算法设计与分析

Dexter C □ Kozen
康奈尔大学

十二月 □ □ □ □

给我妻子 Fran
和我的儿子
Alexander、GeoGerey、和 Timothy

前言

这些是我在康奈尔大学教授的一门为期一个学期的研究生课程CS这些这些算法设计与分析的讲义这门课程有两个目的，一是为计算机科学研究生准备博士资格考试的核心算法材料，二是为理论学生介绍算法设计与分析的一些高级主题。因此，课程内容是核心和高级主题的混合。

起初，我打算这些笔记是作为教材的补充而不是替代起初是在这三年里，它们逐渐形成了自己的生命。除了笔记外，我还严重依赖教材。

A V Aho J E Hopcroft 和 J D Ullman 计算机算法的设计与分析 AddisonAdesley

M R Garey 和 D S Johnson 计算机和难解性问题 NP理论指南 完备性 W H Freeman

R E Tarjan 数据结构和网络算法 SIAM区域 应用数学系列

并且仍然推荐它们作为优秀的参考资料

该课程包括 节讲座 这些讲座的笔记是使用写作人准备的 在每节讲座开始时 我会指定一个写作人为整个班级做笔记并准备原始的LaTeX源代码 然后我会进行修改和分发 除了 讲座 我还包括了 作业集和几个杂项作业练习 都有完整的解答 分发的笔记基本上就是现在的样子 没有进行重大的重新组织有很多有趣的主题 既有经典的也有当前的 我本来想涉及的 但由于时间不足而无法涉及 很多这样的主题比如计算几何和因子分解算法 可以填满整个学期 实际上 最困难的任务之一就是决定如何最好地利用有限的 讲座时间

我要感谢所有帮助准备这些笔记并保持我诚实的学生，马克・阿加德、玛丽・安・布兰奇、卡尔・弗里德里希・博林格、托马斯・布雷索德、苏雷什・查里、索福克利斯・埃弗雷米迪斯、罗宁・费尔德曼、特德

Fischer Richard HuHu Michael Kalantar Steve Kautz Dani Lischinski Pe ter Bro Miltersen Marc Parmet David Pearson Dan Proskauer Uday Rao Mike Reiter Gene Ressler Alex Russell Laura Sabel Aravind Srinivasan Sridhar Sundaram Ida Szafranska Filippo Tampieri 和 Sam Weber 我特别感谢我的助教 Mark Novick fall Alessandro Panconesi fall 和 Kjartan StefStnsson fall，他们帮助我校对解答集的准备，并偶尔讲课。我还要感谢我的同事 L aszl o Babai Gianfranco Bilardi Michael Luby Keith Marzullo Erik Meineche Schmidt Bernd Sturmfels Ev a Tardos SteveVavasis Sue Whitesides 和 Rich Zippel 提供的宝贵意见和兴趣的练习。最后，我要衷心感谢我的同事 Vijay VaziraniVa他在秋季学期教授了这门课程，对我帮助很大。

我将非常感激读者提出的任何建议或批评

康奈尔大学
伊萨卡 纽约

德克斯特・科森
十二月

目录

Ⅰ　讲座

讲座 □ 算法及其复杂性

这是一门关于算法设计与分析的课程，面向计算机科学专业的研究生。它的目的是多方面的，一方面，我们希望涵盖一些相当高级的主题，以提供

为了那些可能希望专攻这个领域的人的利益，我们将提供一些当前研究的概览。另一方面，我们也希望介绍一些核心结果和技术，对那些计划专攻其他领域的人无疑会有用。

我们将假设学生熟悉在设计和分析算法的高级本科课程中通常教授的经典材料。特别是，我们将假设熟悉

顺序机器模型，包括图灵机和随机访问机器（RAMs）

离散数学结构，包括图、树和有向无环图（dags），以及它们的常见表示方法（邻接表和矩阵）
基本数据结构，包括列表、栈、队列、数组、平衡树

渐近分析的基本原理，包括 O□□□ □O□□□ □和□ □□□ □表示法，以及解决递归问题的技巧
基本的编程技术，如递归，分治，动态规划

基本的排序和搜索算法
这些概念在早期章节中介绍

熟悉初等代数，数论和离散概率论将有助于理解。特别是，我们将偶尔使用以下概念：线性独立性，基，行列式，特征值，多项式，素数，模，欧几里得算法，最大公约数，群，环，域，随机变量，期望，条件概率，条件期望。一些优秀的经典参考资料是。主要重点将放在渐近最坏情况复杂性上。这衡量了问题的最坏情况时间或空间复杂性随输入规模增长的方式。我们还将花一些时间讨论概率算法和分析。

渐近复杂度

设 f和 g为函数 N N 其中 N 表示自然数
f g 形式上

f是 O g 如果

$$\exists c \exists N \forall n f(n) \le c \cdot g(n)$$

这个符号 表示 几乎所有 或者 除了有限多个之外都成立
直观地说 f在渐近意义下不比 g快
多一个常数倍

f是 o g 如果

$$\forall c \exists N \forall n f(n) \le \frac{1}{c} \cdot g(n)$$

这是一个更强的说法 直观地说 f 增长比 g 慢
比 g 的任意小正常数倍都要慢 例如
n 是 o $\log_n$

f是 g 如果 g是 O f 换句话说 f是 g 如果

$$\exists c \exists N \forall n f(n) \ge \frac{1}{c} \cdot g(n)$$

f是 g 如果 f是 both O g 和 g

有一个基本规则

始终使用 O和 o作为上界，使用 作为下界，永远不要使用 O作为下界

关于 的定义存在一些争议。一些作者，如 更喜欢上述给出的定义。其他一些作者，如 更喜欢 f是 g 如果 g不是 of 换句话说 f是 g 如果

c N f c g

表示法 意味着存在无限多个 后者较弱且可能更容易建立 但前者给出了更精确的结果 我们不会在这里深入讨论 但只是评论说没有定义排除算法在某些输入上所花费的时间少于规定的界限 例如 断言 归并排序的运行时间是 log 表示存在一个 c使得除了有限多个 以外的所有输入序列长度为的归并排序至少需要

$\frac{1}{c} n \log n$ 比较

没有任何限制使得归并排序在其他长度为
的输入上花费更少的时间

涉及 O和 的语句的确切解释取决于对计算模型的假设 输入的呈现方式 输入大小的确定方式 以及计算的单步操作是什么 在实践中在实者通常不费心去写这些内容 例如例如归并排序的运行时间是 O表示存在一个固定的常数 c使得对于任意从

对于一个完全有序的集合对于多需要 cn log n次比较才能产生一个排序好的数组。在运行时间中，除了个别元素之间的比较次数之外，不计算任何其他操作。同样地，输入大小中也不计算元素的数量，只计算每个元素的大小次比论这意味着什么次比

在阅读该领域的论文时，了解这些未明确说明的假设并明确和形式化它们是很重要的。当自己做出这样的陈述时，始终要记住底层的假设。虽然许多作者不会费心，但在撰写任何论文时，明确说明计算模型的任何假设是一个好习惯。

什么假设是合理的问题往往是审美的问题。通过阅读文献，您将熟悉标准模型和假设什么此之外什么必须依靠自己的良心。

计算模型

我们的主要计算模型将是单位我们本的随机访问机器（RAM）其他模型，如均匀电路和PRAM，在需要时会被引入。RAM模型允许随机访问和使用

数组数组及任意大整数上的单位数组本算术和位向量操作数组见数组

对于图算法来说，算术运算通常是不必要的。图的两种主要表示方式是邻接矩阵和邻接表，前者需要随机访问和对于n数组存储，后者只需要线性存储和无随机访问。对于图来说，线性意味着 O n m ，其中n是图的顶点数，m是边数。最纯粹的图算法是那些使用邻接表表示并且只操作指针的算法。为了表达这样的算法，可以构建一个非常弱的计算模型，其中包含与纯LISP的 car cdr cons eq和il等价的原始操作。参见也等价等价

一粒盐

没有数学模型能完全准确地反映现实，数学模型是抽象的，因此它们必然存在缺陷。例如，众所周知，可以通过在大整数中编码非常复杂的计
算并在多项式时间内解决棘手的问题来滥用单位成本RAM的能力。然而，这违反了良好品味的不成文规定。一种可能的预防措施是使用对数成本模型，但是当按预期使用时，单位成本模型更准确地反映了中等大小数据的实验观察，因为乘法确实需要一单位的时间，此外，还使数学分析变得简单得多。

一些理论家认为渐近最优结果是一种圣杯，并以无情的狂热追求它们（不一定包括在内）。这通常导致人为和晦涩的解决方案，从渐近复杂度的角度来看可能更优，但其常数非常大或其实现非常繁琐，以至于技术的改进永远无法使它们变得可行。这样的结果有什么价值呢？有时它们会引发新的数据结构或分析技术，这些技术在一系列问题上都很有用，但更多时候，它们只是纯粹的数学兴趣。一些从业者将这种活动视为渐近复杂度本身的谴责，并拒绝承认渐近性在实际软件工程中有任何有趣的内容。

在并行计算理论中，争论最激烈的地方莫过于这个观点。有人认为，许多常用的计算模型，如均匀电路和PRAM，是如此不准确，以至于理论结果毫无用处。我们将在稍后讨论并行机器模型时再回到这个争议。

两种极端的态度都是不幸的和反生产的。到现在为止，渐近复杂度在我们的计算机科学意识中占据了一个不可动摇的位置，并且可能对我们的指导更多。

在算法设计和分析方面，改进技术比任何其他数学抽象更重要。另一方面，人们应该意识到它的局限性，并意识到渐近最优解不一定是最好的解决方案

在算法的设计和分析中，一个好的经验法则，就像在生活中一样，是运用常识，发挥良好的品味，并始终倾听自己的良心。

斯特拉森矩阵乘法算法

在算法的渐近快速设计中，分治法可能是最重要的技术之一。让我们来看看斯特拉森的经典矩阵乘法算法及其一些衍生算法。这些例子也将说明渐近分析有时可能会走向极端的程度

通常的矩阵乘法方法需要 次乘法和 次加法来相乘两个 矩阵 或者一般情况下 O n 算术运算来相乘两个 n n矩阵 然而然而法的次数可以减少 斯特拉森 发表了一种用于相乘的算法

矩阵只使用乘法和 加法

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} e & f \\ g & h \end{bmatrix} = \begin{bmatrix} s & s & s & s & s & s \\ s & s & s & s & s & s \end{bmatrix}$$

哪里

$$s = (b - d)(g + h)$$
$$s = (a - d)(e + h)$$
$$s = (a - c)(e + f)$$
$$s = h(a + b)$$
$$s = a(f - h)$$
$$s = d(g - e)$$
$$s = e(c + d)$$

为了简单起见，假设n是一个是一幂次方是一不是你最后一次听到这个是一一对n n矩阵上递归地应用 算法
通过将它们分解成大小为四的方形子矩阵来破解 $\frac{n}{}$ $\frac{n}{}$

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix} \begin{bmatrix} E & F \\ G & H \end{bmatrix} = \begin{bmatrix} S & S & S & S & S & S \\ S & S & S & S & S & S \end{bmatrix}$$

哪里

$$S = (B - D)(G + H)$$

S # A D E H
S # A C E F
S # H A B
S # A F H
S # D G E
S # E C D

一切都与 情况相同，只是现在我们在操作矩阵而不是标量情况因此我们需要稍微小心 矩阵而不是标量 我们需要稍微小心，因为矩阵乘法不是可交换的 最终最终个递归算法在乘法两个矩阵时执行多少标量操作 我们得到递归式

$$T(n) = 7T\left(\frac{n}{2}\right) + dn^2$$

有解

$$T(n) = \sum_{i=0}^{\log n} 7^i d\left(\frac{n}{2^i}\right)^2 = O(n^2)$$
$$= O\left(n^2 \sum_{i=0}^{\log n} \left(\frac{7}{4}\right)^i\right)$$
$$= O(n^{\log 7})$$

这是一个显著的渐近改进，超过了朴素算法但是我们能做得更好吗" 一般来说，一个使用c乘法来乘法两个 d d矩阵的算法，作为这样一个递归算法的基础，将产生一个O nlog d

这已经是一个显著的渐近改进，超过了朴素算法但是我们能做得更好吗" 一般来说，一个使用 c乘法来乘法两个 d d矩阵的算法，作为这样一个递归算法的基础，将产生一个 $O(n^{\log_d c})$

算法要超越 Strassen 的算法 我们必须有 c d 对于一个 矩阵 我们需要 c log # 但是目前已知的最好的算法使用了 次乘法

在 Victor Pan 展示了如何使用 次乘法来相乘 矩阵 这给出了一个大约为 $O(n^{2.81})$ 的算法 迄今为止已知的渐近最优算法 通过完全不同的方法实现 是 $O(n^{2.376})$

每个算法都必须是 n 因为它必须查看矩阵的所有条目 目前没有更好的下界被知道

讲座 □ 拓扑排序和最小生成树

渐近分析中的一个重要主题是，通常可以通过保留关于结构的额外信息来获得更好的渐近性能。更新这些额外信息可能会减慢每个单独步骤的速度单独种额外的代价有时被称为开销。然而单独常情况下，一点点开销就可以在算法的渐近复杂度上产生巨大的改进

为了说明为了我们来看一下拓扑排序。设G# □V□E□是一个有向无环图G的边集E引起了一个偏序关系 □一个自反自反对称自反递的二元关系自反V上，我们用E 表示

如果存在一条长度为如果更长的有向路径从u到v，则称 E 关系为 E 的自反传递闭包

的定义为 E，其中 uE v如果存在一条长度为如果更长的有向路径从 u到 v。关系 E 被称为 E 的自反传递闭包

命题 □ 每个偏序都可以扩展为一个全序 □一个偏序
其中每一对元素都是可比较的其中

证明□ 如果 R是一个偏序但不是一个全序 □ 那么存在 u □ v 使得既不是 uRv也不是 vRu 通过设置来扩展 R

R # R f□x□ y□ j xRu 和 vRy g

新的 R是扩展旧的 R的一个偏序□ 并且现在 uRv
重复直到没有更多的不可比较的对

对于一个有向无环图 G # V E 与相关的偏序 E 来说
如果存在一个全序 extends E 则可以说
如果uEv则u v
这样的全序被称为有向无环图的拓扑排序G 一个朴素的O n
算法可以从上述命题的证明中得到
这里有一个更快的算法这里然仍然不是最优的

算法 拓扑排序 II
从任意顶点开始，沿着边反向前进，直到找到一个没有入边的顶点。这样的顶点最终会被遇到，因为没有循环且有向无环图是有限的 将 u设为下一个顶点的总序 删除 u 及其所有相邻的边，并进行下一步

使用邻接表表示法，该算法的运行时间为每次迭代O n步，共进行n次迭代，即O n

这里的瓶颈是步骤A。通过一些小的修改，我们可以在常数时间内执行此步骤。假设图的邻接表表示与每个顶点关联两个单独的列表，一个用于入边，一个用于出边。如果表示形式尚未处于此形式，则可以在线性时间内将其转换为此形式。算法将维护一个没有入边的顶点队列。这将减少查找没有入边的顶点的成本，使其在常数时间内完成，但稍微增加了维护队列的开销

算法 拓扑排序 III
通过遍历图并插入每个 v 其入边列表为空来初始化队列 选择一个顶点 u o 队列并将其作为 u总序中的下一个顶点 删除 u及其所有出边 u v 对于每个这样的 v 如果其入边列表变为空 将 v放入队列 转到步骤

步骤需要时间 O n 步骤 需要常数时间 因此在所有迭代中的时间为 O n 步骤 在所有迭代中需要时间 O m 因为每条边最多只能被删除一次 总体时间为 O m n稍后我们将看到一种不同的方法，涉及深度优先搜索

□ □最小生成树

设 G # □V□ E□ 是一个连通的无向图

定义 □ G中的森林是一个子图 F # □V□E □没有循环的图注意 F和 G具有相同的顶点集合 G中的生成树是一个森林， G中的生成树是一个森林，具有恰好一个连通分量 w E □ N □边被分配了自然数的权重边被一个最小的权重生成树 MST是一个生成树 T其边的总权重 □是所有生成树中最小的

引理 □设 F # □VE□是一个无向图 c F的连通分量的数量 m # j Ej 和 n # jV j□那么 F没有循环，即 c m # n□

证明□

□□□通过对 m的归纳证明如果 m # □ 那么有 n个顶点，每个顶点都形成一个连通分量个顶以 c # n如果添加一条边而不形成
如果存在一个循环如果那么它必须连接两个组件。因此 m增加，并且 c减少了减少所以方程 c m # n保持不变

□ □ 假设 F 至少有一个循环。选择一个任意的循环并从中删除一条边。然后 m减少了减少但 c和 n保持不变。重复此过程，直到没有更多的循环。完成后，方程c m # n成立成立前面的段落所述成立是它最初可能不成立

我们使用贪心算法来生成最小权重生成树。这个算法最初由Kruskal提出我们我们

> 算法 □ 最小生成树的贪心算法
>
> 按权重对边进行排序
>
> □ 按照递增的权重顺序对列表中的每条边进行处理，如果它与已经选择的边不形成循环，则将其包含在生成树中，否则将其丢弃

只要 n就可以停止算法 自引理 □，我们知道我们有一棵生成树，因此边已经保留□

步骤需要时间 O□m log m□ # O□m log n□ 使用任何一种通用排序方法都可以完成使用在某些情况下可以更快地完成使用如，如果权重是小整数，则可以使用桶排序稍后，我们将给出一个几乎线性时间的步骤实现 □

□ 但现在我们将接受 O□n log n□ 我们将考虑将边 e包含在生成树中，即将两个不相交的顶点集合的并集作为边 e的两个端点的连通分量中的顶点

正在构建中，我们将每个连通分量表示为一个链表，每个链表元素指向下一个元素，并具有指向链表头部的反向指针最初没有边，因此我们有n个链表，每个链表包含一个顶点 当遇到新的边 □ u□v□ 时，我们检查它是否会形成一个环，即检查 u和v是否在同一个连通分量中

列表元素指向下一个元素，并具有指向列表头部的反向指针最初没有边，因此我们有 n个链表，每个链表包含一个顶点 当遇到新的边 u v 时，我们检查它是否会形成一个环，即检查 u和 v是否在同一个连通分量中

通过比较后向指针来判断 u和 v是否在同一个列表中。如果不是的话，我们将u v添加到生成树中，并将两个连接的组件合并。注意，列表始终是不相交的，因此我们不需要检查重复项。

检查 u和 v是否在同一个连接的组件中需要常数时间。每次合并两个列表可能需要线性时间，因为我们需要遍历一个列表并更改后向指针，并且有n个合并。

合并操作的次数为 O n ，如果我们不小心的话。然而，如果我们维护包含每个组件大小的计数器，并始终将较小的合并到较大的组件中，那么每个顶点的后向指针最多会改变log n次，因为每次其组件的大小至少加倍。如果我们将后向指针的更改归因于顶点本身，那么每个顶点最多有log n次更改，或者总共最多有n log n次更改。因此，所有列表合并的总时间为 O n logn 。

蓝色和红色规则

这里有一个更一般的方法，涵盖了大多数已知的最小生成树问题的算法有关详细信息和参考资料，请参见第有关有关 ，该章证明了贪心算法作为这种更一般方法的特例的正确性在下一堂课上，我们将给出一个更一般的处理方法

设 G # V E 是一个带有边权重 w E N 的无向连通图 考虑以下两个对 G的边进行着色的规则 Tarjan 将其称为蓝色规则和红色规则

蓝色规则 找到一个切割 将 V 划分为两个不相交的集合 X和 V X，使得没有蓝色边穿过切割 在X和V 之间选择一条未着色的最小权重边并将其着色为蓝色 红色规则 找到一个循环 在 G中起始和结束于同一顶点顶点其中不包含红色边 在该循环上选择一条未着色的最大权重边并将其着色为红色

贪婪算法只是特例的重复应用蓝色规则的一种我们下次会展示

定理 从所有边未着色开始从所果蓝色规则和红色规则以任意顺序应用从所到两者都不适用从所么最终的蓝色边集合形成最小生成树

讲座 □ 马特罗伊德和独立性

在我们证明最小生成树的蓝色规则和红色规则的正确性之前在我我们先讨论一个抽象的组合结构，称为马特罗伊德我们将展示最小生成树问题是在马特罗伊德中找到最小权重最大独立集的特例然后我们将把蓝色规则和红色规则推广到任意马特罗伊德，并在这个更一般的情况下证明它们的正确性我们将展示每个马特罗伊德都有一个对偶马特罗伊德我们且马特罗伊德的蓝色规则和红色规则分别是其对偶的红色规则和蓝色规则我们此我们旦我们证明了蓝色规则的正确性我们们就免费得到了红色规则

我们还将证明，如果贪心算法总是产生最小我们重的最大独立集，则结构是一个拟阵

定义 □一个拟阵是一个二元组 □S□ I□ 其中 S是一个有限集合， I 是 S的子集族，使得

□i□ 如果 J □ I且 I □ J□，则 I □ I□

□ii□ 如果 I□ J □ I 且 jIj □ jJj□，则存在一个 x □ J I使得
我 fxg □ 我

I的元素被称为独立集合，而不在 I
中的S的子集被称为依赖集合

这个定义应该以一种普遍的方式捕捉到独立性的概念。以下是一些例子

让 V 是一个向量空间 让 S是 V的一个有限子集 让 I 是 线性无关子集的族族 S 这个例子证明了术语这个立

让 A是一个矩阵在一个域上让 S是 A的行集合的行且让 我 是线性无关子集的族族 S

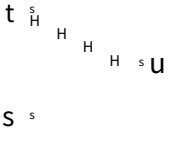
让 G # V E 是一个连通的无向图让 S # E并且让 I 是 G中的森林集合这个例子给出了前一讲的MST问题

让 G # V E 是一个连通的无向图让 S # E并且让 我 是 E的子集集合 使得图 V E E 是连通的

元素 在一个子域 k上代数无关如果没有非平凡多项式 p x x n 与 coecoients in k such that p # 让 S是一个有限集合的元素，让 I 是 S 上的代数无关的子集的集合

定义 一个循环 或电路 是一个对于集合包含关系来说最小的集合 i e 最小的，对于集合包含关系来说最小的与集合S相交的集合是一个集合最小的集合

术语电路和余电路在拟阵理论中是标准的术语我们将继续使用循环和割来保持与最小生成树的特殊情况的直观联系术语请注意，图中的割是在这里定义的割的并集。例如，在图中



在MST的意义上，集合 f s u t u g形成了一个割，但在拟阵的意义上，它不是一个割，因为它不是最小的。然而，稍加思考就会发现，这种差异对于蓝色规则来说是无关紧要的。

让集合 S的元素具有权重。我们希望找到一个总权重最小的集合最大独立集。在这种更一般的情况下，蓝色规则和红色规则变为

蓝色规则：找到一个没有蓝色元素的割。选择割中权重最小的未着色元素，并将其着色为蓝色。红色规则：找到一个没有红色元素的循环。选择循环中权重最大的元素，并将其着色为红色。

□ □拟阵对偶性

细心的读者现在可能已经注意到了，有某种对偶性在起作用。蓝色规则和红色规则之间的相似性太过明显，不可能仅仅是巧合。

定义和定义让□S□ I□成为一个拟阵，其对偶拟阵为□S□I □ □
其中

我 # 子集 S与某个极大元素 Ig 不相交

换句话说□ I的极大元素是 S的补集
I的极大元素

上面的例子上面上面对偶的。注意，一个集合在一个拟阵中是独立的，不一定在其对偶中是依赖的。例如，除了平凡情况外， □在两个拟阵中都是独立的

定理定理

□在 □S□ I□中的割是在 □S□I □中的循环□

□在□S□ I□中的蓝色规则是在□S□ I□中的红色规则，只是权重的顺序相反□

□ □ 蓝色和红色规则的正确性

现在我们证明任意拟阵中蓝色和红色规则的正确性
最小生成树特殊情况的证明可以在Tarjan的书中找到 □ □
第 □ □ 章 Lawler □ □ 提出了任意拟阵的蓝色和红色规则
但没有给出正确性证明

定义 □ 设 □S□ I□ 是一个拟阵及其对偶 □S□ I □ 一个可接受的着色是一对不相交的集合 B □ I □蓝色元素□ 和 R □ I □红色元素□ 一个可接受的着色 B□R是全局的如果 B R # S□ 即 B是一个最大独立集合且 R是对偶中的最大独立集合 一个可接受的着色 B

□ R 扩展或是一个可接受着色的扩展 B□ R如果 B □ B和 R □ R

引理 □ □任何可接受的着色都有一个完全可接受的扩展 □

证明□ 让 B□R是一个可接受的着色 让 U 是I 中的一个最大元素
扩展 R□ 并且让 U # S U 那么 U 是一个最大元素与 R 不相交的I的元素 只要 jBj □ jUj□ 选择 U的元素并将它们添加到 B□ 保持独立性 这是可能的 根据拟阵的公理 □ii□让
让B是结果集 因为所有的最大独立集具有相同的
基数基数习 a□ 作业 □ □ 让B是 I中包含
B并且与 R 不相交的一个最大元素 所需的完全扩展是 B□ S B

引理 一条割和一个环不能只有一个交点

证明 让 C是一条割和 D是一个环 假设 C D # fxg 那么 D fxg是独立的而 C fxg是对偶中的独立集 根据引理 这种着色可以扩展为一个完全可接受的着色 但是根据x的颜色不同 C要么全是红色要么D要么全是蓝色这在一个可接受的着色中是不可能的 因为 D是依赖的而 C是依赖的在对偶中

假设 B 是独立的，而 B fxg 是依赖的。那么 B fxg 包含一个最小的依赖子集或循环 C 被称为基本循环 of x and B 循环 C必须包含 x 因为 C fxg包含在B中，并且 因此是独立的

引理 交换引理 让 B R 是一个完全可接受的着色

(i) 让 x R并且让 y位于 x和 B 的基本循环上。如果 x和 y的颜色交换 则得到的着色是可接受的

(ii) 让 y B并且让 x位于 y和 R的基本割上。 y和 R的基本割是 y和 R的基本循环在对偶矩阵中的表示矩阵 如果 x和 y的颜色交换 则得到的着色是可接受的

证明 通过对偶性 我们只需要证明 (i) 设 C为基本循环 of x and B and let y lie on C 如果 y # x 那么没有什么需要证明的 否则 y B 集合 C fyg是独立的 因为 C是最小的 将 C fyg扩展 通过添加 jBj的元素 如引理 的证明中所示 直到达到最大独立集 B 然后 B # B fyg fxg 并且总可接受的着色 B S B 通过交换 x和 y的颜色得到 B R

一个总可接受的着色 B R被称为最优的 如果 B是最小权重的 在所有最大独立集中 等价地 如果 R是最大权重的 在对偶子群中的所有最大独立集中

引理 如果一个可接受的着色在执行蓝色或红色规则之前具有最优的总扩展 那么在之后得到的着色也是如此

证明 我们证明蓝色规则的情况 红色规则由对偶性得出 设 B R是一个可接受的着色，具有最优的总扩展 $B^{b} R^{b}$ 设 A是一个不包含蓝色元素的割集是一且设 x是 A中的一个未着色元素 具有最小权重 如果 $x B^{b}$ 我们完成了 因此假设 $x b$ R 设 C是的基本循环 B^{b} 根据引理 A C必须包含另一个

我们称之为 这个 因为它是唯一的 练习 b 作业 尽管我们不需要知道这一点来支持我们的论证

元素除了 x□ 假设 y 那么 y □ Þ□ 和 y □ B因为 A中没有蓝色元素 根据引理 □ □x和 y的颜色可以 Þ□ Þ可以交换来获得一个完全可接受的着色 Þ□□Þ□ 扩展 B fxg□R 此外□Þ□ 是最小权重的是最为 x的权重不超过 y的权重

我们还需要知道

引理 □ □如果一个可接受的着色不是总的如果么要么蓝色规则要么红色规则适用□

证明证明设 B□R是一个可接受的着色，其中未着色的元素是 x 根据引理引理 B□R 有一个总扩展 Þ□ Þ根据对偶性根据设不失一般性地 x □ Þ 假设 C是 x的基本割集，并且 Þ由于 C中除了 x之外的所有元素都在 Þ□它们在 B中都不是蓝色的因此蓝色规则适用

结合引理结合结合们有

定理 □ □如果我们从一个未着色的加权拟阵开始，并按任意顺序应用蓝色或红色规则，直到两者都不适用如果么得到的着色是一个最优的总可接受着色□

这里真正发生的是，所有最小权重的最大独立集的子集构成了一个子矩阵，□S□I□和蓝色规则给出了实现这个矩阵的公理□ii□的方法□参见杂项练习

□ □ 矩阵和贪婪算法

我们已经证明，如果 □S□I□是一个矩阵，那么贪婪算法会产生一个最小权重的最大独立集这里我们展示了反过来，如果□S□I□不是一个矩阵，那么贪婪算法在某些选择整数权重的情况下会失败，因此矩阵的抽象概念恰好捕捉了贪婪算法何时有效

定理 □ □ □□ □参见也 □ □ □一个满足矩阵公理 □i□的系统□S□I□是一个矩阵 □i□e□ 它满足 □ii□ □当且仅当对于所有权重分配 w S □ N 贪婪算法给出一个最小权重的最大独立集

证明□ 方向 □□□ 已经被证明 对于 □ □ □让 □S□I□满足 □i □ 但不满足 □ii □ 必然存在 A□B 使得 A□B □I 。jAj □ jBj。但对于任意的 x □ B A是 A fxg □I

不失一般性地假设 B是一个最大独立集
如果不是 □ 我们可以向 B中添加元素而保持 B的独立性质□ 对于

任何添加到 B中的元素如果也可以添加到 A中而保持
A的独立性质□ 我们就这样做 这个过程不会改变以下事实
jAj □ jBj且对于任意的 x □ B A是 A fxg □ I
现在我们给权重 w S □ N 让 a # jA B j 和 b # jB A j
然后 a □ b让 h是一个很大的数□ h □ a □ b □实际上 h □ b□就可以了 □

情况 □ 如果 A是一个最大独立集 □ 分配

w□x□ # a 对于 x □ B A
w□x□ # b 对于 x □ A B
w□x□ # 对于 x □ A □ B
w□x□ # h 对于 x □ A B

因此

w□A□ # a□b □ # ab a
w□B□ # b□a □ # ab b

这个权重分配强制贪心算法在实际上是一个较小权重的最大独立集时选择 B

情况 如果 A不是一个最大独立集 □ 分配

w□x□ # 对于 x □ A
w□x□ # b 对于 x □ B A
w□x□ # h 对于 x □ A B

首先选择 A的所有元素 □ 然后选择一个在
A B之外的巨大元素□ 因为 A不是最大的 因此最小权重的
最大独立集 B没有被选择