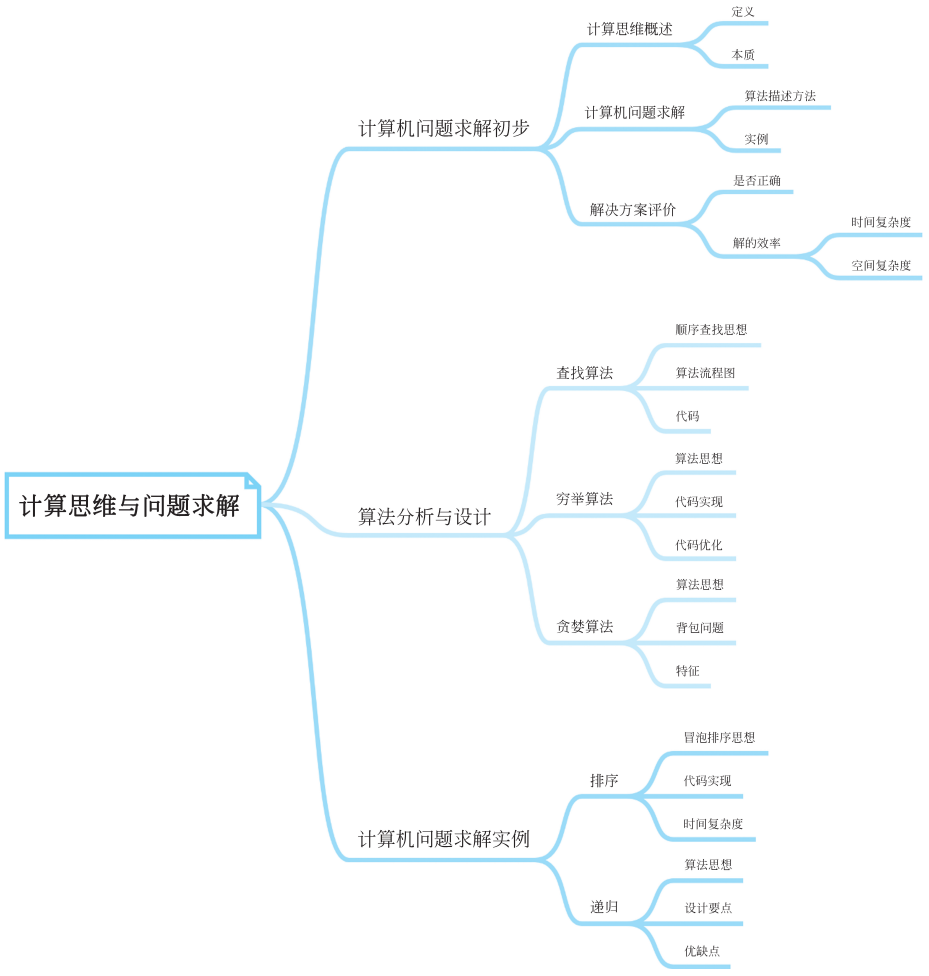
第3章 计算思维与问题求解

计算思维是对问题进行抽象并形成自动化解决方案的思维活动，是当今信息社会每个人必须具备的基本技能。本章围绕计算思维的核心——抽象和自动化，介绍计算机问题求解的步骤和算法描述，探讨查找、穷举、贪婪、排序、递归等典型算法，通过问题求解的部分实例，展现计算思维及其过程。



学习任务单(一)

一、学习指南	
章节名称	第 3 章 计算思维与问题求解 3.1 计算机问题求解初步
学习目标	(1) 能阐述计算思维的本质、数据结构的基本概念。 (2) 能描述计算机问题求解的一般步骤与方法。 (3) 能读懂用自然语言、流程图或伪代码描述的算法,并利用 Python 进行代码实现。 (4) 解释说明算法正确性、时间复杂度、空间复杂度等算法评价标准。
学习内容	(1) 计算思维概述。 (2) 计算机问题求解。 (3) 解决方案评价。
重点与难点	重点:计算机问题求解步骤;算法的概念与描述方法。 难点:算法描述方法。
二、学习任务	
线上学习	中国大学 MOOC 平台“大学计算机基础” 自主观看以下内容的视频:“第一单元 1.2 计算思维概述 1.3 计算的自动化 1.4 计算的抽象 1.5 本单元小结”。
研讨问题	(1) 输入两个正整数,求最大公约数和最小公倍数。 (2) 输入正整数 n,判断 n 是否为素数。 (3) 输出 100 以内的素数,并求和。
三、学习测评	
内容	习题 3(一)

学习任务单(二)

一、学习指南	
章节名称	第 3 章 计算思维与问题求解 3.2 算法分析与设计
学习目标	(1) 能描述穷举法、贪婪法等常用算法设计策略。 (2) 能对查找、素数、背包等典型问题进行算法分析与设计。
学习内容	(1) 查找算法。 (2) 穷举算法。 (3) 贪婪算法。
重点与难点	重点：穷举法、贪婪法的算法分析与设计。 难点：穷举法的算法分析与设计。
二、学习任务	
线上学习	中国大学 MOOC 平台“大学计算机基础”。 自主观看以下内容的视频：“开启 Python 之旅(二)、(三)”。
研讨问题	(1) 以每行 5 个数来输出 1～300(不含)能被 7 或 17 整除的偶数，并求其和。 (2) 有一些数，除以 10 余 7，除以 7 余 4，除以 4 余 1，求满足条件的最小正整数。 (3) 有 1 元、2 元、5 元若干张，要凑齐 20 元，有哪几种方案？哪种方案用得张数最少？
三、学习测评	
内容	习题 3(二)

学习任务单(三)

一、学习指南	
章节名称	第 3 章 计算思维与问题求解 3.3 计算机问题求解实例
学习目标	(1) 能阐述排序算法和递归策略的设计思路。 (2) 能利用 Python 实现冒泡排序和递归问题的求解。
学习内容	(1) 计算机问题求解实例：排序与递归。 (2) 航空飞行训练成绩分析与统计。
重点与难点	重点：冒泡排序、递归策略的思想及应用。 难点：冒泡排序的实现；递归策略的思想及应用。
二、学习任务	
线上学习	中国大学 MOOC 平台“Python 编程基础”(南开大学 王恺 第 3 次课开课)。 自主观看以下内容的视频：“开启 Python 之旅(二)、(三)”。
研讨问题	(1) 输入 n 个数，采用冒泡排序将这 n 个数从小到大输出。 (2) 用递归方法求 $n!$。 (3) 输出斐波那契数列的前 20 个数。斐波那契数列为 1、1、2、3、5、8、13、…此数列从第三项开始，每一项都等于前两项之和，递推公式为 $F(n)=F(n-1)+F(n-2)\quad n\geqslant 3,\quad F(1)=1,\ F(2)=1。$
三、学习测评	
内容	习题 3(三)

3.1 计算机问题求解初步

3.1.1 计算思维概述

1. 计算思维

计算思维是对问题进行抽象并形成自动化解决方案的思维活动。它包括一系列计算机科学的思维方法，如逻辑思维、算法思维、分解、抽象等。

逻辑是研究推理的科学，是一种区分正确和不正确论证的系统，逻辑思维帮助人们理解事物、建立和检查事实。

算法由求解问题的有限个步骤组成。算法中动作步骤的组织有 3 种方式：顺序、选择、循环。算法思维能让人们设计出借助计算机解决问题的步骤。算法通常用算法流程图进行描述。算法描述的是过程性知识，这是利用计算思维设计问题解决方案的基石。因此，需要掌握算法思维来正确地组织解决方案的动作序列。

分解是对问题进行划分，得到一组子问题，这些子问题是易于理解的。通常这种分解过程会一直持续下去，直到每个子问题的解都很简单为止。它有助于人们解决复杂问题。

2006 年 3 月，美国卡内基梅隆大学的周以真教授在美国计算机权威杂志 *Communication of ACM* 上，发表了计算思维(Computational Thinking)的论文。论文指出，计算思维和阅读、写作及算术一样，是 21 世纪每个人的基本技能，而不仅仅属于计算机科学家。

2. 计算思维的本质

计算思维的本质是抽象(Abstraction)与自动化(Automation)(简称两个 A)，即首先将现实世界抽象为数据模型，称为建模，然后设计能自动执行的算法进行模拟。

抽象是从众多的事物中抽取出共同的、本质性的特征，而舍弃其非本质的特征。在讨论抽象时，经常出现的一个词是建模，它是对现实世界事物的描述，这种描述通常会舍弃一些细节。建模的结果是各种模型，是对现实世界事物的各种表示，即抽象后的表现形式。

计算机中数据之间的关系抽象为数据结构。数据结构是指相互之间存在一种或多种特定关系的数据元素的集合。根据数据之间的逻辑关系，数据结构可分为集合结构、线性结构、树结构和图结构。

集合结构的元素之间无逻辑关系。

线性结构是一个有序数据元素的集合。常用的线性结构有线性表、栈、队列等。其中，线性表的数据元素是一对一关系，即除了第一个和最后一个元素外，其他元素都只有一个前驱和一个后继。栈是按先进后出(FILO)的原则组织信息的一种线性结构。队列是按先进先出(FIFO)的原则组织信息的一种线性结构。

非线性结构包含树和图。其中，树结构是具有层次的嵌套结构，该结构中的数据成一对多的关系，如家族谱。图结构是一种复杂的数据结构，该结构内的数据存在多对多的关系，也称网状结构。

计算机科学家、图灵奖得主 N. Wirth(沃斯)提出，程序＝数据结构＋算法。借助计算机进行问题求解，首先分析问题，将现实世界中的对象及对象之间的关系抽象为数据模型，并选择合适的数据结构，保存数据；然后设计合理高效的算法，对数据进行操作，编写出计算机能自动执行的程序，得到问题的解。

3.1.2 计算机问题求解

用计算机求解问题的一般步骤是分析问题、建立模型、设计算法、编程/调试、得到结果。其中，算法设计是问题求解的关键。

1. 算法描述方法

算法的描述方法有多种，包括文字描述、图形描述、伪代码描述等。

流程图是算法的一种图形化表示方式，将一个过程中的指令或流动的流程绘制成图，并使用符号表示其中的每个活动。流程图基本符号如图 3.1 所示，图 3.2 所示流程图的功能是判断成绩是否及格。

流程图基本符号	说明
	程序的开始或结束
	计算步骤(动作/操作)
	输入输出指令
	判断和分支
	连接符
	流程线

图 3.1　流程图基本符号

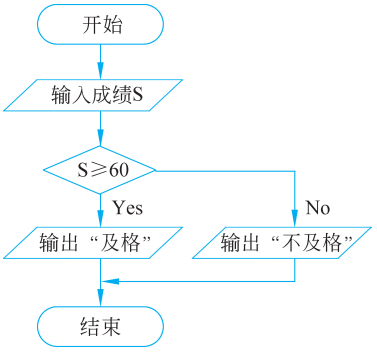


图 3.2　算法流程图举例

2. 实例

1）最大公约数与最小公倍数问题

方法一：概念法。

流程图如图 3.3 所示，代码实现如下：

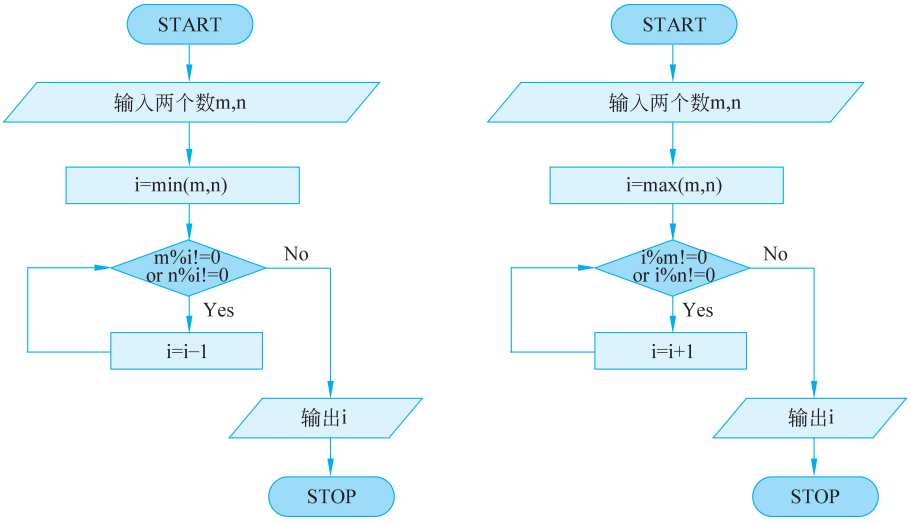


图 3.3　概念法求两数最大公约数和最小公倍数流程图

```
m=int(input('输入数字 1:'))
n=int(input('输入数字 2:'))
i=min(m,n)
while m%i!=0 or n%i!=0:
    i=i-1
print(i,'最大公约数')
i=max(m,n)
while i%m!=0 or i%n!=0:
    i=i+1
print(i,'最小公倍数')
```

说明：

(1) 求最大公约数时，除数的范围可以是 $2\sim i-1$，也可以是 $2\sim i/2$ 或 $2\sim\sqrt{i}$。可以采用递减方式，也可采用递增方式。递增方式参考代码如下：

```
m=int(input('输入数字 1:'))
n=int(input('输入数字 2:'))
i=min(m,n)
for j in range(2,i+1,1):
    if(m%j==0 and n%j==0):
        p=j
print(p,'最大公约数')
```

(2) 求最小公倍数，也可以 m 或 n 的倍数增长，效率更高。参考代码如下：

```
m=int(input('输入数字 1:'))
n=int(input('输入数字 2:'))
i=max(m,n)
mul=1
while (i*mul)%m!=0 or (i*mul)%n!=0:
    mul=mul+1
print(i*mul,'最小公倍数')
```

方法二："辗转相除"法。

原理：两个正整数的最大公约数等于其中较小的数与两数余数的最大公约数。

算法：用两个变量 m 和 n 表示两个正整数，用自然语言描述算法如下。

(1) 如果 m＜n，则交换 m 和 n。

(2) 判断 n 是否等于 0，不等于 0 则下一步，否则 m 即最大公约数，输出 m。

(3) m 除以 n，得到余数 r。将 n 赋值给 m，将 r 赋值给 n，转到步骤(2)继续执行。

流程图如图 3.4 所示，参考代码如下：

```
m=int(input('输入数字 1:'))
n=int(input('输入数字 2:'))
t=m*n
while n!=0:
```

```
    r=m%n
    m=n
    n=r
print('最大公约数:',m)
print('最小公倍数:',t//m)
```

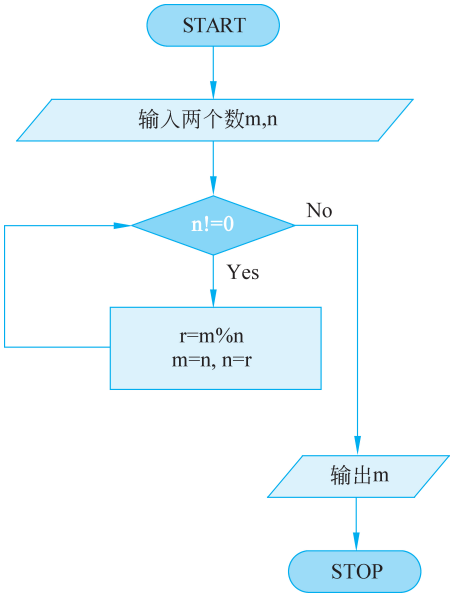


图 3.4　辗转相除法求两数最大公约数与最小公倍数流程图

2）素数问题

实例 1　输入 x 判断它是否为素数。

素数(质数)：除 1 和它本身以外不再有其他因数，如：2、3、5、7、…

思路：遍历 2～x－1 的所有整数 i，若 x 除以 i 的余数为 0，则 x 不是素数；若 x 除以 i 的余数都不为 0，则 x 是素数。

流程图如图 3.5 所示，参考代码如下：

```
x=eval(input('请输入一个数:'))
i=2
while i<x:
    if x%i == 0:
        break
    i=i+1
if i == x:
    print(x, '是素数')
else:
    print(x,'不是素数')
```

说明：

(1) 本例使用的是 while 语句，也可使用 for 语句，但由于 for 语句的运行机制，导致素数 2 的判断异常，可采用单独处理 2 的方式进行纠正。参考代码如下：

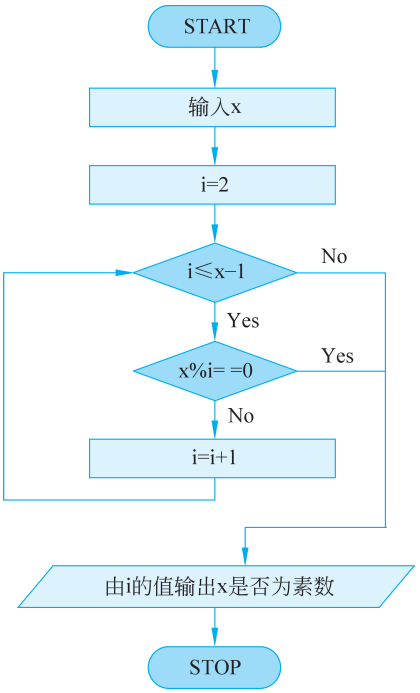


图 3.5　判断素数流程图

```
x= eval(input('请输入一个数:'))
if x==2:
    print("是素数")
else:
    for i in range(2, x, 1):
        if x%i == 0:
            break
    if i == x-1:
        print(x, '是素数')
    else:
        print(x, '不是素数')
```

(2) 本例可引入标记变量 flag,标记 x 是否为素数。如 flag 为 1 代表 x 是素数,flag 为 0 代表 x 不是素数。参考代码如下:

```
x= eval(input('请输入一个数:'))
flag=1
i=2
while i<x:
    if x%i == 0:
        flag=0
        break
    i=i+1
if flag:
    print(x, '是素数')
else:
```

```
        print(x,'不是素数')
```

实例 2　输出 100 以内的所有素数。

思路：对 100 以内的每个数 x，遍历 2～x－1 的所有整数 i，若 x 除以 i 的余数都不为 0，则 x 是素数，输出。

参考代码：

```
n = 100
print(n,'以内素数包括:')
for x in range(2,n,1):
    i=2
    while i<x:
        if(x%i==0):
            break
        i=i+1
    if(i==x):
        print(x)
```

实例 3　求 100 以内所有素数的和。

思路：对 100 以内的每个数 x，遍历 2～x－1 的所有整数 i，若 x 除以 i 的余数都不为 0，则 x 是素数，将 x 累加，输出累加和。

参考代码：

```
n = 100
print(n,'以内素数之和为:')
s = 0
for x in range(2,n,1):
    i=2
    while i<x:
        if(x%i==0):
            break
        i=i+1
    if(i==x):
        s+= x
print(s)
```

3.1.3　解决方案评价

问题求解的最后一步，是确保问题的解是一个“好”解，必须对解进行评价，即解的质量如何。评价涉及很多方面，如正确性、可读性、健壮性等。主要是正确性和解的效率。

1. 解是否正确

通常采用系统化、有计划的测试来评价解的正确性。测试的主要任务就是设法使软件发生故障、暴露软件错误，尽可能多地查找错误。测试阶段的根本目标是尽可能多地发现并排除软件中潜藏的错误。

2. 解的效率如何

解的效率关心的就是程序在占用计算资源方面的表现。通常用主要操作步骤的数目以及所需的空间来度量时间和空间复杂度，即用时间和空间两个指标来度量算法效率。

时间复杂度：为了便于比较同一问题的不同算法，通常从算法中选取一种与求解问题相关的基本操作，以该基本操作的重复执行次数作为算法时间量度的依据。

空间复杂度：算法所处理的数据所需的存储空间（与数据结构密切相关）与算法操作所需的辅助空间（工作单元）之和，通常又以后者为主要考虑对象。

3.2 算法分析与设计

3.2.1 查找算法

问题：某期班有 10 名学员，参加飞行训练考核，现需查找考核成绩低于合格线的学员，以进行预警提醒。输入合格分数线，输出需预警提醒的学员编号（设列表元素下标为学员编号）。

1. 算法分析

（1）构造成绩列表 a＝[a0，a1，…，a9]。

（2）遍历列表，顺序查找低于合格线的学员，输出其下标。

2. 算法流程图

顺序查找算法流程图如图 3.6 所示。

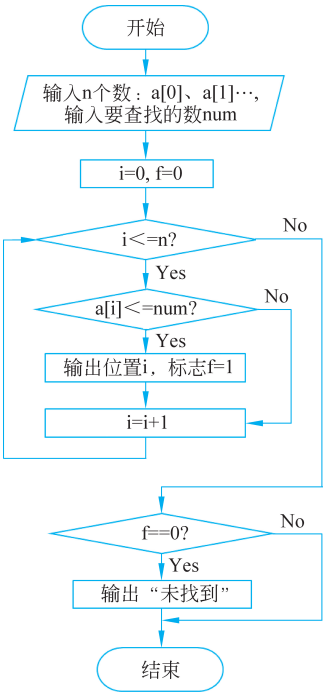


图 3.6 顺序查找算法流程图

3. 代码实现

```
n=int(input("请输入数列元素的个数:"))
a=[ ]
f=0
for i in range(n):
    print ("输入第",i+1,"数:")
    a.append(int(input()))
print ('成绩列表为:',a)
num=int(input("请输入合格分数线:"))
for i in range(n):
    if(a[i]<num):
        print (i,"号学员低于合格线")
        f=1
if f==0:
    print ("没有需要预警提醒的学员")
```

4. 查找算法应用——用凯撒密码为军事情报加密

古罗马凯撒大帝发明来对军事情报进行加解密的算法。它是一种替换加密的技术，明文中的所有字母都在字母表上向后(或向前)按照一个固定数目进行偏移后被替换成密文。例如，当偏移量是 3 时，所有的字母 A 将被替换成 D，B 变成 E，以此类推，小写字母和数字也一样处理，其他字符不做任何改变。

假如有这样一条指令：

```
this is a secret.
```

用凯撒密码加密后就成为

```
wklv lv d vhfuhw.
```

编写程序，输入一个字符串，对字符串中的字母和数字进行凯撒加密，输出加密后的字符串，并对加密后的字符串进行解密。

凯撒加密参考代码：

```
LETTERS="abcdefghijklmnopqrstuvwxyz"
KEYS="defghijklmnopqrstuvwxyzabc"
s=input("请输入明文:")
t=""
for i in range(len(s)):
    if s[i] in LETTERS:
        for j in range(len(LETTERS)):
          if(s[i]==LETTERS[j]):
            t=t+KEYS[j]
    else:
        t=t+s[i]
print('密文为:',t)
```

凯撒解密参考代码：

```
LETTERS="abcdefghijklmnopqrstuvwxyz"
KEYS="defghijklmnopqrstuvwxyzabc"
t=input("请输入密文:")
s=""
for i in range(len(t)):
    if t[i] in KEYS:
        for j in range(len(KEYS)):
          if(t[i]==KEYS[j]):
            s=s+LETTERS[j]
    else:
        s=s+t[i]
print('明文为:',s)
```

5. 顺序查找总结

思想：从头到尾，依次比较，找到要找的。

适用问题：在无序的数组中查找。

优点：思想直观，实现简单。

缺点：效率不高。

6. 二分查找

二分查找又称折半查找，它是一种效率较高的查找方法。

二分查找要求：必须按关键字大小有序排列。

算法思想：首先，将列表中间位置记录的关键字与查找关键字比较，如果二者相等，则查找成功；否则利用中间位置记录将列表分成前后两个子列表，如果中间位置记录的关键字大于查找关键字，则进一步查找前一子列表，否则进一步查找后一子列表。

3.2.2 穷举算法

1. 引入：百钱买百鸡问题

在公元 5 世纪我国数学家张丘建在其《算经》一书中提出“百鸡问题”：鸡翁一，值钱 5；鸡母一，值钱 3；鸡雏三，值钱 1。百钱买百鸡，问鸡翁、母、雏各几何？

2. 问题分析

建立数学模型：设鸡翁 x 只、鸡母 y 只、鸡雏 z 只，x、y、z 取值范围为 0～100。

$$\begin{cases} x+y+z=100 \\ 5x+3y+\dfrac{z}{3}=100 \end{cases}$$

3. 穷举法思想

根据问题列出所有可能情况，然后根据问题中的条件检验哪些情况是需要的。

4. 实现及时间复杂度分析

```
for Cock in range(101):
```

```
    for Hen in range(101):
        for Chick in range(101):
            if Cock+Hen+Chick==100 and 5*Cock+3* Hen + Chick /3==100:
                print ("Cock:",Cock, "Hen:",Hen,"Chick:",Chick)
```

算法时间复杂度：$O(n^3)$。

此穷举法需尝试 100×100×100＝1 000 000 次。

思考：能否优化？

优化 1：百钱只买一种类别的鸡，则鸡翁最多能买 20 只，鸡母最多能买 33 只，鸡雏最多能买 100 只。

```
for Cock in range(21):
    for Hen in range(34):
        for Chick in range(101):
            if Cock+Hen+Chick==100 and 5*Cock+3* Hen + Chick /3==100:
                print ("Cock:",Cock, "Hen:",Hen,"Chick:",Chick)
```

算法时间复杂度：$O(n^3)$。

此穷举法需尝试 20×33×100＝66 000 次。

优化 2：购买鸡翁和鸡母后，鸡雏的数量可利用“百鸡”的条件获得。

```
for Cock in range(21):
    for Hen in range(34):
            if  5*Cock+3* Hen + (100-Cock-Hen) /3==100:
                print ("Cock:",Cock, "Hen:",Hen,"Chick:",100-Cock-Hen )
```

算法时间复杂度：$O(n^2)$。

此穷举法需尝试 20×33＝660 次。

5. 穷举总结

适用问题：可以枚举各种情况。

设计要点：注意方案的优化，减少运算量。

3.2.3 贪婪算法

1. 引入

思考：有 1 元、2 元、5 元纸币若干张，要凑齐 20 元，有几种方法？

```
count=0
for x in range(21):
    for y in range(11):
        for z in range(5):
            if  x+2*y+5* z==20:
                print ("1元:",x, "张,2元:",y, "张,5元:",z, "张")
                count=count+1
print('共',count,'种方法。')
```

运行结果如图 3.7 所示。

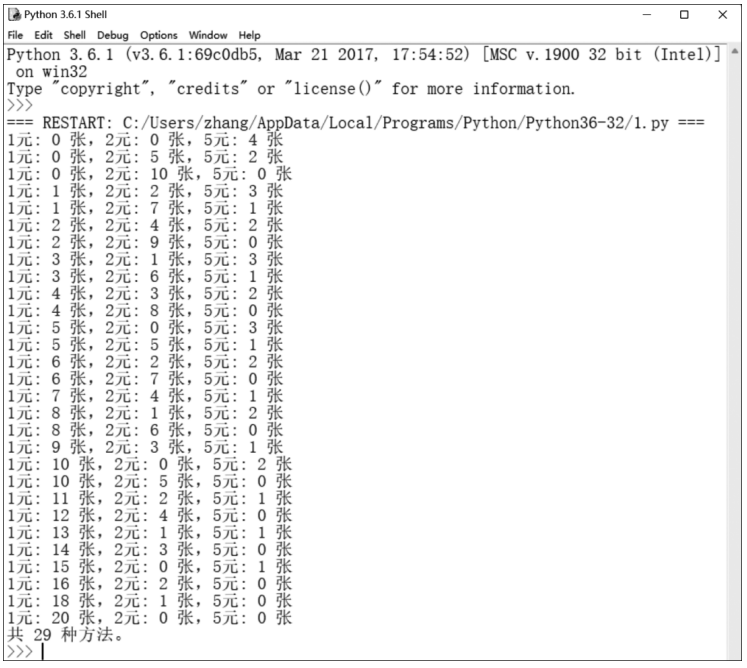


图 3.7　运行结果

进阶：编程给出所用张数最少的方案。

```
a=[1,2,5]
a.sort(reverse=True)
s=eval(input('总钱数:'))
for i in range(len(a)):
    print (a[i],"元:",s//a[i], "张")
    s=s-s//a[i]*a[i]
```

如输入总钱数 20 元，运行结果如图 3.8 所示。

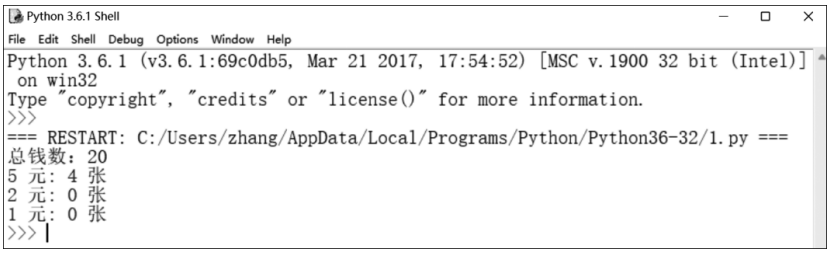


图 3.8　运行结果

2. 贪婪算法思想

贪婪算法在每一步选择中都采取在当前状态下最好或最优(即最有利)的选择，从而希望导致结果是最好或最优。此算法不一定能求出最优解。

3. 背包问题

1）问题描述

有 n 种物品，物品 j 的重量为 w_j，价格为 p_j，背包所能承受的最大重量为 W；限定每种物品只能选择 0 个或 1 个；求解将哪些物品装入背包可使这些物品的重量总和不超过背包重量限制，且价格总和尽可能大。

应用领域：商业、组合数学和密码学等。

2）思路

采用贪婪算法求解背包问题，通过多步过程来完成背包的装入，在每一步过程中利用贪婪准则选择一个物品装入背包。

3）算法文字描述

(1) 将背包清空。

(2) 如果背包中的物品重量已达到背包的重量限制，则转(5)。

(3) 否则(即背包中的物品重量未达到背包的重量限制)，按照贪婪准则从剩下的物品中选择一个加入背包，转(2)。

(4) 如果找不到这样的物品，则转(5)。

(5) 结束。

4）贪婪准则

(1) 价格准则：优先选价格高的。

(2) 重量准则：优先选重量轻的。

(3) 价格重量比准则：优先选价格重量比高的。

考虑 $n=3$ 个物品，这 3 个物品的重量和价格分别为 $w=[50,30,20]$，$p=[40,30,30]$，背包的重量限制为 $W=60$。

(1) 价格准则：$x=[1,0,0]$，总价格为 40。

(2) 重量准则：$x=[0,1,1]$，总价格为 60。

(3) 价格重量比准则：$x=[0,1,1]$，总价格为 60。

最优解：重量准则、价格重量比准则，总价格为 60。

贪婪算法每一步作选择时，都是按照某种标准采取在当前状态下最有利的选择，以期望获得较好的解。贪婪算法效率较高，但并非在任何情况下都能找到问题的最优解。

3.3 计算机问题求解实例

3.3.1 排序

所谓排序，就是使一串记录，按照其中的某个或某些关键字的大小，递增或递减排列起来的操作。排序算法就是如何使得记录按照要求排列的方法。排序算法在很多领域有广泛应用，尤其在大量数据的处理方面。排序算法有很多，比较经典的有冒泡排序、选择排序、快速排序等。

1. 冒泡排序

冒泡排序思想：相邻数两两比较，满足条件，互换位置。

规律：n 个数需要比较 $n-1$ 趟，第 j 趟比较 $n-1-j$ 次。

n 个元素的列表 a 排序过程代码：

```
for j in range(n-1):
    for i in range(n-1-j):
        if (a[i]>a[i+1]):
            t=a[i]
            a[i]=a[i+1]
            a[i+1]=t
```

算法复杂度分析：

最佳情况——数据序列的初始状态为“正序”。$n(n-1)/2$ 次比较，无须交换数据。

最坏情况——数据序列的初始状态为“逆序”。$n(n-1)/2$ 次比较、$n(n-1)/2$ 次交换。

时间复杂度为 $O(n^2)$。

例 3-1 某期班 7 名学员某科目飞行训练成绩列表为 a=[90,75,80,95,65,70,85]，请用冒泡排序方法将成绩排序。参考代码如下：

```
a=[90,75,80,95,65,70,85]
n=len(a)
for  j in range(n-1):
     for i in range(n-1-j):
         if (a[i]>a[i+1]):
             t=a[i]
             a[i]=a[i+1]
             a[i+1]=t
print("排好序的成绩列表为:",a)
```

2. 选择排序

算法思想：从头到尾扫描所有的 n 个元素，从中找出最小或最大的元素并和第一个元素进行交换，然后从除第一个以外的 $n-1$ 个元素中扫描，找出最小或最大的元素并和第一个（$n-1$ 个中）元素进行交换，不断迭代此操作剩下的元素，最终就是一个有序的序列。

实现原理：以[54,226,93,17,77,31,44,55,20]为例。

首先我们认为第一个元素为最小元素，然后从列表中找到最小元素，并与第一个元素替换：

[17,226,93,54,77,31,44,55,20]

然后接着假设第二个为最小，再从列表中寻找最小元素进行替换：

[17,20,93,54,77,31,44,55,226]

以此类推：

[17,20,31,54,77,93,44,55,226]
[17,20,31,44,77,93,54,55,226]
[17,20,31,44,54,93,77,55,226]
[17,20,31,44,54,55,77,93,226]
[17,20,31,44,54,55,77,93,226]
参考代码如下：

```
def selectionSort(a):
    for i in range(len(a) - 1):
        #记录最小数的索引
        minIndex = i
        for j in range(i + 1, len(a)):
            #print("内层索引为 j",j)
            if a[j] < a[minIndex]:
                minIndex = j
        #i 不是最小数时,将 i 和最小数进行交换
        if i != minIndex:
            a[i], a[minIndex] = a[minIndex], a[i]
    return a
a = [3,44,2,7,67,57,21]
a = selectionSort(a)
print(a)
```

选择排序时间复杂度为 $O(n^2)$。

3.3.2 递归

1. 引入

有 5 个学生坐在一起，问第 5 个学生多少岁？他说比第 4 个学生大 2 岁；问第 4 个学生岁数，他说比第 3 个学生大 2 岁；问第 3 个学生，又说比第 2 个学生大 2 岁；问第 2 个学生，说比第 1 个学生大 2 岁；最后问第 1 个学生，他说是 10 岁。请问第 5 个学生多大？

分析：

```
age(5)=age(4)+2
age(4)=age(3)+2
age(3)=age(2)+2
age(2)=age(1)+2
age(1)=10
```

建立关于年龄的函数模型：

$$\text{age}(n)=\begin{cases}10 & (n=1)\\ \text{age}(n-1)+2 & (n>1)\end{cases}$$

参考代码：

```
def age(n):
    if n==1:
        return 10
    else:
        return age(n-1)+2
print("第 5 个人的年龄是:",age(5))
```

在定义函数的同时，又调用了函数本身，这就是递归。

2. 递归

定义：在函数内部直接或间接调用自己的函数称为递归函数，如

$$f(x)=x \cdot f(x-1)$$

主要思想：把问题转化为规模缩小了的同类问题的子问题加以解决。

设计要点：将复杂问题转化为同类子问题，规模极小时，直接给出解，作为“出口”。

优点：思路简洁，清晰。

缺点：递归函数执行时空间开销大。

3. 汉诺塔（Hanoi）问题

在印度，有一个古老的传说：在世界中心贝拿勒斯（在印度北部）的圣庙里，一块黄铜板上插着 3 根宝石针。印度教的主神梵天在创造世界的时候，在其中一根针上从下到上穿好了由大到小的 64 个金盘，这就是所谓的汉诺塔。不论白天黑夜，总有一个僧侣在按照下面的法则移动这些金盘：一次只移动一个金盘，不管在哪根针上，小金盘必须在大金盘上面。当所有的金盘都从梵天穿好的那根针上移到另外一根针上时，看需多长时间。这就是有名的“汉诺塔”问题，是一个典型的递归问题，如图 3.9 所示。

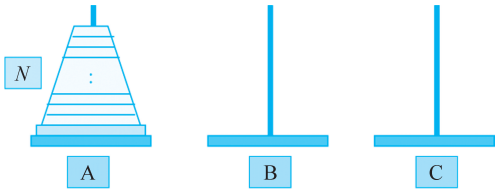


图 3.9　汉诺塔问题

将 N 个金盘从 A 移到 C，借助 B，可划分为以下 3 步：

（1）将 $N-1$ 个金盘从 A 移到 B，借助 C。

（2）将 A 上的金盘移到 C。

（3）将 $N-1$ 个金盘从 B 移到 C，借助 A。

将汉诺塔问题写成递归函数 hanoi，参考代码如下：

```
def hanoi(n, from_, with_, to_):
    if n ==1:
        print("%s———>%s"%(from_,to_))
```

```
    else:
        hanoi(n-1, from_, to_, with_)
        print("%s---->%s"%(from_,to_))
        hanoi(n-1, with_, from_, to_)
```

假设有 3 个金盘，则用以下调用语句实现。

```
hanoi(3,"A","B","C")
```

现对汉诺塔问题进行时间复杂度分析。对于 n 个金盘，共需移动 2^n-1 次，即 $T(n)=2^n-1$。对于 64 个金盘，$T(64)=18446744073709551615$ 次。假设每秒移动一次，也需要 5845 亿年以上。

4. 举例

例 3-2 用递归算法求 $n!$。$n!=\begin{cases}1 & n=1\\ n\cdot(n-1)! & n>1\end{cases}$

```
def f(n):
    if(n==1):
        v=1
    else:
        v=n*f(n-1)
    return v
n=eval(input('请输入自然数 n:'))
print('n!=',f(n))
```

3.4 航空飞行训练成绩分析与统计

本节在 2.7 节航空飞行训练成绩管理系统设计与实现的基础上，继续完善系统功能，实现学员信息的批量录入，增加成绩分析与统计模块。

3.4.1 系统功能

(1) 按需批量录入学员信息，继续录入时输入 Y 或 y，结束录入时输入 N 或 n。运行参考图 3.10。

(2) 实现按学员学号升序排列或按学员训练成绩降序排列。运行参考图 3.11。

(3) 飞行训练成绩采用十分制，规定飞行训练成绩≥9，等级为优秀；7≤飞行训练成绩<9，等级为良好；6≤飞行训练成绩<7，等级为合格；飞行训练成绩<6，等级为不合格。请统计各等级人数。运行参考图 3.12。