结构与算法》的实习指导用书。它可以方便读者进行实习训练（包括书面和上机训练），以取得更好的学习效果。

本书内容包括对主教材中各章习题的解答和上机实习指导两部分。

习题解答部分：对教材中的各章习题作了详细的解答，在问题解答中有意识地突出了对重要概念和知识点的解释，对可以用多种方法和思路解决的问题，同时给出不同的求解方法或算法。其目的是希望通过对问题的解答，帮助读者进一步加深对课程中重要概念和方法的掌握与理解。

上机实习指导部分：包括“实验指导”和“上机大作业”两部分。其中，“实验指导”部分配合课程内容、以循序渐进的原则，给出了 12 个实验。各个实验之间，既有层次关系又相对独立，以方便不同基础和不同要求的读者更有针对性地选择实验训练内容。“上机大作业”要求读者在规定时间内独立上机完成，给出了 6 个与实际问题比较贴近、有一定综合性的作业题。对于课程教学，如果条件允许的话，按照作业中的具体要求，很容易通过作业测试系统在计算机上统一批改、评分。为了兼顾给读者留有足够的思考空间和难以下手两种情况，在实验和上机大作业题中，除了明确目的、要求以外，还给出了问题求解的思路或实现算法的提示。

针对非计算机专业学生的实际情况，进行上机实验和完成上机大作业时，读者可以根据自身的实际情况，确定选用 C 语言或 C++语言。

通过这些书面和上机的练习，读者不仅可以加深对课程中所学的重要概念和知识点的理解，还能够掌握重要知识的应用、锻炼独立分析问题和解决问题的能力，从而达到更好的学习效果。

本书第 5～7 章的习题解答由吴及执笔，其余部分由朱明方执笔，朱峰做了部分算法实现与验证的工作，全书由朱明方统稿。

本书的编写得到了清华大学教务处和清华大学出版社的大力支持，在此表示衷心的感谢。

限于作者的水平，书中难免存在疏漏和不足之处，敬请读者批评指正。

作　者

2011 年 2 月

目 录

第一部分 习 题 解 答

第二部分 上机实习指导

第一部分 习题解答

第1章 绪 论

1.1 重点与难点

本章主要介绍数据结构和算法的基本概念、算法设计的常用思路、算法分析的基本方法，为以后章节的学习作准备。

本章重点

（1）“关系”的定义和基本性质，它在实际问题中所表示的实际意义。
（2）数据结构的概念，包括数据的逻辑结构、存储结构，以及它们之间的关系。
（3）抽象数据类型的概念及它与数据结构之间的关系。
（4）算法设计常用的方法思路。
（5）算法的时间复杂度和空间复杂度的概念，以及分析方法。

内容难点

（1）掌握二元关系的基本性质和几种常用关系的特点。
（2）掌握算法的时间复杂度和空间复杂度的基本分析方法。

1.2 习题解答

习题 1.1 什么是二元关系？“给定集合 M 上的一个关系 R”，其含义是什么？

[**解答**] 集合 A 对集合 B 的笛卡儿积的任意一个子集称为 A 到 B 的一个二元关系。

在实际问题中，二元关系表示了集合中元素之间的某种关联性。例如，假设有一个父亲集合和一个孩子集合，那么，父亲集合到孩子集合的一个二元关系，就可以表示父亲集合中的父亲和孩子集合中的孩子之间的“父子”关系。若集合 A 和集合 B 相等，此时 A 到 B 的二元关系表示的是集合 A（或集合 B）中元素之间的某种相关性。例如，一个学生集合的一个二元关系就可以表示学生之间的“同班”关系。当然根据需要也可以有表示“同寝室”或“同兴趣小组”等其他的二元关系。所谓“给定集合 M 上的一个关系 R”，就是指对集合 M 自身的笛卡儿积所取的一个子集 R，对实际问题而言，它表示的是集合 M 中元素的一种相关性，正如前面所说的学生集合的“同班”关系、“同寝室”关系那样。

习题 1.2 设有集合 $M=\{d_1,d_2,d_3,d_4,d_5\}$上的一个关系：

$R=\{(d_1,d_2),(d_2,d_4),(d_4,d_5),(d_2,d_5),(d_1,d_4),(d_1,d_5),(d_3,d_5),(d_1,d_3)\}$，试说明关系 R 具有什么样的性质。

［**解答**］ 从二元关系的基本性质容易验证，该关系 R 是反自反的、反对称的、传递的关系。因为关系 R 中没有（d_i,d_i）这样的元素，所以它是反自反的。因为关系 R 中没有（元素 d_i,d_j）和（d_j,d_i）同时存在的情况，所以它是反对称的。

关系 R 的传递性表现在：

有元素$(d_1,d_2),(d_2,d_4)$，同时有元素(d_1,d_4)；

有元素$(d_1,d_2),(d_2,d_5)$，同时有元素(d_1,d_5)；

有元素$(d_1,d_3),(d_3,d_5)$，同时有元素(d_1,d_5)；

有元素$(d_1,d_4),(d_4,d_5)$，同时有元素(d_1,d_5)；

有元素$(d_2,d_4),(d_4,d_5)$，同时有元素(d_2,d_5)。

习题 1.3 什么是数据结构？什么是线性结构？什么是非线性结构？举例说明。

［**解答**］ 概括地说，数据结构是互相有关联的数据元素的集合。也就是说，数据结构是由某个数据元素的集合和该集合中的数据元素之间的关系组成的，因此数据结构可以用一个二元组来表示。例如，$B=(D,R)$，其中 D 是某一数据元素的集合，R 是 D 上的关系的有限集。R 所表示的是集合 D 的数据元素之间的逻辑关系，它表示的可能是数据元素之间客观存在的某种关系，也可能是为了处理问题的需要而人为组织的数据元素之间的某种关系，因此，称之为数据的逻辑结构。例如，一个农历节气表，就构成了一个数据结构，其数据元素是一年的农历二十四节气，数据元素之间的关系是节气的时间先后关系。又如，一个某年级学生的成绩排序表，也是一个数据结构，其数据元素是包含成绩项的该年级的学生记录，数据元素之间的关系是学生之间的成绩高低关系。为了在计算机中进行数据处理，必须把从实际问题中抽象出来的数据的逻辑结构映象到计算机的存储器中，即要把抽象出来的数据元素集合 D 和数据元素之间的关系存储到计算机的存储器中，称为数据的物理结构或存储结构，它是数据的逻辑结构在计算机中的表示。

数据的逻辑结构可以划分为两大类，即线性结构与非线性结构。它们的主要区别是：线性结构表示的是数据元素之间一对一的关系，而非线性结构表示的是数据元素之间一对多或多对多的关系。

线性结构具有以下特点：

① 存在唯一的没有前驱、只有一个直接后继的“头”元素。

② 存在唯一的没有后继、只有一个直接前驱的“尾”元素。

③ 除了“头”元素和“尾”元素之外，集合中的每个元素有且只有一个直接前驱、有且只有一个直接后继。

由以上特点可以看出，线性结构中的数据元素之间存在一对一的关系，结构中的数据元素依照它们的逻辑关系可以排成一个有“头”、有“尾”的序列。例如，前面所说的农历节气表，就是一个线性结构，它是一个从“春分”开始，然后是“雨水”，……，最后是“大寒”，这样一个序列。

除线性结构以外的结构称为非线性结构。在非线性结构中，各数据元素的关系不一定

保持一个线性序列，每个数据元素可能与零个或多个数据元素有关系。也就是说，非线性结构中的数据元素之间存在一对多或者是多对多的关系。

例如，一个学校的教学组织关系可以构成一个有明显层次的数据结构：学校下属有若干学院，每个学院下设若干个系，每个系有多个研究所和教研组，有若干的学生班级。这个一对多的关系的抽象就是非线性结构。

又如，对一个销售系统的各个连锁店及相互之间的联系的抽象是一个非线性结构，这个数据结构中的数据元素是各连锁店，数据元素之间的关系是各连锁店之间的联系，因为各连锁店之间都可以有联系，显然各连锁店之间的关系是多对多的关系。也就是说，每一个连锁店都可以与其余多个连锁店发生联系，这个结构也是非线性结构。

习题 1.4 设数据元素的集合为 $D=\{a_1, a_2, a_3, a_4, a_5, a_6\}$，请分别画出与以下各关系 R 对应的数据结构 $B=(D, R)$ 的结构示意图，并指出它属于哪类结构。

（1）$R=\{(a_3, a_4), (a_4, a_5), (a_1, a_2), (a_2, a_3), (a_5, a_6)\}$

（2）$R=\{(a_3, a_2), (a_2, a_4), (a_3, a_1), (a_2, a_5), (a_2, a_6)\}$

（3）$R=\{(a_{i+1}, a_i) \mid i=5, 4, 3, 2, 1\}$

（4）$R=\{(a_i, a_j) | i>j\}$

（5）$R=\{\}$

[解答] （1）为线性结构，其图形表示如图 1-1（a）所示。

（2）为非线性结构，其图形表示如图 1-1（b）所示。

（3）为线性结构，其图形表示如图 1-1（c）所示。

（4）非线性结构，其图形表示如图 1-1（d）所示。

（5）集合结构，除了同属一个集合外，数据元素之间无其他关系。

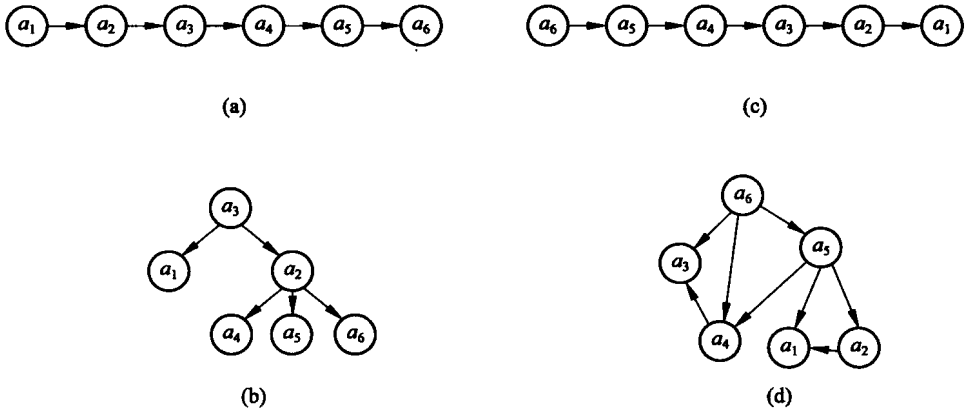


图 1-1

习题 1.5 数据的存储结构有哪几种？它们各有什么特点？

[解答] 数据存储结构又称物理结构。最常用的存储结构有顺序存储和链式存储两种存储方式。

在顺序存储方式中，要开辟一块连续的存储空间来存放数据结构；对每个数据元素给以等长的数据单元，结构中的数据元素按照它们之间的逻辑顺序依次存放于连续的内存单

元中。顺序存储方式的特点是：除了存储数据元素以外，不必耗费另外的空间，数据元素之间的关系是由数据元素在存储器中的邻接关系来表示的。由于数据元素在存储器中的物理顺序和它们之间的逻辑顺序一致，因此这种存储方式是非常直观的一种存储方式。

在链式存储方式中，数据元素可以存放在不连续的内存单元中，数据元素在存储器中的物理存放顺序可以和逻辑顺序不一致，数据元素之间的逻辑关系是通过指示数据元素存储地址的指针来表示的。因此，每个数据元素除了存储自身以外，同时还要存储指示其后件（或前件）的存储地址的指针，它们构成一个结点。也就是说，在链式存储方式中每个数据元素的存储映象是一个结点，它包括存储数据元素的数据域（又称值域）和存储指针的指针域两部分，通过各结点的指针把各数据元素按照它们的逻辑关系链成一条“链”，从而清晰的表示了数据元素之间的逻辑关系。链式存储的明显优点是存储空间的利用比较灵活，数据元素的增减操作比较方便。

除了上述两种常用的存储方式以外，还有索引存储方式和散列存储方式。

在索引存储方式中，按照某种性质把一个大表的元素划分成若干个子表，使每个子表中的元素具有相同的性质。存储时以子表为单位存放，同时建立一个索引表，索引表中的每个索引项对应一个子表，指出该子表的起始地址、长度和子表的性质，这样能够给查找等操作带来很大的方便。显然，在该存储方式下数据元素之间的逻辑关系是通过数据元素在索引表中的位置得以反映的。

在散列存储方式中，通过数据元素的关键字值来确定数据元素的存储位置，因而可以直接通过计算查找到相应的数据元素。使得它比通过“比较”查找有更高的效率。

习题 1.6 什么是抽象数据类型？抽象数据类型和面向对象的程序设计方法有什么关系？

[**解答**] 抽象数据类型是指用以表示应用问题的一个数据模型以及定义在该模型上的一组操作。抽象数据类型与一般的数据类型的概念在本质上是一致的，都是对数据类型的数学特性的抽象，其目的是可以使程序设计者，在程序设计中更专注于数据的逻辑特性，而不必关心抽象数据类型实现的具体细节。但抽象数据类型比一般数据类型的抽象层次更高、范畴更广，它不局限于计算机系统中已定义和实现的数据类型，通常它是由用户根据实际问题的需要而定义，且通过计算机系统中已经定义的数据类型来表示和实现。因此，抽象数据类型是基于一般数据类型的更高层次上的一种数据抽象。

抽象数据类型的概念是由于程序设计方法和技术的发展而提出来的。为了更好地提高软件模块的可复用性和可扩充性，现代程序设计方法强调以数据为基础来构建软件系统，更加强调“封装”和“信息隐蔽”策略。面向对象的程序设计方法正是符合这种要求的方法，“类”是面向对象的程序设计方法中的核心概念，它是数据抽象的结果，“类”不但体现了“封装”和“信息隐蔽”的原则，而且具有继承性，因而为模块的复用提供了很好的条件。抽象数据类型具有封装和信息隐蔽的特点，可以做到使用与实现分离。由此可见，抽象数据类型与面向对象的方法思想是一致的，从抽象数据类型出发来进行面向对象的程序设计，会使程序设计更加顺理成章。

习题 1.7 集合是由无重复的、相同类型元素组成、元素之间没有顺序关系的一种数据结构。假定集合是 1～Size 的整数集，用一个整型数组 m[Size+1]来表示它，若整数 i（$1 \leqslant i \leqslant$Size）在集合中，则 m[i]=1，若 i 不在集合中，则 m[i]=0；所需要的操作包括：对集合

初始化，向集合中加入一个元素，从集合中删除一个元素，对两个集合实现并、交、差运算。请定义出该类集合的抽象数据类型。

若定义类型：

```
struct Set
 { int m[Size+1]
 };
```

为数据成员，并通过重载加法、乘法和减法运算符的方法实现集合的并、交、差运算，若有C++基础，请用C++写出抽象数据类型中各操作的具体实现。

[解答] 对题中所要求的整数集合的抽象数据类型定义如下：

```
ADT Integ_Set
 { 数据:
     正整数集合Set，以整型数组元素m[1]~m[Size]表示;
   操作:
     void InitSet(Set &s);                  // 初始化集合中的数据成员
     void Insert(Set &s, int n);            // 向集合中加入一个元素
     void Delete(Set &s, int n);            // 从集合中删除一个元素
     Set operator+(Set s1, Set s2);         // 重载加法运算符实现集合的并运算
     Set operator*(Set s1, Set s2);         // 重载乘法运算符实现集合的交运算
     Set operator-(Set s1, Set s2);         // 重载减法运算符实现集合的差运算
 }
```

各操作实现如下：

(1) 初始化集合中的数据成员（集合中的所有元素清零）。

```
void InitSet(Set &s)
 { int i;
    for (i=1; i<=Size; i++)  s.m[i]=0;
 }
```

(2) 向集合中加入一个元素。

```
void Insert(Set &s, int n)
 { s.m[n]=1;
 }
```

(3) 从集合中删除一个元素。

```
void Delete(Set &s, int n)
 { s.m[n]=0;
 }
```

(4) 重载加法运算符实现集合的并运算。

```
Set operator+(Set s1, Set s2)
{ Set s;
```

```
    InitSet(s);
    for (int i=1;i<=Size;i++)
     if ((s1.m[i]==1)|| (s2.m[i]==1))  s.m[i]=1;
    return s;
}
```

(5) 重载乘法运算符实现集合的交运算。

```
Set operator*(Set s1, Set s2)
{ Set s;
    InitSet(s);
    for (int i=1;i<=Size;i++)
     if ((s1.m[i]==1)&&(s2.m[i]==1))  s.m[i]=1;
    return s;
}
```

(6) 重载减法运算符实现集合的差运算。

```
Set operator-(Set s1, Set s2)
{ Set s;
    InitSet(s);
    for (int i=1;i<=Size;i++)
      { if ((s1.m[i]==1)&&(s2.m[i]==1))  s.m[i]=0;
        else  s.m[i]=s1.m[i];
      }
    return s;
}
```

习题 1.8 完成以下问题：

(1) 设计二次多项式 ax^2+bx+c 的一种抽象数据类型，其数据部分为多项式的 3 个系数项 a、b、c；操作部分包括：初始化数据成员 a、b、c，实现两个多项式相加，给定 x 求多项式的值，求方程 $ax^2+bx+c=0$ 的两个实根，按照 $ax**2+bx+c$ 的格式输出二次多项式。

(2) 假定用记录类型定义数据成员 a、b、c，请写出上述各操作的具体实现。

[解答] (1) 对题中的二次多项式抽象数据类型可以作以下定义：

```
ADT Quadratic
 {数据:
一元二次多项式的二次项系数 a，一次项系数 b，常数项 c; 其结构类型为: Quadratic 操作:
  // 初始化一元二次多项式
  Quadratic InitQuadratic(float aa=0, float bb=0, float cc=0);
  Quadratic Add(Quadratic f1, Quadratic f2);        // 两个多项式相加
  float Value(Quadratic f, float x);                // 计算一元二次多项式的值
  int Root(quadratic f, float& r1, float& r2);      // 求一元二次方程的两个实根
  void print(Quadratic f);                          // 按指定格式输出多项式
}
```

(2) 数据成员定义和指定的操作的实现如下：

数据成员定义：

```
struct Quadratic
{ float a, b, c;
};
```

数据成员初始化：

```
Quadratic InitQuadratic(float aa=0, float bb=0, float cc=0)
{ Quadratic f;
    f.a=aa;
    f.b=bb;
    f.c=cc;
    return f;
}
```

两个二次多项式相加：

```
Quadratic Add(Quadratic f1, Quadratic f2)
{ Quadratic f;
  f.a=f1.a+f2.a;
  f.b=f1.b+f2.b;
  f.c=f1.c+f2.c;
  return f;
}
```

由给定的 x，求二次多项式的值：

```
float Value(Quadratic f, float x)
{ return (f.a*x*x+f.b*x+f.c);
}
```

求一元二次方程的实根：

```
int Root(quadratic f, float& r1, float& r2)
{ if (f.a==0)   return -1;
   float d=f.b*f.b-4*f.a*f.c;
   if(d>=0)
     { r1=(-f.b+sqrt(d))/(2*f.a);
       r2=(-f.b-sqrt(d))/(2*f.a);
       return 1;
     }
    else  return 0;
}
```

按照要求的形式输出二次多项式：

```
void print(Quadratic f)
 { if (f.a)  cout<<f.a<<"x**2";
```

```
    if (f.b)
        { if (f.b>0)  cout<<"+"<<f.b<<"x";
          else  cout<<f.b<<"x";
        }
    if (f.c)
        { if (f.c>0)  cout<<"+"<<f.c;
          else  cout<<f.c;
        }
      cout<<endl;
}
```

习题 1.9 算法的基本特征是什么？算法分析主要针对哪些方面？

[解答] 算法是解决问题方案的准确而完整的描述。算法是为解决某一特定问题而确定的一个指令序列。算法具有以下特性：

（1）有穷性。一个算法必须在执行有穷步之后结束，而且每一步都应该能够在有限时间内完成。

（2）确定性。算法中的每一步含义都必须是确切的、无歧义的。并且在任何情况下算法只有一条唯一的执行路径。

（3）可执行性。算法中描述的运算都应该能够准确的执行。

（4）有输入。一个算法应该有 0 个或多个取自于特定对象的集合的输入。

（5）有输出。一个算法应该有 0 个或多个经算法计算得到输出。

对同一个问题可以设计出不同的算法，各个算法特点不同，性能也会不一样，因而对一个算法需要进行性能的分析。对算法的性能分析包括算法的正确性、可读性、健壮性、执行效率等方面，但通常对算法的分析主要是针对算法的执行效率进行分析，即对算法执行时的时间和空间代价进行分析比较，也就是分析算法的时间复杂度和空间复杂度。

习题 1.10 分治法与减治法的思路有什么相同之处？又有什么不同之处？

[解答] 分治法和减治法的共同之处是，它们都是在“分而治之” 思想的指导下发展起来的，基本思路就是把一个规模较大的问题划分为若干个规模较小的子问题，通过对子问题的求解，得到原问题的解。

分治法和减治法又各自适用于不同的情况，因此它们的求解过程有所不同。用分治法求解的问题，所划分的子问题是互相独立的，且原问题的解需要由各子问题的解合并而成。因此，需要对各子问题分别求解，并合并子问题的解，才能得到原问题的解。可以用减治法求解的问题，虽然也要对原问题进行划分，但因为原问题的解只在其中一个子问题中，或者是只与其中的一个子问题的解之间有着某种对应关系，因此只要对相关的一个子问题进行求解，就可以得到原问题的解。当然它也就不存在合并解的过程。可以说，减治法是一种退化了的减治法。

习题 1.11 简述回溯法的基本思想，采用这种算法的关键是什么？试举出可以用回溯法求解的一个问题。

[解答] 回溯法是一种有组织的系统化搜索问题解的方法，它是对穷举搜索的改进，其采用的是“向前走，碰壁回头”思想。具体地说，首先要对问题进行分析，确定问题的

所有可能解，即确定问题的解空间，然后沿着所确定的路线向前搜索。在搜索过程中，对每一步得到的部分解进行判断，如果该部分解有可能构成一个完整解，说明这一步走得通，则继续向前走，也就是进一步构造问题解；否则，说明“此路不通”，则需要回退，找另一条路线再搜索，也就是回溯，从新的路线上继续构造问题的解。

由回溯法的求解过程可以看出，采用回溯法的关键是确定正确的解空间，即确定解的搜索范围，并确定搜索的路线，只有做到这两步，才可能有效地对问题解进行搜索。

例如，“给定一个正整数集合 $X=\{x_1, x_2, \cdots, x_n\}$和一个正整数 y，求集合 X 的一个子集 Y，使得集合 Y 中的元素之和等于 y。”这个问题就可以采用回溯法来求解。其求解思路是：从 X 集合中依次选取各元素并将其与 Y 中的元素相加，考察相加结果。

具体做法是：

① 初始子集 Y 为空，其元素和等于 0；

② 选取 x_1，将其与子集 Y 的元素和相加，检查结果：

若相加结果大于 y，则放弃当前所选 x_i：

若 X 中还有后续元素，继续选取 x_{i+1} 再试；

否则，回溯：放弃 x_i 之前一个选中的元素，继续向后选取；

若相加结果小于 y，做 $Y=Y+x_i$，继续向后选取 x_{i+1} 再试；

若相加结果等于 y，输出 Y 中的元素，并回溯：

放弃当前所选 x_i，继续选取 x_{i+1} 再试，求其余解。

习题 1.12 简述贪心法和动态规划法思路的异同。

[**解答**] 贪心法和动态规划法都是用于解决多阶段决策的最优化问题。基本的求解思路，都是把一个复杂的问题分解为若干子问题，通过对子问题求解的一系列的决策或选择，最后得到原问题的解。

两者的决策方法不相同。贪心法总是把原问题分解为一系列较为简单的局部最优选择，每一步选择都是在当前状态下做出的最优选择，同时扩展了当前的部分解，直到求得问题的完整解。这个贪心选择过程是以自顶向下的方式进行的，即从原问题出发，每做一步贪心选择都把问题简化为规模更小的子问题，直到对规模最小的子问题做出贪心选择。动态规划法中，分解成的子问题具有两个特点：其一，子问题之间往往不是互相独立的，而是有重叠的部分，这种重叠关系通过动态规划函数表现出来，为了避免对重叠部分的重复计算，以表格形式保存每一步对子问题的求解结果，当需要再次求解已经解决的子问题时，只要做简单的查表操作即可；其二，每个子问题对应决策过程的一个阶段，每步所做的决策往往依赖于相关子问题的解，因此，只有在解决了相关的子问题之后，才能做出决策或选择。正因为此，其求解子问题时，通常以自底向上的方式进行，即从求解最后分解得到的子问题开始，逐层向前，最后求得原问题的解。

习题 1.13 什么是算法的渐近时间复杂度？如何分析一个算法的渐近时间复杂度？

[**解答**] 算法的渐近时间复杂度是对算法的时间效率的度量。也就是对一个算法执行所需要的时间进行分析。一个算法执行所需要的具体时间与所使用的计算机系统的软、硬件性能及问题的规模等因素有关。为了比较算法本身的时间性能，应该采用能够反映算法本身的时间性能的度量。实际上，通常算法运行所需要的时间 T 是问题规模 n

的函数，可记为 $T(n)$。所谓算法的渐近时间复杂度，是当问题规模充分大时，对算法运行时间的增长趋势的度量。因为增长率的上限对算法的比较更具意义，所以经常分析的是算法运行时间的增长率的上限，这就是算法的时间复杂度的大 O 表示，也常简称算法的时间复杂度。

为了分析一个算法的时间复杂度，一般情况下需要考察算法中基本语句的执行次数，找出其与问题规模 n 的函数关系 $f(n)$，从而得到算法的渐近时间复杂度。所谓基本语句是执行次数与算法的执行次数成正比的语句，它是算法中的关键操作。算法的基本语句大多包含在循环和递归结构中；对于单循环结构，循环体中的简单语句就是基本语句，其执行次数的大 O 表示就是该算法段的渐近时间复杂度；对于并列的循环结构，要先分析各个循环结构的渐近时间复杂度，然后利用大 O 表示法的加法规则求出算法的时间复杂度；对于多层嵌套的循环结构，最内层循环中的简单语句就是算法的基本语句，要自外向内逐层分析各层循环的渐近时间复杂度，再利用大 O 表示法的乘法规则来求出算法的渐近时间复杂度；对于递归结构，则可以根据递归过程递推出基本语句的执行次数，进而得到它的大 O 表示。总之，只要分析求出算法中关键操作的执行次数与问题规模的函数关系，也就得到了该次数的大 O 表示，从而也就求出了算法的渐近时间复杂度。

习题 1.14 什么是算法的渐近空间复杂度？如何分析一个算法的渐近空间复杂度？

[**解答**] 算法的渐近空间复杂度是对算法的空间效率的度量，也就是对一个算法执行所需要的存储空间进行分析。一个算法执行时所需要的空间包括几个方面，如存储程序指令所需要的空间，存储输入数据的空间等。与分析算法的时间复杂度类似，为了能够反映一个算法的空间性能，要排除与算法性能无关的存储空间需求，仅考虑算法执行时所需要的辅助存储空间，因为它直接与算法的空间性能有关。一个算法执行时所需要的辅助存储空间量也可以表示为问题规模 n 的函数，其大 O 表示称为算法的渐近时间复杂度，也简称算法的空间复杂度。

根据上述概念，分析算法的渐近空间复杂度就是要考察和分析算法执行时所需要的临时工作单元、动态使用的空间、递归工作栈所占空间等辅助空间的需求量，然后将其表示为问题规模的函数，也就是用大 O 表示法表示它，即可得到算法的渐近空间复杂度。

习题 1.15 给以下程序加上必要的注释，指出其功能，并分析时间复杂度。

(1)

```
int sum(int n)
{ int s=0;
   for(int i=1;i<=n;i++)
   {int p=1;
      for(int j=1;j<=i;j++) p*=j;
      s+=p;
   }
    return s;
}
```

(2)

```
int fac(int n)
{ int p=1,s=0;
   for(int i=1;i<=n;i++)
    { p*=i;
      s+=p;
    }
   return s;
}
```

(3)

```
void writ(int n)
{ if (n!=0)
   { writ(n-1);cout<<n<<endl;}
   return ;
}
```

(4)

```
void Order(float a[], int k, int m)
{ if (k<m)
      { for(int i=k;i<=m;i++)
          if (a[i]<a[k])
           { int temp=a[i];a[i]=a[k];a[k]=temp;}
        k++;
        order(k,m);
      }
 }
```

(5)

```
void insort(int a[], int n)
{ int i,t;
  for(int j=1;j<=n;j++)
   { t=a[j];
     i=j-1;
     while(i>=0 && a[i]>t)
      { a[i+1]=a[i];i--;}
     a[i+1]=t;
  }
}
```

[解答]

(1)

```
int sum(int n)                          // 求n! 的累加和
{ int s=0;                              //  s 存放n! 的累加和
   for (int i=1; i<=n; i++)
    {int p=1;
```