HZ BOOKS
华章教育

面向CS2013计算机专业规划教材

数据结构与算法

Python语言描述

裘宗燕
北京大学 著

Data Structures and Algorithms in Python

机械工业出版社
China Machine Press

面向CS2013计算机专业规划教材

数据结构与算法
Python语言描述

裘宗燕
北京大学 著

Data Structures and Algorithms in Python

机械工业出版社
China Machine Press

图书在版编目（CIP）数据

数据结构与算法：Python 语言描述 / 裘宗燕著 . —北京：机械工业出版社，2015.12
（面向 CS2013 计算机专业规划教材）

ISBN 978-7-111-52118-1

I. 数… II. 裘… III. ① 数据结构－高等学校－教材 ② 算法分析－高等学校－教材 ③ 软件工具－程序设计－高等学校－教材 IV. ① TP311.12 ② TP311.56

中国版本图书馆 CIP 数据核字（2015）第 271070 号

Python 是目前国际上流行的用于教授第一门程序设计课程的语言，国内高校也开始使用。本书是结合国内数据结构课程现状，采用 Python 作为工作语言，全新编撰的一本数据结构教程。书中结合抽象数据类型结构的思想，基于 Python 的面向对象机制，阐述各种基本数据结构的性质、问题和实现，讨论一些相关算法的设计、实现和特性。书中还结合研究了一些数据结构的应用案例。

本书要求学习者已有基本 Python 程序设计的知识和经验，可以作为基于 Python 的计算机基础课程中的数据结构课程教材，也可以作为学习 Python 语言基本内容之后的一本面向对象等高级编程技术的进阶读物。

出版发行：机械工业出版社（北京市西城区百万庄大街 22 号 邮政编码：100037）
责任编辑：佘 洁 责任校对：董纪丽
印 刷：三河市宏图印务有限公司 版 次：2016 年 1 月第 1 版第 1 次印刷
开 本：185mm×260mm 1/16 印 张：22
书 号：ISBN 978-7-111-52118-1 定 价：45.00 元

凡购本书，如有缺页、倒页、脱页，由本社发行部调换
客服热线：(010) 88378991 88361066 投稿热线：(010) 88379604
购书热线：(010) 68326294 88379649 68995259 读者信箱：hzjsj@hzbook.com

前　言

本书基于作者在北京大学用 Python 讲授相应课程的工作，用 Python 作为工作语言讨论数据结构和算法的基本问题，其撰写主要有下面几方面考虑：

- 作为以 Python 为第一门计算机课程之后相应的数据结构课程的教材。
- 结合数据结构和算法，讨论 Python 中重要数据类型的实现情况和性质，帮助读者理解 Python 语言程序，理解如何写出高效的 Python 程序。
- 展示 Python 的面向对象技术可以怎样运用。书中构造了一批相互关联的数据结构类，前面定义的类被反复应用在后续章节的数据结构和算法中。

基于这些情况，本书不但可以作为数据结构课程的教材，也可以作为学习 Python 语言编程技术的后续读物（假设读者已经有了 Python 编程的基本知识）。

由于 Python 语言的一些优点，近年来，国外已经有不少大学采用它作为第一门计算机科学技术课程的教学语言，包括许多一流大学。国内院校也可能参考这种趋势，出现这种变化。作者在北京大学数学学院开设了基于 Python 语言的程序设计和数据结构课程，通过亲身实践，发现用 Python 讲授这两门课程也是一种很好的安排。

用 Python 学习数据结构，最大的优点就是可以看到复杂的数据结构可以怎样一步步地从基本的语言机制构造起来。在一个章节里定义的数据结构，经常可以在后续章节的算法和数据结构中直接使用，如果不适用，常常可以通过简单的类派生来调整。还可以非常方便地用在各种习题和练习里，或用于解决实际问题。学生可以看到，书中的（或他们自己写的）代码不是玩具，而是切实有用的软件构件。在基于本书的课程中，很容易安排一些有一定规模的面向实际应用的开发课题，使学生得到更好的实际锻炼。

书中标 * 的节或小节作为选讲内容，或留给学生自己阅读。

本书的成型也得益于作者多年讲授基于 C 语言的数据结构课程的经验，张乃孝老师的《算法和数据结构》是作者一直使用的教材，本书在编写中也参考了该书的一些体例。此外，北京大学数学学院 2013 级的同学在学习中提出了许多很好的问题，参加课程辅导工作的刘海洋、胡婷婷、张可和陈晨也提供了很多帮助，在此表示特别的感谢。

裘宗燕

2015 年 8 月于北京

目　录

第1章 绪 论

作为基于 Python 语言的“数据结构与算法”教程，本章首先讨论一些与数据结构和算法有关的基础问题，还将特别关注 Python 语言的一些相关情况。

1.1 计算机问题求解

使用计算机是为了解决实际问题。计算机具有通用性，其本身的功能很简单，就是能执行程序，按程序的指示完成一系列操作，得到某些结果，或者产生某些效果。要想用计算机处理一个具体问题，就需要有一个解决该问题的程序。经过长期努力，人们已经为各种计算机开发了许多有用的程序。在面对一个需要解决的问题时，如果恰好有一个适用的程序，事情就很方便了：运行这个程序，让它去完成所需工作。

实际中的计算需求无穷无尽，不可能都有现成的程序。如果面对一个问题，但没有适用的程序，可能就需要编写一个。一般而言，人们需要的不是解决一个具体问题的程序，而是解决一类问题的程序。例如，一个文本编辑器不应该只能编辑出一个具体的文本文件，而应该能用于编辑各种文本文件；Python 解释器不是只能执行一个具体的 Python 程序，而是可以执行所有可能的 Python 程序。对于求平方根这样的简单问题，人们希望的也不是专用于求某个数（例如 2）的平方根的函数，而是能求任何数的平方根的函数。求平方根是一个问题，求 2 的平方根是求平方根问题的一个实例。人们开发（设计，编写）一个程序，通常是为了解决一个问题，该程序的每次执行能处理该问题的一个实例。

简言之，用计算机解决问题的过程分为两个阶段：程序开发者针对要解决的问题开发出相应的程序，使用者运行程序处理问题的具体实例，完成具体计算（实际上，是计算机按程序的指示完成计算。为简单起见，人们常说程序完成计算，这样说不会引起误解）。开发程序的工作只要做一次，完成的程序可以多次使用，每次处理一个问题实例。当然，对于复杂的程序，完成后通常还需要修改完善，消除错误，升级功能。但这些是后话，无论如何，用计算机解决问题的第一步是开发出能解决问题的程序。

1.1.1 程序开发过程

程序开发就是根据面对的问题，最终得到一个可以解决问题的程序的工作过程。真正的问题来自实际，是不清晰和不明确的，而程序是对计算机操作过程的精确描述，两者之间有着非常大的距离。因此，一般而言，程序开发工作需要经过一系列工作阶段才能完成。由于人的认识能力的限制，其中还可能出现反复。图 1.1 刻画了这一过程中的各个工作阶段，以及实际程序开发的工作流程。

分析阶段：程序开发的第一步是弄清问题。实际中提出（发现）的问题往往是模糊的，缺乏许多细节，是一种含糊的需求。因此，程序开发的第一步就是深入分析问题，弄清其方方面面的情况和细节，将问题严格化，最终得到一个比较详尽的尽可能严格表述的问题描述。在软件开发领域，这一工作阶段通常被称为*需求分析*。

设计阶段：问题的严格描述仍然是描述性的，而计算机求解是一个操作过程。“一个问

题是什么”与“怎样做才能解决它”并不是一件事，在真正编程之前，需要先有一个能解决这个问题的计算过程模型。这种模型包括两个方面，一方面需要表示计算中处理的数据，另一方面必须有求解问题的计算方法，即通常所说的算法。由于问题可能很复杂，其中牵涉的数据不仅可能很多，数据项之间还可能有错综复杂的关系。为了有效操作，就需要把这些数据组织好。数据结构课程的主要内容就是研究数据的组织技术。如何在良好组织的数据结构上完成计算是算法设计的问题，是本书讨论的另一个重点。

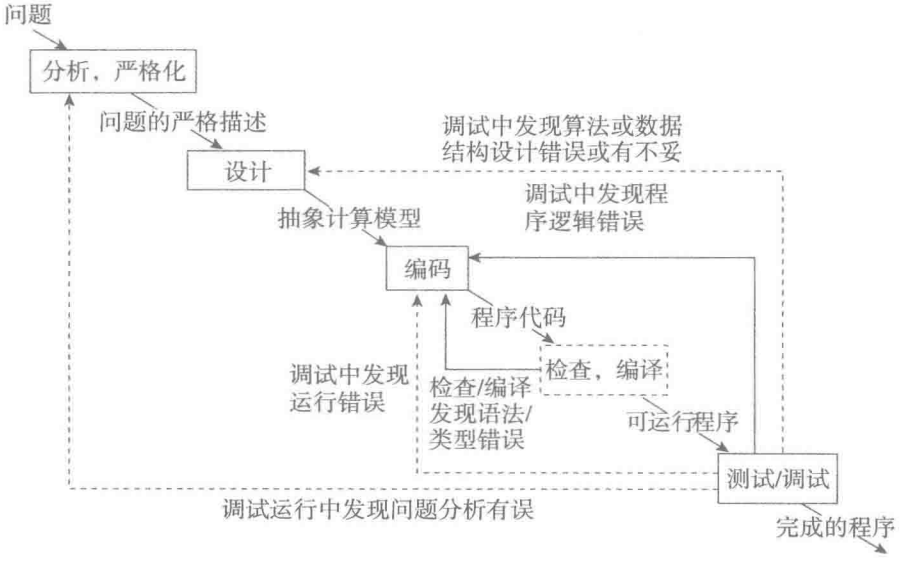

图 1.1 程序开发过程

编码阶段：有了解决问题的抽象计算模型，下一步工作就是用某种适当的编程语言实现这个模型，做出一个可能由计算机执行的实际计算模型，也就是一个程序。针对抽象计算模型的两个方面，编程中需要通过语言的各种数据机制实现抽象模型中设计的数据结构，用语言的命令和控制结构实现解决问题的算法。

检查测试阶段：复杂的程序通常不可能一蹴而就，写出的代码中可能有各种错误，最简单的是语法和类型错误。通过人或计算机（语言系统，编译器）的检查，可以发现这些简单错误。经过反复检查修改，最后得到了一个可以运行的程序。

测试 / 调试阶段：程序可以运行并不代表它就是所需的那个程序，还需要通过尝试性的运行确定其功能是否满足需要，这一工作阶段称为测试和调试。程序运行中可能出现动态运行错误，需要回到前面阶段去修改程序，消除这种错误。也可能发现得到的结果或效果不满足问题需要，这种错误称为*逻辑错误*。逻辑错误可能反映出编程中的失误，也可能是前面的算法设计有问题，甚至是开始的问题分析没做好。无论如何，发现错误之后，需要设法确定造成问题的原因，回到前面某个工作阶段去做适当的修正。然后根据情况在开发的后续步骤中做相应调整。这些工作需要反复进行，直至得到令人满意的程序。

图 1.1 和上面的说明阐释了从问题出发，最终得到可用程序的开发过程。在工作的第二和第三个阶段，算法和数据结构的设计和运用技术都扮演着重要角色：在第二个阶段需要设计抽象的数据结构和算法，第三个阶段考虑它们在具体编程语言中的实现。在设计阶段针对具体问题，建立一个可以用计算机实现的问题求解模型，而编码阶段（加上后续工作）真正实现这个

求解模型，完成一个可以在计算机上运行的程序。

相对而言，设计阶段的工作更困难一些。其工作基础是问题的说明性描述，有关信息并不能简单地映射到问题的操作性求解过程中，需要人的智力参与。为了完成这一工作，需要考虑被求解问题的性质和特点，参考人们用计算机解决问题的已有经验、已经开发的技术和方法。这方面的一些讨论将是本课程的重要内容。

编码阶段的工作相对容易一些。例如要用 Python 作为编程语言来解决问题，就需要把已经建立的抽象数据模型映射到 Python 语言可以表示的结构，把实际问题的抽象求解过程映射到一个用 Python 语言描述的计算过程。这两方面配合就得到了一个用 Python 语言写出的解决问题的程序。

下面将通过实例说明程序开发中的一些情况。

1.1.2　一个简单例子

虽然一个问题的说明性描述与其操作性描述表达的是同一个问题，但它们却非常不同。前者说明了需要解决的问题是什么，针对什么样的问题，期望什么样的解；而后者说明通过怎样的操作过程可以得到所要的解。对于一个给定的问题，用某种严格方式描述一个求解过程，且对该问题的每个实例，该过程都能给出解，这个描述就是解决该问题的一个*算法*。从算法到与之对应的程序，映射关系比较清晰。

现在用一个最简单的问题来说明。假设需要求出任一个非负实数的平方根。这句话是问题的一个非形式描述，工作的第一步就是需要把它严格化。

首先假设实数是一个已经清楚的概念，基于它考虑这个问题。在上面说明中，没有讲清楚的概念是平方根。根据数学中平方与平方根的定义，非负实数 x 的平方根就是满足等式 $y\times y=x$ 的非负实数 y。这是一个严格的数学定义，说明了结果 y 应该满足的条件。但是，它并没有给出一种从任一 x 求出满足这个条件的 y 的方法。

从计算的角度看，上面定义还有一个重大缺陷：对于给定的数值，即使它只包含有穷位小数，其平方根通常也是一个无理数，不能写成数字的有穷表示形式。计算都需要在有穷步内完成，应该是一种有穷过程。因此一般而言，通过计算只能得到实数的平方根的近似值。在考虑求平方根的计算方法（算法）时，这个问题必须考虑，必须把近似误差作为参数事先给定。基于这一看法，上述问题可以修改为：对任意非负实数 x，设法找到一个非负实数 y，使得 $|y\times y-x|<e$，其中 e 是事先给定的允许误差。

这样就有了问题的一个严格描述。但这个描述是说明性的，说明了需要什么样的 y，并没有告诉人们怎么得到这个结果。平方根是一个数学概念，要找到计算平方根的过程性描述（算法），也需要通过数学领域的知识。

人们已经提出了一些求平方根的方法。基本算术课程中介绍过如何求任一正实数的平方根，但在那个方法里需要做试除，不太适合机械进行（可以实现，但比较麻烦）。而求平方根的另一种算法称为牛顿迭代法，描述如下：

0. 对给定正实数 x 和允许误差 e，令变量 y 取任意正实数值，如令 $y=x$；

1. 如果 $y\times y$ 与 x 足够接近，即 $|y\times y-x|<e$，计算结束并把 y 作为结果；

2. 取 $z=(y+x/y)/2$；

3. 将 z 作为 y 的新值，回到步骤 1。

首先，这是一个算法，因为它描述了一个计算过程。只要能做实数的算术运算、求绝对值和比较大小，就可以执行上面描述说明的计算过程。

但是，要确定这个算法能求出实数的平方根，还需要证明两个断言：①对任一正实数 x，如果算法结束，它一定能给出 x 的平方根的近似值；②对任意给定的误差 e，这个算法一定结束（实际上，这件事还与误差 e 和计算机实数表示精度有关）。

第一个断言很清楚，步骤 1 的条件 $|y\times y-x|<e$ 说明了这个断言成立。第二个断言则需要一个数学证明，证明计算过程中 y 值的序列一定收敛，其极限是 x 的平方根。这样，只要迭代的次数足够多，$|y\times y-x|$ 就能任意小，因此对任何允许误差 e，这个循环都能结束。这个问题请读者自己考虑，这里不进一步讨论。

有了上面算法，写出相应 Python 程序已经不困难了。很容易定义一个完成平方根计算的 Python 函数，实现上述算法。下面是一个定义：

```
def sqrt(x):
    y = 1.0
    while abs(y * y - x) > 1e-6:
        y = (y + x/y)/2
    return y
```

其中变量 y 的初值为 1.0，允许误差为 10^{-6}。通过用各种数值测试，可以看到这个函数确实能完成所需要的工作。

从这个简单实例可以看到从问题的描述出发最终得到一个可用程序的工作过程。由于求平方根的问题比较简单，特别是其中涉及的数据只是几个简单实数，数据组织的工作非常简单。下一节的实例将更好展现数据组织的有关情况。

还有一个情况值得注意。在上述例子中，最不清晰的一步就是从平方根的定义到求平方根的算法。算法设计是一种创造性工作，依赖于对问题的认识和相关领域的知识，没有放之四海而皆准的路径可循。计算机科学领域有一个研究方向是*算法的设计与分析*，计算机教育中有相应课程，其中讨论算法设计和研究的许多经验，总结算法设计中一些规律性的线索和思路。但经验也只是经验，在设计新算法时可以参考，但不能保证有效。算法分析则是分析算法的性质，将在 1.3 节进一步介绍。

1.2 问题求解：交叉路口的红绿灯安排

本节展示一个具体问题的计算机求解过程，以此说明在这种过程中可能出现的一些情况，需要考虑的一些问题，以及一些可能的处理方法。

交叉路口是现代城市路网中最重要的组成部分，繁忙的交叉路口需要用红绿灯指挥车辆通行与等待，正确的红绿灯调度安排对于保证交通安全、道路运行秩序和效率都非常重要。交叉路口的情况多种多样，常见形式有三岔路口、十字路口，也有较为少见的更复杂形式的路口。进一步说，有些道路是单行线，在中国的交叉路口还有人车分流和专门的人行 / 自行车红绿灯等许多情况，这些都进一步增加了路口交通控制的复杂性。要开发一个程序生成交通路口的红绿灯安排和切换，需要考虑许多复杂情况。

现在计划考虑的问题很简单，只考虑红绿灯（实际上是相应允许行驶路线）如何安排，可以保证相应的允许行驶路线互不交错，从而保证路口的交通安全。

为把问题看得更清楚，先考虑一个具体实例，希望从中发现解决问题的线索。作为实例的

交叉路口如图 1.2 所示[⊖]：这是一个五条路的交叉口，其中两条路是单行线（图中用箭头标出），其余是正常的双向行驶道路。实际上，这个图形本身已经是原问题实例的一个抽象，与行驶方向无关的因素都已被抽象去除，例如道路的方位、不同道路交叉的角度、各条道路的实际宽度和通常的车流量等。

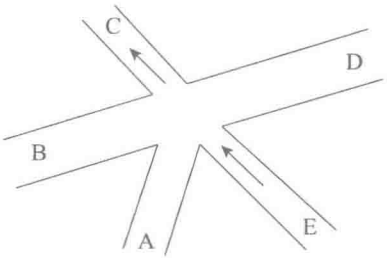

图 1.2　一个交叉路口实例的模型

根据图形中表示的情况不难看到，从路口各驶入方向来的车辆，都可能向多个方向驶出，各种行驶方向的轨迹相互交错，形成很复杂的局面。按不同方向行驶的车辆可能相互冲突，存在着很实际的安全问题，因此，红绿灯的设计必须慎重！

现在的基本想法是对行驶方向分组，使得：

- 同属一组的各个方向行驶的车辆，均可以同时行驶，不出现相互交错的行驶路线，因此保证了安全和路线畅通。
- 所做的分组应该尽可能大些，以提高路口的效率（经济问题）。

第一条是基本的安全问题，丝毫不能妥协。而第二条只是经济问题，这个要求具有相对性。不难看出，允许同时行驶的方向越少，就越能保证安全。例如，每次只放行一个行驶方向，肯定能保证安全，但道路通行效率太低，因此完全不可取。另外还可以看到，这不是一个一目了然的问题，需要深入分析，才能找到较好的统一解决方法。

1.2.1　问题分析和严格化

图形的表达方式一目了然，特别有助于人们把握问题的全局。但是，如图 1.2 这样的图形并不适合表达问题细节和分析结果。如果把所有允许行驶方向画在图中，看到的将是来来去去、相互交错的许多有向线段，不容易看清其相互关系。

为了理清情况，应该先罗列出路口的所有可能行驶方向，例如从道路 A 驶向道路 B，从道路 A 驶向道路 C。为简便易读，用 AB、AC 表示这两个行驶方向，其他方向都采用类似的表示方式。不难列出路口总共有 13 个可行方向：AB, AC, AD, BA, BC, BD, DA, DB, DC, EA, EB, EC, ED。

采用这种抽象表示，问题的实质看得更清楚了。有了（给定了）一集不同的行驶方向，需要从中确定一些可以同时开绿灯的方向组。也就是说，需要为所有可能方向做出一种安全分组，保证位于每组里的行驶方向相互不冲突，可以同时放行。

这里的“冲突”又是一个需要进一步明确的概念。显然，两个行驶路线交叉是最明显的冲突情况[⊖]。如果对这样两个方向同时开绿灯，按绿灯行驶的车辆就有在路口撞车的危险，这是不能允许的。因此，这样的两个方向不能放入同一个分组。

为了弄清安全分组，需要设想一种表示冲突的方式，更清晰地描述冲突的情况。如果把所有行驶方向画在一张纸上，在相互冲突的方向之间画一条连线，就做出了一个冲突图。图 1.3 就是上面实例的冲突图，其中的 13 个小矩形表示所有的可能行驶方向，两个矩形之间有连线表示它们代表的行驶路线相互冲突。注意，在这个图形中，各个矩形的位置、大小等都

⊖ 本例参考张乃孝编著的《数据结构和算法》(由高等教育出版社出版)。

⊖ 注意，“冲突”是对可能出现危险情况的认识，其定义也应该根据实际情况考虑。例如，在允许 BD 方向的情况下，是否同时允许 AD 和 ED，就可能有不同考虑。对冲突的不同定义将得到不同的解。

不代表任何信息，只有矩形中的符号和矩形之间的连线有意义。这样的图形构成了一种数据结构，也称为图，图中元素称为顶点，连线称为边或者弧。相互之间有边的顶点称为邻接顶点。第7章将会详细讨论这种结构。

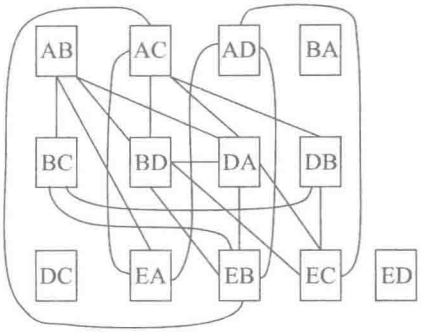

图 1.3 行驶线路冲突图

有了冲突图，寻找安全分组的问题就可以换一种说法：为冲突图中的顶点确定一种分组，保证属于同一组的所有顶点互不邻接。显然，可能的分组方法很多。如前所述，将每个顶点作为一个组一定满足要求。但从原问题看，这里期待一种比较少的分组或者最少的分组。回到原问题，就是希望在一个周期中的红绿灯转换最少。

1.2.2 图的顶点分组和算法

经过一步步抽象和严格化，要解决的问题从交叉路口的红绿灯安排变成了一类抽象的图数据结构上的顶点分组。假设有了这样一个图结构，现在需要考虑一个算法，其计算的结果给出一个满足需要的分组。对这个问题，安全性是第一位的基本要求，必须满足；而分组最少是进一步的附加要求，是一种追求。

以非相邻为条件的最佳顶点分组问题，实际上对应于有名的图最佳着色问题：把顶点看作区域，把相邻看作两个区域有边界，把不同分组看作给相邻顶点以不同颜色着色。著名的四色问题表明，对于以任一方式分割为区域的平面图，只需要用4种不同颜色着色，就能保证相邻区域都具有不同的颜色。但请注意，从交叉路口情况构造出的冲突图可能不是平面图[⊖]，因此完全可能需要更多的颜色。

人们对图着色的算法做过一些研究，目前的情况是：已经找到的最佳着色算法（得到最佳分组），基本上相当于枚举出所有可能的着色方案，从中选出使用颜色最少的方案。例如，一种直截了当的方法是首先基于图中顶点枚举出所有可能分组，从中筛选出满足基本要求的分组（各组中的顶点互不相邻），再从中选出分组数最小的分组。由于计算中需要考虑各种可能组合，这种组合数显然是顶点个数的指数函数，这个算法的代价太高了。能得到最佳分组的其他算法可能更复杂，但在效率上没有本质性提高。

下面考虑一种简单算法，它被称为是一种*贪婪算法*，或称*贪心法*。贪心法也是一种典型的算法设计思路（或称算法设计模式），其中的基本想法就是根据当时掌握的信息，尽可能地向得到解的方向前进，直到不能继续再换一个方向。这样做通常不能找到最优解，但能找到“可接受的”解，即给出一种较好的分组。

算法的梗概（伪代码）如下：

```
输入：图G                          # 记录图中的顶点连接关系
集合verts保存G中所有顶点           # 建立初始状态
设置集合groups为空集               # 记录得到的分组，元素是顶点集合
while存在未着色顶点：              # 每次迭代用贪心法找一个新分组
    选一种新颜色
    在未着色顶点中给尽量多的无互连边的点着色（构建一个分组）
    记录新着色的顶点组
```

⊖ 这里所说的平面图（图形），指在保持邻接关系不变的前提下调整顶点位置，可以将原图变换为另一个图，所得到的图中顶点间的边互不交叉。

```
# 算法结束时集合groups里记录着一种分组方式
# 算法细节还需要进一步考虑
```

现在考虑用这个算法处理图 1.3 中的冲突图，假定操作按照前面列出的 13 个方向的顺序进行处理。操作中的情况如下：

1. 选第 1 种颜色构造第 1 个分组：顺序选出相互不冲突的 AB、AC、AD，以及与任何方向都不冲突的 BA、DC 和 ED，做成一组；
2. 选出 BC、BD 和与它们不冲突的 EA 作为第二组；
3. 选出 DA 和 DB 作为第三组；
4. 剩下的 EA 和 EB 作为第四组。

由于上面算法中这几个分组内互不冲突，而且每个行驶方向属于一个分组，因此是满足问题要求的一个解。得到的分组如下：

{AB, AC, AD, BA, DC, ED}, {BC, BD, EA}, {DA, DB}, {EA, EB}

上面算法还有重要的细节缺失：一种新颜色的着色处理。现在考虑这个问题。

假设图 G 保存需着色图中顶点的邻接信息，集合 verts 是图中所有尚未着色的顶点的集合。显然，算法开始时 verts 应该是 G 中所有顶点的集合。用另一个变量 new_group 记录正在构造的用当前新颜色着色的顶点（一个集合），在上面算法的每次迭代中重新构造，每次开始做分组时将这个集合重新设置为空集。

在上面安排的基础上，找出 verts 中可用新颜色着色的顶点集的算法是：

```
new_group = 空集
for v in verts:
    if v与new_group中所有顶点之间都没有边:
        从verts中去掉v
        把v加入new_group

# 循环结束时new_group是可以用一种新颜色着色的顶点集合
# 用这段代替前面程序框架中主循环体里的一部分
```

这样就有了一个完整的算法。

检查上面算法，可以看到算法中假设了一些集合操作和一些图操作。集合操作包括判断一个集合是否为空集，构造一个空集，从一个集合里删除元素，向一个集合里加入元素，顺序获得集合里的各个元素（上面算法中 for 循环做这件事）。图操作包括获取图中所有顶点、判断两个顶点是否相邻。

上面讨论中实际上也介绍了两种最基本的算法设计方法：

- 枚举和选择（选优）：根据问题，设法枚举出所有可能的情况，首先筛选出问题的解（为此需要判断枚举生成的结果是否为问题的解），而后从得到的解中找出最优解（这一步需要能评价解的优劣）。
- 贪心法：根据当时已知的局部信息，完成尽可能多的工作。这样做通常可以得到正确的解，但可能并非最优。对于一个复杂的问题，全面考虑的工作代价可能太高，为得到实际可用的算法，常常需要在最优方面做出妥协。

1.2.3　算法的精化和 Python 描述

前面给出了一个解决图着色的抽象算法，它与实际程序还有很大距离。要进一步考虑如何实现，还有很多需要处理的细节，例如：

- 如何表示颜色？例如，可以考虑用顺序的整数。
- 如何记录得到的分组？可以考虑用集合表示分组，把构造好的分组（集合）加入 groups 作为元素（也就是说，groups 是集合的集合）。
- 如何表示图结构？

实际上，是否把“颜色”记入 groups 并不重要，可以记录或者不记录。下面的考虑是记录颜色，将“颜色”和新分组做成二元组。

现在考虑如何把上述算法映射到一个 Python 程序中，填充其中的许多细节。前面考虑的集合操作大都可以直接用 Python 的集合操作：

- 判断一个集合 `vs` 是空集，对应于直接判断 `not vs`。
- 设置一个集合为空，对应于赋值语句 `vs = set()`。
- 从集合中去掉一个元素的对应操作是 `vs.remove(v)`。
- 向集合里增加一个元素的对应操作是 `vs.add(v)`。

Python 的集合数据类型不支持元素遍历，但上述算法中需要遍历集合元素，还要在遍历中修改集合。处理这个问题的方法是在每次需要遍历时从当时的 `verts` 生成一个表，而后对表做遍历（并不直接对集合遍历）。

算法中需要的图操作依赖于图的表示，需要考虑如何在 Python 中实现图数据结构。图是一种复杂数据结构，应该支持一些操作。图的可能实现方法很多，第 7 章将详细讨论这方面问题。这里假定两个与图结构有关的操作（依赖于图的表示）：

- 函数 `vertices(G)` 得到 `G` 中所有顶点的集合。
- 谓词 `not_adjacent_with_set(v,group,G)` 检查顶点 `v` 与顶点集 `group` 中各顶点在图 `G` 中是否有边连接。

假设有了图结构及其操作，程序实现就不难了。下面是一个程序（算法）：

```
def coloring(G):                    # 做图G的着色
    color = 0
    groups = set()
    verts = vertices(G)             # 取得G的所有顶点，依赖于图的表示
    while verts:                    # 如果集合不空
        new_group = set()
        for v in list(verts):
            if not_adjacent_with_set(v, newgroup, G):
                new_group.add(v)
                verts.remove(v)
        groups.add((color, new_group))
        color += 1
    return groups
```

这个算法实际上已经是一个 Python 函数，能对任何一个图算出顶点分组。其中欠缺的细节就是图的表示，以及函数里涉及的两个图操作。

1.2.4 讨论

完成了工作并不是结束。任何时候完成了一个算法或者程序，都应该回过头去仔细检查做出的结果，考虑其中有没有问题。做出了一个图着色程序，传递给它一个冲突图，它将返回一个分组的集合。现在应该回过头进一步认真考虑工作中遇到的各种情况和问题，包括一些前面没有关注的情况。

解唯一吗?

首先，可以看到，对于给定的交叉路口实例，可行的分组可能不唯一。除了前面几次提到的每个方向独立成组的平凡解之外，完全可能找到一些与前面给出的解的分组数相同的解。对前面问题实例，下面是另一种满足要求的分组：

{AB, EB, EC}, {AC, AD, BC}, {BA, BD, DB, ED}, {DA, DC, EA}

读者不难验证这一分组确实是该问题实例的解。

算法给出的解是确定的，依赖于算法中选择顶点的具体策略，以及对图中顶点的遍历顺序，即 `list(verts)` 给出的顶点序列中的顺序。

求解算法和原问题的关系

回顾前面从问题出发最终做出一个 Python 程序的工作过程：

1. 有关工作开始于交叉路口的抽象图示，首先枚举出所有允许通行方向；
2. 根据通行方向和有关不同方向冲突的定义，画出冲突图；
3. 把通行方向的分组问题归结为冲突图中不相邻顶点的划分问题，用求出不相邻顶点的分组作为交叉路口中可以同时通行的方向分组。

问题是，这样得到的结果满足原交叉路口问题实例的需要吗?

仔细分析可以看到，上面算法中采用的定义不统一：算法给出的结果是行驶方向的一种划分（各分组互不相交，每个顶点只属于一个分组）；而工作开始考虑安全分组时，只要求同属一组的顶点（行驶方向）是互不冲突的。也就是说，原问题允许一个行驶方向属于不同分组的情况（现实情况也是这样，可能某个行驶方向在多种情况下都是绿灯。典型的例子如无害的右转弯）。出现分组不相交的情况，原因是在构建新分组时一旦选择了某个顶点，就将其从未分组顶点集中删除，这样就产生了划分的效果。

要回到求解之前的考虑，并不一定要推翻得到的算法，也可能在原算法上调整。例如，对于图 1.2 的交叉路口实例，前面算法给出：

{AB, AC, AD, BA, DC, ED}, {BC, BD, EA}, {DA, DB}, {EA, EB}

将分组尽可能扩充，加入与已有成员不冲突的方向，得到：

{AB, AC, AD, BA, DC, ED}, {BC, BD, EA, BA, DC, ED},
{DA, DB, BA, DC, ED, AD}, {EA, EB, BA, DC, ED, EA}

根据前面的定义，这样得到的各分组仍然是安全的，请读者检查。如何修改前面算法，使之能给出这样分组的问题留给读者考虑。

另一个问题前面已有所讨论，就是冲突概念的定义问题。前面采用行驶方向交叉作为不安全情况的定义，这只是一种合理选择。另外的情况是否看作冲突，可能存在不同考虑。如前面提到的直行与旁边道路的右转问题（例如，在允许 BD 方向的情况下，是否允许 AD 和 ED）。对同一个路口，如果冲突的定义不同，做出的冲突图就会不同。实际定义可能需要反映交管部门对具体路口实际情况的考量，需要根据具体情况确定。

还有许多实际问题在前面算法中都没有考虑。例如，在用于控制交叉路口的红绿灯时，得到的行驶方向分组应该按怎样的顺序循环更替？这里能不能有一些调度的原则，能不能通过算法得出结果？另一些问题更实际，可能更难通过计算机处理。例如各个分组绿灯持续的时间，这里牵涉到公平性、实际需要等。可能还有其他问题。

从以上讨论中可以看到，前面工作中从问题出发逐步抽象，得到的算法处理的问题与原问题已经有了很大的距离。该算法的输入是经过人工分析和加工而得到的冲突图，做出冲突图需