第1讲和第2讲 矩阵乘法与矩阵求逆
记录员：Jessica Su 日期：2015年9月25日
编辑：Kathy Cooper

这门课程是关于矩阵乘法及其在图算法中的应用。

1 矩阵乘法的先前工作

定义1.1. (矩阵乘法) 设A和B是$n\times n$矩阵（其中的元素有$O(\log n)$位）。
那么乘积C，其中$AB=C$，是一个$n\times n$矩阵，定义为$([i,j]) = \sum n_k{=}_1 A(i,_k)B(_k,j)$。

我们将假设在$O(\log n)$位整数上的操作（加法和乘法）需要$O(1)$的时间，即我们将在一个字-RAM计算模型中工作，字大小为$O(\log n)$。

为了改进矩阵乘法的运行时间，已经做出了很多努力。 简单的算法在$O(\ n\text{^}3)$的时间内进行矩阵乘法。斯特拉森（'69）通过提供一个$O(\ n\text{^}2.81)$的时间复杂度的算法，让所有人都感到惊讶。
从那时起，一系列的改进开始了，直到1986年，科波斯密斯和维诺格拉德实现了$O(n\text{^}2.376)$的时间复杂度。经过24年的停滞，2010年爱丁堡大学的研究生安德鲁・斯托瑟改进了运行时间，将其降低到$O(n\text{^}2.3734)$。 2011年，弗吉尼亚・威廉姆斯获得了$O(n\text{^}2.3729)$的时间复杂度，这是最好的界限，直到2014年勒・加尔获得了$O(n\text{^}2.3728^7)$的时间复杂度。 许多人相信最终的界限将是$n\text{^}2+\ O(1)$，但这尚未被证明。

今天我们将讨论矩阵求逆和矩阵乘法之间的关系。

2 矩阵乘法等同于矩阵求逆。

矩阵求逆很重要，因为它用于解线性方程组。 乘法等同于求逆，从某种意义上说，任何乘法算法都可以用来得到一个类似运行时间的求逆算法，反之亦然。

2.1 乘法可以简化为求逆。

定理2.1. 如果可以在$T(\ n)$时间内求解$n\times\ n$矩阵的逆，那么可以在$O(\ T(3\ n))$时间内相乘$n\times\ n$矩阵。

证明。设 A和 B为矩阵。假设

$$D = \begin{bmatrix} I & A & 0 \\ 0 & I & B \\ 0 & 0 & I \end{bmatrix}$$

其中 I是 n乘n的单位矩阵。 通过直接计算可以验证

$$D^{-1} = \begin{bmatrix} I & -A & AB \\ 0 & I & -B \\ 0 & 0 & I \end{bmatrix}$$

求逆 D的时间复杂度为 $O(T(3n))$，我们可以通过求逆 C来找到 AB。 注意 C始终可逆，因为其行列式为1。 □

2.2 求逆可以简化为乘法

定理2.2. 假设 $T(n)$满足 $T(2n) \geq (2+\varepsilon)T(n)$，其中$\varepsilon > 0$且对于所有的 n成立。 如果可以在 $T(n)$时间内进行n乘n矩阵的乘法，则可以在 $O(T(n))$时间内求逆n乘 n矩阵。

证明思路：首先，我们给出一个算法来求解对称正定矩阵的逆。 然后我们使用这个算法来求解任意可逆矩阵的逆。

第2节的其余部分专门讲解这个证明。

2.2.1 对称正定矩阵

定义2.1. 一个矩阵 A是对称正定的，如果

1. A是对称的，即 $A = A^t$，所以 $A(i,j) = A(j,i)$对所有的i, j成立
2. A是正定的，即对于所有的 $x = 0$，有 $x^t A x > 0$。

2.2.2 对称正定矩阵的性质

命题1. 所有对称正定矩阵都是可逆的。

证明。假设 A不可逆。 那么存在一个非零向量 x使得 $Ax = 0$。 但是$x^t Ax = 0$，所以 A不是对称正定的。 因此我们得出结论，所有对称正定矩阵都是可逆的。

□

命题2. 对称正定矩阵的任何主子矩阵也是对称正定的。
（一个 m行 n列的矩阵 M是一个n行 n列的矩阵 A的主子矩阵，如果 M是通过删除 A的最后m行和列得到的。）

证明。设向量 x有 m个分量。 我们需要证明 $x^t M x > 0$。 考虑向量 y，它是向量 x后面补充了 $n-m$个零。 由于矩阵 A是对称正定的， $y^t A y > 0$。 但是 $y^t A y = x^t M x$，因为除了前 m个分量外，其他都是零。

□

命题3. 对于任意可逆矩阵 A， $A^t A$是对称正定的。

证明。 设向量 x为非零向量。. 我们现在证明 $||Ax||^2 > 0$。
对于任意的 $x = 0$，由于 A是可逆的， Ax不为零。 因此，对于任意的 $x = 0$， $||Ax||^2 > 0$。 所以 $A^t A$是正定的。 此外，它是对称的，因为（$A^t A$）$^t = A^t A$。 □

第4个命题。假设 n是偶数， A是一个 $n \times n$对称正定矩阵。 将 A分成四个方块（每个方块大小为 $n/2 \times n/2$）：

$$A = \begin{bmatrix} M & B^t \\ B & C \end{bmatrix}.$$

然后，舒尔补， $S = C - BM^{-1}B^t$，是对称正定的。

上述命题的证明将在作业中给出。

2.2.3 对称正定矩阵的约化

假设 A是对称正定的，并将其分成方块 M、 B^t、 B和 C。 再次，令 $S = C - BM^{-1}B^t$。 通过直接计算，我们可以验证

$$A^{-1} = \begin{bmatrix} M^{-1} + M^{-1}B^t S^{-1} BM^{-1} & -M^{-1}B^t S^{-1} \\ -S^{-1}BM^{-1} & S^{-1} \end{bmatrix}$$

因此，我们可以递归地计算 A^{-1}，如下所示：（让运行时间为 t（n））

算法1：对称正定矩阵的求逆
递归计算 M^{-1}（这需要 t（$n/2$）时间） 使用矩阵乘法计算 $S = C - BM^{-1}B^t$（这需要 O（T（n））时间） 递归计算 S^{-1}（这需要 t（$n/2$）时间） 计算 A^{-1}的所有元素（这需要 O（T（n））时间）

该过程的总运行时间为

$$t(n) \leq 2t(n/2) + O(T(n)) \leq O(\sum_j 2^j T(n/2^j))$$

$$\leq O(\sum_j (2/(2+\varepsilon))^j T(n)) \leq O(T(n)).$$

2.2.4 任意矩阵的简化

假设将求逆对称正定矩阵简化为矩阵乘法。 然后考虑求逆任意可逆矩阵 A的问题。 根据第3个命题，我们知道 A^tA是对称的正定矩阵，因此我们可以轻松找到 C= $(A^tA)^{-1}$。 然后 $CA^t = A^{-1}A^{-t}A^t = A^{-1}$，因此我们可以通过将 C与A^t相乘来计算 A^{-1}。

3 布尔矩阵乘法（介绍）

记录员：罗比·奥斯特罗
编辑：凯西·库珀

给定两个$n\times n$矩阵A，B，其中A，B的元素取值为{0,1}，我们定义布尔矩阵乘法（BMM）如下：

$$(AB)[i,\, j] = \bigvee_k (A(i,\, k) \wedge B(k,\, j))$$

注意，布尔矩阵乘法可以使用整数矩阵乘法的算法来计算，因此对于$n\times n$矩阵的BMM的时间复杂度为$O(n^{\omega}+O(^{1))}$，其中$\omega < 2.373$（整数矩阵乘法的当前界限）。

大多数理论上快速的矩阵乘法算法在实际中是不可行的。因此，所谓的“组合算法”是可取的。 “组合算法”没有严格的定义，但具有以下特性：

- 不使用减法
- 所有操作都是相对实际的（就像查找表）

注**1.**对于$\varepsilon > 0$，甚至对于BMM，目前没有已知的时间复杂度为$O(n^{\wedge 3}\text{-}\square)$的组合算法！这样的算法被称为“真正的亚立方算法。”

4个俄罗斯人

1970年，Arlazarov、Dinic、Kronrod和Faradzev（他们似乎并不全是俄罗斯人）开发了一种组合算法，用于BMM，时间复杂度为$O(\frac{n^3}{\log(n)})$，称为Four-Russia$_n$s算法。通过对算法进行小的改变，可以将其运行时间降低为$O(\frac{n^3}{\log^2 n})$。2009年，Bansal和Williams提出了一种改进的算法

，其运行时间为$O(\frac{n^3}{\log(^{2.25}n)})$。2014年，Chan提出了一种运行时间为$O(\frac{n^3}{\log^3 n})$ 然后，在2015年，斯坦福大学的研究生Yu提出了一种运行时间为 $O($的算法。$\frac{n^3}{\log^4 n})$。今天我们将介绍Four-Russians算法。

4.1 Four-Russians算法

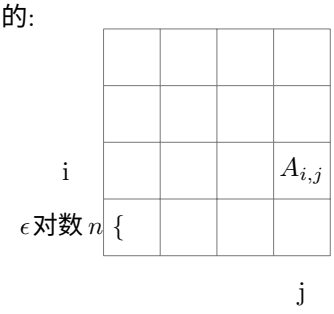
我们从一个假设开始：

- 我们可以存储多项式数量的查找表 T，大小为^c，其中 $c \leq 2+\epsilon$，给定表 T的索引和任意 $O(\log n)$位向量 x，我们可以在常数时间$O(1)$内查找 $T(x)$。

定理4.1. 矩阵乘法的时间复杂度为 $O(\frac{n^3}{\log^2 n})$（在字长为 $O(\log n)$的字RAM上）。

证明。我们给出FourRussians算法。（更多是算法的描述而不是完整的正确性证明。）

设 A和 B为 $n \times n$的布尔矩阵。选择任意的 ϵ，我们可以将 A分割成大小为 $\epsilon \log n \times \epsilon \log n$的块。也就是说， A被划分为块 $A_{i,j}$，其中$i, j \in [$ ~~ϵ~~对数$_n]$。下面我们给出一个简单的例子 A的:



对于每个选择的i，j我们创建一个查找表 T_i，$_j$对应于 A_i，$_j$具有以下规格:

对于每个长度为 ϵ对数 n的位向量 v:

$$T_{i,j}[v] = A_{i,j} \cdot v.$$

也就是说， Ti，$_j$接受长度为 ϵ对数 n的位序列作为键，并存储长度为 ϵ对数 n的位序列。由于存在$^\epsilon$对数 n位的 ϵ位向量，并且 Ai，$_j \cdot v$是 ϵ对数 n位，我们有 $|T$i，$_j| = n^\epsilon \epsilon$对数 n.

这些表的整个计算时间在渐近意义上是

$$(\frac{n}{\log n})_2 n^\epsilon \text{对数}^2\, n = n^{2+\epsilon}$$

因为有 $(\frac{n}{\log n})_2$个选择，对于$i, j,\ n^\varepsilon$向量 v，对于每个 Ai,$_j$和每个 v，计算 $A_{i,j}v$需要 $O(\log^2 n)$ 时间，对于常数 ε。

给定我们在次立方时间内创建的表，我们现在可以在常数时间内查找任何 $A_{ij} \cdot v$。

我们现在考虑矩阵 B。将 B的每一列分割成 $_{\epsilon \log n}$个部分，每个部分包含 $\varepsilon \log _n$个连续的条目。设 $B_j{}^k$为B的第 j个部分的第 k列。每个 $A_{ij}B_j{}^k$可以在常数时间内计算，因为它可以从预处理中创建的表 T_{ij}[B$_j{}^k$] 中访问。

要计算乘积 $Q = AB$，我们可以按以下步骤进行。

从 j= 1 到 $\frac{n}{\epsilon \log n}$: $Q_{ik} = Q_{ik} \vee (A_{ij} \wedge B_j^k)$, 根据定义。通过我们的表 T，我们可以在常数时间内计算出按位“与”（或*），但“或”（或求和）仍然需要 $O(\log n)$ 时间。这给我们提供了一个在时间 $O(n \cdot \frac{n}{\log n}{}^2 \cdot \log n) = O(\frac{n^3}{\log n})$ 时间，四个俄罗斯人的原始结果。

我们如何摆脱由求和创建的额外$\log n$项?

我们可以预先计算所有可能的成对求和！创建一个表 S ，使得 $S(u, v) = \ u \vee v$ 其中 $u, v \in \{0,1\}^{\epsilon \log n}$. 这需要我们 $O(n^{2\epsilon}\epsilon \log n)$ 的时间，因为有 $n^{2\epsilon}$ pairs u, v ，每个组件只需要只需要 $O(\log n)$ 的时间。
这种预计算使我们能够在常数时间内查找任意可能的两两和 $\epsilon \log n$位向量。

因此，每个 $Q_{ik} = Q_{ik} \vee (A_{ij} \wedge B_j{}^k)$ 操作都需要 $O(1)$ 时间，而最终算法的渐近运行时间为

$$n \cdot (n/\varepsilon \log n)^2 = n^3/\log^2 n,$$

其中第一个 n计算了 B的列数 k，剩下的项是i, j对的数量。

因此，我们有一个运行时间为 $O($的组合算法 $\frac{n^3}{\log^2 n}$)。

□

注意：我们能否节省超过对数因子？ 这是一个重要的未解决问题。

5 传递闭包

定义**5.1.** 有向图 $G=(V, E)$的传递闭包（TC）是一个 $n \times n$矩阵
使得 $\forall u, v \in E\ (T(u, v) = \begin{cases} 1 \text{ 如果}v\text{从}u\text{可达} \\ 0 \text{ 否则} \end{cases}$

对于无向图的传递闭包在线性时间内是微不足道的-只需计算连通分量。 相反，我们将证明对于有向图，该问题等价于BMM。

定理**5.1.** 传递闭包等价于BMM

证明。我们分别证明等价性。

命题**5.** 如果TC在 $T(n)$时间内，则BMM在 $O(T(3\ n))$时间内。

证明。考虑下面这样的图，其中有三行N个顶点。 给定两个布尔矩阵 A和 B，我们可以通过在第一行的第 i个顶点和第二行的第 j个顶点之间添加一条边来创建这样的图，当且仅当 $A_{ij}== 1$。 类似地，对于 B，在第二行和第三行之间构建边。 因此，矩阵乘积 AB可以通过计算该图的传递闭包来得到。 通过简单地取 $ij \to jk$的 $\wedge$，它等价于BMM。 由于该图有3N个节点，给定一个 $T(\ n)$算法来计算TC，我们就有了一个 $O(T(3n))$算法来计算BMM。 （当然，创建图需要N^2的时间，但这被 $T(n)$所包含，因为 $T(n) \geq N^2$，至少需要打印 AB的输出。）

□

命题**6.** 如果BMM在 $T(n)$时间内，满足 $T(n/2) \leq T(n)/(2+\epsilon)$，那么$T$C在 $O(T(n))$时间内。

我们注意到命题中的条件是非常自然的。例如，对于任意的 $c > 1$，都有 T (n) = n c成立。

证明。设 A是某个图 G的邻接矩阵。 那么$(A+I)^n$是 G的传递闭包。 由于我们有一个 $T(n)$的BMM算法，我们可以使用对$A+I$进行$\log^n$次连续平方运算，在$O(T(\ n)\log n)$的时间内计算出$(\ A+I)\ n$。 我们需要消除$\log$项以证明等价性。

我们使用以下算法来实现：

1. 计算 G的强连通分量并将其合并得到 G'，这是一个有向无环图。我们可以在线性时间内完成这个步骤。

2. 计算 G'的拓扑排序并根据其重新排序顶点（线性时间）

3. 设 A为 G'的邻接矩阵。 $(A+I)$ 是上三角矩阵。 计算 $C=(A+I)^*$，即 G'的传递闭包。

4. 恢复合并的强连通分量（线性时间）

除了步骤（3）外，所有步骤都可以在线性时间内完成。我们将详细讨论步骤（3）。
考虑将矩阵 $(A+I)$ 分割成四个子矩阵M、C、B 和 0，每个子矩阵的大小为 $n/2 \times n/2$。
$(A+I) = \begin{bmatrix} M & C \\ 0 & B \end{bmatrix}$
我们声称 $(\quad A+I)^* = \begin{bmatrix} M^* & M^*CB^* \\ 0 & B^* \end{bmatrix}$
这背后的推理如下所示。 让 U是拓扑排序中的前$\mathrm{n}/2$个节点，让 V是其余的节点。 那么 M是由 U诱导的子图的邻接矩阵， B是由V诱导的邻接矩阵。 在 U 和 V之间的唯一边是从 U 到 V。 因此， M^*和 U^*表示限制在 $U \times U$和 $V \times V$上的传递闭包。 对于 U中的 u和 V中的 v的TC条目，我们注意到从 u到 v的唯一方法是从 u可能经过另一个 $u' \in U$在U内部的路径，然后取一条边 (u', v') 到一个节点 $v' \in V$，然后在 V内部从 v'到 v的路径。即 $U \times V$的TC矩阵的条目正好是 M^*CB^*。

假设我们的算法在 n节点图上的运行时间为$\mathrm{TC}(n)$。 为了计算传递闭包矩阵，我们递归地计算 M^*和 B^*。 由于这些矩阵的维度为 $n/2$，所以这需要$2TC(n/2)$的时间。

然后我们计算 M^*CB^*，这需要 $O(T(n))$的时间，其中 $T(n)$是计算布尔乘积的时间。

最后，我们有$TC(n) \leq 2TC(\frac{n}{2}) + O(T(n))$。 如果我们假设存在某个$\varepsilon > 0$，使得$T(n/2) \leq T(n)/(2+\epsilon)$，那么递归解为$TC(n) = O(T(n))$。

□

由第1和第2个命题可知，$n \times n$矩阵的布尔矩阵乘法在渐近运行时间上等价于图的传递闭包，其中图有n个节点。 □

6其他注意事项

布尔矩阵乘法还可以解决看起来更简单的问题。 例如：一个有向图是否有三角形?
我们可以通过取邻接矩阵的立方并检查结果矩阵的对角线是否包含1来轻松解决这个问题。

有点令人惊讶的是，已知对于任何$\epsilon > 0$，一个三角形查找的 $O(n^{3-\epsilon})$时间组合算法也意味着一个布尔矩阵乘法的 $O(n^{3-\epsilon/3})$时间组合算法。因此，在组合算法方面，布尔矩阵乘法和三角形查找是“亚立方”等价的。

1 引言

在本讲中，我们将讨论使用矩阵乘法计算全对全最短路径（APSP）的问题。形式上，给定一个图 $G=(V, E)$，目标是计算所有节点对$u, v \in V$的距离 $d(u, v)$。假设 G有 m个节点和m条边，我们可以很容易地设计出运行时间为$\tilde{O}(mn)$的算法 [1]– 例如，在具有非负边权重的图上，我们可以从每个节点运行Dijkstra算法。
当 G是稠密图时，这个时间的数量级是O(n^3)；自然的问题是我们能否做得更好?

对于带权图，已知的最佳APSP算法的运行时间为 $O\left(\frac{n^3}{2^{\Omega(\sqrt{\log n})}}\right)$ 时间，由Ryan Williams（2014）提供。一个重要的开放问题是是否存在这个版本的APSP的“真正的亚立方”算法，即运行时间为 $O(n^{3-\varepsilon})$（其中$\varepsilon > 0$）的算法。

对于无权图，我们知道有算法可以达到这种亚立方性能。特别地，对于无向图，Seidel（1992）提出了一种运行时间为 $O(n^\omega \log n)$的算法，对于有向图，Zwick（2002）提出了一种运行时间为 $O(n^{2.575})$的算法。即使在矩阵乘法指数 ω改进的情况下，运行时间的差异仍然存在。例如，如果 $\omega=2$，那么Seidel的算法将在$\tilde{O}(n^2)$的时间内运行，而Zwick的算法将在$\tilde{O}(n^{2.5})$的时间内运行。在本讲中，我们将讨论无权图的算法。特别地，我们将讨论使用一个命中集的基准算法，该算法适用于有向或无向图，并描述Seidel的无向图算法。

2 命中集算法

给定 G，特别是表示 G的邻接矩阵 A，我们希望计算所有节点对之间的距离。一个自然的第一个算法是通过连续的布尔矩阵乘法计算 A与自身的距离。在 A的第(i, j)个元素 A^k中，如果且仅如果 i到j存在长度为 k的路径，则该元素为1。因此，如果图的直径有限，则对于所有的i, j， $d(i, j) = \min\{k \mid \mathrm{A}^k[i, j] = 1\}$。

事实**2.1.**如果 G的直径为 D，则我们可以在 $O(Dn^\omega)$时间内计算$APSP$。

如果 G的直径较小，则我们找到了一种快速计算APSP的算法，但是 D可能是$O(n)$，在这种情况下，与$O(n^3)$的运行时间相比没有改进。然而，我们可以在 $O(kn^\omega)$的时间内计算所有小于某个 k的短距离，然后使用另一种技术计算更长的距离。
关键思想是使用“命中集合”.

引理**2.1**. *(*命中集合*)* 让 S是一个大小为 $\geq k$的 n^2个集合的集合 $V=[n]$. 高概率下，一个大小为 $O(\frac{n}{k}\log n)$的随机子集 $T \subseteq V$可以命中 S中的所有集合。

有了这个命中集合引理，我们可以使用算法1来计算大于等于 k的距离。

高概率下，该算法将正确计算距离 $\geq k$（因为这些路径至少涉及k个节点，所以高概率下 T命中该路径）。该算法需要从 $O(\frac{n}{k}\log n)$个节点运行Dijkstra算法，因此需要 $\tilde{O}(\frac{n}{k}n^2)$时间。如果我们使用该算法计算长距离，并使用迭代矩阵乘法计算短距离，我们就有了一个全对全最短路径的算法。

定理**2.1.** 设 G是一个有向或无向图，有n个节点，单位权重。G的$APSP$可以在 $\tilde{O}(kn^\omega + \frac{n}{k}n^2)$时间内计算。

[1] 换句话说，多项式对数项已被省略。

算法1： $\textbf{LoNgDist}(V, E)$

随机选择 $T \subseteq V$，使得 $|T| = c \cdot {}^{n}{}_{k} \log n$，其中$c$足够大
对于每个 $t \in T$执行以下操作
  计算 $\textsc{Dijkstra}(t)$
对于每个$u, v \in V$执行以下操作
  计算 $d(u, v) = \min_{t \in T} d(u, t) + d(t, v)$

当我们优化选择 k并将其设置为 $n^{(3-\omega)/2}$时，运行时间为$\tilde{O}\left(n^{\frac{3+\omega}{2}}\right)$ 大约为$\tilde{O}(n^{2.69})$.

3 Seidel算法

虽然这个第一个算法为我们提供了一个快速计算APSP的算法，但问题仍然存在，我们能做得更好吗？ 特别是，我们能避免分别计算短距离和长距离吗？ 我们能利用矩阵乘法计算所有最短路径吗？

事实上，我们可以改进无向图的命中集算法如下。 给定一个具有邻接矩阵 A的图G，考虑其布尔平方 $A^2 = A \cdot A$，其中 $\cdot$表示布尔矩阵乘法。 考虑一个具有邻接矩阵 $A' = A^2 \vee A$的图 G'。

事实**3.1.** $d_{G'}(s, t) = \left\lceil \frac{d(s,t)}{2} \right\rceil$

为了证明这个事实，注意到 A^2中的边表示原图 G中长度为2的路径，而 G'也包含 G的边。 因此，在 G中长度为$2k$的任意路径都可以通过 A^2中的边来构成长度为 k的路径，而在 G中长度为$2k+1$的任意路径则可以通过 A^2中的边构成长度为 k的路径，然后再跟随一条原始边，从而形成长度为 $k+1$的路径。

现在假设我们有一种方法来确定所有节点对之间距离的奇偶性。
那么我们可以使用以下递归策略来计算APSP。

算法2： APSP思路

给定一个邻接矩阵 A
计算 $A^2 \vee A$
递归计算 $d' \leftarrow \text{APSP}(A^2 \vee A)$
对于每个$u, v \in V$执行
  如果 $d'(u, v)$是偶数则
    $d(u, v) = 2d'(u, v)$
  否则
    $d(u, v) = 2d'(u, v) - 1$

请注意，在每次递归调用中，图的直径减少2，并且在$\log n$次迭代后，A将成为全1矩阵，对角线上为0，我们可以检测到。 因此，如果我们可以有效地确定u, v路径的奇偶性，我们应该能够获得一个高效的递归算法来解决APSP问题。考虑任意一对节点$i, j \in V$以及另一个是 j的邻居节点 $k \in N(j)$。 根据三角不等式（在无权、无向图中成立），我们知道 $d(i, j) - 1 \leq d(i, k) \leq d(i, j) + 1$。

命题**1.** 如果 $d(i, j) \equiv d(i, k) \bmod 2$, 那么 $d(i, j) = d(i, k)$.

证明.根据三角不等式， $d(i, j)$ 和 $d(i, k)$ 最多相差1. 因此，如果它们的奇偶性相同，
它们也必须相等. □

命题**2.**设$_{G^2}(i,j)$是 G^2中 i和 j之间的距离，由 $A^2 \vee A$定义。 那么，*(*a*)* 如果 $d(i,j)$是偶数且 $d(i,\ k)$是奇数，则 $d_{G^2}(i,\ k) \geq d_{G^2}(i,\ j)$. *(b)* 如果 $d(i,\ j)$是奇数且 $d(i,\ k)$是偶数， $d_{G^2}(i,\ k) \leq d_{G^2}(i,\ j)$且存在 $k' \in N(j)$使得 $d_{G^2}(i,\ k') < d_{G^2}(i,\ j)$.

（a）的证明

$$d_{G^2}(i,\ j) = \frac{d(i,\ j)}{2}$$
$$d_{G^2}(i,\ k) = \frac{d(i,\ k)+1}{2} \geq \frac{d(i,\ j)}{2}$$

所以 d_{G^2}（i,k） $\geq d_{G^2}$（i,j）. □

（b）的证明

$$d_{G^2}(i,\ j) = \frac{d(i,\ j)+1}{2} \geq \frac{d(i,\ k)}{2} = d_{G^2}(i,\ k)$$

所以一般来说， d_{G^2}（i, k） $\leq d_{G^2}$（i, j），对于从 i到 j的最短路径上的邻居 k'，我们知道 d（i, k'）$< d$（i, j）. □

命题**3.** 如果 $d(i,j)$是偶数，则

$$\sum_{k \in N(j)} d_{G^2}(i,\ k) \geq deg(j) d_{G^2}(i,\ j)$$

如果 $d(i,\ j)$ 是奇数，则

$$\sum_{k \in N(j)} d_{G^2}(i,\ k) < deg(j) d_{G^2}(i,\ j)$$

这第三个命题直接由前两个得出。此外，如果我们可以在$O(n^\omega)$ 时间内计算这里的求和，那么总体运行时间将为 $O(n^\omega \log n)$，正如所期望的那样。 右边的表达式可以在 $O(n^2)$ 时间内计算，这将被 $O(n^\omega)$ 项所包含。

考虑 D，一个 $n \times n$矩阵，其中 $D(i,\ j) = d_{G^2}(i,\ j)$。 我们想要对于每个$i,\ j$对决定是否$\sum$考虑整数矩阵乘积 DA。注意

$$(D \cdot A)[i,\ j] = \sum_{k \in N(j)} d_{G^2}(i,\ k)$$

因此，这个矩阵乘积允许我们计算左边的表达式。

现在我们准备完整地陈述Seidel算法。

断言**4.** *Se*i*de*l算法的运行时间为 $O(n^\omega \log d)$，其中 d指的是图的直径。

证明。运行时间可以表示为以下递归关系。

$$T(n,\ d) \leq T(n,\ \frac{d}{2}) + O(n^\omega)$$
$$\implies T(n,\ d) \leq O(n^\omega \log d)$$

因为 $d \leq n$，所以这个运行时间的上界是 $O(n^\omega \log n)$。 □

请注意，Seidel算法依赖于快速整数矩阵乘法，其运行时间为 $O(n^\omega)$，但目前没有已知的快速组合算法存在。 仍然有一些未解决的问题，它们的答案可以在理论和实践中加速APSP的计算：整数矩阵乘法步骤是否可避免?
是否存在快速的整数组合矩阵乘法算法?

算法3： SEIDEL(A)

```
如果 A除了对角线外都是1，则返回 A
|
否则
    计算布尔乘积 A^2
    D ← SEIDEL(A^2 ∨ A)
    计算整数乘积 D · A
    R ← 0^{n×n}
    对于每个i, j ∈ V执行以下操作
        如果 DA(i, j)< deg(j)D(i, j)，则
            R(i, j) ← 2D(i, j) − 1
        否则
            R(i, j) ← 2D(i, j)
    return R
```

参考文献

[1] Raimund Seidel, *On the All-Pairs-Shortest-Path Problem in Unweighted Undirected Graphs*, Journal of Computer and System Sciences 51, pp. 400-403 (1995).

本周的目标是证明Zwick的以下定理：

定理0.1.在无权有向图上，全对最短路径（$APSP$）可以在 $\widetilde{O}(n^{2+1/(4-\omega)})$ 时间内解决，其中 ω是矩阵乘法指数。

Zwick算法分别计算接近节点（节点u, v满足$(u, v) < P$）和远离节点（节点u, v满足$(u, v) \geq P$）之间的距离。

然而，这两种情况都利用了命中集引理，该引理可以通过标准的概率论论证得到证明。

引理0.1. 假设我们有大小 $\geq L$的多个集合 $S_1, \ldots, S_k \ of \ \{1, \ldots, n\}$。 那么对于足够大的常数 c，随机样本 $S \subseteq \{1, \ldots, n\}$满足 $|S| = c(n/L \lg n)$，有很高的概率能够命中至少一个 S_i中的一个元素。

我们将从解决距离大于等于 P的节点开始。

命题1. 设 $G = (V, E)$为一个无权有向图。固定参数 P。. 在$\widetilde{O}(n^3/P)$时间内，可以计算出每对u, v的距离 $d(u, v)$满足 $d(u, v) \geq P$。

证明. 对于每对至少相距 P的节点u, v在 V中，设 $S_{u,v}$为一组包含某条从 u到 v的最短路径上的节点。. 选择一个大小为$\Theta(n/P \lg n)$的击中集合 S，使得 S几乎肯定击中每个 $S_{u,v}$。

对于每个 S中的元素 s，使用广度优先搜索在 $O(n^2)$时间内计算出所有 $v \in V$的距离 $d(s, v)$。 类似地，对于每个 S中的元素s，使用广度优先搜索（在 G的边反转后）在 $O(n^2)$时间内计算出所有 $v \in V$的距离 $d(v, s)$。

由于 S中的每个 $S_{u,v}$对于每个u, v满足 $d(u, v) \geq P$，因此存在 S中的某个 s使得 $d(u, v) = d(u, s) + d(s, v)$。 因此，我们可以在 $O(n^2|S|)$的时间内计算

$$d'(u, v) = \underset{\text{最小值}}{} d(u, s) + d(s, v),$$

这是所有u, v至少 P距离的正确距离。 □

为了处理长度小于 P的最短路径，我们引入了距离乘积。

定义0.1. 设A, B为 $n \times n$矩阵。 通过以下方式定义距离乘积

$$(A \star B)[i, j] = \underset{k}{\text{最小值}} \{A(i, k) + B(k, j)\}.$$

虽然我们不会证明，但Fisher、Mayer等人的一个定理表明，如果$A \star B$可以在 $T(n)$时间内计算出来，那么加权图中的APSP可以在 $O(T(n))$时间内完成，反之亦然。

事实证明，距离乘积可以相对快速地计算出来。

定理0.2. 如果A, B是 $n \times n$矩阵，其元素在 $\{-M, \ldots, M\}$范围内，则$A \star B$可以在 $\widetilde{O}(M\, n^{\omega})$时间内计算出来。

证明。 定义矩阵 A'和 B'的元素

$$A'[i, j] = (n+1)^{M - A(i,j)}$$
$$B'[i, j] = (n+1)^{M - B(i,j)}$$

计算 A'和 B'的整数乘积，得到 C'的元素

$$C'[i,\, j] = \sum_k (n+1)^{2M-(A(i,k)+B(k,j))}.$$

对于给定的i, j，我们可以计算 $(A \star B)[i, j] = \min_k A(i, k) + B(k, j)$，我们将其称为 L，如下所示。

观察到 $(n+1)^{2M-L} \leq C'[i, j]$，因为 $(n+1)^{2M-L}$是 $C'[i, j]$的一个加数。同时，$C'[i, j] \leq (n+1)^{2M-L} \cdot n$，因为 $(n+1)^{2M-L}$是$C'[i, j]$中最大的加数，而 $C'[i, j]$只有n个加数。因此，我们可以将 L设置为最小的整数，使得 $C'[i, j] \geq (\ n+1)^{2M-L}$。

请注意，我们处理的是在 C'中具有 $O(M \lg n)$位的整数，操作需要 $\widetilde{\mathrm{O}}(\ M)$时间。
记住这一点，很容易看出上述方法在 $\widetilde{\mathrm{O}}(\ M\, n^{\omega})$时间内计算$A \star B$。
□

通过将距离乘积的快速计算与命中集的概念相结合，我们现在可以获得计算紧密节点之间距离的快速算法。

命题**2**。 设 $G{=}\,(V, E)$为一个无权有向图， P为一个固定参数。. 我们可以在时间复杂度为 P以下的节点对(u, v)上计算 $d(u, v)$。

$$\widetilde{O}\left(n^{\omega} P^{3-\omega}\right).$$

证明。 我们将有 $\lceil \lg_{3/2} P \rceil$个阶段。. 设 V_j 为一对顶点(u, v)的集合，使得 $d(u, v)\ \in [(3/2)_j{}^{-1}, (3/2)^j)$，并且设 $V_{\leq}{}^j$为 $\cup^j{}_{i=1} V_i$。在第$_j$阶段，我们将计算每个 $V\, j$中的 $d(u, v)$。更具体地说，我们将计算一个矩阵 D_j，使得对于所有$(x, y)\ \in\ V_{\leq j}$， $D_j[x, y] =\ d(x, y)$；并且对于所有$(x, y) \in V_{\leq j}$，$D_j[x, y] = \inf$。注意 D_1可以很容易地从 G的邻接矩阵中获得。可以通过简单地计算和清理 $D_{j-1} \star D_{j-1}$来获得有效的 D_j。然而，我们无法承担计算 $n \times n$矩阵距离乘积的代价。相反，我们将利用命中集。

对于每个 $(u, v) \in V_j$，考虑从 u到 v的最短路径 $P_{u,v}$。 路径 $P_{u,v}$的中间三分之一是一组连续出现的节点，其中至多有 $(3/2)^{j-1}$个节点在它们之前，至多有 $(3/2)^{j-1}$个节点在它们之后。

在第 j阶段，取一个随机的 $S_j \subseteq V$，使得 $|S_j| \in \Theta(\frac{n}{(3/2)^{j-1}} \lg n)$ 以便对于所有 $(u, v) \in V_j$，高概率下 V会命中路径 $P_{u,v}$的中间节点。注意，因为路径 $P_{u,v}$的中间部分包含节点 $s_{u,v}$，所以我们得到 $(u, s_{u,v})$和 $(\ s_{u,v}, v) \in D_{\leq j-1}$。因此，高概率下，对于所有 $(u, v) \in V_j$，

$$d(u,\, v) = \underset{s \in S_j}{\text{最小值}}\ D_j[u,\, s] + D_j[s,\, v].$$

因此我们可以计算 $D_j(u,\, v)$ 为

$$\min(D_{j-1}[u,\, v], \min_{s \in S_j} D_j[u,\, s] + D_j[s,\, v]).$$

一旦我们已经计算出每个 $\min_{s \in S} D_j[u, s] + D_j[s, v]$，这在^{2}时间内很容易完成，可以通过计算矩阵乘积 $X \star Y$其中 X包含与 S_j的元素对应的 D_{j-1}中的列， Y包含与 S_j的元素对应的 D_{j-1}中的行。 换句话说，通过选择一个击中集 S_j，我们能够使用比 D_{j-1}小得多的矩阵距离乘积来计算 D_j。

将X和Y分解成边长约为$(3/2)^j$的正方形（或更小）块，我们可以利用所有$(3/2)^{2j}$对块的距离乘积轻松恢复$X \star Y$。 根据定理0.2，由于D_j的条目在$\{0, \ldots, (3/2)^j\}$中（以及具有值inf的条目，定理0.2可以轻松适应处理），这需要时间

$$\widetilde{\mathrm{O}}\left((3/2)^{2j} \left(\frac{n}{(3/2)^j}\right)^{\omega} (3/2)^j\right) = \widetilde{\mathrm{O}}\left(n^{\omega} ((3/2)^{3-\omega})^j\right).$$

在$\lceil \lg_{3/2} P \rceil$阶段求和，我们得到时间

$$\widetilde{\mathrm{O}}\left(n^{\omega} \sum_{j:(3/2)^j<P} ((3/2)^j)^{3-\omega}\right) \leq \widetilde{\mathrm{O}}\left(n^{\omega} P^{3-\omega}\right).$$

□

我们现在可以完成Zwick定理的证明。实际上，结合命题1和命题2，并优化 P $(\text{at } P = n^{(3-\omega)/(4-\omega)})$，我们得到总运行时间为$\widetilde{\mathrm{O}}(n^{2+1/(4-\omega)})$。
观察到命题1和命题2的算法都计算节点对之间的距离的正确值或者过估计值（对于远距离节点是正确的）；因此，通过最小化这两个算法输出的距离，可以得到所有$u, v \in V$的确切距离 $d(u, v)$。

1 距离乘积

上次我们定义了$n\times n$矩阵的距离乘积：

$$(A \star B)[i, j] = \min_k \{A(i, k) + B(k, j)\}$$

定理**1.1.** 给定两个$n\times n$矩阵A, B，其元素范围为[-M, M]，可以在$O(\tilde{} Mn^{\omega})$时间内计算$A\star B$。

2 全对最短路径的预处理

定理**2.1** (Yust**er**, Zwick '05)。假设G是一个有向图，边权值范围为[-M, M]，且没有负环。 那么我们可以在$O(\tilde{} Mn^{\omega})$时间内计算出一个$n\times n$矩阵D，对于每对u, $v\in V$，高概率下成立：

$$(D \star D)[u, v] = d(u, v)$$

它被称为距离乘积，因为从节点 u到节点 v的最短路径可以通过先沿着从 u到中间节点 k的最短路径，然后从 k到 v来创建。 我们正在寻找最方便的节点 k，以使这两个节点之间的距离之和最小。

请注意，这并不立即意味着存在一个快速的APSP算法，因为 D可能具有较大的条目，使得计算 $D\star D$变得昂贵。 然而，对于单源最短路径问题，我们有以下推论：

推论**2.1.** 设 $G = (V, E)$为一个具有边权值在 $\{-M, M\}$范围内且没有负环的有向图。
那么从源点 s到其他节点的单源最短路径可以在 $\tilde{O}(Mn^{\omega})$时间内计算出来。

证明。根据定理2.1，我们可以计算一个N×N矩阵 D，使得 $D\star D$是正确的全对最短路径矩阵，在$\tilde{O}(Mn^{\omega})$时间内。
那么对于所有的 $v\in V$，我们知道：

$$d(s, v) = \min_k D[s, k] + D[k, v]$$

对于所有的 $v\in V$，计算这个值只需要 $O(n^2)$时间。 由于 $\omega \geq 2$，整个计算过程的时间复杂度为$\tilde{O}(Mn^{\omega})$。 □

类似地，我们可以证明检测负环的速度很快，因为任何负环都包含一个简单的负权重环，因此对应着从 i到 i的长度≤N的路径。

推论**2.2**。设 G为一个有向图，边的权重在 $\{-M, M\}$之间。 那么负环检测可以在 $\tilde{O}(Mn^{\omega})$时间内完成。

注意：为了方便表示，假设 A是一个$m \times n$矩阵，$S, T \subseteq \{1, \ldots, n\}$。 那么，$A[S, T]$是由 S索引的行和 T索引的列组成的A的子矩阵。

现在我们证明我们的主要定理：

主要算法再次使用随机性和命中集引理，但这次我们不是每次都取新的随机样本，而是将 B_{j+1}作为 B_j 的随机样本。

定理*2.1*的证明。设 $\ell(u, v)$是从 u到 v的最短路径上的节点数。

算法**1**： YZ(A)

A是一个加权邻接矩阵；
将 D设为 A；
将 B_0设为 V；
对于 $j = 1, \ldots, \log_{3/2} n$执行以下操作
  令 D'为 D，但将所有大于 $M(3/2)^j$的元素替换为 ∞；选择 B_j为大小为 S_j的随机集合 $B_{j-1} =$ $\frac{c \cdot n}{(3/2)^j}$ 计算 $D_j \leftarrow D'[V, B_{j-1}] \star D'[B_{j-1}, B_j]$；计算 $D_j \leftarrow D'[B_j, B_{j-1}] \star D'[\overline{B_{j-1}}, V]$；对于每个 $u \in V$，$b \in B_j$，执行以下操作
    设 $D[u, b] = \min(D[u, b], D_j[u, b])$；
    设 $D[b, u] = \min(D[b, u], D_j[b, u])$；
返回 D;

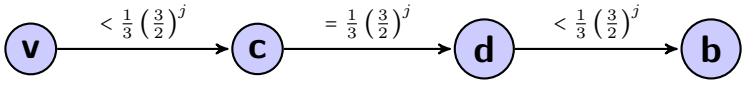
我们声称算法1是我们期望的算法。(期望的运行时间和正确性)
运行时间：在迭代 j中，我们将一个 $n \times \tilde{O}\left($矩阵与一个$\tilde{O}\left($矩阵相乘$\frac{n}{(3/2)^{j-1}}$ 矩阵，其中所有元素最多为$(3/2)^j$ 个节点的路径。$\frac{n}{(3/2)^{j-1}}\right) \times \tilde{O}\left(\frac{n}{(3/2)^j}\right)$矩阵，其中所有元素最多为 $(3/2)^j M$(我们将展示迭代 j只需要考虑最多有 $(3/2)^j$个节点的路径)。

因此，迭代 j的运行时间是$\tilde{O}\left(M(3/2)^j (3/2)^j \left(\frac{n}{(3/2)^j}\right)^\omega\right) = \tilde{O}\left(\frac{M n^\omega}{3/2)^{j(\omega-2)}}\right)$. 这个术语 $((3/2)^j (\frac{n}{(3/2)^j})^\omega)$是由于计算 D_j和 D_j时使用的阻塞造成的。 在所有迭代中，运行时间是渐近的，忽略多项式对数因子。

$$M n^\omega \sum_j ((3/2)^{\omega-2})^j \leq \tilde{O}(M n^\omega).$$

如果 $\omega > 2$，那么在$\tilde{O}$中的对数因子之一可以省略。
正确性：我们将通过证明两个命题来证明正确性。
命题**1**：对于所有的 $j = 0, \ldots, \log_{3/2} n$, $v \in V$, $b \in B_j$, 如果 $\ell(v, b) < (3/2)^j$那么在第j次迭代之后，几乎可以肯定有$D[v, b] = d(v, b)$
命题**1**的证明：我们将通过归纳法来证明它。 基本情况（$j = 0, \ell(v, b) < (3/2)^0 = \mathbf{1}$）是显然的，因为一跳路径的距离恰好是邻接矩阵。 现在，假设归纳假设对于 $j-1$成立，即如果最短路径（v, b）的长度为 $\ell(v, b) < (3/2)^{j-1}$，则我们已经正确存储了 $D[u, b] = d[v, b]$。 我们将展示 j的正确性。 考虑某个 $v \in V$和 $b \in B_j$。 我们根据节点 b距离 v的远近来考虑两种可能情况。情况一： $\ell(v, b) < (3/2)^{j-1}$**1**（b靠近）

根据我们的归纳假设， $D[v, b] = d(v, b)$ w.h.p.!
情况二： $\ell(v, b) \in [(3/2)^{j-1}, (3/2)^j)$ (b离得很远)
我们将需要使用上一堂课学到的“三分之一中间”技巧。



我们可以选择 $c, d \in V$ 使得：

$$\ell(v, c) < \frac{1}{3}\left(\frac{3}{2}\right)^j$$
$$\ell(d, b) < \frac{1}{3}\left(\frac{3}{2}\right)^j$$
$$\ell(c, d) = \frac{1}{3}\left(\frac{3}{2}\right)^j < \left(\frac{3}{2}\right)^{j-1}$$

通过一个击中集的论证，如果 c足够大， $B_{j-1} \cap$ “三分之一中间” $\neq \varnothing$(w.h.p. 取决于 c) 因为 $|B_{j-1}| = c \frac{n}{(3/2)^{j-1}}$ 对数 n.

令 x属于 $B_{j-1} \cap$ “中间的三分之一”。 那么 $\ell(v, x) \leq \ell(v, c) + \ell(c, d) < \frac{2}{3}(\frac{3}{2})^j = (\frac{3}{2})^{j-1}$. 由于 x属于 B_{j-1}，根据归纳 $D[v, x] = d(v, x)$在迭代 j时几乎必然发生。 通过类似的论证，我们得到几乎必然发生的是 $D[x, b] = d(x, b)$在迭代 j时（在迭代 j开始时）。

因此，在这次迭代之后， $D[v, b] \leq D[v, x] + D[x, b] = d(v, b)$.

作为一个小的技术说明，在进行乘法之前，我们需要实际上从D中删除大于 $(3/2)^j M$的条目，但它们是不需要的。

命题**2**：对于所有的$u, v \in V$，w.h.p. $(D \star D)[u, v] = d(u, v)$。

命题**2**的证明：固定$u, v \in V$，并且令 j为使得 $\ell(u, v) \in [(3/2)^{j-1}, (3/2)^j)$的值。观察 u和 v之间的最短路径。它的中间三分之一的长度为 $(1/3)(3/2)^j$。

但是 w.h.p. B_j在某个 $x \in V$上击中了这条路径，使得 $\ell(u, x)$、$\ell(x, v) < (3/2)^{j-1}$。 根据命题1，$D(u, x) = d(u, x)$且 $D(x, b) = d(x, b)$。因此：

$$d(u, v) \leq (D \star D)[u, v] \leq min_{x \in B_{j-1}} D(u, x) + D(x, v) \leq d(u, v)$$

证明完成。

□

3节点加权全对最短路径

现在我们将看到一个有趣的APSP变体，称为节点加权全对最短路径问题（现在权重与节点而不是边相关联），对于这个问题，我们可以得到一个真正的亚立方解，尽管对于APSP问题没有已知的真正的亚立方解。这个差距可能是固有的，因为将 $\approx$的 n^2权值映射到只有 n并且仍然保持成对最短路径信息是困难的。

这里我们证明了Chan的一个定理[1]

定理3.1. 节点加权 $APSP$可以在 $O(n^{9+\omega})$时间内计算。 $\frac{}{4}$)或 $O(2.84)$时间。

思路是通过一个击中集合的论证和多次运行Dijkstra算法来计算长路径（>跳数），运行时间为$\tilde{O}(^3)$ s)。然后，在 $O(sn^{3+\omega})$内处理短跳数)(^{2})时间通过一个专门的矩阵乘法。

设 G为一个带有节点权重 $w : V \to Z$的有向图。 假设我们只想计算长度为两个的路径上的距离。

设 A为无权重的邻接矩阵。 注意 $d_2(u, v) = w(u) + w(v) + \min\{w(j) \mid A[u, j] = A[j, v] = 1\}$（我们正在寻找通过的最便宜的邻居）。

假设我们复制了两份 A，并将其中一份的列按 $w(j)$的非递减顺序排序，将另一份的行按 $w(j)$的非递减顺序排序。

然后只需计算 $\min\{j \mid A[i,j] = A[j,k] = 1\}$，或者称为“最小证明”矩阵乘积。 我们使用了Kowaluk和Lingas [3]提供的算法：

引理3.1 (Kowaluk, Lingas '05). A, B的最小证人 ($n \times n$ 矩阵)在 $O(n^{2.616})$ or $O(n^{2+\frac{1}{4-\omega}})$时间内找到。

请注意，这个算法已经被Czumaj, Kowaluk和Lingas [2]改进过。

证明。设 p为我们稍后选择的某个参数。 将矩阵 A按列分成大小为 p的桶。
将矩阵 B按行分成大小为 p的桶。

对于每个桶 $b \in \{1, \ldots, \frac{n}{p}\}$，计算 $A_b \cdot B_b$（布尔矩阵乘积）。 这需要 $O((\frac{n}{p})^2 p^\omega)$时间 each，或 $O(n^2 p^{\omega-2})$时间 each。 但是有 $\frac{n}{p}$ 个这样的桶，所以总共需要 $O(\frac{n^3}{p^{3-\omega}})$ 时间。

然后对于所有的$i, j \in \{1, \ldots, n\}$，执行以下操作。设 b_{ij}为最小的 b，使得 $(A_b \cdot B_b)[i,j] = 1$。因此我们只需尝试所有 k在桶 b_{ij}中的选择，并返回最小的 k，使得 $A_b[i,k]B_b[k,j] = 1$。
这只是 n^2个穷举搜索，所以这一步的运行时间为 $O(n^2 p)$。 这个的直觉是，在排序之后，最小的证人 k，使得我们有 $A_b[i,k]B_b[k,j] = 1$ 是 i和 j之间寻找最短路径时最方便的邻居。将上述运行时间设置为相等（$\frac{n^3}{p^{3-\omega}} = n^2 p$）并平衡，我们得到应该设置 $p = n^{\frac{1}{4-\omega}}$ to 使得总时间为 $O(n^{2+\frac{1}{4-\omega}})$. □

直觉：在排序后，我们对矩阵进行的阻塞有以下解释：在排序后的邻接矩阵上，当我们进行阻塞时，就像根据它们的权重值将节点分组在一起。 所以如果我们有2个块，那么我们将有2个节点组：廉价节点，昂贵节点。 然后通过乘法过程，我们试图弄清楚通过不同的节点组可以到达哪些节点。

现在我们已经看到了如何处理距离为2的情况，我们可以继续处理更长的路径。 我们如何计算超过两个跳数的路径的距离？

我们将有两个参数p、s，我们稍后会选择它们以使运行时间最小化。 参数s用于区分“短”路径和“长”路径。 参数 p再次与矩阵的阻塞相关，这是方便的。 对于每个 $\ell \le s$，我们希望计算 D_ℓ，使得：

$$D_\ell[u,v] = d(u,v) - w(u) - w(v) \text{ if } \ell(u,v) = \ell$$
$$D_\ell[u,v] = \min_{j \in N(u)} \{w(j) + D_{\ell-1}[j,v]\}$$

这引出了一个新的矩阵乘积！假设我们已经给出了 $D_{\ell-1}$。 令 $\overline{D_{\ell-1}[u,v]} = w(u) + D_{\ell-1}[u,v]$。
然后我们对 $(A \odot D_{\ell-1})[u,v] = \overline{\min}\{D_{\ell-1}[j,v] \mid \overline{A}[u,j] = 1\}$感兴趣。

我们可以按照以下方式计算这个乘积。 再次，让 p成为我们稍后选择的参数。 对 $D_{\ell-1}$的列进行排序，使用 $O(n^2 \log n)$时间。 然后将每一列划分为长度为 p的块。如果 $D_{\ell-1}[u,v]$在 $\left(b^{\frac{n}{p}}\right)^{th}$ 和 $\left((b+1)\frac{n}{p}\right)^{th}$ 之间，则 $D_b[u,v]$=1。对所有的 b计算 A和 D_b的布尔矩阵乘积。 注意，如果存在一个 x使得$A[u,x]$ =1且 $D_{\ell-1}[x,v]$在排序后的 v^{th}列的第 b个块中，则 $(A \cdot D_b)[u,v]$=1。 我们可以通过穷举搜索的方式完成，尝试所有的 j使得$D_{\ell-1}[j,v]$在列v的第b个块中。

这需要 $O(\frac{n}{p} n^\omega)$时间进行乘法运算，并且需要 $O(n^2 p)$时间进行穷举搜索。这导致 $O(n^{3+\omega}/s^\omega + n^2 p)$时间。然而，我们需要这样做 s次。
因此，总运行时间为 $O(n^{3+\omega}/s^{\omega-1} + n^3/s)$时间，这在平衡后需要 $s = n^{\frac{3+\omega}{4}}$ 时间。

今天我们将介绍和解决两种All Pairs Shortest Paths (APSP)的变体，其运行时间为 $O(n^{3-\delta})$，其中某个常数 $\delta > 0$。在此过程中，我们还将介绍两个更多的矩阵乘积，即 $(\min, \leq)$ 乘积和支配乘积。

第4个最早到达

我们将研究的第一个APSP变体是最早到达问题。给定一个包含 n个机场的集合 V和一个包含 n个航班的集合 F。每个航班 $f \in F$由一个起始机场$ts \in V$，目的地机场 $t \in V$，出发时间和到达时间组成。

定义**4.1.**从 s到 t的有效行程是一个航班序列 $f_1, \ldots, f_k$，使得对于所有的 $i \in \{1, \ldots, k\}$，起点(f_{i+1})=终点(f_i)且出发时间(f_{i+1})≥到达时间(f_i)。

最早到达问题是计算所有机场$u, v \in V$的最早到达时间的问题。这个问题有一个自然的图解释。考虑一个二分图 $G = (V \cup F, E)$。对于每个航班 $f \in F$，我们添加一条从起点 (f)到f的有向边$(source(f), f)$，权重为出发时间 (f)。然后，我们再添加一条从f到终点(f)的有向边$(f, destination(\mathrm{f}))$，权重为到达时间(f)。

在这个图上，一个有效的行程是一个非递减序列的边构成的路径，到达时间由最后一条边的权重给出。因此，最早到达时间等价于找到所有非递减的起点到终点路径上的最小最后一条边的权重。

定义**4.2.** 让A，B是$n \times n$矩阵。矩阵A和B的（最小，$\leq$）乘积，表示为$A \otimes B$，给出如下：

$$(A \otimes B)\ (i, j) = \min_k \{B(k, j) \mid A(i, k) \leq B(k, j)\}$$

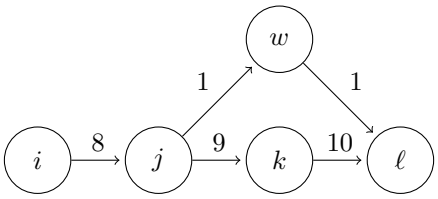
如果不存在这样的k，则为∞。

如果我们以自然的方式定义图G的邻接矩阵A，

$$A(i, j) = \begin{cases} w(i, j) & (i, j) \in E \\ 0 & \text{否则} \end{cases}$$

（在这里我们假设所有权重都是正数）然后我们发现（$A \otimes A$）(i, j)是长度为2的路径上的最小最后一条边的权重。通过迭代这个关系，我们发现（$A \otimes \cdots \otimes A$（$\ell - 1$次）$) \otimes A(i, j)$ 是最小的最后一条边在长度为 ℓ的所有路径上。

然而，我们必须小心，因为 $(\min, \leq)$ 乘积在一般情况下不是结合的，如下面的例子所示。考虑下面的图。



观察到（$A \otimes A$）$\otimes A$（i, ℓ）=10，但 $A \otimes$（$A \otimes A$）（i, ℓ）=∞。因此，我们不能简单地使用连续平方来解决最早到达问题。相反，我们的方法是计算“短路径”的最早到达时间，并使用前面讲过的随机抽样技术来处理“长路径”。大致思路如下。

算法2：最早到达(G)

生成邻接矩阵 A
设置 $D = A$
  对于 $i := 1$到 s 做
    计算 $D = D \otimes A$
  结束循环
  计算大小为 $c * n$的随机样本, S $\frac{}{s}$ 对于
  所有 $x \in S$做
    计算通过 x的所有对最早到达路径
  结束循环
  对于所有$i, j \in V$ 做
    $EA(i, j) = \min_{x \in S}$通过有效行程 $i \to x \to j$的最后一条边的最小权重
    $EA(i, j) = \min\{EA(i, j), D(i, j)\}$
  结束循环
  返回 EA

留作家庭作业练习，证明对于任意节点 $x \in S$我们可以在 $O(n^2 \log n)$时间内计算通过 x的所有对最早到达路径。

引理4.1. 如果可以在 $O(n^c)$时间内计算出 $n \times n$矩阵的$(\min, \leq)$乘积，则我们可以在 $O(n^{3+c}$时间内解决最早到达问题。 $\frac{}{2}$) 次.

*引理4.1*的证明。 使用上述概述的算法，我们得到一个运行时间为 $O(\frac{n^3}{s} + s(n^c))$。优化 s，我们设置 $s = n^{\frac{3-c}{2}}$ 并获得总运行时间为 $O(n^{\frac{3+c}{2}})$，如所需。 □

5个所有对瓶颈路径

（我们在课堂上跳过了，但你仍然应该学习它）

设图 $G = (V, E)$为具有边权重 $w : E \to \mathbb{Z}$的图。

定义5.1. 给定图 G中的路径 p，其瓶颈边是权重最小的边。

定义5.2. 全对最大瓶颈路径问题（$APBP$）是要找到所有对$u, v \in V$的最大瓶颈权重，其中 $u \to v$是所有路径中的最大瓶颈权重。

为了解决这个问题，我们需要定义另一个矩阵乘积。

定义5.3. 让 A和 B是一个 $n \times n$矩阵。 矩阵 A和 B的 $(\max, \min)$乘积，记作 $A \otimes B$，定义如下：

$$(A \otimes B)(i, j) = \max_k \min(A(i, k), B(k, j))$$

观察到 $(\max, \min)$乘积恰好是直径为2的图中的瓶颈路径问题。 留作练习，验证 $\otimes$是可结合的。 由于 $\otimes$是可结合的，并且 $A \otimes A$给出了长度为2的路径的最大瓶颈，我们可以使用连续平方来解决全对最大瓶颈路径问题。 这给出了以下引理。

引理5.1. 如果两个$n \times n$矩阵的（最大，最小）乘积可以在O~$(n^{\wedge c})$时间内计算，则我们可以在O~$(n^{\wedge c})$时间内解决所有对瓶颈路径的求解。

事实上，我们将证明计算$\otimes$等价于两个$\odot$乘积计算。

引理5.2. 如果存在一个$O(n^{\wedge c})$的算法来计算（最小，$\leq$）乘积，则存在一个$O(n^{\wedge c})$的算法来计算（最大，最小）乘积。

证明。考虑由（$A \circledcirc B$）$(\mathrm{i}, j) \circledcirc \max_k\{A(\mathrm{i}, k) \mid A(\mathrm{i}, k) \le B(\mathrm{i}, k)\}$定义的矩阵乘积。 注意，这个乘积实际上是一个（最小，≤）乘积。 特别地，它是通过对A和B中的所有条目a_i,j和b_i,j取反，然后交换矩阵A和B得到的乘积$-B \circledcirc -A$。
使用这个乘积，我们可以计算

$$(A \circledast B)(i, j) = \max\{(A \circledcirc B)(i, j), (B \circledcirc A)(i, j)\}.$$

因此，我们可以使用两个 $(\min, \le)$计算来计算 $A \circledast B$，如所需。 □

通过上述讨论，我们可以使用快速算法来解决所有对最早到达问题和所有对瓶颈路径问题进行 $(\min, \le)$ 计算。 本文的其余部分致力于寻找这样的算法。

6 用于计算 $(\min, \le)$乘积的快速算法

我们将在计算 $(\min, \le)$的算法中使用另一种特殊的矩阵乘积。

定义**6.1.** 矩阵 A和 B的支配乘积，记作 $A \odot B$，定义如下

$$(A \odot B)(i, j) = |\{k \mid A(i, k) \le B(k, j)\}|$$

定理**6.1.**（*Matousek*）两个$n \times n$矩阵A和B的支配乘积$A \odot B$可以在$O(n^{3+d})$时间内计算得到，则最小、≤乘积可以在 $O(n^{3+d})$时间内计算得到

$O(n^{\frac{3+d}{2}})$ 次.

假设6.1，我们首先证明6.2。

定理*6.2*的证明. 让A, B为两个$n \times n$矩阵。 我们将按照以下步骤计算 $A \circledcirc B$。

1. 对矩阵 B的每一列 j进行排序
2. 固定参数 p。 将每个排序后的列分成 $\frac{n}{p}$个连续的桶，每个桶包含 p个元素。 给每个桶命名，使得对于所有的桶 $b \le b'$，对于列 j中的桶 b中的每个元素$B(i, j)$，以及列j中的桶 b'中的每个元素$B(\ell, j)$，我们有 $B(i, j) \le B(\ell, j)$。
3. 对于每个 $b \in \{1, \ldots, \frac{n}{p}\}$，创建一个 $n \times n$矩阵 B_b，使得

$$B_b(i, j) = \begin{cases} B(i, j) & \text{if } B(i, j) \text{在列 } j \text{的桶 } b \text{中} \\ -\infty & \text{否则} \end{cases}$$

4. 计算所有桶 b的 $A \odot B_b$，即

$$(A \odot B_b)(i, j) = \begin{cases} \neq 0 \text{ 如果 } \exists k \text{使得 } B_b(k, j) \neq -\infty \text{且 } A(i, k) \le B(k, j) \\ 0 \text{ 否则} \end{cases}$$

5. 对于所有i, j确定 $b_{i,j}=$ 最小的 b使得 $(A \odot B_b)(i, j) \neq 0$。这等价于

$$\min\{B[k, j] \mid B(k, j) \text{在bucket } b(i, j) \text{中和 } A(i, k) \le B(k, j)\}.$$

因此，我们可以使用暴力法，如下所示。 对于所有的i, j检查每个 $B(k, j)$在bucket $b_{i,j}$ of j中，将其与 $A(i, k)$进行比较，并输出满足 $A(i, k) \le B(k, j)$的最小 $B(k, j)$. 观察到
这是 $(A \circledcirc B)(i, j)$.

该算法的运行时间主要由第4步中计算支配乘积和第5步中的暴力法决定。 使用6.1，我们可以在 $O(n^d)$时间内计算支配乘积。 因此，它的时间复杂度为 $O(\frac{n^{d+1}}{p})$ 计算所需的 $\binom{n}{p}$ 支配乘积的时间。 暴力步骤需要 $O(n^2 p)$时间。选择 $p = n^{\frac{d-1}{2}}$，我们得到总运行时间为 $O(n^{\frac{3+d}{2}})$，如所需。 □

还需要证明6.1

定理*6.1*的证明。 设A，B为 $n \times n$矩阵。 我们按以下方式计算 $A \odot B$。

1. 对于所有的 j，将 A的第 j列和 B的第 j行的条目集合排序在一起。 这将产生一个包含$2n$个元素的列表。

2. 将此列表分成每个包含 p个元素的桶。每个j有 $\frac{2n}{p}$ 个桶。

3. 对于所有的 $b \in \{1, \ldots, \frac{2n}{p}\}$，创建 $n \times n$ 矩阵

$$A_b(i,j) = \begin{cases} 1，\text{如果 } A(i,j)\text{在 } j\text{的第 } b\text{个桶中} \\ \text{否则为}0 \end{cases}$$

$$B_b(j,k) = \begin{cases} 1 \text{ 如果对于某个 } b' \le b \text{ 使得 } B(j,k)\text{在桶 } b'\text{中的}0 \text{ 否则} \end{cases}$$

4. 对于不同的桶 b，计算整数矩阵乘积

$$(A_b B_b)(i,j) = |\{k \mid A(i,k) \in b,\ A(i,k) \le B(k,j),\ \text{and } B(k,j) \notin b\}$$

   我们使用暴力搜索处理相同的桶 b。 对于所有的i, j和桶 b，将 $A(i,k)$与在与 $A(i,k)$相同的桶中的所有 $B(k,j)$进行比较，并更新输出中的总和。

暴力搜索步骤需要 $O(n^2 p)$时间。 然后，我们需要O($\binom{n}{p} n \omega$)时间来执行矩阵乘法。 我们通过取 $p = n^{3-\omega}$来最小化 $\frac{}{2}$ 从而得到最终运行时间为 $O(n^{3+\omega})\frac{}{2}$)，如所需。 □

第7节 结论

在本讲中，我们看到了许多在许多应用中有用的不同矩阵乘积。 距离乘积对于APSP预言机非常有用，它的一个轻微变体对于节点加权APSP问题（NW-APSP）也很有用。 $(\min, \le)$乘积（不是结合的）在搜索非递减路径和在所有对最早到达问题（APEA）中有应用时很有用。 $(\max,\min)$乘积在搜索所有对瓶颈路径（APBP）时使用。 最后，我们定义了优势乘积。 使用所有这些，我们得出结论，如果我们使用当前值 ω，则NW-APSP、APEA、APBP确实是亚立方的，分别具有 $O(n^{2.84}), O(n^{2.9}), O(n^{2.8})$的时间复杂度算法。 然而，对于一般的APSP问题，找到一个真正的亚立方算法仍然是一个重要的未解问题。

参考文献

[1] T.M. Chan. 更多算法用于加权图中的全对最短路径. *SIAM J. Comput.*, 39(5):20752089, 2010.

[2] Artur Czumaj, Miroslaw Kowaluk, 和 Andrzej Lingas. 更快的算法用于在有向无环图中找到最低共同祖先.理论计算机科学, 380(1):3746, 2007.

[3] Miroslaw Kowaluk 和 Andrzej Lingas. 有向无环图中的LCA查询. 在 *Automata, Languages and Programming*, 页码241248. Springer, 2005.

[4] Raphael Yuster 和 Uri Zwick. 使用快速矩阵乘法回答有向图中的距离查询. 在 *FOCS*, 页码389396, 2005.

CS367 第8讲 完美匹配
记录员：朴宰贤 日期：2014年4月28日
编辑者：伊兰・古德曼 更新日期：2015年10月16日

1 最短路径的后继矩阵

到目前为止，我们学习了如何在各种矩阵乘积下计算最短路径距离，但从未展示如何计算对应于这些最短距离的路径。 这些笔记将重点介绍如何通过修改我们之前学习的算法来找到实际路径。

在笔记中，我们将只使用一种类型的矩阵乘积，即$(\min,)$乘积，定义为$(A\ B)[i, j] = \min_k(A[i, k]\ \ B[k, j]$)，其中操作： $\mathbb{Z} \to \mathbb{Z}$。 我们只考虑使得这种矩阵乘积满足结合律的操作。 请注意，我们在之前的讲座中考虑的许多问题，如布尔矩阵乘法或有向加权图上的全对最短路径，都属于这个框架。 此外，在这些笔记中，我们将使用符号 i

j表示从 i到 j的路径，它是否表示最短路径应该从上下文中清楚。

为了激发讨论，假设我们想要计算所有节点对之间的最短路径。
然而，一般情况下，输出的大小可以达到 $\Omega(n^3)$，因此许多相关问题变得不再有趣。 因此，我们将寻找一个后继矩阵，它基本上具有与所有最短路径列表相同的表示能力，但只使用$\tilde{O}(n^2)$ 的内存。

定义一个后继矩阵 S是一个 $n \times n$矩阵 S，其中我们定义S$[i, j] = k$，使得 k是在 i之后的下一个节点上的 i最短路径。 j最短路径。 对于退化情况 $i = j$，则 $S[i, i] = i$。

这个定义立即给出了一种使用后继矩阵检索任意给定节点对之间最短路径的过程，假设路径是简单的[1]；从一个节点开始，继续到由后继矩阵给出的下一个节点，然后重复。 这表明找到一个后继矩阵基本上就像拥有整个最短路径列表一样有用。

命题**1.1.** 给定 S，对于任意i, j，我们可以输出一个最短 i j路径，时间复杂度与该路径的边数成线性关系。

证明。查看矩阵中的适当条目，并继续跟随，直到找到目标节点。 □

2证人矩阵

定义 A和 B的布尔矩阵乘积的证人矩阵 W是一个矩阵，使得对于任意i, j，如果存在一个 k使得 $A[i, k] = B[k, j] = 1$，则$W[i, j] =$ 这个 k，否则等于 ∞。

备注 一个证人矩阵不一定是唯一的：我们只需要计算一个这样的 k对于每个i, j，但可能存在多个。

假设我们可以在$\tilde{O}(\ n^\omega)$时间内计算两个{n} ×{n}矩阵的布尔乘积的证人矩阵。 我们将展示如何计算以下的后继矩阵：

1. 传递闭包
2. Zwick算法
3. Seidel算法

[1]我们考虑的最短路径问题都具有这样的特性：总是存在一条简单的最短路径。

2.1 传递闭包

回想一下，我们可以通过连续平方邻接矩阵（在布尔乘积下）来计算传递闭包。 布尔矩阵乘法（BMM）实际上可以看作是（max, ×）对 $\{0,1\}$条目的限制，其中 ×是普通的乘法；通过对其中一个矩阵的条目取反，BMM也可以表示为（min,）乘积。

如果存在一条从 i到 j的路径，则后继矩阵 S的(i,j)元素应该给出 i在某条简单路径中的下一个节点 j. 如果不存在路径，则 $S[i,j]$可以是任意值（这与后继矩阵的定义一致）。 然后我们可以使用以下算法：

算法1： TC(A)

$A^0 \leftarrow A$

对于所有的i,j，如果$(i,j)\in E$，则 $S^0[i,j]\leftarrow j$

对于每个 $k\in\{1,\ldots,\log n\}$执行

  $A^{(k)} \leftarrow A^{(k-1)}\cdot A^{(k-1)}$

  $W^{(k)} \leftarrow$上述乘积的证人矩阵

  对于所有的i,j， $S^{(k)} \leftarrow S^{(k-1)}[i, W^{(k)}[i,j]]$

返回 $A^{(\log n)}, S^{(\log n)}$

在这个算法中，如果存在一条长度$\leqslant 2^k$的 i到j的路径，则 $A^{(k)}[i,j]=1$。 路径长度$\leq 2^k$。 算法1返回传递闭包 T，以及后继矩阵 S。 为了验证其正确性，我们证明对于所有可达性路径长度$\leq 2^k$的节点对i,j， $S^{(k)}$是一个正确的后继矩阵。如果存在一条长度不超过2k的路径 i到j，那么W(k)的相应条目给出一个中间节点w，使得存在路径i到w和w到 j，它们的长度都不超过2k−1。路径i到j的后继可以从路径i到w的后继中获取。 路径长度最多为2^k，那么 $W^{(k)}$的相应条目给出一个中间节点 w，使得存在路径 i到w和w到j，它们的长度都不超过2k−1。路径i到j的后继可以从路径i到w的后继中获取。 w 和 w j，它们的长度都不超过2^{k-1}。路径 i到j的后继可以从路径i到w的后继中获取。 j可以从路径 i到w的后继中获取。 w，它已经在 $S^{(k-1)}$中计算出来。 基本情况 $k=0$ 直接从邻接矩阵中获得。 这表明 $S^{(k)}$确实是一个正确的后继矩阵。

2.2 Zwick算法

在Zwick算法中，我们查看一个最短路径，使得最短路径的边数在 $\left(\frac{3}{2}\right)^{k-1}$ 和 $\left(\frac{3}{2}\right)^{k}$ 之间。 这个想法是中间部分的长度约为 $\sim \left(\frac{3}{2}\right)^{k-1}$，所以我们从这个随机样本中选择 s，并且两边的尾部（路径的部分）的长度都$\leq$

在第 j步，我们计算以下维度的矩阵的一些（min,+）乘积：$\binom{\frac{3}{2}}{n}^{k-1}\times$ $\frac{n}{\left(\frac{3}{2}\right)^k}\star\frac{n}{\left(\frac{3}{2}\right)^k}\times n$。 我们希望找到这些乘积的证明。

定义A $(\min,\ )$-乘积为A, B的 C，使得 $C[i,j]=\min_k A[i,k]\quad B[k,j]$.

备注 布尔乘积是一个（最小，+）乘积：$A^{-}\leftarrow$整数$\{0,1\}$对应于布尔A和$B^{-}[\mathrm{i},j]=$-1如果$B[\mathrm{i},j]=1$，否则为0.

定义 A见证矩阵对于（最小，）是W，使得对于所有$\mathbf{i},j$，$W[\mathbf{i},j]=arg\ m\mathbf{in}_k A[i,k]\quad B[k,j]$.

如果在$\{$-$M,\cdots,M\}$范围内的矩阵上的（最小，+）见证可以在$O^{\sim}(M^{\wedge}n^{\omega})$时间内计算出来，则可以在$O^{\sim}($ M-$0^{\cdot 68}n^{2.575})$时间内找到Zwick算法的后继矩阵·

备注 这些证人矩阵的目的是，我们不是要找到最短路径的长度，而是要得到实际的最短路径。 这样做会多花一点钱，但不会多到可观。

2.3 Seidel算法

我们介绍了Seidel在无向图中的APSP算法。 然而，对于Seidel算法，上述计算后继矩阵的方法不适用，因为Seidel算法在一个新图上递归运行，并且它以一种特殊的方式使用整数矩阵乘法来计算距离。 然而，我们可以证明，在无向图中，仅仅给出两两距离矩阵就足以在 $O(n^\omega)$时间内获得后继矩阵。

Seidel算法的一个主要思想是，对于所有的 i、 j和邻居 k， $d[i,j]$和 $d[k,j]$之间的差值最多为1，其中 d 是最短距离矩阵。 此外，任何满足 $d[k,j]=d[i,j]-1$的 k都是 i在路径 i中的有效后继。

j. 现在，因为 $d[k,j]$只能与 $d[i,j]$相差1，我们知道$d[k,j]\equiv d[i,j]-1\pmod 3$意味着 k是一个后继。 根据距离矩阵 $D[i,j]=d[i,j]$，可以使用这个事实来计算后继矩阵，如下所示。

在$\tilde{O}(n^\omega)$时间内计算所有i,j的 $d(i,j)$（例如，通过Seidel算法）。对于所有的 $s\in\{0,1,2\}$，设置

$$D^{(s)}[k,j]=\begin{cases}\text{如果 } D[k,j]\equiv s-1 \pmod 3\text{，则为 } 1\\ \text{否则为0.}\end{cases}.$$

设 A为邻接矩阵。 计算 $A\cdot D^{(s)}$（布尔乘积，时间复杂度为 $O(n^\omega)$）对于每个选择的 s
和这个乘积的证明矩阵 $W^{(s)}$。 对于所有的i,j，设 $s_{ij}\equiv d(i,j)\bmod 3$，并设置 $S[i,j]=W^{(s_{ij})}[i,j]$。
为了看出这是正确的，固定 i和 j，考虑乘积 $A\cdot D^{(s_{ij})}$的(i,j)项。 对于这个项有贡献的索引 k满足 $A[i,k]=D^{(s_{ij})}[k,j]=1$。 根据构造，这意味着 $D[k,j]\equiv D[i,j]-1\pmod 3$，根据先前的观察，我们可以得出结论 k是路径 i的正确后继。

j. 证人是一些 k，使得 $A[i,k]=1$ 并且 $d(k,j)=d(i,j)-1 \bmod 3$，即 $W^{(s_{ij})}[i,j]$ 是后继。

3 计算证人矩阵

在本节中，我们展示了一个用于计算与矩阵乘积
$A\quad B$相关的证人矩阵的算法，以便可以按照前面部分的描述计算后继矩阵。

在处理一般情况之前，我们首先关注一个更简单的问题变体，其中证人是唯一的。 从那里，我们将展示如何轻松找到任何证人矩阵。

定义 A和 B的（最小，）-乘积的唯一证人矩阵 U是一个矩阵，使得对于所有的i,j，
$U[i,j]=k$，其中 k是唯一的列，使得（$A\ B$）$[i,j]=A[i,k]\quad B[k,j]$，如果不存在唯一的证人，则为∞。

特殊情况：BMM的唯一证人。给定A，B，创建 A'，使得 $A'[i,j]=j$如果 $A[i,j]=1$，否则为0。 乘以 $A'\cdot B$（整数乘积）。 如果 k是i,j的唯一证人，则 $\sum_l A'[i,l]B[l,j]=k$。

策略： $C\leftarrow A'\cdot B$，并对所有i,j检查 $A[i,C[i,j]]=B[C[i,j],j]=1$。 如果是这样的话， $U[i,j]\leftarrow C[i,j]$，否则 $U[i,j]\leftarrow\infty$。

定理3.1。如果可以在 $T(\ n\ )$时间内计算$(n\times n)$矩阵的 $(\min,)$乘积，则可以在 $O(T(n)\log n)$时间内计算 U。

符号：对于 $S\subseteq[n]$， $A[\cdot,S]$ 是由 S中的列组成的 A的子矩阵。
定理*3.1*的证明。对于所有的 b从*1*到$\log n$定义 $S_b=\{j\mid j$的第b位是*1*$\}$。令 $C\leftarrow A\ B$。 计算$(\min,)$： $C\ b\leftarrow A(\cdot,S_b)\quad B(S_b,\cdot)$。 对于所有的$i,j,b$，如果 $C[i,j]=C_b[i,j]$，则将 $U[i,j]$ 的第$_b$位设为*1*，否则设为0。 最后，检查结果 $\forall i,j$，如果 $A[i,U[i,j]]\quad B[U[i,j],j]=\ C[i,j]$，则$U[i,j]\leftarrow\infty$（如果证人不唯一，可能会得到垃圾值）。

该过程计算（最小，）乘积对数 n次，因此总运行时间为 O（T（n）$\log n$）。 很容易看出为什么该算法是正确的：固定 i和 j，它们有唯一的证明 k。 然后在每个 b处， $W[i,j]$的第 b位将被正确设置。

□

最后，我们展示如何计算更一般的证明矩阵，给定一个计算唯一证明的算法（从计算 U中获取 W）。为了做到这一点，我们进行随机抽样并使用上述算法。 首先，我们声明以下内容：

引理3.2. 让 s是 0和 $\log n$之间的某个值。 让（i, j）在 A和 B的（最小，）乘积中有 c个证明，使得 $\frac{n}{2^{s+1}} \leq c \leq \frac{n}{2秒}$。 设 S为大小为 2的随机样本，其中元素取自 $\{1, \ldots, n\}$。 那么 S以概率$\geq$ [1]包含一个唯一的证人 (i, j)。 $\overline{2次方}$

如果引理3.2成立，对于所有的$\log n$值为 s，重复以下步骤 $d \log n$次（其中d >3为某个常数）：从 $[n]$中随机选择一个大小为2^s的样本 S，找到矩阵 U作为 $A(\cdot, S)\ \ B(S, \cdot)$的唯一证人，并且对于所有的$i, j$，如果 $U[i, j] < \infty$，则将 $W[i, j] \leftarrow U[i, j]$。

对于任意的(i, j)和 s，使得对于(i, j)的证人数量在 $\left[\frac{n}{2加1}, \frac{n}{2秒}\right)$，在任何一个 $d \log n$个样本中，(i, j)没有唯一的证人的概率是 $\leq \left(1 - \frac{1}{2的e}\right)^{d \log n} \sim \frac{1}{n的d}$. 根据并集定，所有的$(i, j)$都有证人的概率是 $\geq 1 - \frac{1}{n的d-2}$.

引理*3.2*的证明。 让 W为(i, j)的证人集合。 由于 S中有*2*的^{s}个元素，并且我们有命中 W中一个元素的概率为 c，但不命中其他元素的概率为：

$$\begin{aligned} Pr[|W \cap S| = 1] &\sim \left(\frac{c}{n}\right) \cdot 2的\left(1 - \frac{c}{n}\right)^{2的^s - 1} \\ &\geq 2的^s \frac{1}{2的s+1} \cdot \left(1 - \frac{1}{2秒}\right)^{2的^s - 1} \\ &\geq \frac{1}{2e}. \end{aligned}$$

□

正如引理3.2和其证明之间所述，这个事实可以用来设计一个随机算法，输出一个正确的证明矩阵。 我们不知道每个个体$(i,\ j)$对的证明数量，但我们知道它必须包含在区间内 $\left[\frac{n}{2加1},\ \frac{n}{2秒}\right]$ 对于某个 $s = 0, 1, \ldots, \log n$. 思路是遍历所有可能的 s值，并确定所有(i, j)索引对的证明，其中证明数量落入正确的区间。具体而言，算法可以写成以下形式。

算法2: 证明矩阵(A, B)

```
C ← A  B
重复
    对于 s = 0, 1, . . . , log n执行以下操作
        S ← {1, . . . . . ., n}的随机子集，大小为2^s
        尝试寻找矩阵乘积的唯一见证矩阵 W′，其中矩阵A[·, S]  B[S, ·]
        对于i, j = 1, . . . . . , n进行如下操作
            如果 A[i, W′[i, j]]  B[W′[i, j], j] = C[i, j]，那么
                W[i, j] ← W′[i, j]
直到 W的所有元素都确定；
返回 W
```

对于每次最外层迭代，每个 $W[i, j]$都以一定的概率填充正确的见证，一旦确定，就不会再改变。 有^{2}个待确定的条目，因此最外层迭代的预期次数为 $O(\log n)$。 我们通过这个最终结果得出结论。

定理3.3. 如果($\mathbf{m}$in,)乘积在$T(n)$时间内可计算，则找到证明矩阵需要$O(\ T(n) \log^3 n)$时间。

记录员：阿米尔・阿布德
编辑：辛西娅・戴
日期：2014年4月28日
更新日期：2015年10月23日

1 图中的匹配

本周我们将讨论在图中找到匹配的问题：一组不共享端点的边。

定义**1.1**(最大匹配).给定一个无向图 $G=(V,E)$，找到一个边的子集 $M \subseteq E$
使得每对边$e,\ e' \in M$都不共享端点 $e \cap e' = \emptyset$。

定义**1.2**（完美匹配）. 给定一个无向图 $G=$（V，E）其中 $|V|$ =N是偶数，找到一个大小为N/2的边集 $M \subseteq E$，使得每对边$\mathbf{e}$，$\mathbf{e}' \in M$不共享端点 $e \cap e' = \emptyset$。
也就是说，每个节点必须被匹配 M覆盖。

显然，任何最大匹配的算法都可以用来解决完美匹配问题。练习是证明如果可以在 T（N）时间内解决完美匹配问题，那么可以在时间$\tilde{O}$（T（2N））内解决最大匹配问题。 思路是二分搜索最大的 k，使得存在一个大小为 k的匹配 M满足$|M| \geq$K。 要检查是否存在这样的 M，我们可以在图上添加一个大小为N- 2K的团，并用所有可能的边将其连接到原始图。 新图将有一个完美匹配，当且仅当原始图有k条边的匹配。

我们将专注于完美匹配，并为其提供代数算法。由于上述简化，这也将意味着最大匹配的算法。思路是定义一个矩阵，当且仅当图有完美匹配时，该矩阵的行列式非零。

1.1 图的图特矩阵

定义**1.3.** 对于一个图 $G=(V,E)$，其中 $|V|=n$，下面的 $m \times n$矩阵 T是图G的图特矩阵：

$$T[i,\ j] = \begin{cases} 0 & \text{如果 } i=j\text{或者 } (i,\ j) \notin E \\ x_{i,j} & \text{如果 } (i,\ j) \in E\text{且}i<j \\ -x_{i,j} & if\ (i,\ j) \in E \text{ 且 } \ i>j \end{cases}$$

下面的定理是我们将讨论的所有完美匹配算法的核心。

定理**1.1**（图特）.对于任意图 $G=(V,E)$，图特矩阵 T的行列式非零当且仅当 G包含一个完美匹配。

$$\det(T)=0 \iff G\text{包含一个完美匹配。}$$

证明。根据行列式的定义：

$$\det(T)=\sum_{\sigma \in S_n}(-1)^{sign(\sigma)} \cdot \underbrace{\prod_{i=1}^{n} T[i,\ \sigma(i)]}_{f_\sigma} \tag{1}$$

其中 S_n是$[n]$的置换集合， $sign(\sigma)$是 σ的逆序对的奇偶性，即对于 σ中的每对$x<y$，有$\sigma(x)>\sigma(y)$。

我们将证明分为三个命题：

命题**1** 奇数循环的排列在 $det(T)$中相互抵消。

设 P为包含至少一个奇数循环的排列集合 S_n。 对于每个 $\sigma \in P$，设 C_σ为 σ中最小元素的奇数循环，并设 σ'为 σ的 C_σ反转。

例如，如果 σ= $(1,5)(2,3,4)(6,7,8)$，那么 σ'= $(1,5)(4,3,2)(6,7,8)$。

$sign(\sigma) = sign(\sigma')$，因此有

$$\prod_{i=1}^{n} T(i,\, \sigma(i)) = -\prod_{i=1}^{n} T(i,\, \sigma'(i))$$

因为奇数环 $C_\sigma = (z_1, ..., z_r)$ 导致第一项中的条目 $T(z_1, z_2)$ …… …… …… …… …… …… $T(z_r, z_1)$ 和条目 $T(z_2, z_1)$ …… $T(z_1, z_r) = (-1)^{\wedge}r \cdot T(z_1, z_2)$ …… $T(z_r, z_1)$。

命题**2** 仅含偶数环的 σ 对应于一个完美匹配。

对于 σ 中的任何偶数环 $C = (z_1, ..., z_r)$，我们可以选择边 (z_1, z_2)，(z_3, z_4)，…， ，(z_{2r-1}, z_{2r}) 加入匹配。 这些边是节点不相交的，并覆盖所有顶点。

请注意，定理1和定理2表明，只要 $det(T) = 0$，就存在一个 σ ，其中只有偶数个循环，因此存在一个完美匹配。

定理**3** 如果 G有一个完美匹配，那么 $det(T) = 0$。

假设 $(a_1, b_1), \ldots, (a_{n/2}, b_{n/2})$ 是一个完美匹配。 考虑排列 σ= $(a_1, b_1) \ldots (a_{n/2}, b_{n/2})$。 那么

$$\prod_{i=1}^{n} T(i,\, \sigma(i)) = \prod_{i=1}^{n/2} T(a_i,\, b_i) T(b_i,\, a_i) = \prod_{i=1}^{n/2} -(x_{a_i,b_i})^2$$

没有其他排列具有相同的变量，因此这个项不能抵消。由此可得 $det(T) = 0$。

□

行列式$\det(T)$是一个关于度数为n的二元多项式，因此计算起来可能很昂贵。

定理**1.2**（Lovasz）。 如果我们从 $\{1, \ldots, n^2\}$中均匀随机选择值 vi_j 为每个 xi_j，并且通过这些替换从T得到矩阵$T(\{v_{ij}\})$，那么当且仅当 $\det(\ T(\{v_{ij}\})) = 0$时，高概率下有 $\det(T) = 0$。

这给出了一个高概率下的完美匹配多项式时间算法。 为了证明这个定理，我们使用：

引理**1.1**（Schwartz-Zippel）。 设 P是一个关于域$\mathbb{F}$上的非零多项式，其度数为 d，并且定义在集合 $\{x_1, \ldots, x_N\}$上。. 如果我们从有限集合 $S \subseteq \mathbb{F}$中随机选择值 $v_1, \ldots, v_N$，并且通过将 $x_1 = v_1, \ldots, x_N = v_N$代入 P，那么 $P(\{\ v_i\})$至少有 $1 -$的概率等于0。 $\frac{d}{|S|}$.

对于$\det(T)$，我们有$\deg(\det(T)) = n$，因此选择 $|S| = n^2$就足够了。 然而，如果我们在$\mathbb{Z}$上工作，这个行列式的元素可能会非常大，我们只能得到 $O(n^{\omega+1})$的运行时间。 相反，选择一个大于等于^{3}的素数 p，并在$\mathbb{Z}_p$上工作。 如果 G有一个完美匹配 M，那么多项式$\det(T) \bmod p$包含非零项 f_σ。 $_M$ 因此，它是一个非零多项式，我们可以应用Schwartz-Zippel引理在 $O(n^\omega)$的时间内检查行列式是否为零。

2 寻找匹配

上述算法告诉我们在 $O(n^{\omega})$的时间内图中是否存在完美匹配。 在本讲和下一讲中，我们将讨论可以为我们找到完美匹配的算法。有一个简单的 $O(n^{\omega+2})$的解决方案：对于每条边 $e \in E$，从图中删除它并检查
是否仍然存在完美匹配，这需要 $O(n^{\omega})$的时间。 如果图不再包含完美匹配，将边放回并继续下一条边，否则将边从图中排除。 最后我们得到一个包含 $n/2$条边且包含完美匹配的图，完成。

今天我们将看到一个 $O(n^{\omega+1})$ 算法，下周我们将看到一个 $O(n^{\omega})$ 的算法。

2.1 Rabin-Vazirani算法

我们将证明这个定理。

定理**2.1**(Rabin-Vazirani)。 可以在 $O(n^{\omega+1})$时间内找到一个完美匹配。

考虑算法1。

算法**1**：RV(G)

$T \leftarrow T(\{v_{ij}\})$：Tutte矩阵模一个足够大的素数的随机替代；
如果 $\det(T) = 0$则
  返回无完美匹配；
设置 $M = \emptyset$；
当 $|M| < n/2$时
  计算 $N = T^{-1}$需要 $O(n^{\omega})$ 时间；
  找到 j使得 $N[1, j] = 0$ 且 $(1, j) \in E$；
  $M \leftarrow M \cup \{(1, j)\}$；
  $T \leftarrow T_{\{1,j\},\{1,j\}}$即从 T中删除行1和 j以及列1和 j；

我们使用符号 $T_{X,Y}$表示子集$_{X,\,Y} \subseteq [n]$，表示从 T中删除由 X索引的行和由 Y索引的列得到的矩阵。

显然，该算法执行 $O(n)$ 次计算，每次计算需要 $O(n^{\omega})$ 的时间，因此总运行时间为 $O(n^{\omega+1})$。 事实上，该算法在某个完美匹配中选择一些 $e= (1, j)$，并在 $G \setminus \{1, j\}$上进行递归。
我们在下面证明正确性。
回顾伴随公式：

$$T^{-1}[i,\, j] = (-1)^{i+j} \cdot \frac{\det(T_{\{i\},\{j\}})}{\det(T)},$$

因此在算法中我们有 $N[1,\, j] = 0$ 当且仅当 $\det(T_{\{i\},\{j\}}) = 0.$
根据行列式的定义：

$$\det(T) = \sum_{j=1}^{n} (-1)^{1+j} \cdot T[1,\, j] \cdot \det(T_{\{1\}\{j\}}),$$

因此，如果 $\det(T) = 0$，则存在 $j \in [n]$，使得 $T[1, j] \cdot \det(T_{\{1\}\{j\}}) = 0$，因此 $(1, j) \in E$并且 $\det(T_{\{1\}\{j\}}) = 0$。 因此，为了证明算法的正确性，只需证明后者也意味着 $\det(T_{\{1,j\}\{1,j\}}) = 0$（即在仅移除 $\{1, j\}, \{1, j\}$而不是 $\{1\}\{j\}$时）。为了证明这一点，我们需要使用 Tutte 矩阵的性质。 注意 T是一个反对称矩阵：$T = -T^{t}.$

命题**1.** 设 A为一个 $n \times n$的反对称矩阵，则：

1. A^{-1}是反对称矩阵。

2. 如果 n是奇数，则 $\det(A) = 0$。

3. (Frobenius) 设 $Y \subseteq [n]$ 满足 $|Y| = rank(A)$ 且 $A[[n], Y]$ 的列秩为 $ank(A)$，则 $\det(A[Y, Y]) = 0$。

证明2: $\det(\quad A) = \det(-A^t) = (-1)^n \det(A)$。 我们将使用3而不证明。

引理2.1. 如果 $\det(T_{\{1\},\{j\}}) = 0$，则 $\det(T_{\{1,j\},\{1,j\}}) = 0$。

证明. 不失一般性地假设 j= 2.. 根据性质2，我们知道 $A = T_{\{1\},\{1\}} = 0$，所以它的秩最多为 $n-2$. 根据我们的假设，$\det(T_{\{1\},\{2\}}) = 0$，所以$\det(T_{\{1\},\{2\}})$的秩为 $n-1$. 因此， $T_{\{1\},\{1,2\}}$的列秩为 $n-2$， A的秩为 $n-2$. A是反对称的，所以根据Frobenius性质，对于 $Y = \{3, \ldots, n\}$，有$\det(\ A[Y, Y] = det(T_{\{1\},\{2\}} = 0.$

□

第10讲 更多关于匹配的内容
记录员：迈克尔・P・金 日期：2014年4月30日
编辑：辛西娅・戴 更新日期：2015年10月23日

1 引言

在上一讲中，我们讨论了两种找到完美匹配的算法，这两种算法都利用了Tutte矩阵的随机替换来找到完美匹配中的边。 回顾一下，Tutte矩阵的定义如下。

$$T(i,j) = \begin{cases} 0 & \text{如果 } (i,j) \in E \\ x_{ij} & \text{if } i < j \\ -x_{ij} & \text{if } i > j \end{cases}$$

朴素算法使用Tutte矩阵的行列式作为判断是否存在完美匹配的依据，而Rabin-Vazirani算法通过利用Tutte矩阵的性质来找到要包含在匹配中的边。

算法1：朴素匹配(G)

对于每个 $e \in E$，执行以下操作
  如果 $\det T_{G\setminus\{e\}} = 0$，则执行以下操作
    从G中删除e;

因为我们可以用一个$O(\tilde{}(n^{\omega}))$的随机算法计算出T的行列式，这给我们一个$O(n^{\omega+2})$的基准。 现在，回想一下Rabin-Vazirani算法。

算法2：RV(G)

$T \leftarrow$Tutte矩阵模一些大于n^3的素数p的随机替换;
$M \leftarrow \emptyset$;
当$|M| < n^2$时 –
  $N \leftarrow T$的逆 // T求逆是运行时间的瓶颈
  找到j使得$N(1,j) = 0, T(1,j) = 0$;
  $M \leftarrow M \cup \{(1,j)\}$;
  $T \leftarrow T_{\{1,j\},\{1,j\}}$

请记住，RV的运行时间为 $O(n^{\omega+1})$，主要是由于每次迭代中计算Tutte矩阵的逆的while循环中的瓶颈。 在本讲中，我们将通过使用专门的算法在 $O(n^2)$时间内求逆Tutte矩阵，从而将其改进为 $O(n^3)$。

2 将Rabin-Vazirani改进为 $O(n^3)$

定理1(Mucha, Sankowski)。 RV可以实现更新N的时间复杂度为 $O(n^2)$。

推论2.1. 完美匹配可以在 $O(n^3)$时间内找到。

为了证明定理1，我们将首先证明一个引理，然后使用该引理在 $O(n^2)$时间内重新计算Tutte矩阵的必要部分。

引理2.1. 设 M为一个 $n \times n$可逆矩阵。令 $N = M^{-1}$。 令 M和 N具有以下形式。

$$M = \begin{array}{c|c|c} & k & n-k \\ \hline k & X & Z \\ \hline n-k & Y & W \\ \hline \end{array} \qquad N = \begin{array}{c|c|c} & k & n-k \\ \hline k & X & Z \\ \hline n-k & Y & W \\ \hline \end{array}$$

如果 X是可逆的，那么 W也是可逆的，且 $W^{-1} = W - Y \cdot X^{-1} \cdot Z$。

> 证明。我们知道 $M \cdot N = I$。 这意味着 $YX + WY = 0$。 根据假设， X是可逆的，所以$Y = -WYX^{-1}$。 我们还知道 $YZ + WW = I$。 我们可以将这些事实结合起来得到以下结果。
>
> $$(-WYX^{-1})Z + WW = I$$
> $$\implies W \cdot (W - YX^{-1}Z) = I$$

因此， W是可逆的，其逆矩阵为 $W - Y \cdot X^{-1} \cdot Z$。 □

注意，这个引理也适用于 M的列/行的排列。 特别地，这意味着我们可以将引理应用于图的Tutte矩阵，其中 X是由第1行和第 j行以及第1列和第 j列组成的2 ×2矩阵， W是 $T_{\{1,j\},\{1,j\}}$。 不失一般性，我们假设 j= 2。 然后，我们得到以下结果。

$$T^{-1}_{\{1,2\},\{1,2\}} = N_{\{1,2\},\{1,2\}} - N_{\{3:n\},\{1,2\}} \cdot N^{-1}_{\{1,2\},\{1,2\}} \cdot N_{\{1,2\},\{3:n\}}$$

我们知道 $N^{-1}_{\{1,2\},\{1,2\}}$ 存在，因为 N 是反对称的。

此外，我们声称我们可以在 $O(n^2)$ 时间内计算这个矩阵的逆。 求逆需要减去两个 $(n-2)\times(n-2)$ 矩阵，这需要 $O(n^2)$ 时间，并且需要通过一个 $(n-2)\times 2$ 矩阵与一个 2×2 矩阵的乘法来计算一个 $(n-2) \times (n-2)$ 矩阵，然后通过一个 $(n-2) \times 2$ 矩阵与一个 $2 \times (n-2)$ 矩阵的乘法来计算一个 $(n-2)\times(n-2)$ 矩阵。 这些乘法也需要 $O(n^2)$ 时间。 因此，通过对拉宾-瓦齐拉尼算法进行微小修改，我们得到了一个 $O(\mathrm{n}^3)$ 的完美匹配算法。 这个算法可以进一步修改，以在二分图中计算完美匹配，时间复杂度为 $O(\mathrm{n}^{\omega})$，但这个结果不能推广到其他图。

3 改进朴素搜索到 $O(n^{\omega})$

命题2（哈维）. 存在一个 $\tilde{O}(n^{\omega})$时间复杂度的算法可以在一般图中进行完美匹配。

哈维算法重新实现了完美匹配的朴素算法。 朴素版本使用图的图特矩阵的行列式作为判断是否存在完美匹配的预言机，并逐步删除不必要的边。 改进版本的关键思想是选择一种访问边的方式，使得检查 $G \setminus \{e\}$是否存在完美匹配变得廉价。 考虑以下引理。

引理3.1.设 M为一个可逆的 $n\times n$矩阵， $S \subseteq [n]$为一些小的条目子集。 设 $\tilde{M}$为一个 $n\times n$矩阵，使得如果 $\tilde{M}\ (i,\ j)\ = M(i,\ j)$，则$i,\ j \in S$。 最后，设$\Gamma = I_{|S|}+(\tilde{M}\ [S,\ S]-M[S,\ S])M^{-1}[S,\ S]$。那么以下陈述是正确的。

1. 矩阵 可逆当且仅当

$$\det(\Gamma) = 0$$

2. 如果矩阵 可逆，则其逆矩阵为

$$\tilde{M}^{-1} = M^{-1} - M^{-1}(:, S)\Gamma^{-1}(\tilde{M}[S, S] - M[S, S])M^{-1}[S, :]$$

根据 *1*的一个直接推论是，我们可以在 $O(|S|^{\omega})$ 时间内检查矩阵 是否可逆。 另外，假设我们只对矩阵 $^{-1}$ $[S, S]$ 感兴趣。 那么，我们可以通过在引理的 *2*中的每个子矩阵中使用一个 $|S| \times |S|$的子矩阵来计算这个子矩阵。 在这种情况下，计算矩阵 $^{-1}[S, S]$ 的总时间为 $O(|S|^{\omega})$，给定矩阵 M和 M^{-1}。

令 $N = T^{-1}$，且$\tilde{T}$ 是 T的转置，其中$\tilde{T}(i, j) = \tilde{T}(j, i) = 0$。

第**3**个命题。 检查$\det\tilde{T} = 0$是否在 $O(|S|^{\omega}) = O(1)$时间内完成。

为了证明这个命题，注意我们需要检查 $\det\Gamma = 0$，其中在这种情况下 Γ 如下所示。

$$\begin{aligned}\Gamma &= \begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix} + \begin{bmatrix}0 & -T(i, j)\\ T(i, j) & 0\end{bmatrix} \cdot \begin{bmatrix}0 & N(i, j)\\ -N(i, j) & 0\end{bmatrix}\\ &= \begin{bmatrix}T(i, j)N(i, j) + 1 & 0\\ 0 & T(i, j)N(i, j) + 1\end{bmatrix}\end{aligned}$$

因此， $G\setminus\{(i, j)\}$有一个完美匹配当且仅当 $T(i, j) \cdot N(i, j) = -1$。

因此，我们可以快速检测到完美匹配的存在，并且我们有$\tilde{T}$ 是替代 Tutte矩阵的 $G\setminus\{(i, j)\}$. 现在我们只需要看到如何重新计算$\tilde{T}^{-1}$，即更新版本 of N. 我们可以通过先前的引理以 $O(n^2)$ 时间来天真地做到这一点，但这只会给我们一个 $O(n^4)$ 算法. 我们需要做一些更复杂的事情来获得所需的 $O(n^{\omega})$ 运行时间.

让我们根据算法3从 $S_1 \times S_2$中删除边.

算法3： DeleteCross(S_1, S_2)

```
如果 |S1| = |S2|= 1那么
    S1 = {s}, S2 = {r};
    如果 T(s,r)· N(s,r) = −1那么
        T(s, r) = T(r, s) = 0    // 移除 (s,r)
        更新N;
否则
    S1 ← S11 ∪ S12, S2 ← S21 ∪ S22    // 将S1和S2分别划分为两个相等的子集
    对于i, j∈ {1, 2}执行
        删除交叉项    (S1i, S2j);
        更新N;
```

虽然这个递归过程可以工作，但问题是如何高效地更新 N。 解决方案是在DeleteCross(S_1, S_2)中每次 DeleteCross调用后只更新 $N[S_1 \cup S_2, S_1 \cup S_2]$。特别地，在DeleteCross$(S_1, S_2)$中，我们将保持不变式 ： $N[S_1 \cup S_2, S_1 \cup S_2]$将是正确的。 然后在基本情况下，当 $|S_1| = |S_2|= 1$时，我们将有正确的值 $T(s, r)$和 $N(s, r)$，并且我们可以正确地确定是否可以删除(r, s)。

在调用DeleteCross(S_1, S_2)后更新 N时，我们有调用前的旧 T和调用内的新$\tilde{T}$。 所有的变化都将在$(S_1 \cup S_2) \times (S_1 \cup S_2)$中进行，因此更新是足够的。
根据我们之前的观察和引理3.1，更新 N将花费 $O((|S_1| + |S_2|)^{\omega})$的时间。因此，DeleteCross$(S_1, S_2)$的总运行时间由以下递归关系给出，其中$n = |S_1| = |S_2|$。

$$\begin{aligned}&T(n) \le 4T(\frac{n}{2}) + 4O(n^{\omega})\\ \implies\ &T(n) = \tilde{O}(n^{\omega})\end{aligned}$$

我们还需要处理集合 S内的边界，以便将其适当地分割为 S_1和 S_2。 考虑
最终算法。

（在下面我们假设 $|S|$是2的幂。如果不是，我们可以向图中添加足够多的新节点：图的大小最多翻倍，新图具有完美匹配，当且仅当旧图有完美匹配。）

算法**4**：删除S内部的元素(S)

如果 $|S| = 1$则
  返回;
$S \leftarrow S_1 \cup S_2$ // 使得|S1| = |S2| = |S|/2
删除S_1内的元素(S1), 更新N(S, S);
删除S_2内的元素(S2), 更新N(S, S);
删除S_1和S_2之间的交叉元素，更新N(S, S);

算法4的运行时间由以下递归关系给出。

$$\begin{aligned} T(n) &\leq 2T(\frac{n}{2}) + T(\textsc{DeleteCross}(\frac{n}{2})) \\ &= 2T(\frac{n}{2}) + T(\tilde{O}(n^{\omega})) \\ \implies T(n) &= \tilde{O(n^{\omega})} \end{aligned}$$

找到匹配的最终算法是

算法**5**：匹配(G)

DeleteWithin(V);
返回剩余的边;

参考文献

[1] Marcin Mucha和Piotr Sankowski，通过高斯消元法求解最大匹配，FOCS，248-255 (2004).

[2] Nicholas J.A. Harvey，用于匹配和拟阵问题的代数算法，SIAM Jour$n$al on Computing, 39(2):679-702, (2009).